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

A deep dive into the Zephyr 2.5 (and 2.6) device model

MARTI BOLIVAR

NORDIC SEMICONDUCTOR

Me

- Zephyr 4.5 years
- NORDIC SEMICONDUCTOR
- West, devicetree

Previously (2019)

A Dive into Zephyr Device Driver Model

Tomasz Bursztyka, Intel Corporation

https://youtu.be/RYbKALYRYCM

Device model

Why

How

What’s new

Why?

Imagine an RTOS...

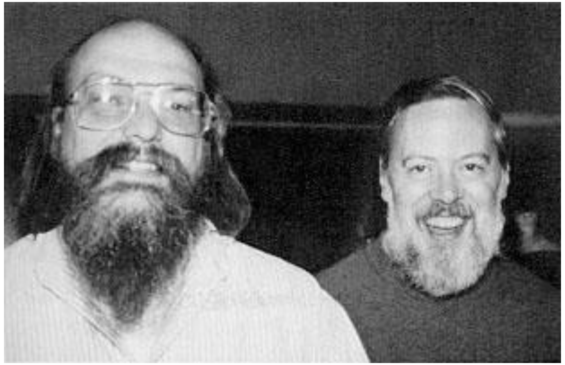
...with batteries included

Lots of people involved

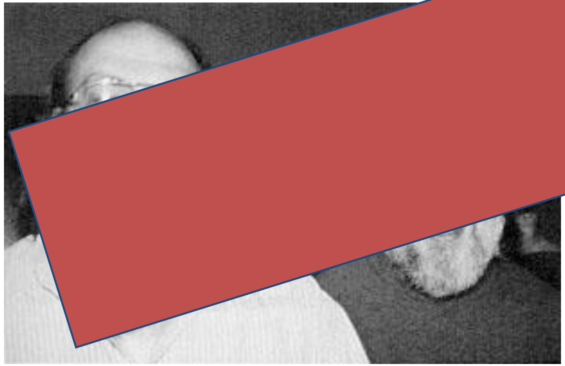
You too

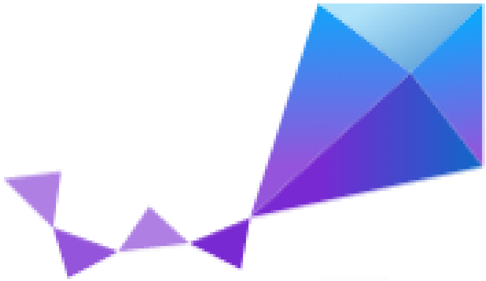
Framework

Everything is a file!



Every





Everything is a `struct device`!

devicetree
.org

Zephyr™ Project
Developer Summit

Devicetree → device

Alice In Wonderland's Cheshire cat

The cat vanishes, leaving its smile

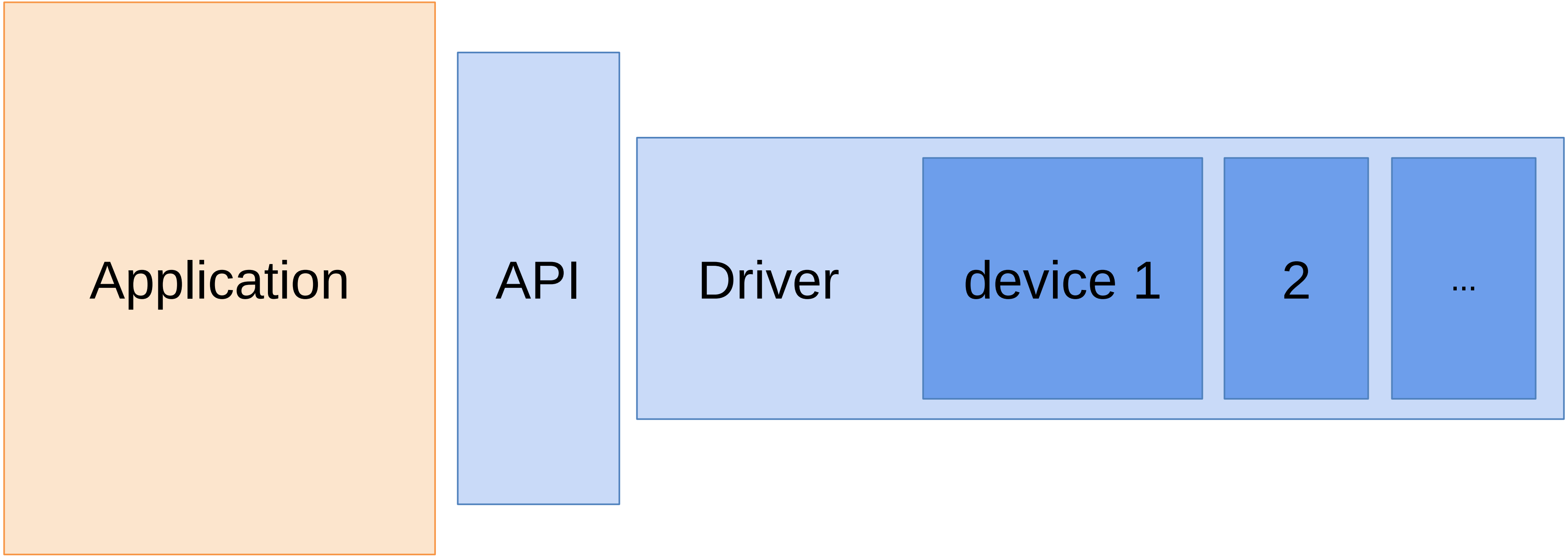
Setting up devices is like this

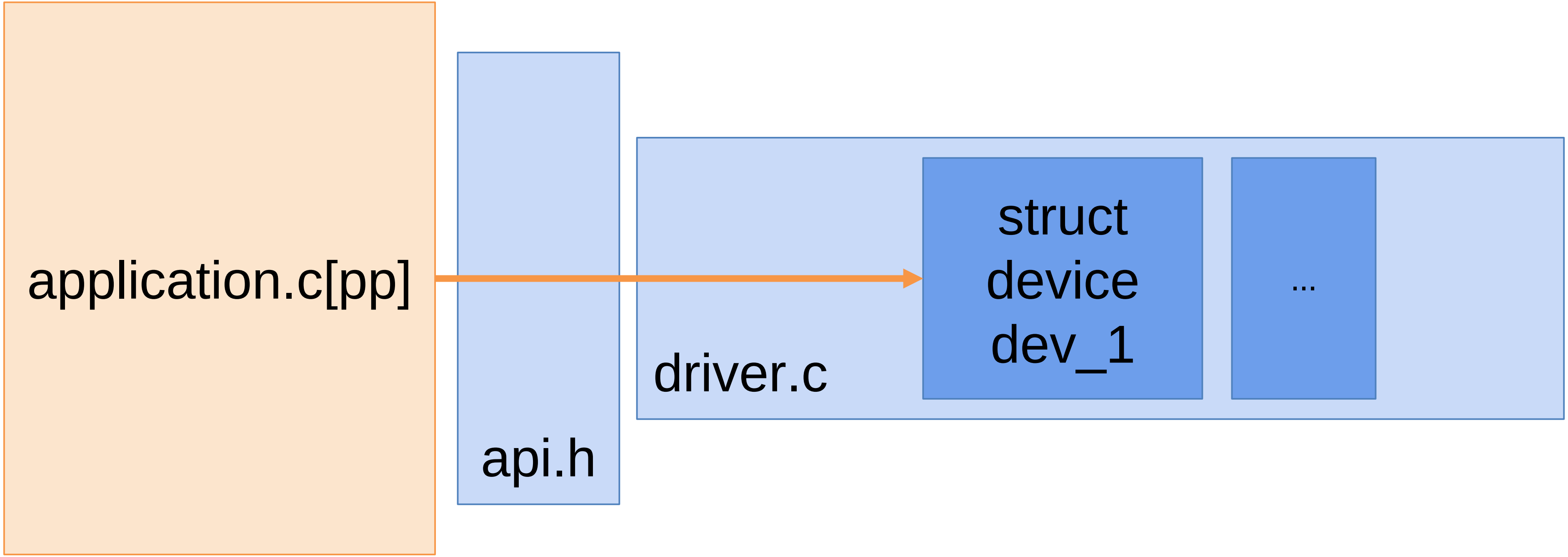
→

Hopefully, you’ll learn to love this

Zephyr™ Project
Developer Summit

APIs, drivers, and devices

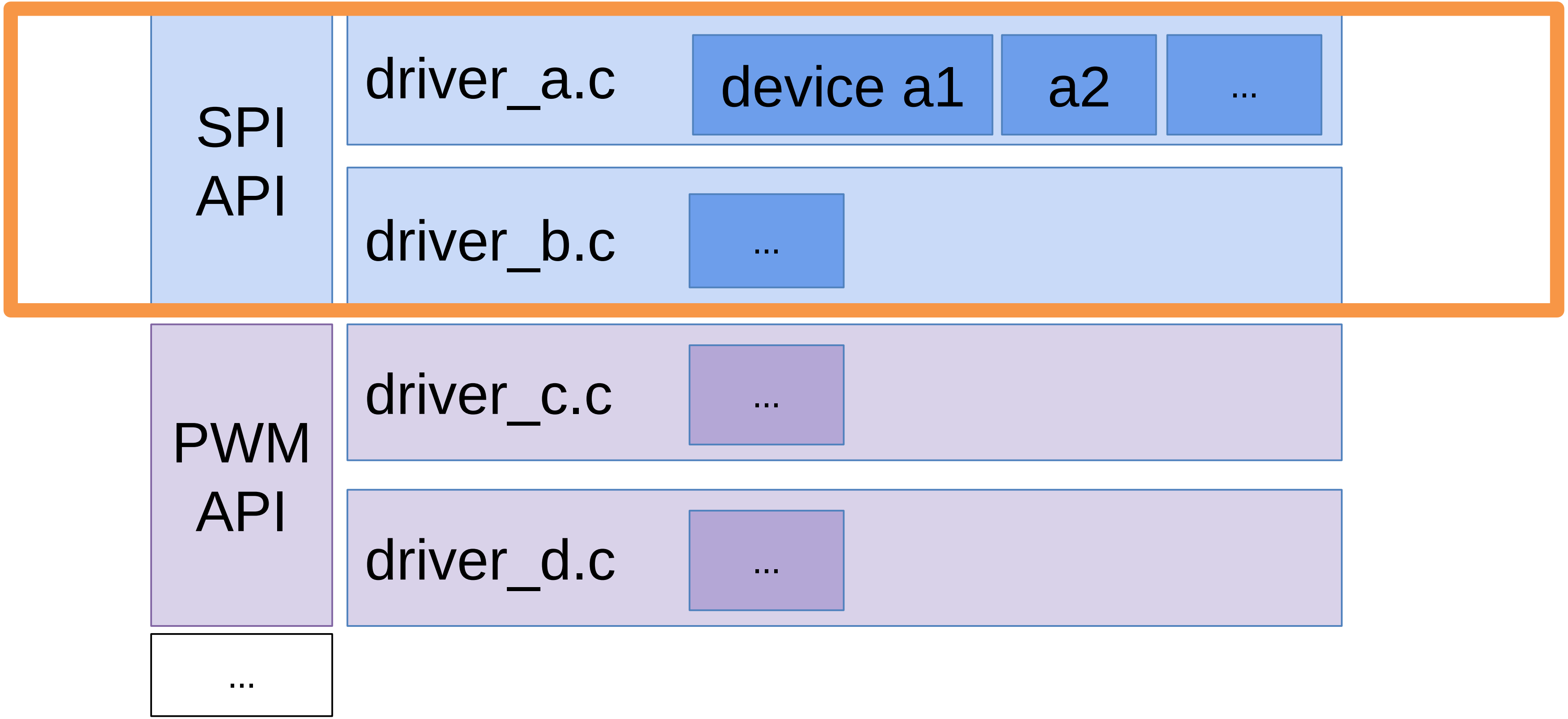

Application
API
Driver
device 1
2
...


application.c[pp]
api.h
driver.c
struct
device
dev_1
...

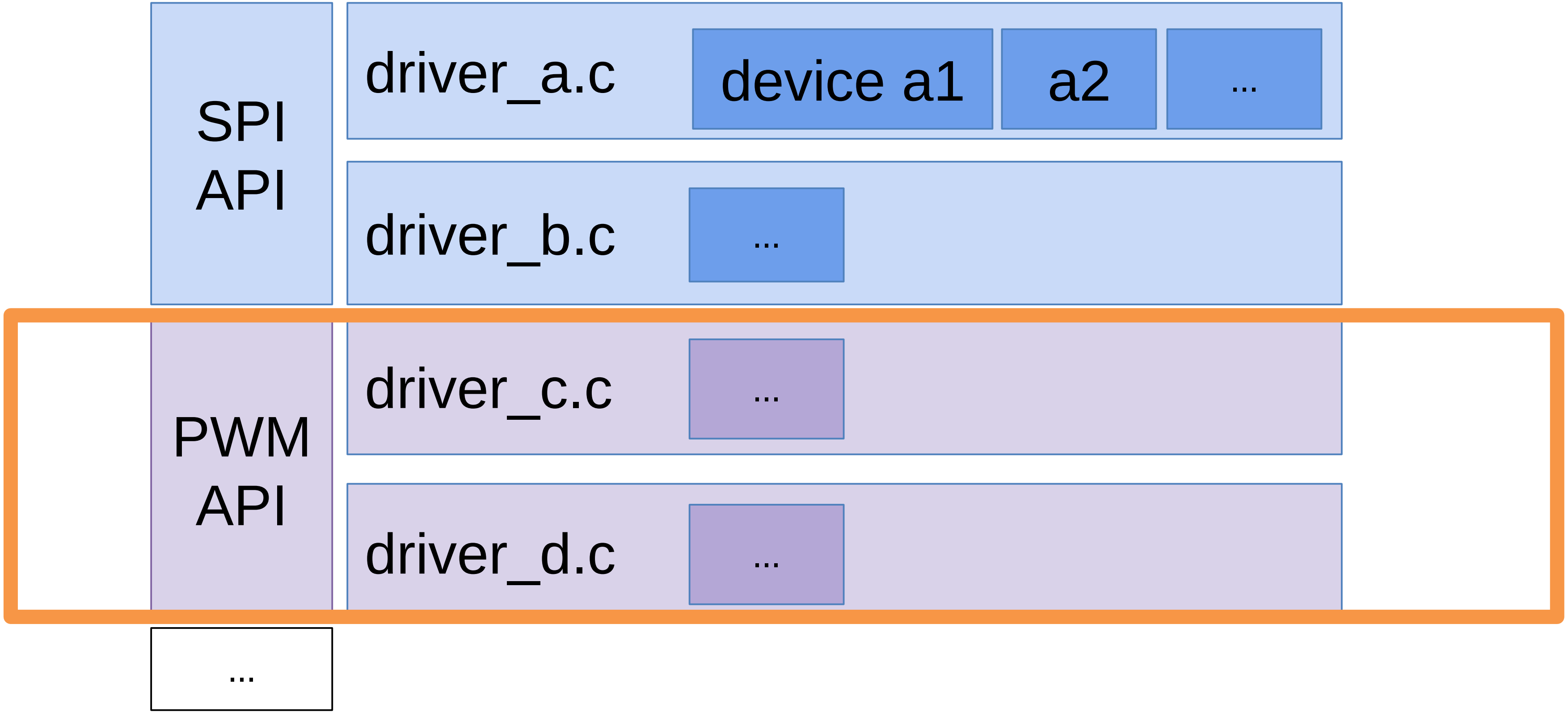
Many

- APIs
- drivers
- devices

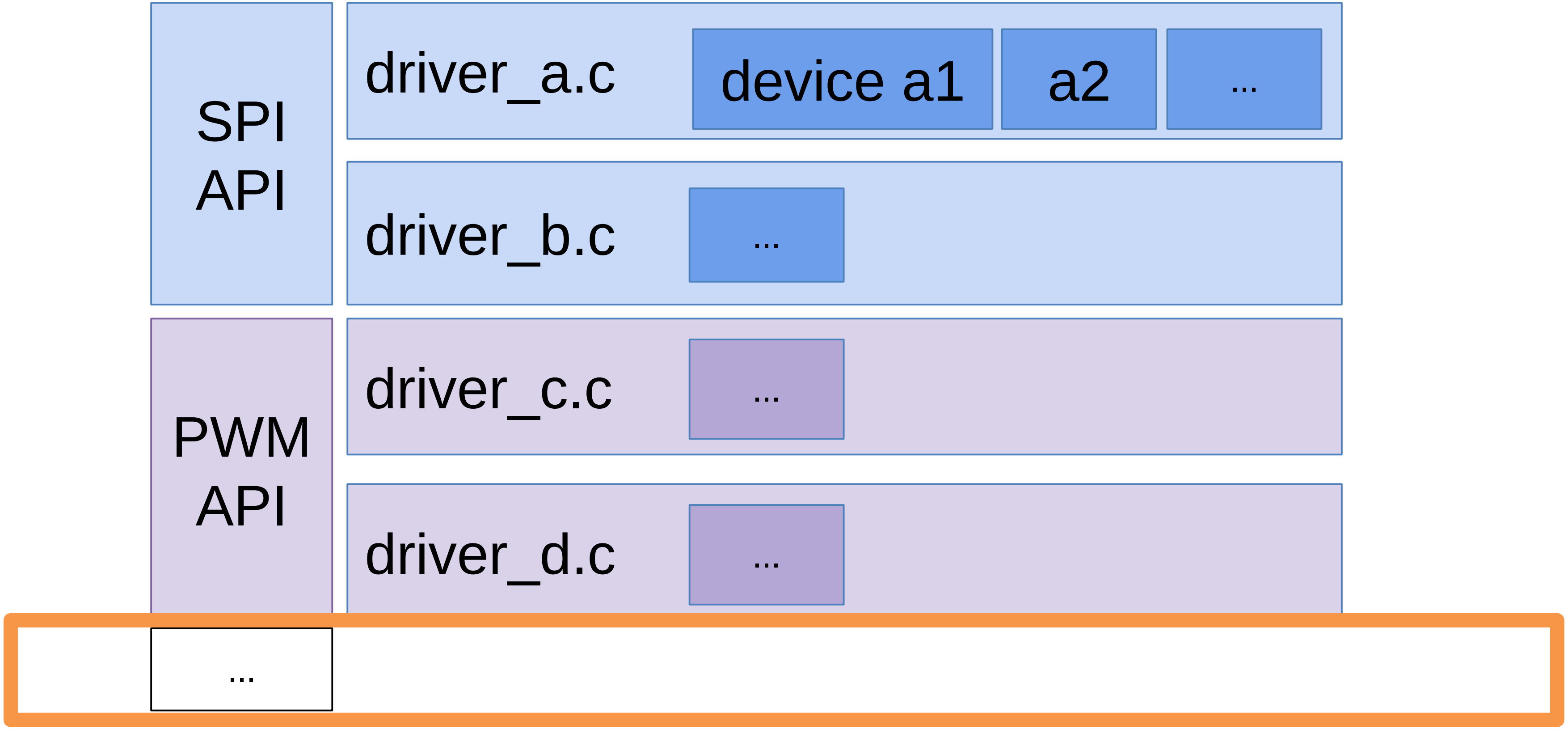
Many APIs, drivers, and devices



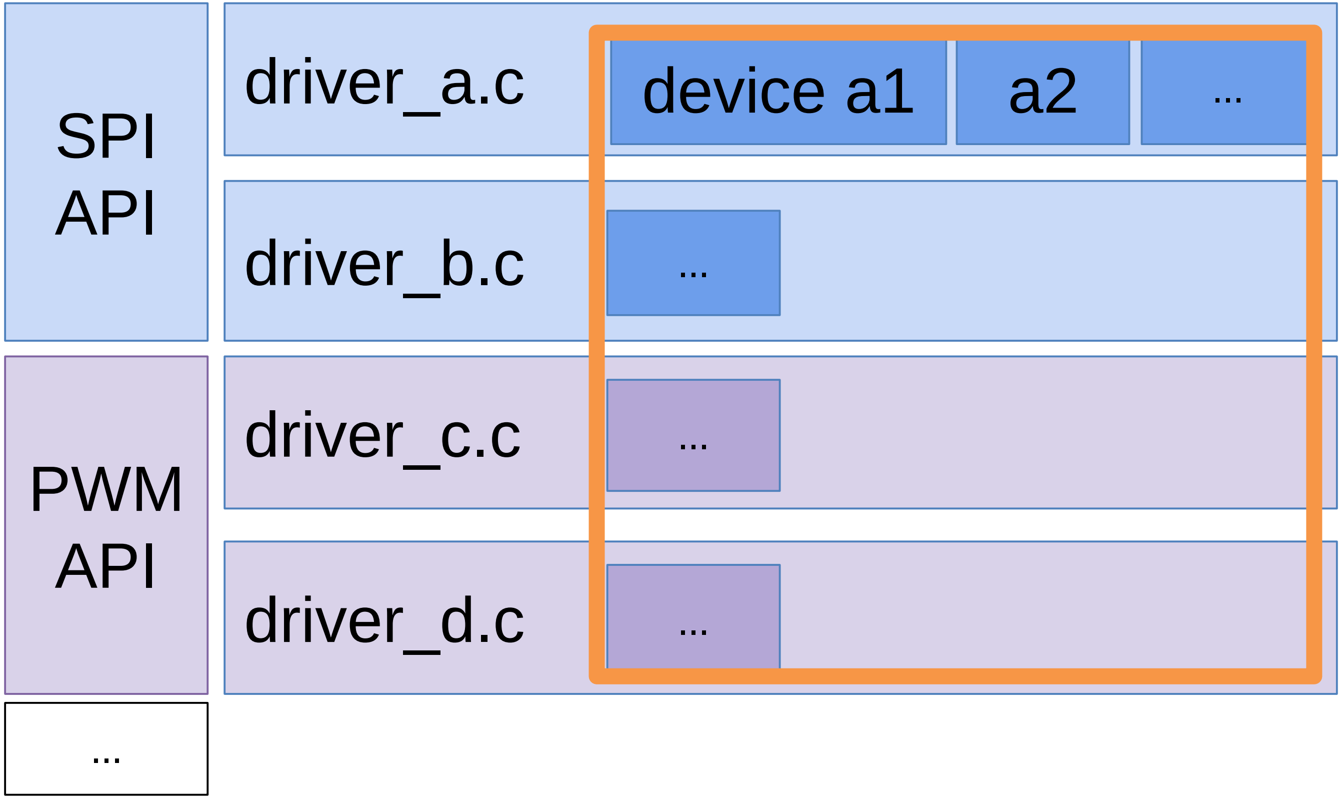
Many APIs, drivers, and devices



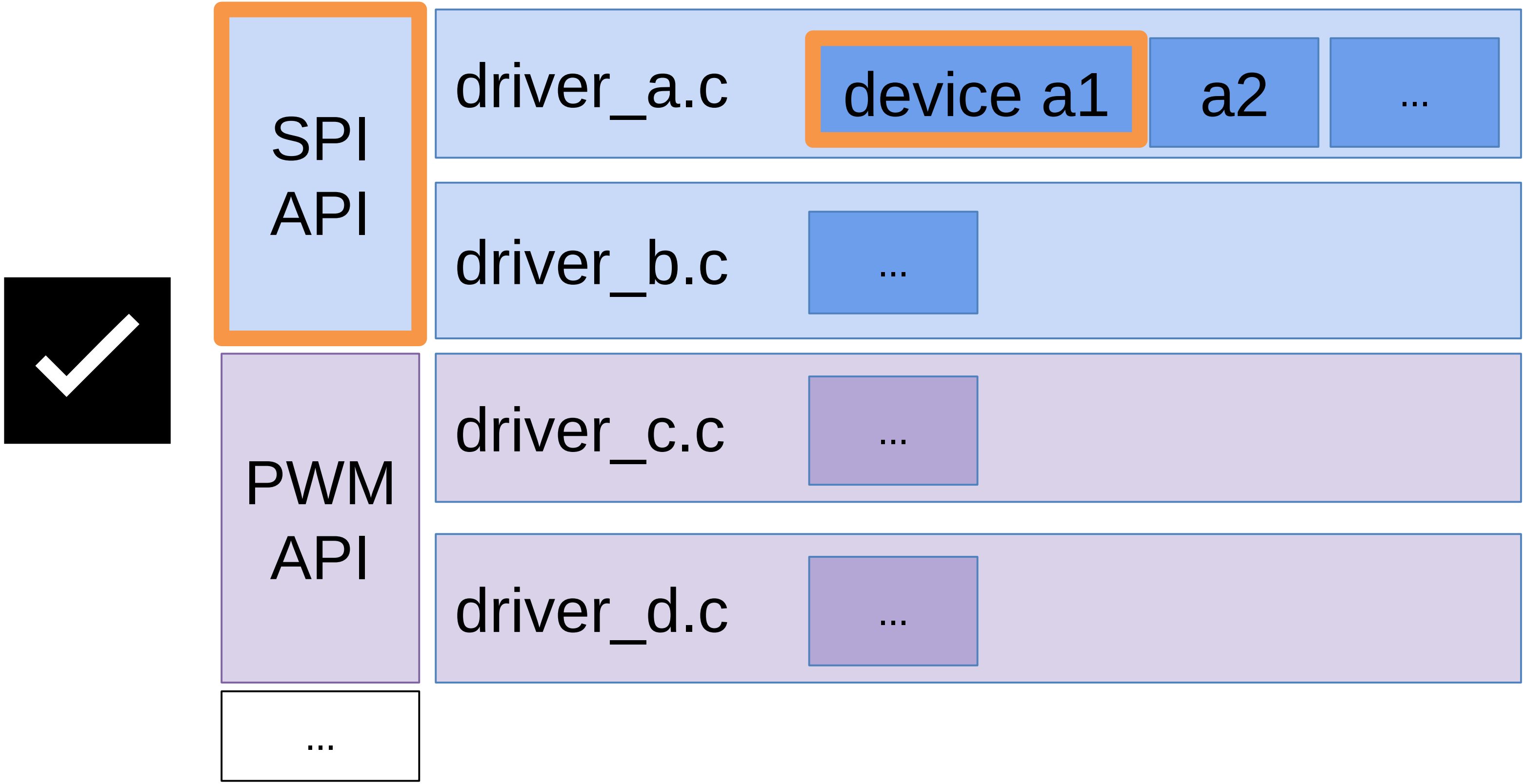
Many APIs, drivers, and devices



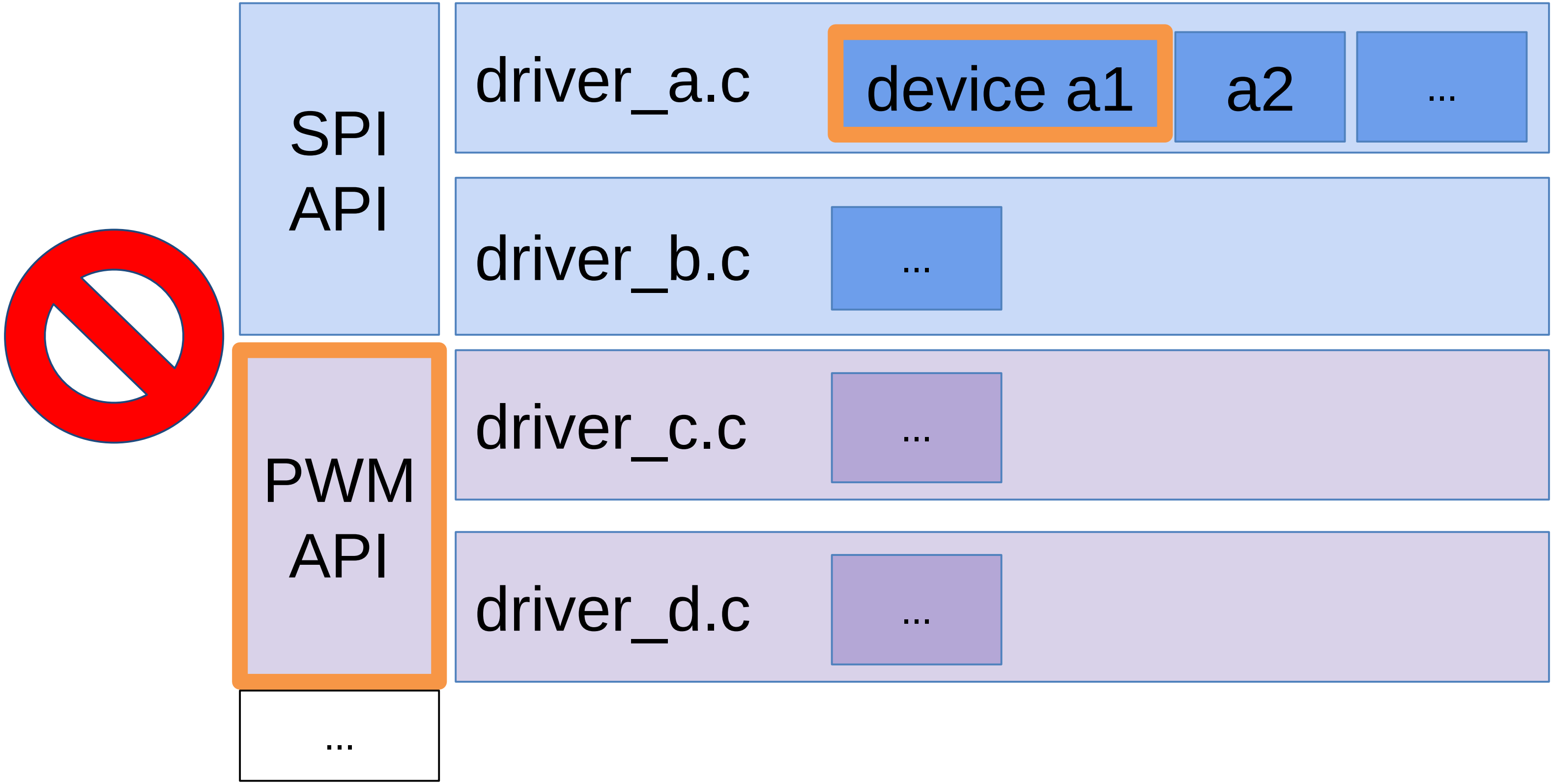
Everything is a `struct device`



Use the right API for each device



Use the right API for each device



SPI API

driver_a.c

device a1

a2

...

driver_b.c

...

PWM API

driver_c.c

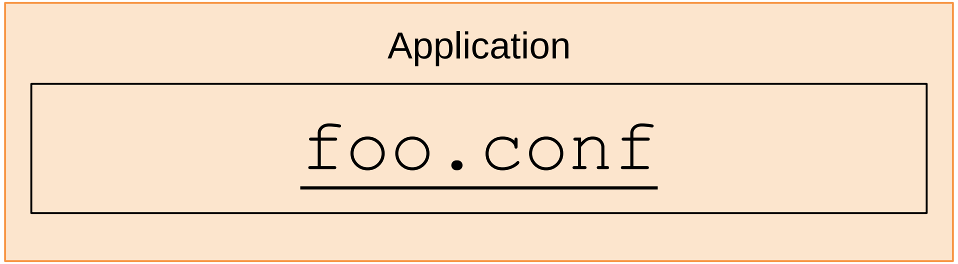
...

driver_d.c

...

...

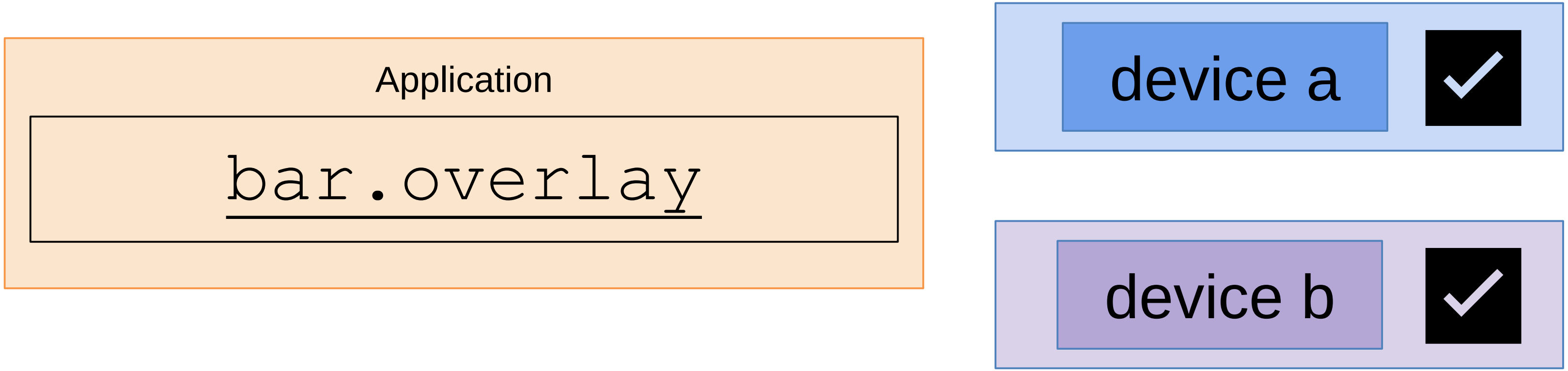
Select drivers with Kconfig



Application

`foo.conf`

Allocate devices with devicetree



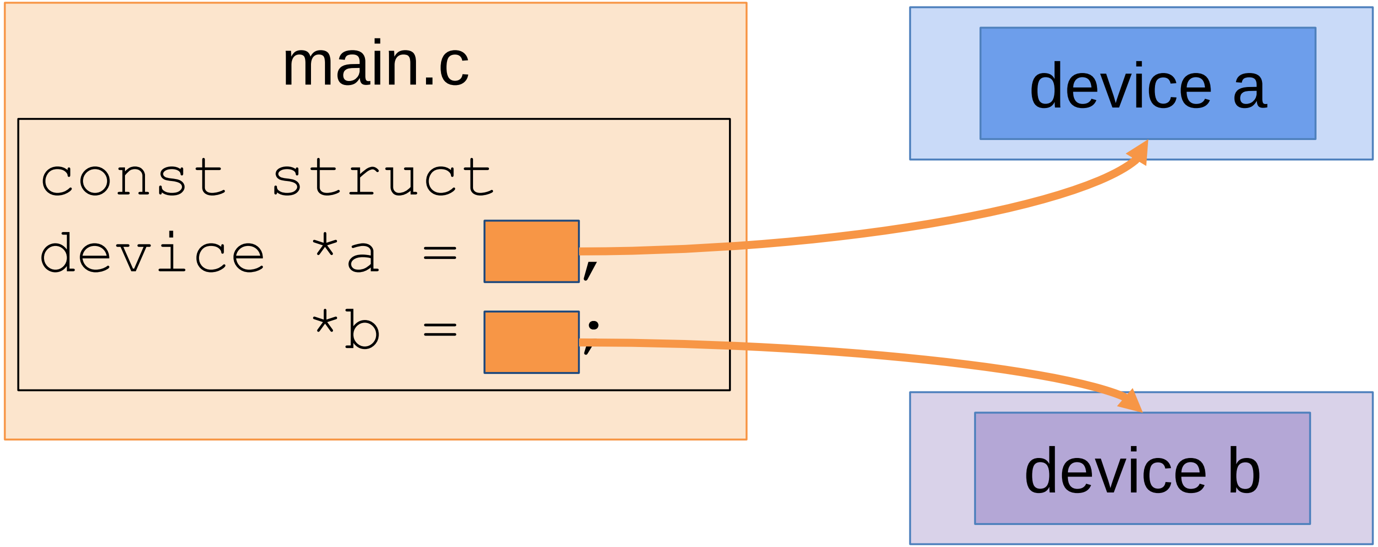
Application

bar.overlay

device a ✓

device b ✓

Get device pointers

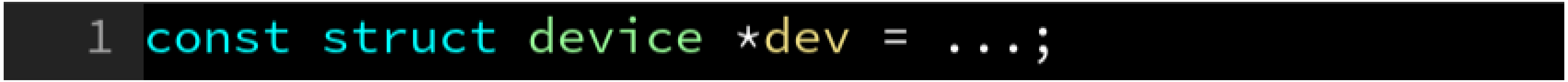


main.c

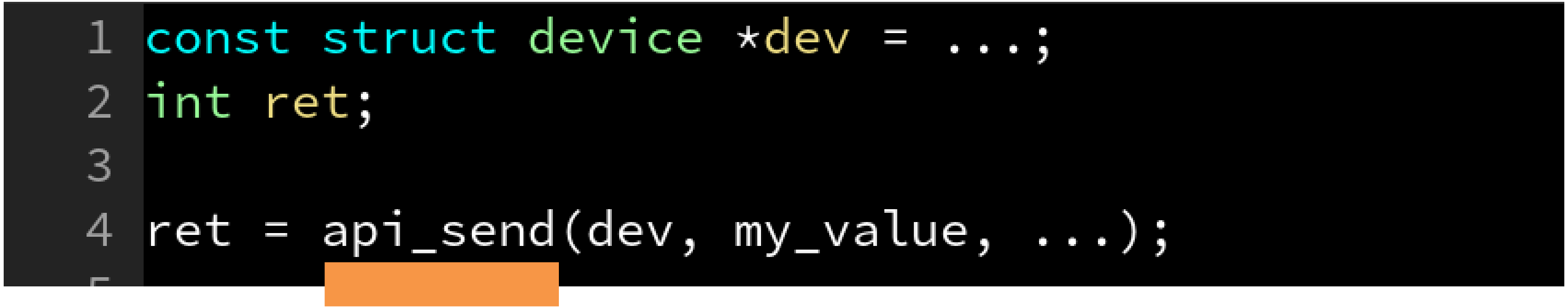
```
const struct
device *a =   ,
       *b =   ;
```

Zephyr™ Project
Developer Summit

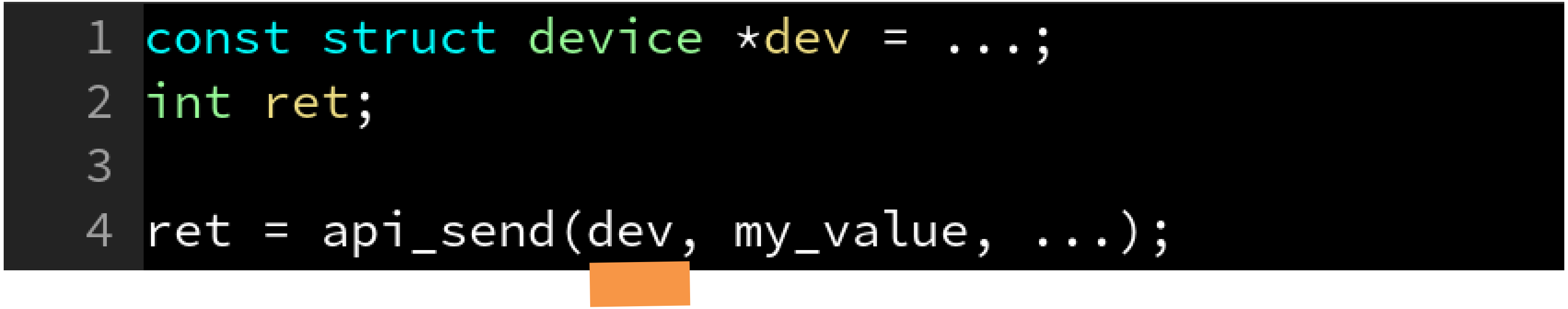
API basics



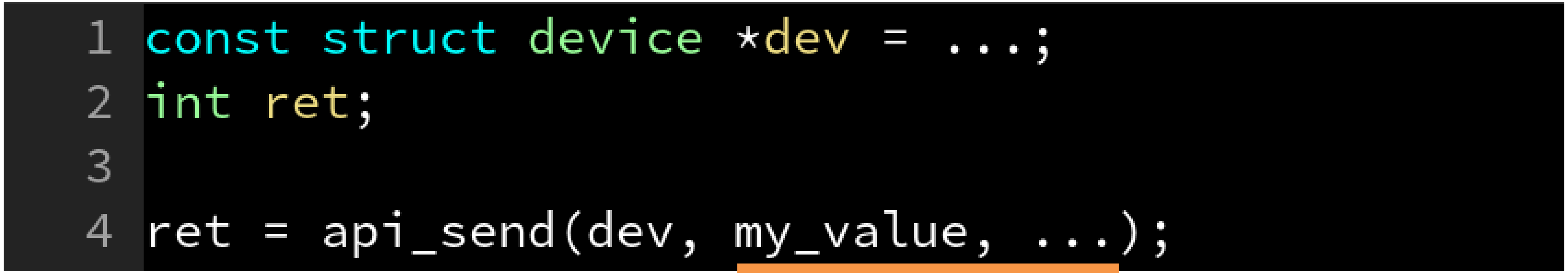
```c
const struct device *dev = ...;
```



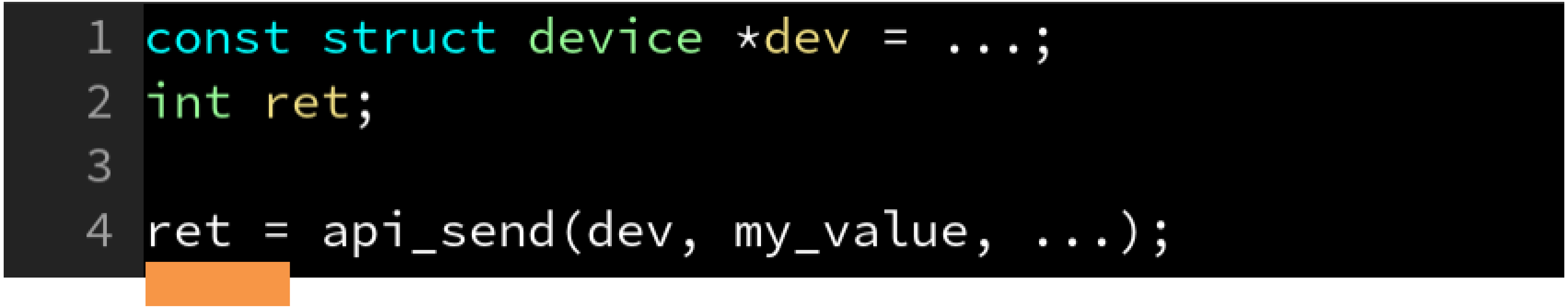
```c
const struct device *dev = ...;
int ret;

ret = api_send(dev, my_value, ...);
```



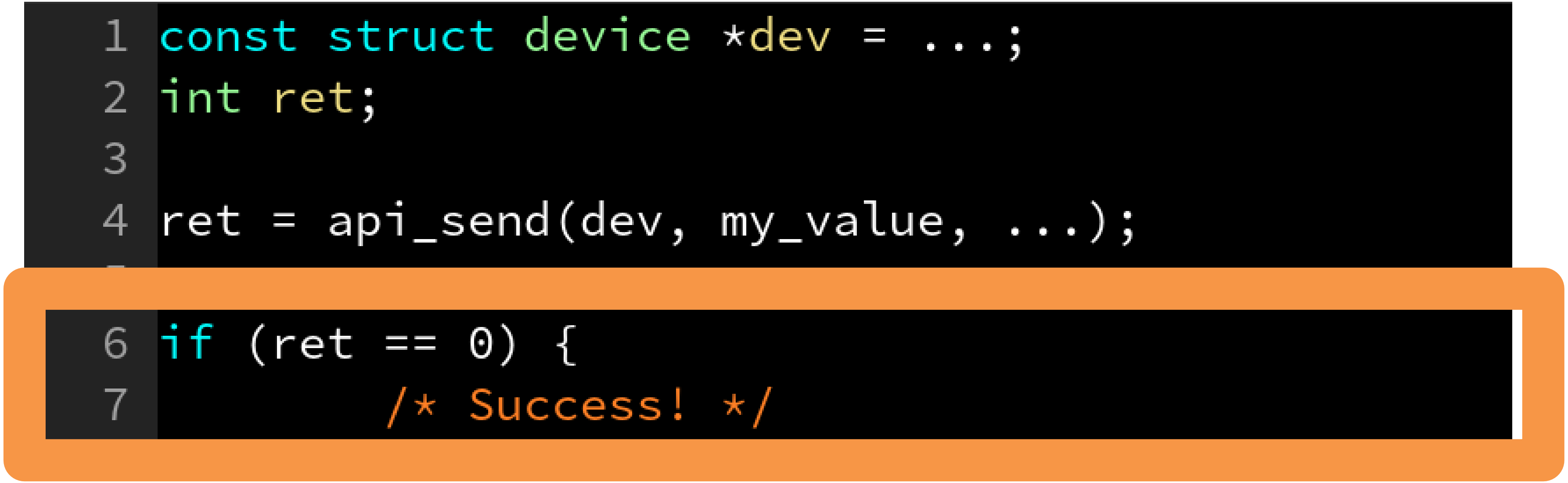
```
const struct device *dev = ...;
int ret;

ret = api_send(dev, my_value, ...);
```



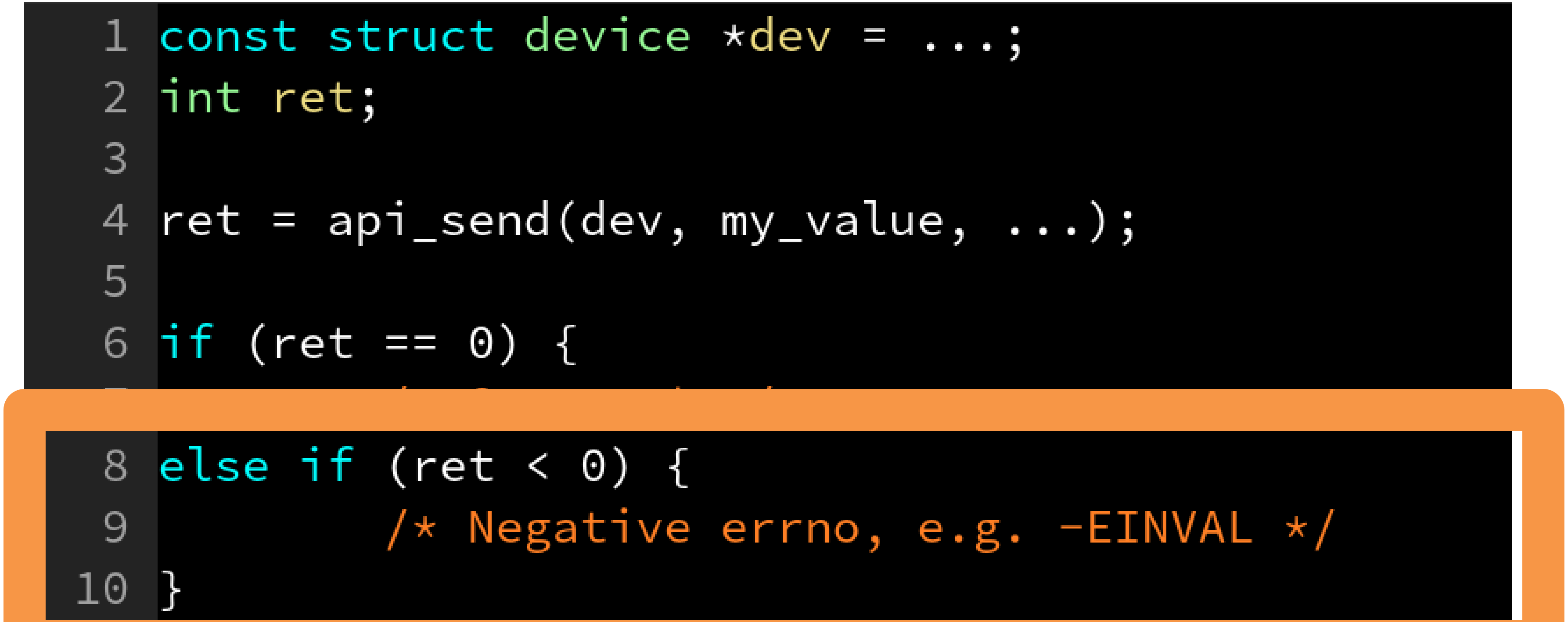
```c
const struct device *dev = ...;
int ret;

ret = api_send(dev, my_value, ...);
```



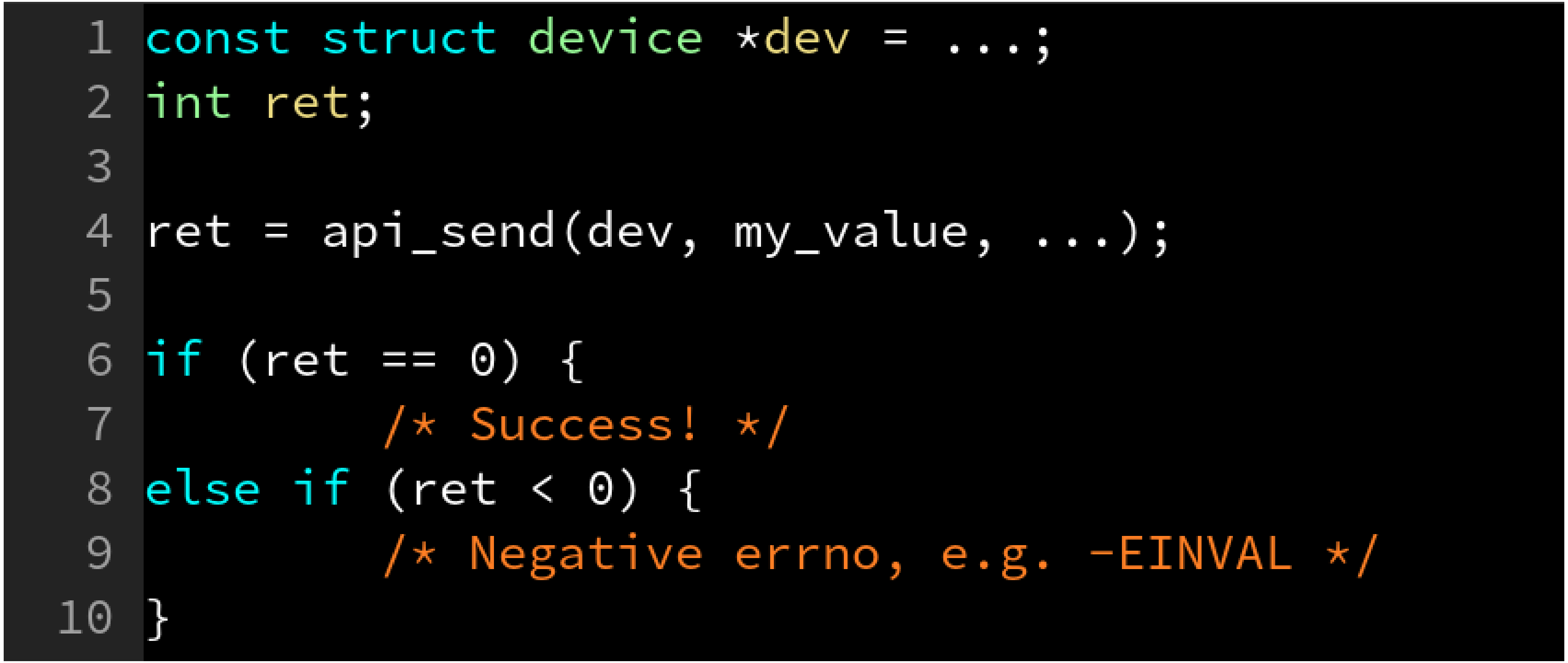
```
const struct device *dev = ...;
int ret;

ret = api_send(dev, my_value, ...);
```



```c
const struct device *dev = ...;
int ret;

ret = api_send(dev, my_value, ...);

if (ret == 0) {
        /* Success! */
```



```c
const struct device *dev = ...;
int ret;

ret = api_send(dev, my_value, ...);

if (ret == 0) {

else if (ret < 0) {
        /* Negative errno, e.g. -EINVAL */
}
```



```c
const struct device *dev = ...;
int ret;

ret = api_send(dev, my_value, ...);

if (ret == 0) {
        /* Success! */
else if (ret < 0) {
        /* Negative errno, e.g. -EINVAL */
}
```

Example APIs

API header	Example API method
<drivers/gpio.h>	
<drivers/pwm.h>	
<drivers/led.h>	

Example APIs

API header	Example API method
`<drivers/gpio.h>`	`gpio_pin_set(gpio_dev, pin, 1);`
`<drivers/pwm.h>`	
`<drivers/led.h>`	

Example APIs

API header	Example API method
<drivers/gpio.h>	`gpio_pin_set(gpio_dev, pin, 1);`

<drivers/pwm.h>	`pwm_pin_set_cycles(pwm_dev, pin, period_in_cycles,`
	`                   pulse_width_in_cycles, 0);`

<drivers/led.h>	

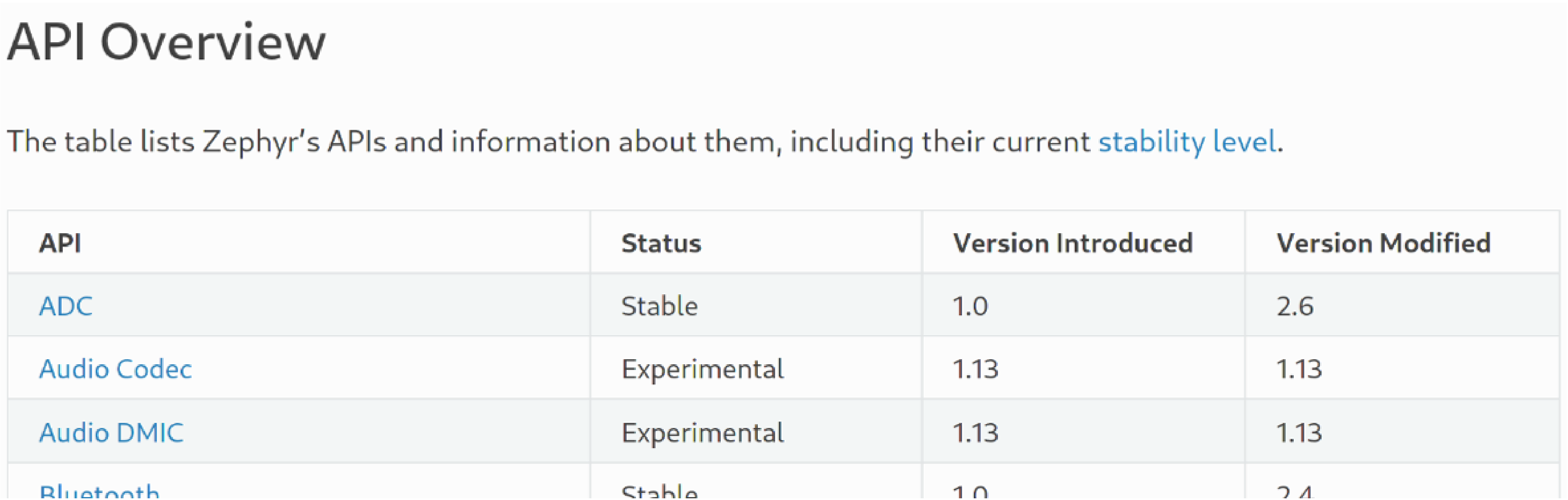
Example APIs

API header	Example API method
`<drivers/gpio.h>`	`gpio_pin_set(gpio_dev, pin, 1);`

`<drivers/pwm.h>`	`pwm_pin_set_cycles(pwm_dev, pin, period_in_cycles,`
	`pulse_width_in_cycles, 0);`

`<drivers/led.h>`	`led_blink(led_dev, led, 200, 400);`

Learning APIs, in theory



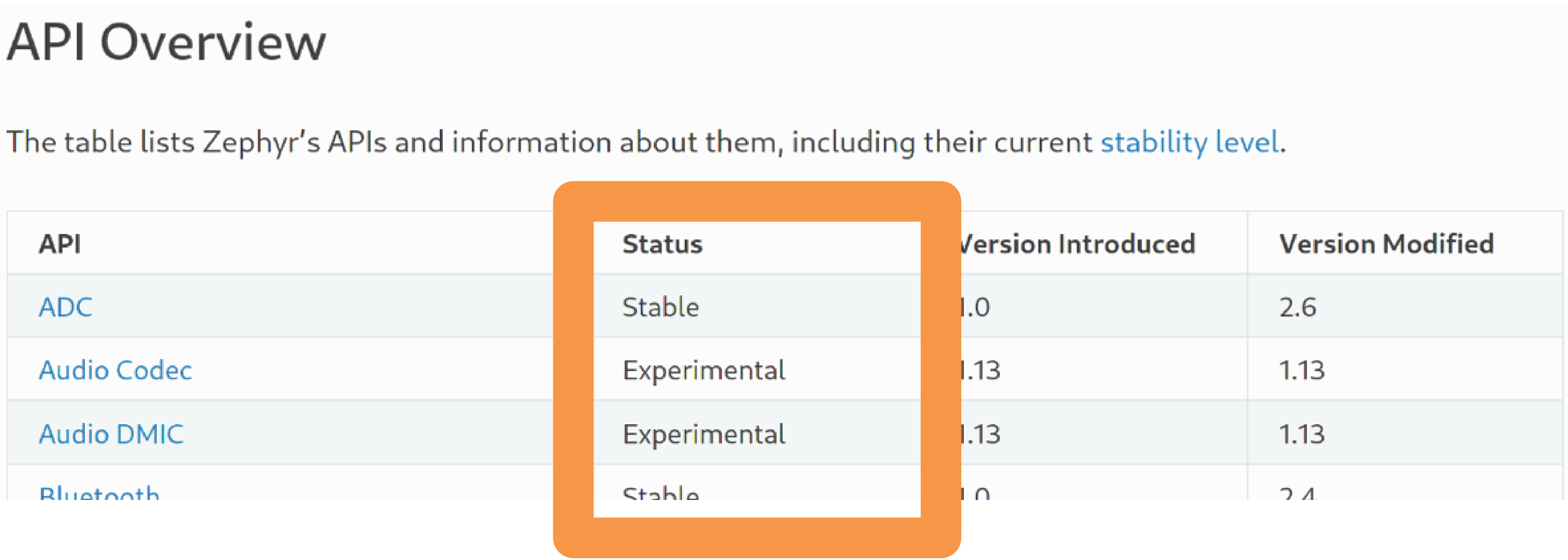
API Overview

The table lists Zephyr's APIs and information about them, including their current stability level.

API	Status	Version Introduced	Version Modified
ADC	Stable	1.0	2.6
Audio Codec	Experimental	1.13	1.13
Audio DMIC	Experimental	1.13	1.13
Bluetooth	Stable	1.0	2.4

https://docs.zephyrproject.org/2.6.0/reference/api/overview.html

Learning APIs, in theory

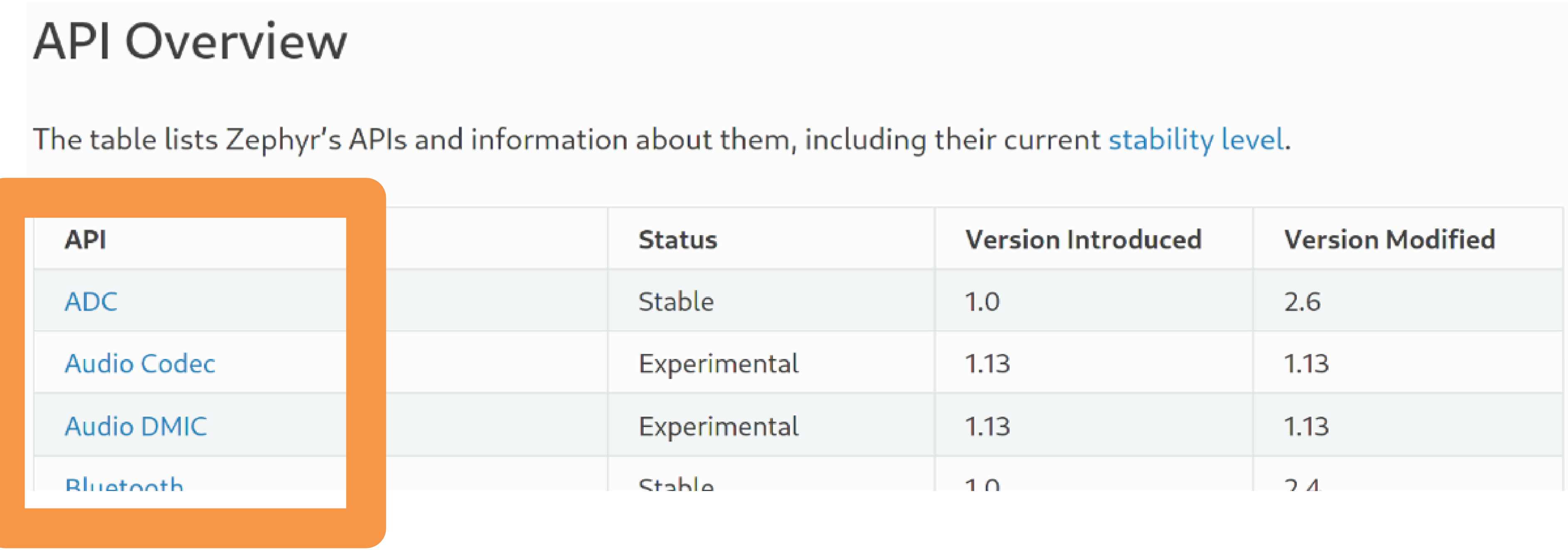


API Overview

The table lists Zephyr's APIs and information about them, including their current stability level.

API	Status	Version Introduced	Version Modified
ADC	Stable	1.0	2.6
Audio Codec	Experimental	1.13	1.13
Audio DMIC	Experimental	1.13	1.13
Bluetooth	Stable	1.0	2.4

Learning APIs, in theory

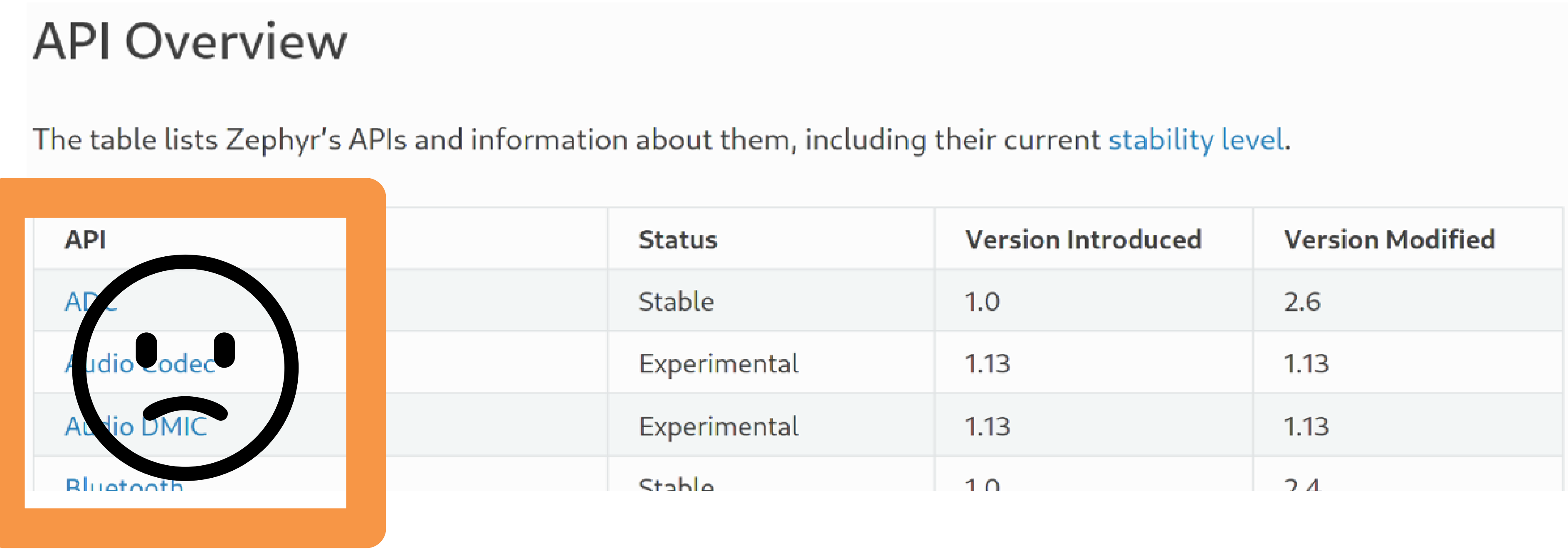


API Overview

The table lists Zephyr's APIs and information about them, including their current stability level.

API	Status	Version Introduced	Version Modified
ADC	Stable	1.0	2.6
Audio Codec	Experimental	1.13	1.13
Audio DMIC	Experimental	1.13	1.13
Bluetooth	Stable	1.0	2.4

Learning APIs, in theory



API Overview

The table lists Zephyr's APIs and information about them, including their current stability level.

API	Status	Version Introduced	Version Modified
ADC	Stable	1.0	2.6
Audio Codec	Experimental	1.13	1.13
Audio DMIC	Experimental	1.13	1.13
Bluetooth	Stable	1.0	2.4

Learning APIs, in practice

```
$ git grep "struct.*_api {" include
```

Then look for samples

```
$ git grep my_api.h samples
```

Worst case, find tests

```
$ git grep my_api.h tests
```

Drivers are by API

```
drivers
├── spi
├── pwm
├── sensor
└── ...
```

Zephyr™ Project
Developer Summit

Devicetree → `device`

DT guide, but with real hardware

https://docs.zephyrproject.org/2.6.0/guides/dts/index.html

How to...

- Allocate, configure `device`
- Get, use `device*`

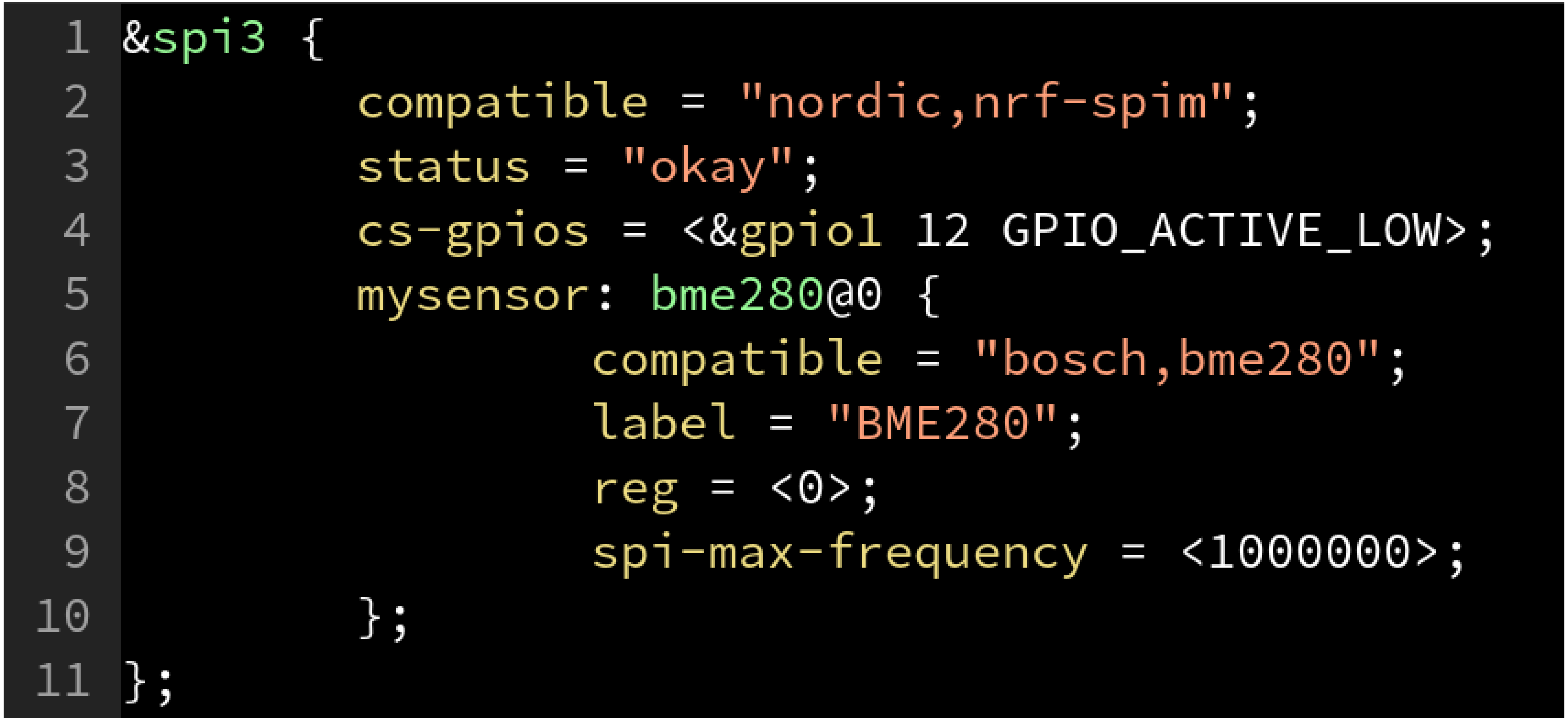
Hardware

- Sensor: Bosch BME280
- Board: nRF52840-DK

Zephyr™ Project
Developer Summit

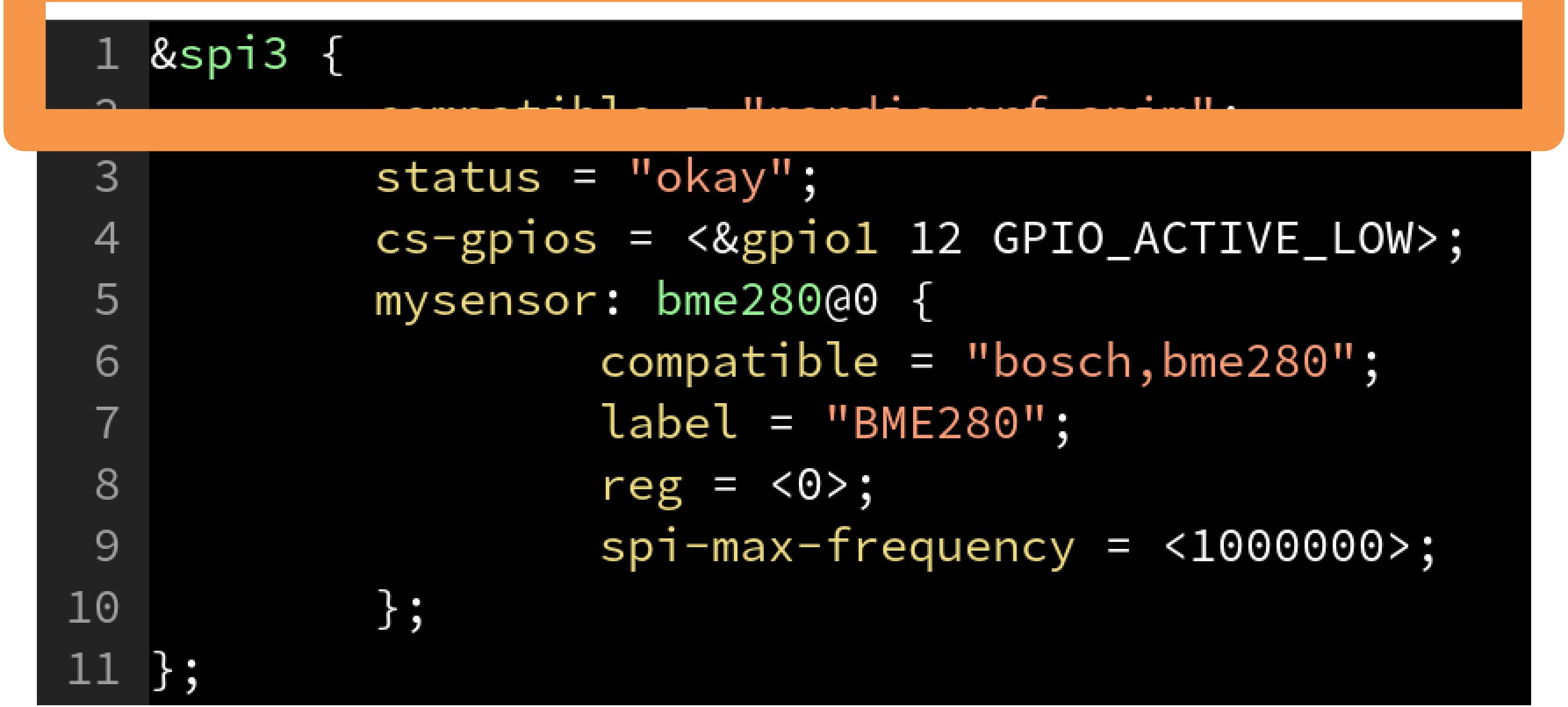
BME280 on SPI, devicetree overlay

SPI DT overlay



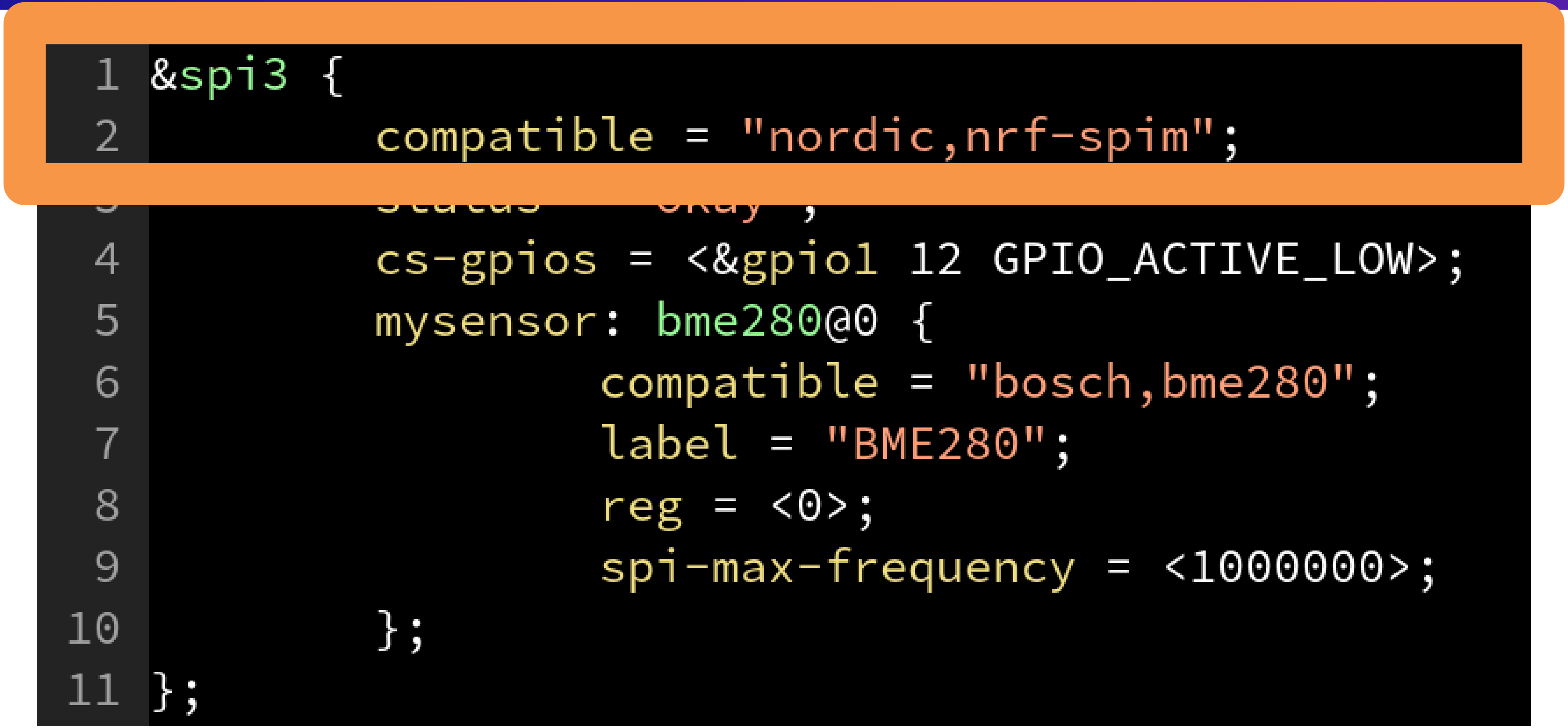
```
&spi3 {
        compatible = "nordic,nrf-spim";
        status = "okay";
        cs-gpios = <&gpio1 12 GPIO_ACTIVE_LOW>;
        mysensor: bme280@0 {
                compatible = "bosch,bme280";
                label = "BME280";
                reg = <0>;
                spi-max-frequency = <1000000>;
        };
};
```

Modify SPI3



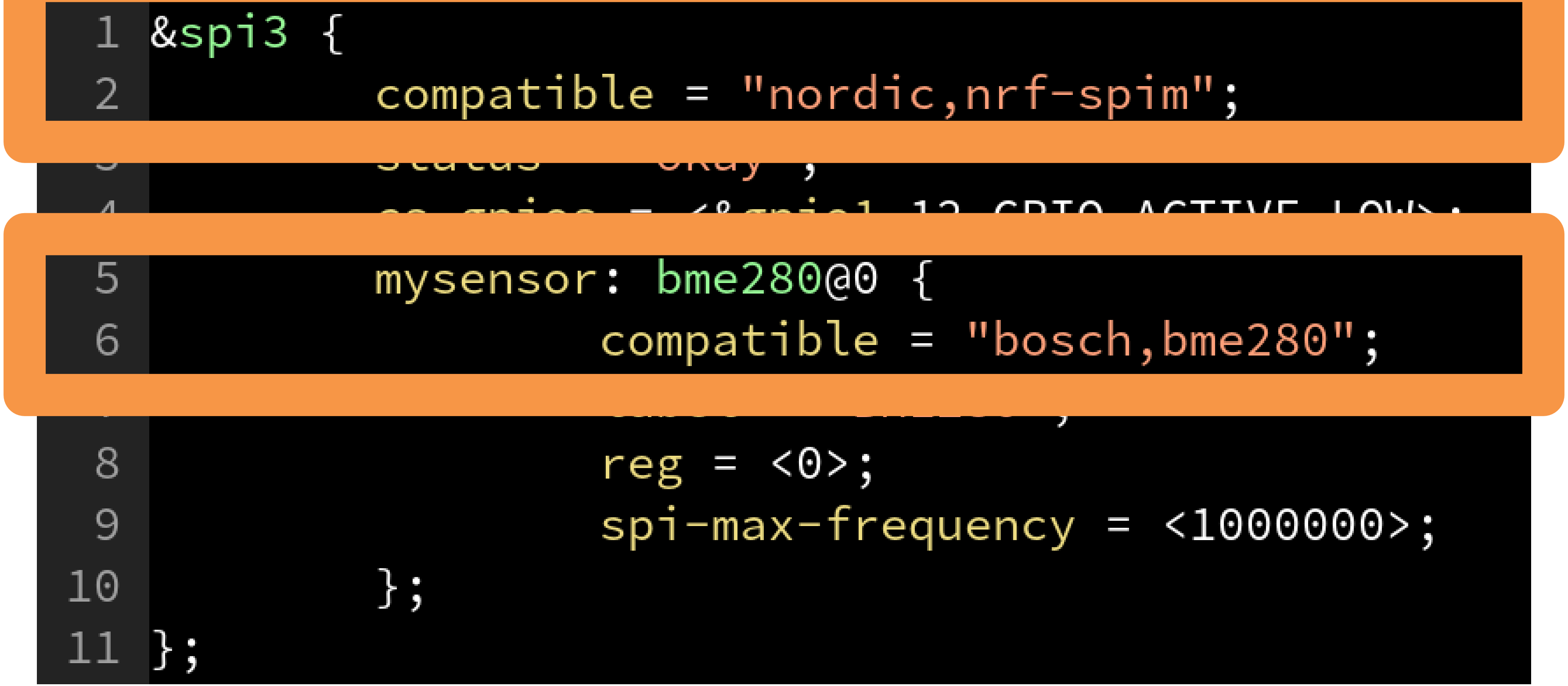
```
&spi3 {
        compatible = "nordic,nrf-spim";
        status = "okay";
        cs-gpios = <&gpio1 12 GPIO_ACTIVE_LOW>;
        mysensor: bme280@0 {
                compatible = "bosch,bme280";
                label = "BME280";
                reg = <0>;
                spi-max-frequency = <1000000>;
        };
};
```

Compatible says the hardware type



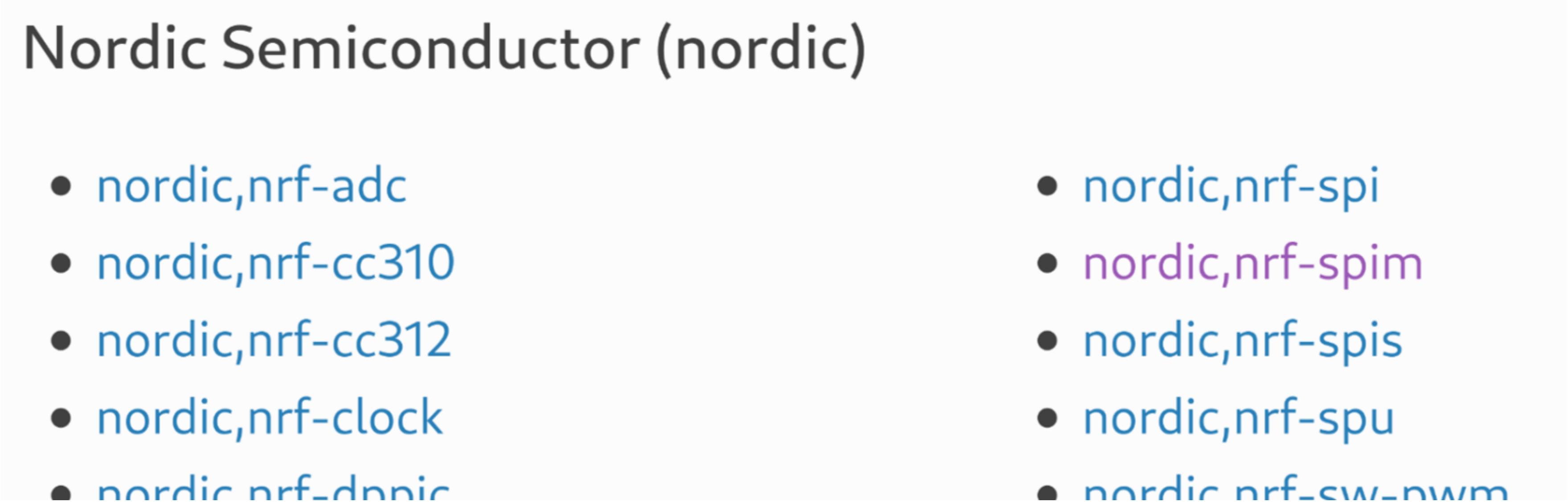
```
&spi3 {
        compatible = "nordic,nrf-spim";
        status = "okay";
        cs-gpios = <&gpio1 12 GPIO_ACTIVE_LOW>;
        mysensor: bme280@0 {
                compatible = "bosch,bme280";
                label = "BME280";
                reg = <0>;
                spi-max-frequency = <1000000>;
        };
};
```

Different hardware, different compatible



```
&spi3 {
        compatible = "nordic,nrf-spim";
        status = "okay";
        cs-gpios = <&gpio1 12 GPIO_ACTIVE_LOW>;
        mysensor: bme280@0 {
                compatible = "bosch,bme280";
                [illegible]
                reg = <0>;
                spi-max-frequency = <1000000>;
        };
};
```

Bindings index documents compatibles

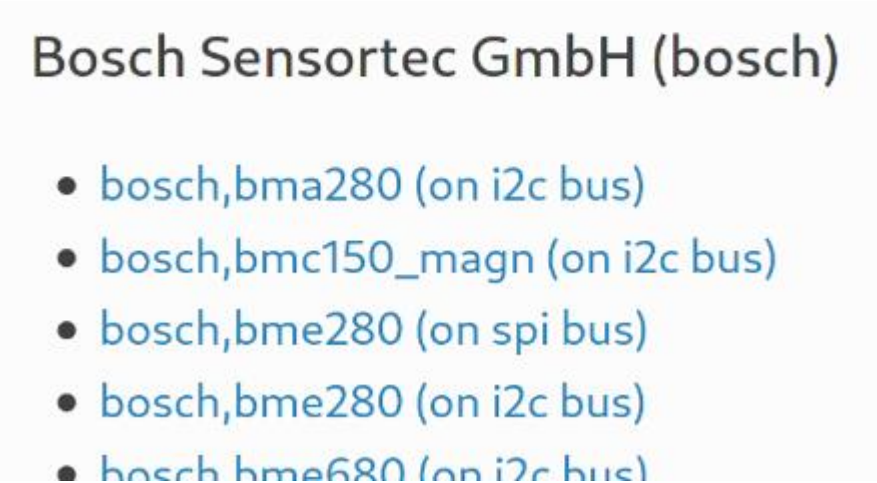


Nordic Semiconductor (nordic)

- nordic,nrf-adc
- nordic,nrf-cc310
- nordic,nrf-cc312
- nordic,nrf-clock
- nordic,nrf-dppic
- nordic,nrf-spi
- nordic,nrf-spim
- nordic,nrf-spis
- nordic,nrf-spu
- nordic,nrf-sw-pwm

https://docs.zephyrproject.org/2.6.0/reference/devicetree/bindings.html#dt-vendor-nordic

Bindings index grouped by vendor

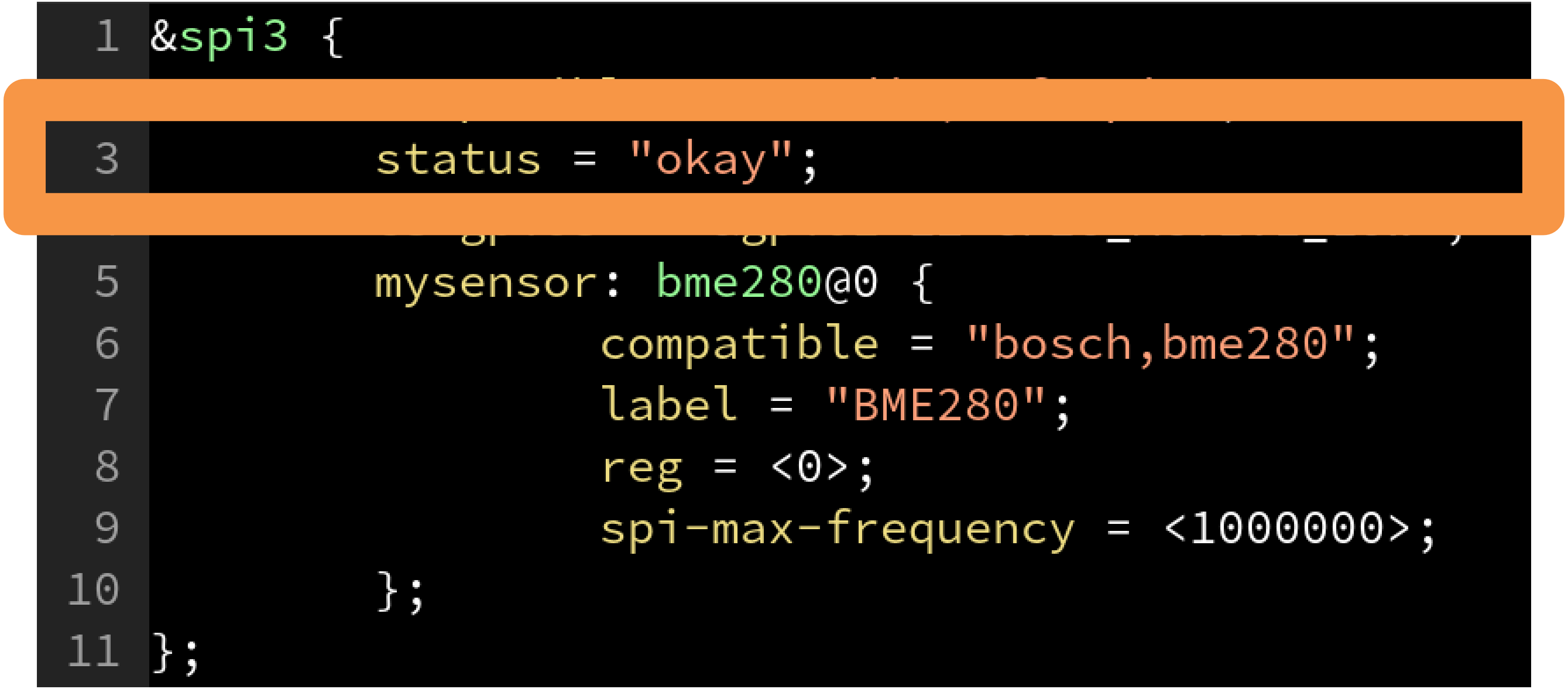


Bosch Sensortec GmbH (bosch)

- bosch,bma280 (on i2c bus)
- bosch,bmc150_magn (on i2c bus)
- bosch,bme280 (on spi bus)
- bosch,bme280 (on i2c bus)
- bosch,bme680 (on i2c bus)

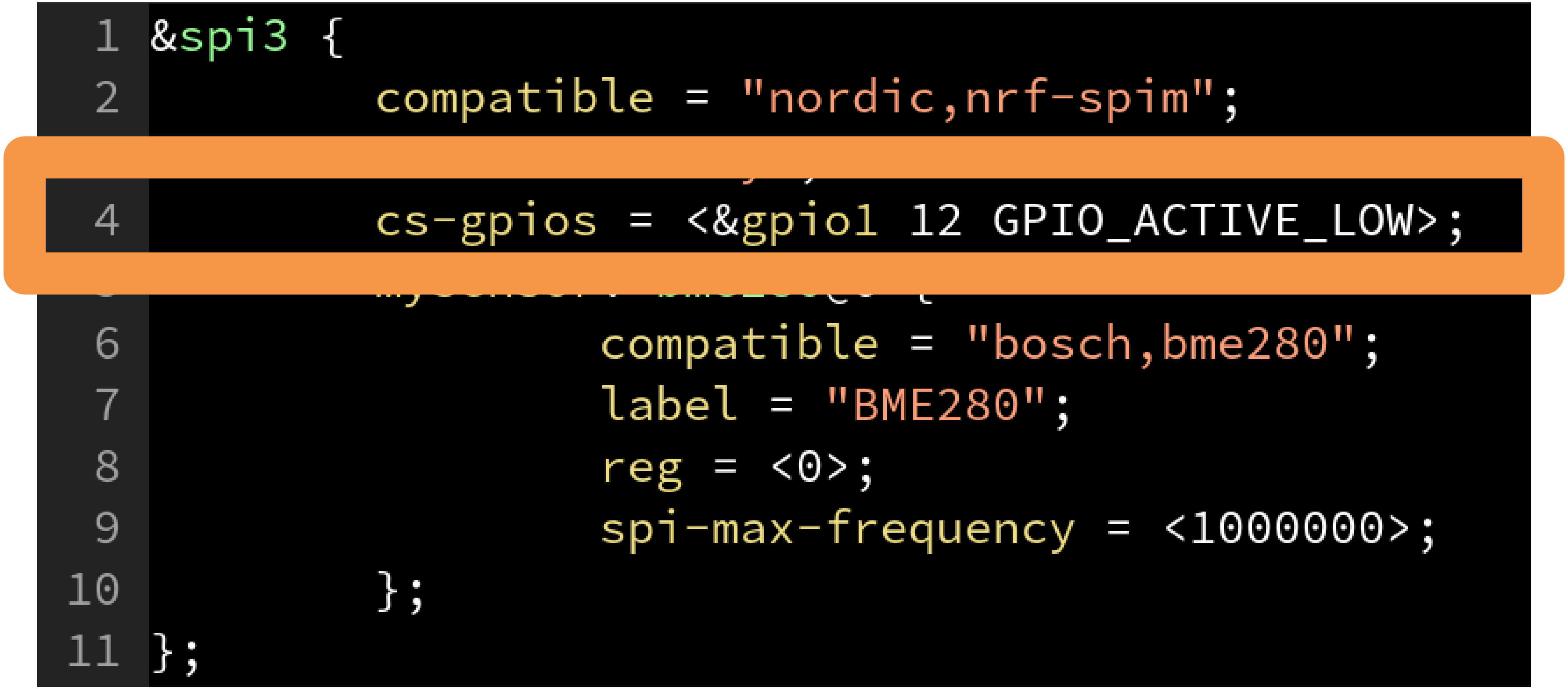
https://docs.zephyrproject.org/2.6.0/reference/devicetree/bindings.html#dt-vendor-bosch

Status controls device allocation



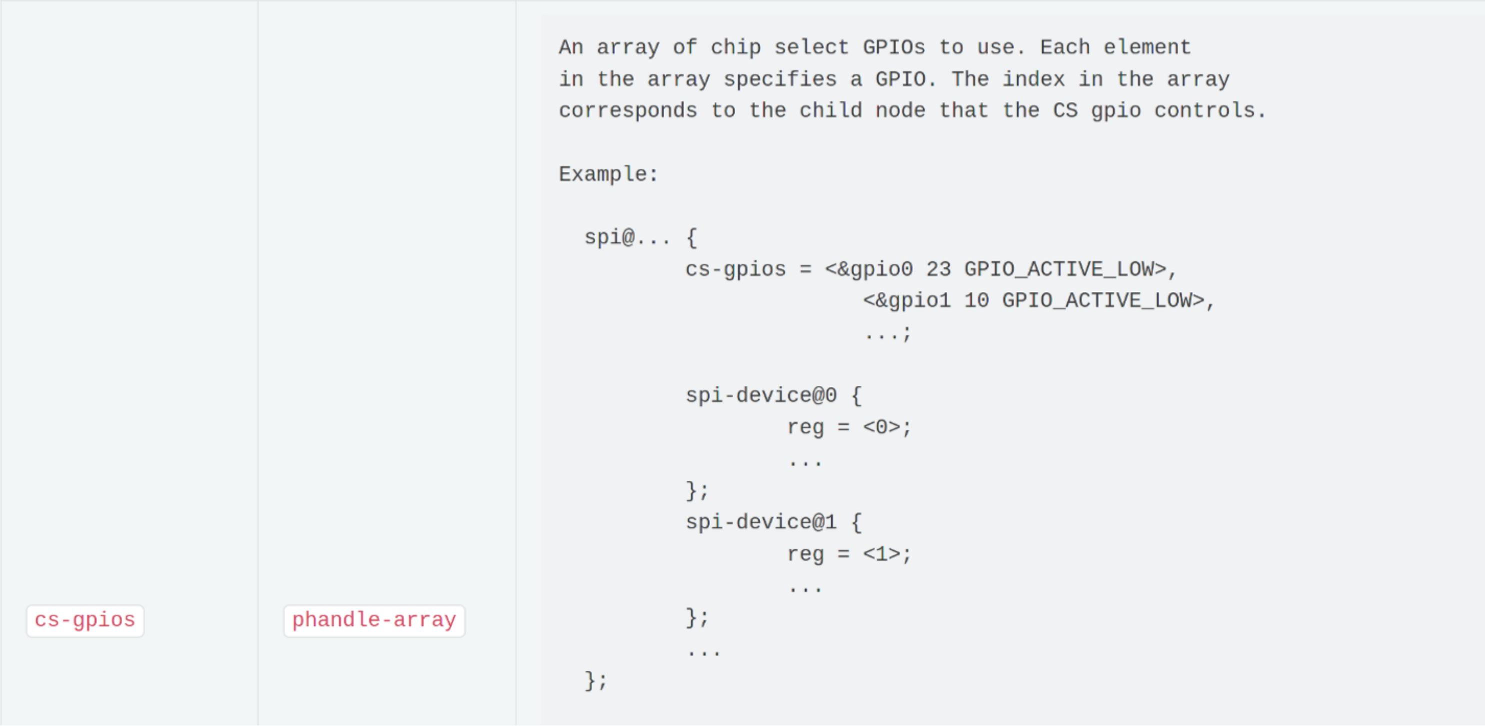
```
&spi3 {
    status = "okay";
    mysensor: bme280@0 {
        compatible = "bosch,bme280";
        label = "BME280";
        reg = <0>;
        spi-max-frequency = <1000000>;
    };
};
```

cs-gpios configures chip select



```
&spi3 {
        compatible = "nordic,nrf-spim";
        cs-gpios = <&gpio1 12 GPIO_ACTIVE_LOW>;
        mysensor: bme280@0 {
                compatible = "bosch,bme280";
                label = "BME280";
                reg = <0>;
                spi-max-frequency = <1000000>;
        };
};
```

More in the bindings index



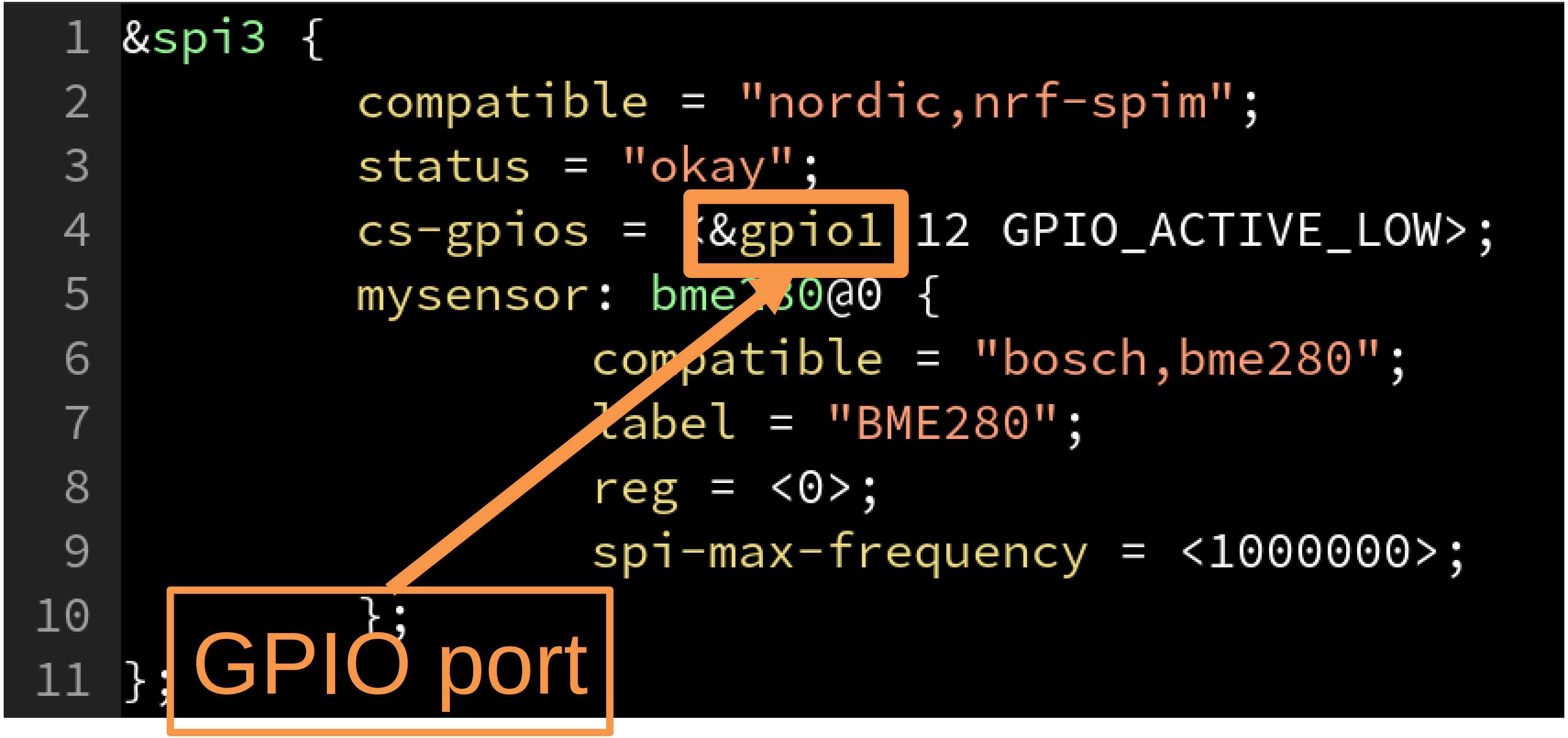
`cs-gpios`

`phandle-array`

```
An array of chip select GPIOs to use. Each element
in the array specifies a GPIO. The index in the array
corresponds to the child node that the CS gpio controls.

Example:

  spi@... {
          cs-gpios = <&gpio0 23 GPIO_ACTIVE_LOW>,
                        <&gpio1 10 GPIO_ACTIVE_LOW>,
                        ...;

          spi-device@0 {
                  reg = <0>;
                  ...
          };
          spi-device@1 {
                  reg = <1>;
                  ...
          };
          ...
  };
```

https://docs.zephyrproject.org/2.6.0/reference/devicetree/bindings/spi/nordic%2Cnrf-spim.html

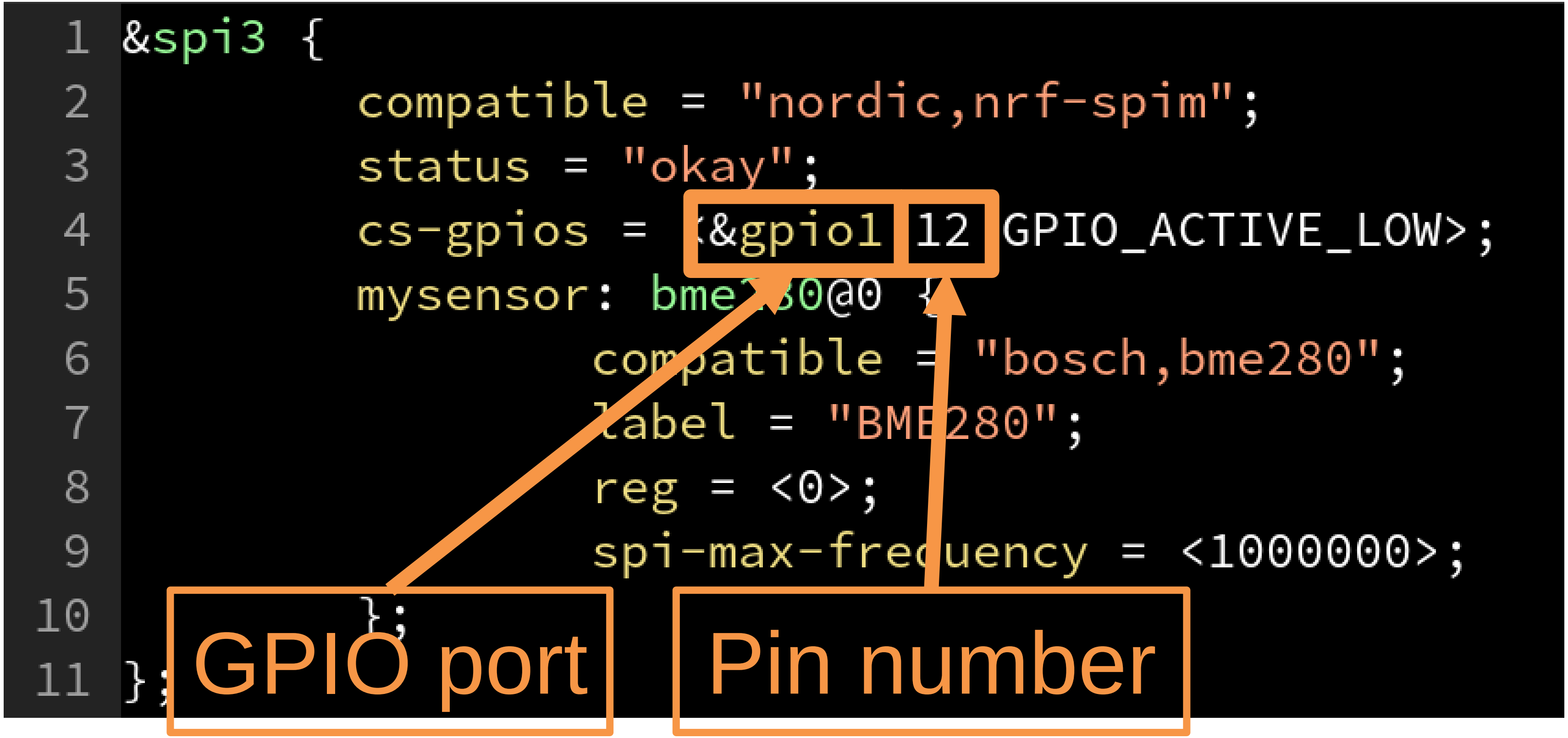
Chip select contents



```
&spi3 {
        compatible = "nordic,nrf-spim";
        status = "okay";
        cs-gpios = <&gpio1 12 GPIO_ACTIVE_LOW>;
        mysensor: bme280@0 {
                compatible = "bosch,bme280";
                label = "BME280";
                reg = <0>;
                spi-max-frequency = <1000000>;
        };
};
```

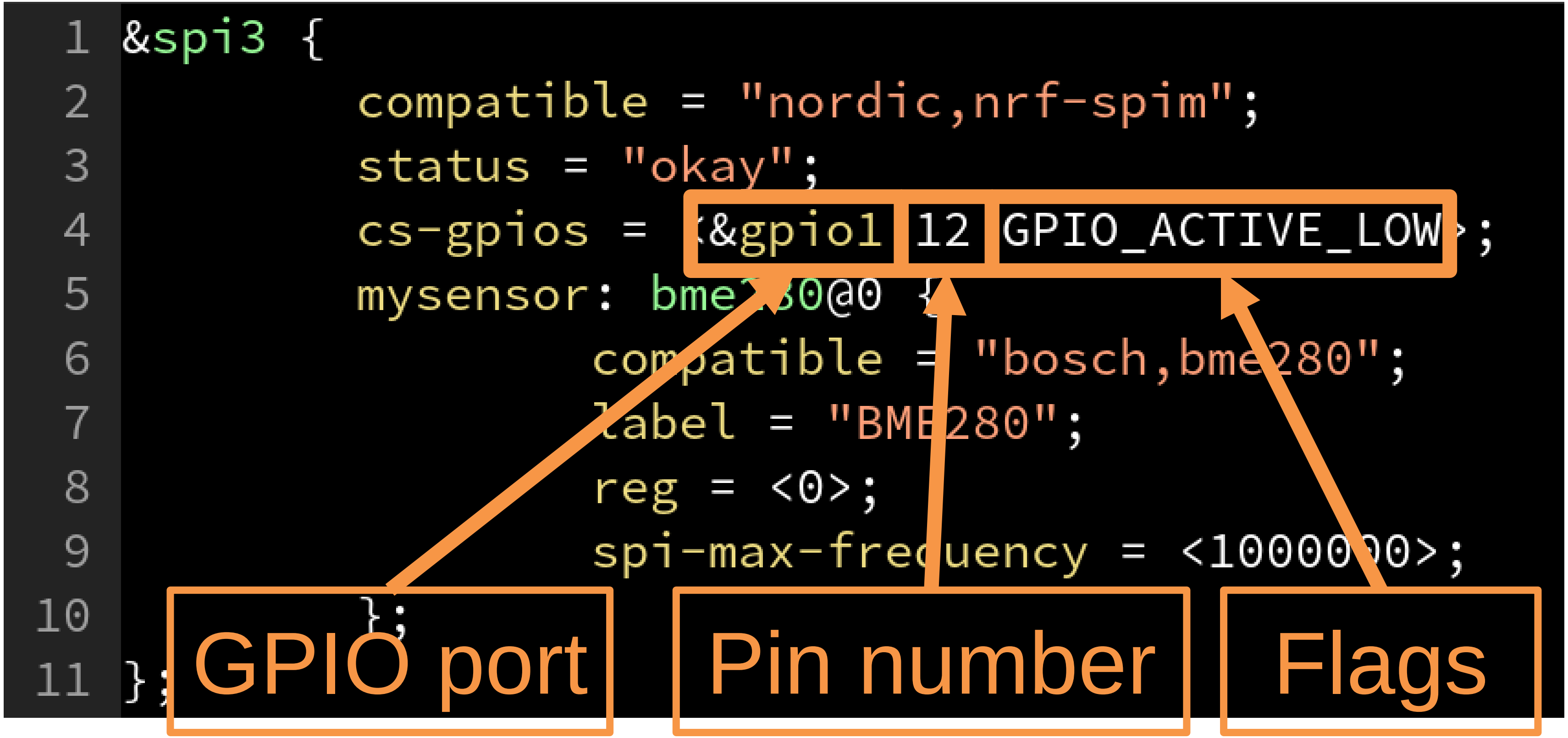
GPIO port

Chip select contents



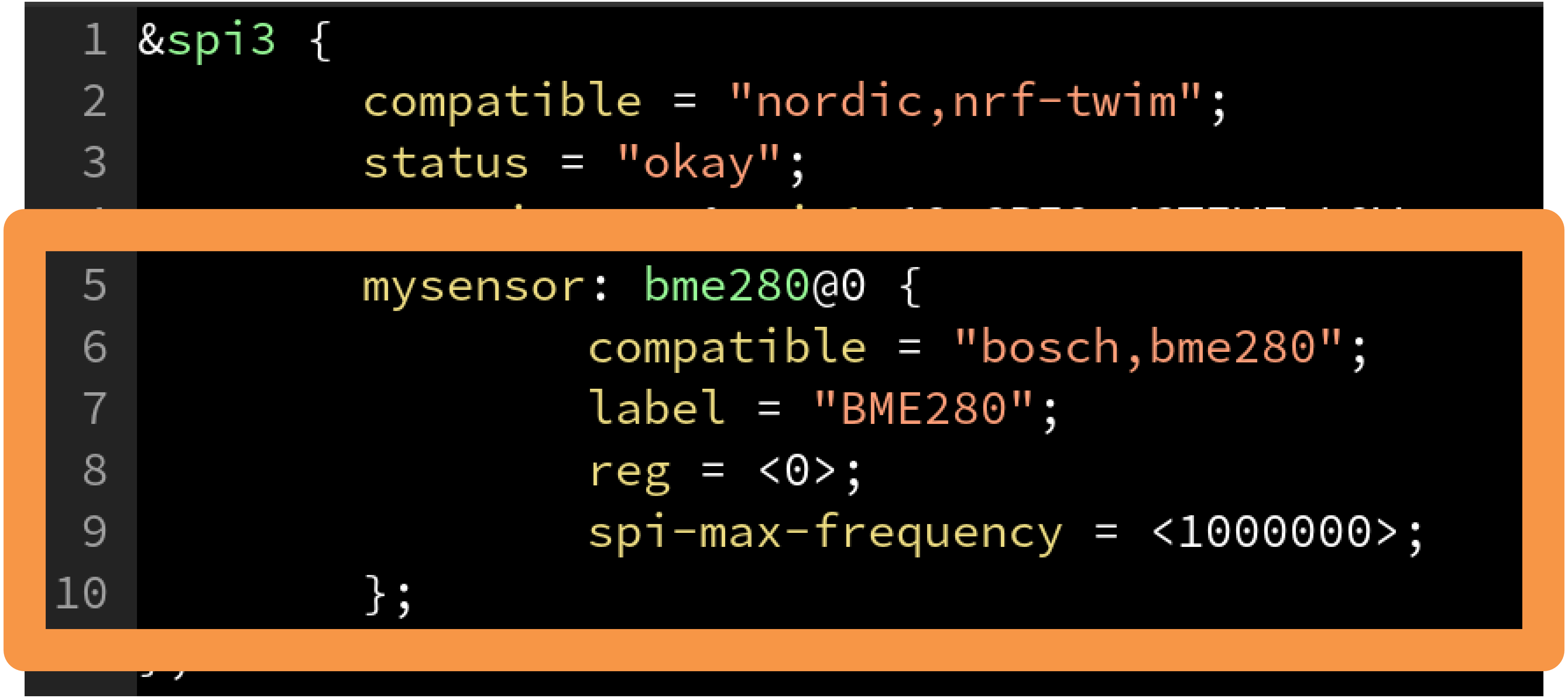
```
&spi3 {
        compatible = "nordic,nrf-spim";
        status = "okay";
        cs-gpios = <&gpio1 12 GPIO_ACTIVE_LOW>;
        mysensor: bme280@0 {
                compatible = "bosch,bme280";
                label = "BME280";
                reg = <0>;
                spi-max-frequency = <1000000>;
        };
};
```

Chip select contents



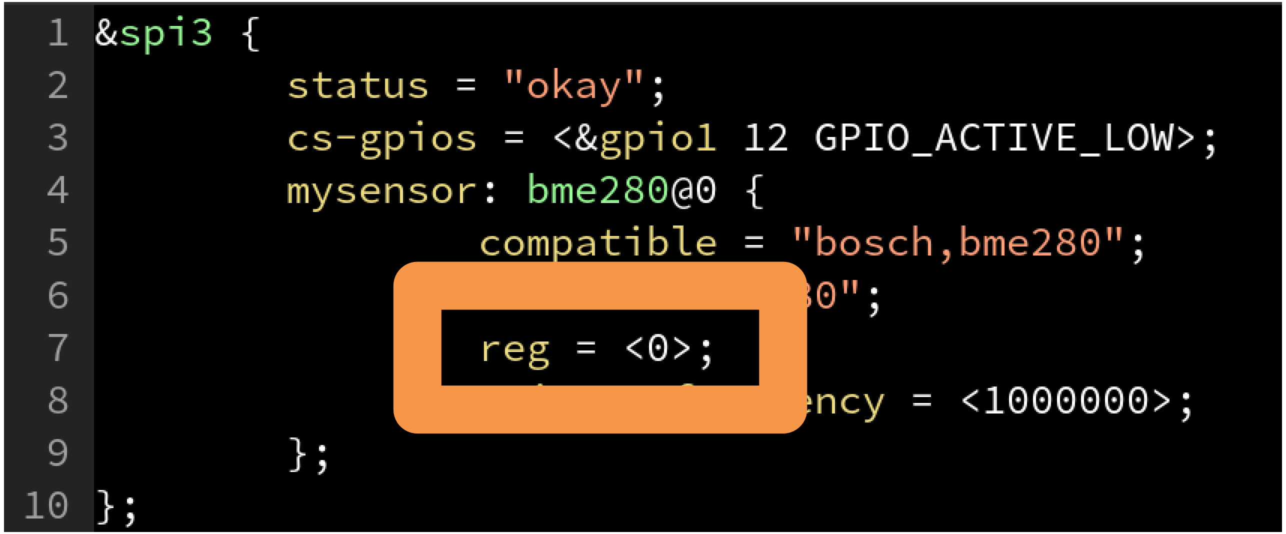
```
&spi3 {
        compatible = "nordic,nrf-spim";
        status = "okay";
        cs-gpios = <&gpio1 12 GPIO_ACTIVE_LOW>;
        mysensor: bme280@0 {
                compatible = "bosch,bme280";
                label = "BME280";
                reg = <0>;
                spi-max-frequency = <1000000>;
        };
};
```

Define a BME280 sensor



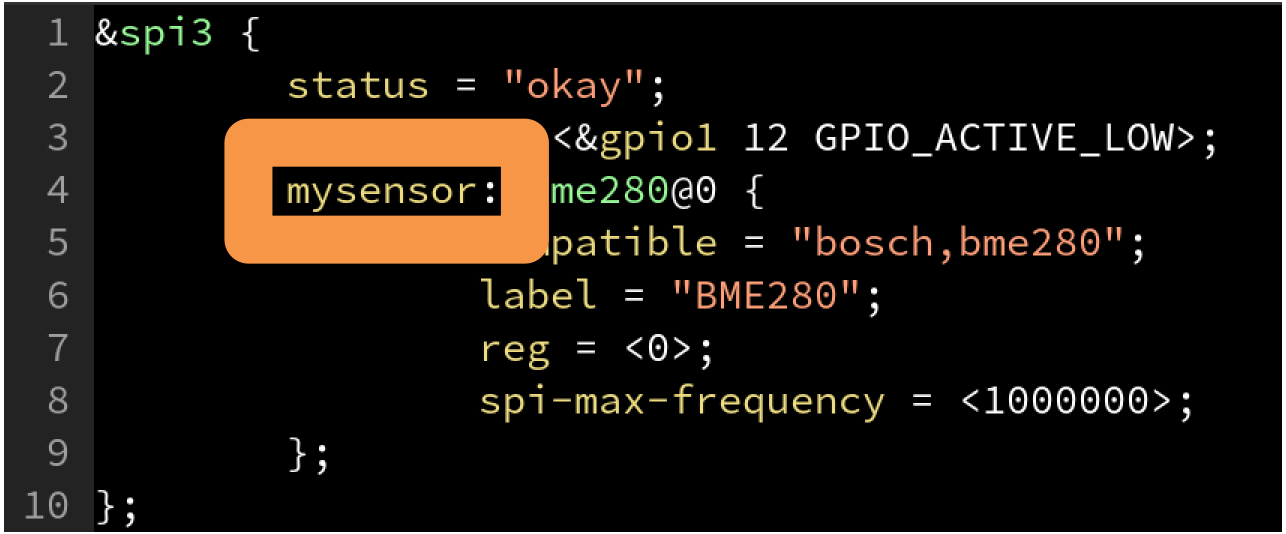
```
&spi3 {
        compatible = "nordic,nrf-twim";
        status = "okay";

        mysensor: bme280@0 {
                compatible = "bosch,bme280";
                label = "BME280";
                reg = <0>;
                spi-max-frequency = <1000000>;
        };
};
```

cs-gpios index 0



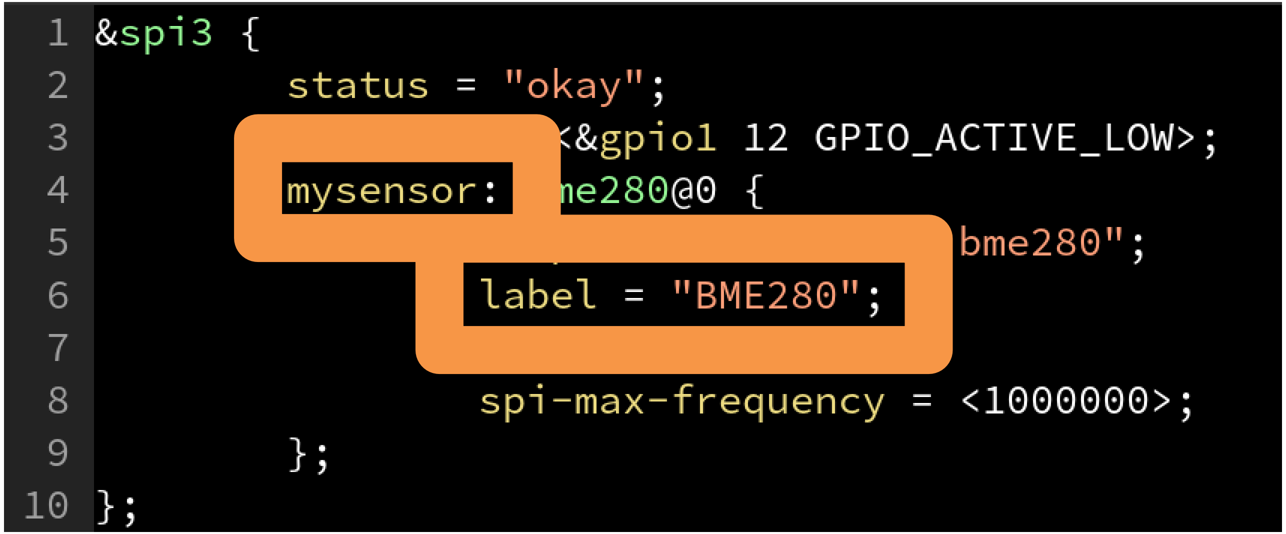
```
&spi3 {
        status = "okay";
        cs-gpios = <&gpio1 12 GPIO_ACTIVE_LOW>;
        mysensor: bme280@0 {
                compatible = "bosch,bme280";
                                  0";
                reg = <0>;
                                  ency = <1000000>;
        };
};
```

Node label “mysensor”



```
&spi3 {
        status = "okay";
                   <&gpio1 12 GPIO_ACTIVE_LOW>;
        mysensor:  me280@0 {
                   patible = "bosch,bme280";
                label = "BME280";
                reg = <0>;
                spi-max-frequency = <1000000>;
        };
};
```

Node label != label property



```
&spi3 {
        status = "okay";
                    <&gpio1 12 GPIO_ACTIVE_LOW>;
        mysensor:   me280@0 {
                                    bme280";
                label = "BME280";

                spi-max-frequency = <1000000>;
        };
};
```

Remember this!

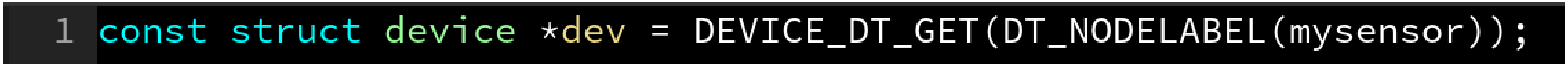
label property

node label

Zephyr™ Project
Developer Summit

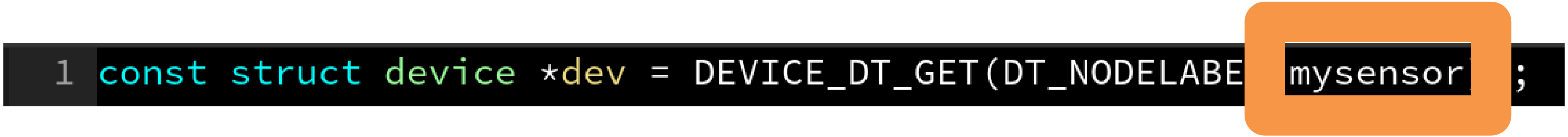
BME280 on SPI, C code

Get the device



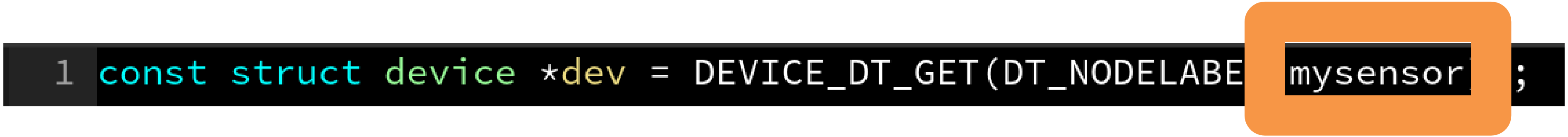
```
const struct device *dev = DEVICE_DT_GET(DT_NODELABEL(mysensor));
```

Get the device

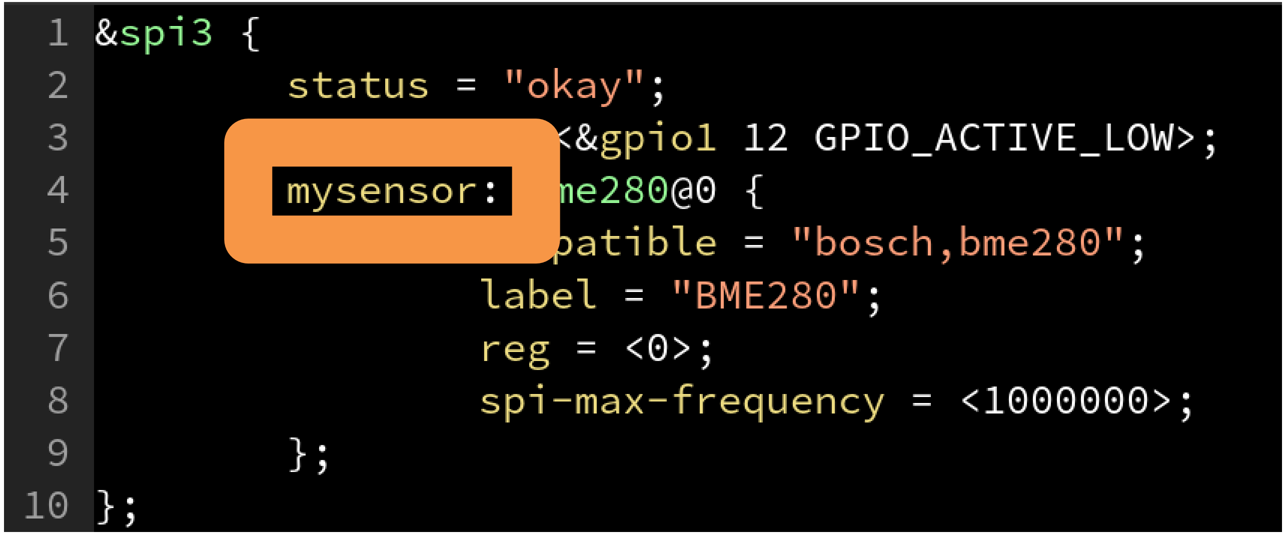


```c
const struct device *dev = DEVICE_DT_GET(DT_NODELABE mysensor ;
```

Device via devicetree node

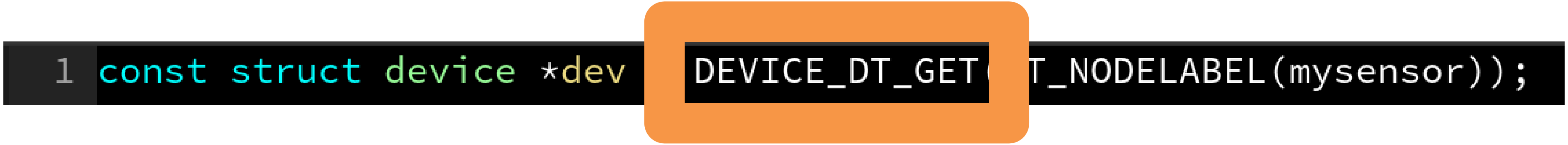


```
const struct device *dev = DEVICE_DT_GET(DT_NODELABE mysensor ;
```



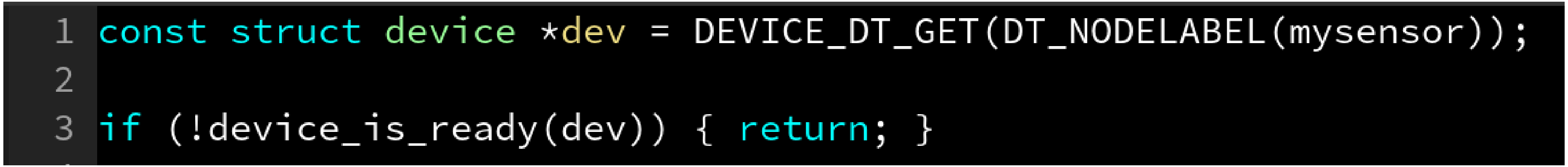
```
&spi3 {
        status = "okay";
                <&gpio1 12 GPIO_ACTIVE_LOW>;
        mysensor: me280@0 {
                patible = "bosch,bme280";
                label = "BME280";
                reg = <0>;
                spi-max-frequency = <1000000>;
        };
};
```

DEVICE_DT_GET



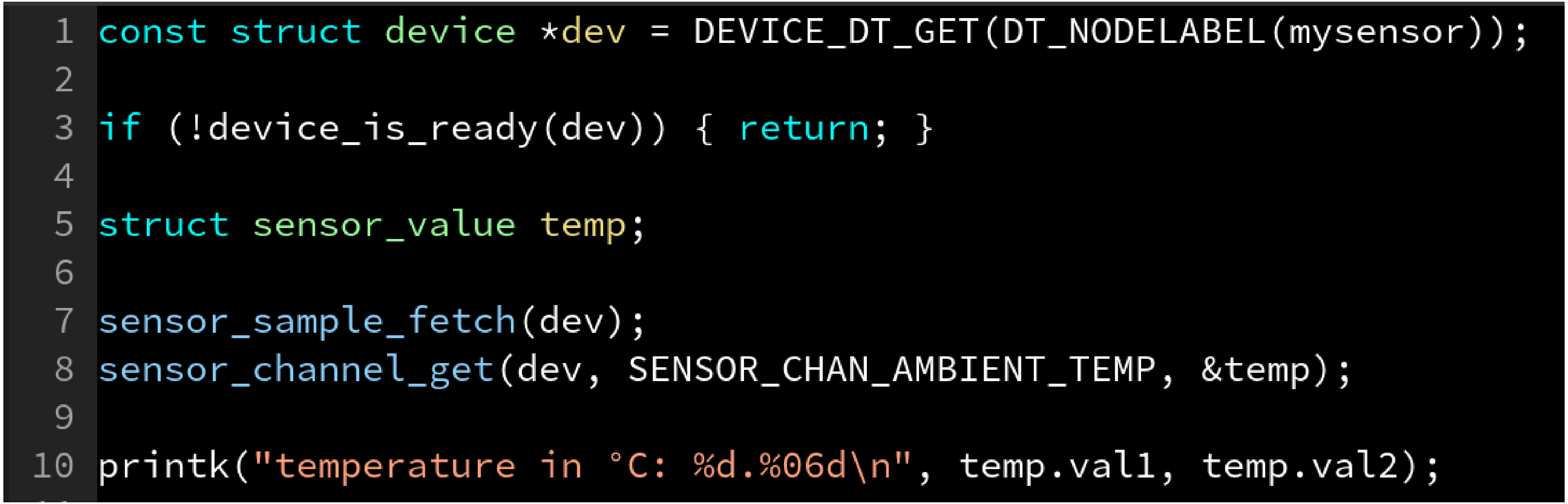
```
const struct device *dev DEVICE_DT_GET(T_NODELABEL(mysensor));
```

Check initialization result



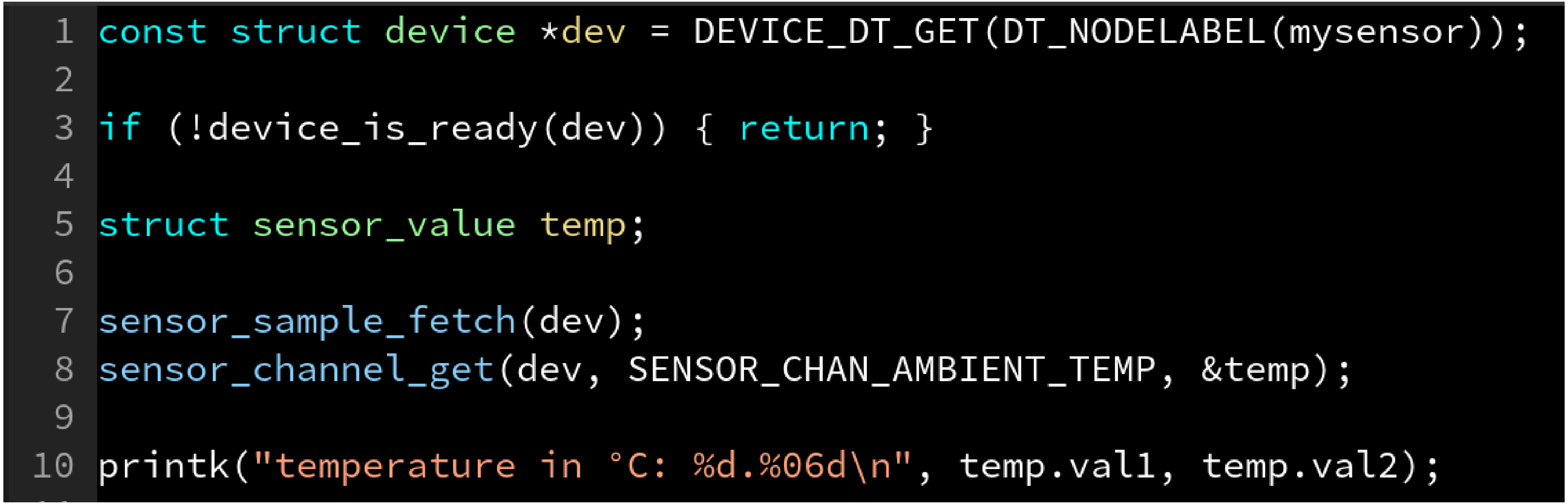
```c
const struct device *dev = DEVICE_DT_GET(DT_NODELABEL(mysensor));

if (!device_is_ready(dev)) { return; }
```

Use sensor API



```c
const struct device *dev = DEVICE_DT_GET(DT_NODELABEL(mysensor));

if (!device_is_ready(dev)) { return; }

struct sensor_value temp;

sensor_sample_fetch(dev);
sensor_channel_get(dev, SENSOR_CHAN_AMBIENT_TEMP, &temp);

printk("temperature in °C: %d.%06d\n", temp.val1, temp.val2);
```

That’s it

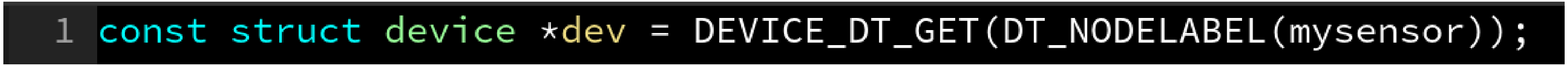


```
const struct device *dev = DEVICE_DT_GET(DT_NODELABEL(mysensor));

if (!device_is_ready(dev)) { return; }

struct sensor_value temp;

sensor_sample_fetch(dev);
sensor_channel_get(dev, SENSOR_CHAN_AMBIENT_TEMP, &temp);

printk("temperature in °C: %d.%06d\n", temp.val1, temp.val2);
```

No bus or vendor specifics! No manual setup!

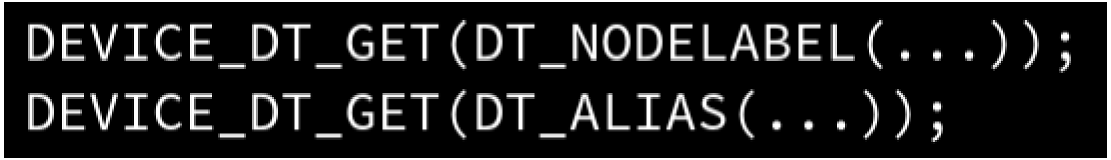
Zephyr™ Project
Developer Summit

DEVICE_DT_GET: awesome



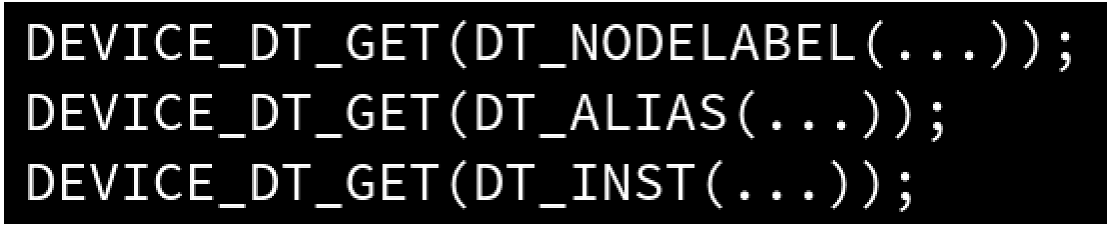
```c
const struct device *dev = DEVICE_DT_GET(DT_NODELABEL(mysensor));
```

All node identifiers work



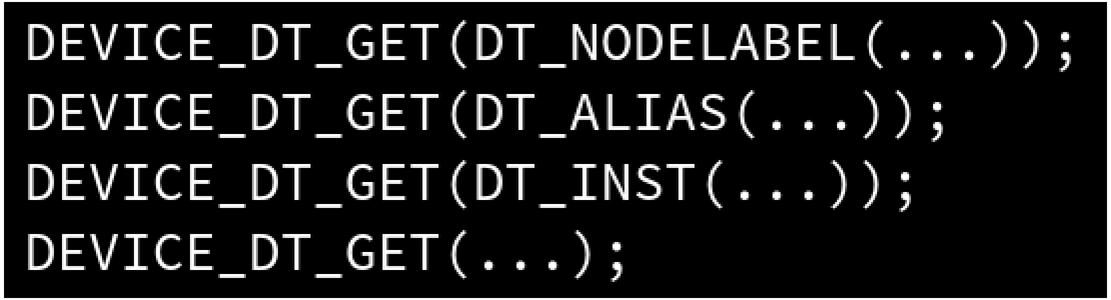
```
DEVICE_DT_GET(DT_NODELABEL(...));
DEVICE_DT_GET(DT_ALIAS(...));
```

All node identifiers work



```
DEVICE_DT_GET(DT_NODELABEL(...));
DEVICE_DT_GET(DT_ALIAS(...));
DEVICE_DT_GET(DT_INST(...));
```

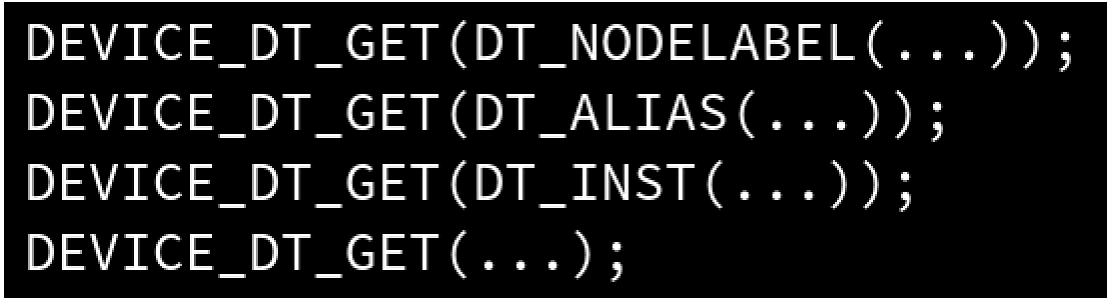
All node identifiers work



```
DEVICE_DT_GET(DT_NODELABEL(...));
DEVICE_DT_GET(DT_ALIAS(...));
DEVICE_DT_GET(DT_INST(...));
DEVICE_DT_GET(...);
```

Avoid DT_INST outside drivers

🚫



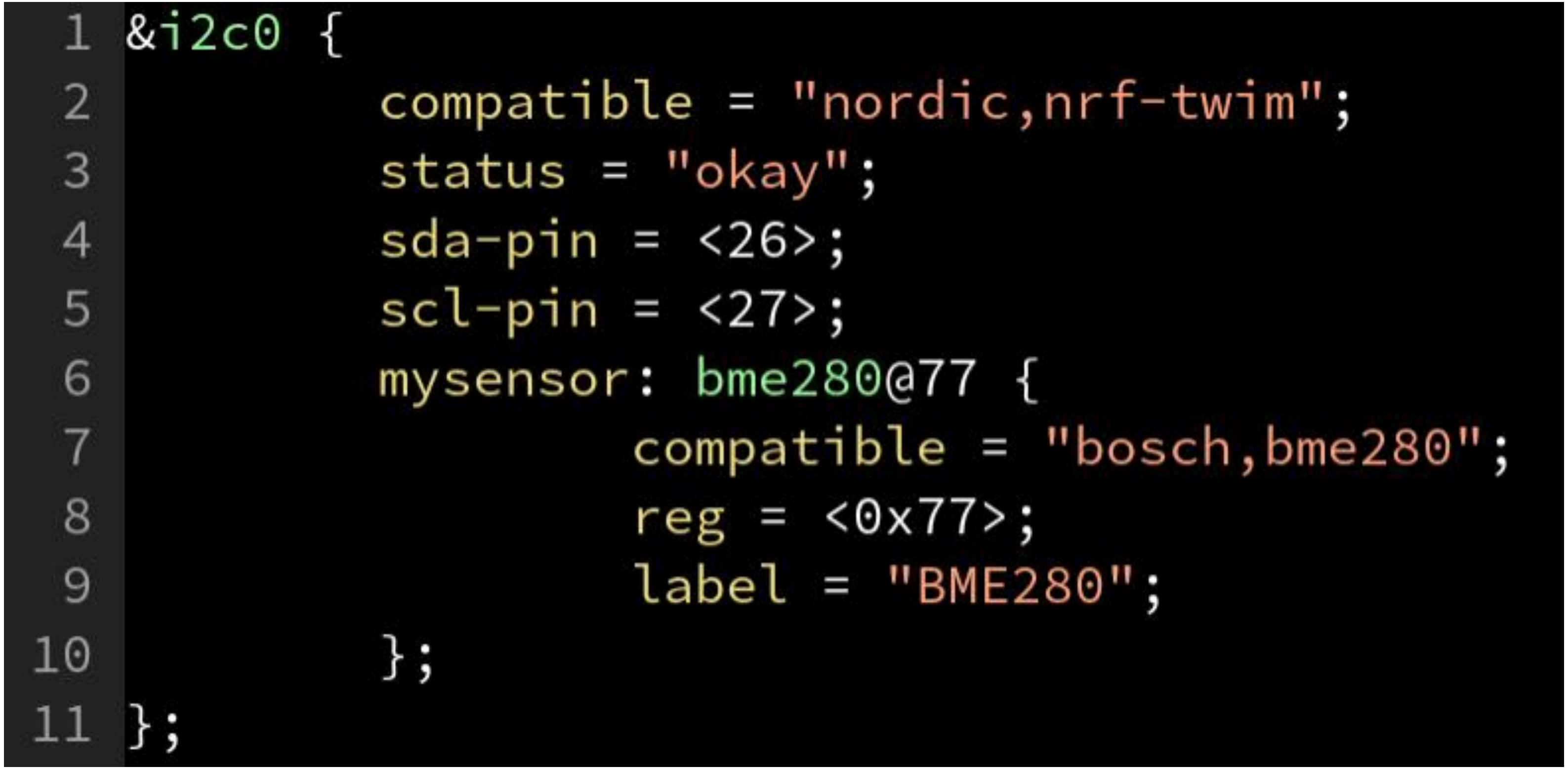
```
DEVICE_DT_GET(DT_NODELABEL(...));
DEVICE_DT_GET(DT_ALIAS(...));
DEVICE_DT_GET(DT_INST(...));
DEVICE_DT_GET(...);
```

🚫

Zephyr™ Project
Developer Summit

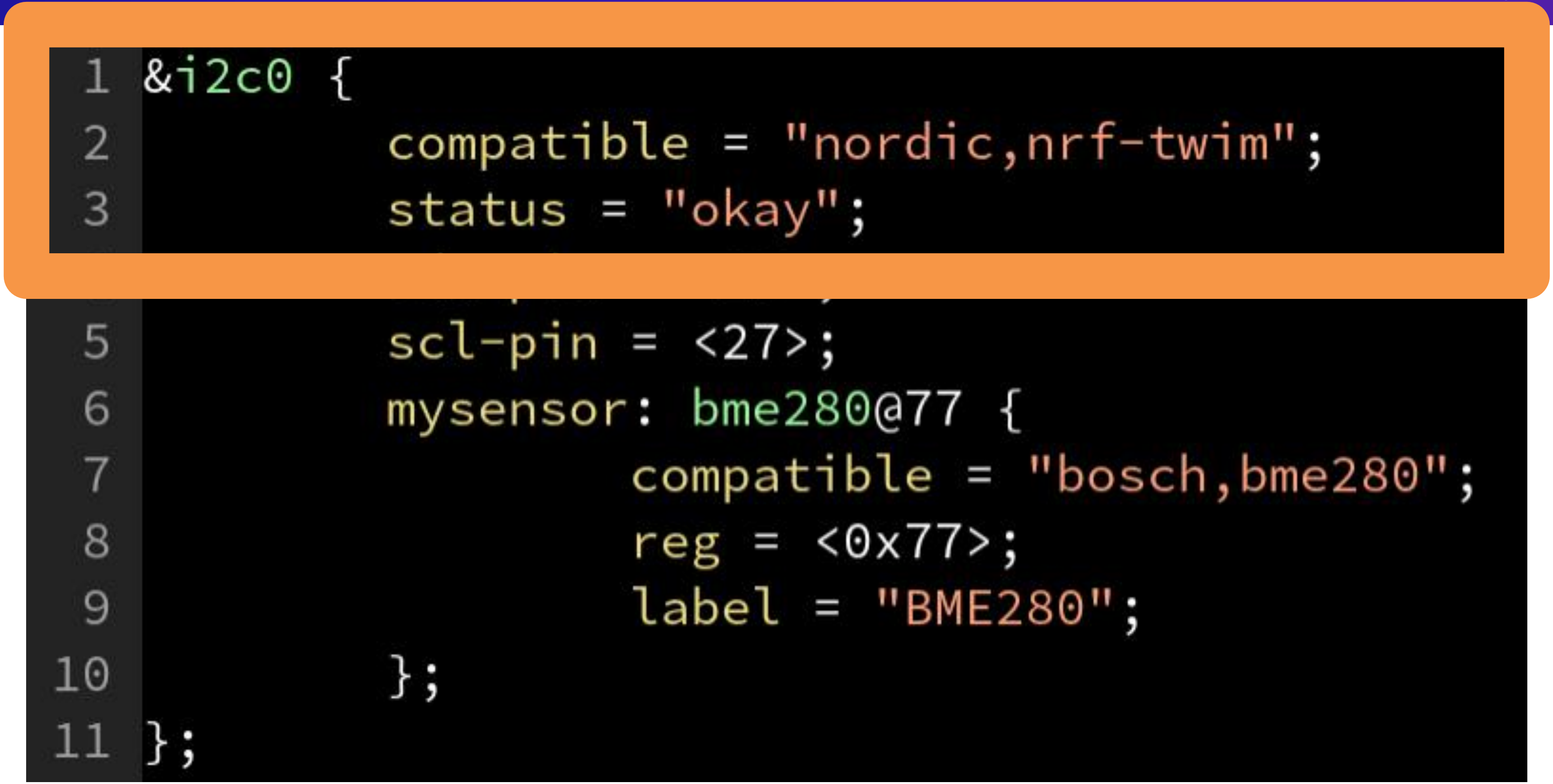
BME280 on I2C, devicetree overlay

DT overlay



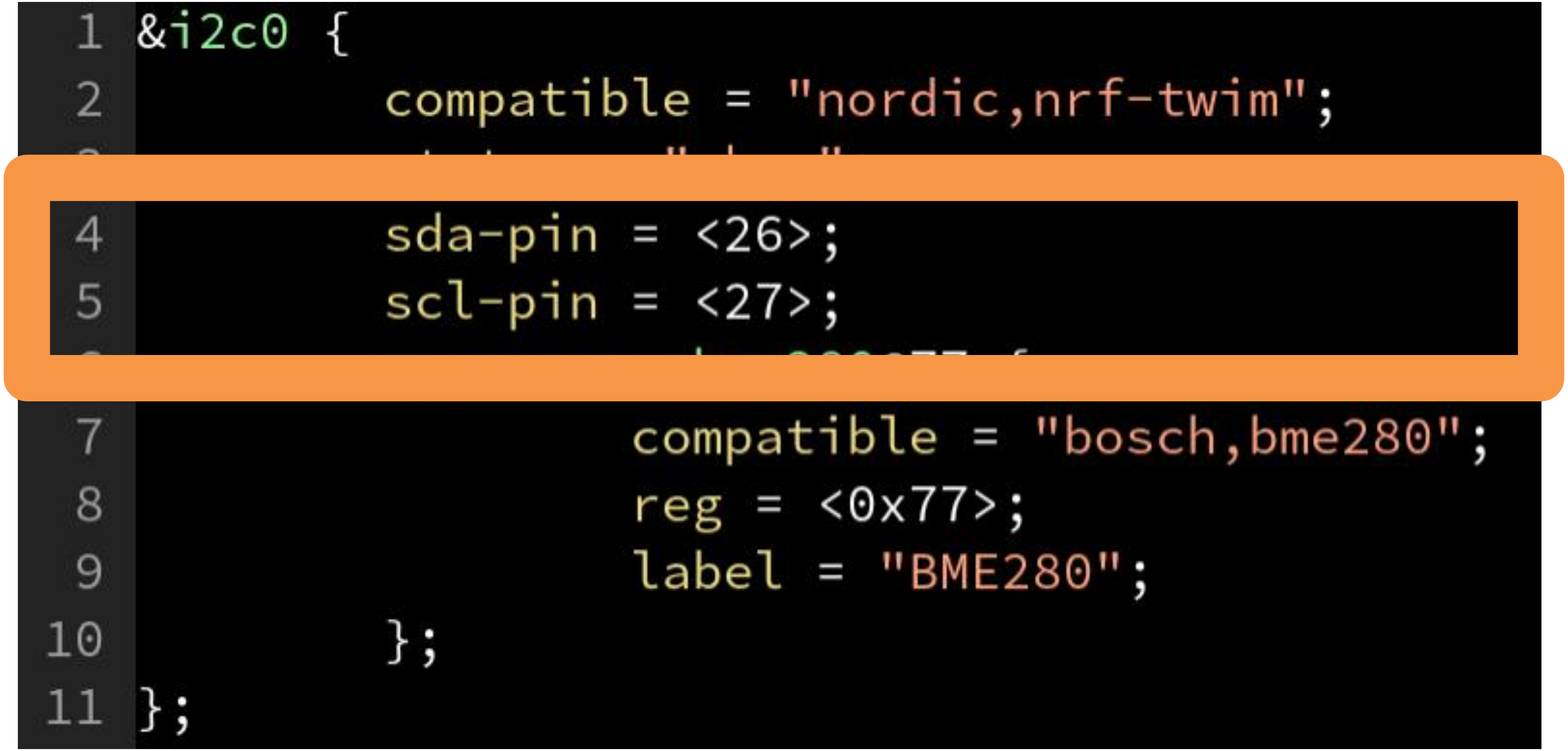
```
&i2c0 {
        compatible = "nordic,nrf-twim";
        status = "okay";
        sda-pin = <26>;
        scl-pin = <27>;
        mysensor: bme280@77 {
                compatible = "bosch,bme280";
                reg = <0x77>;
                label = "BME280";
        };
};
```

Enable i2c0



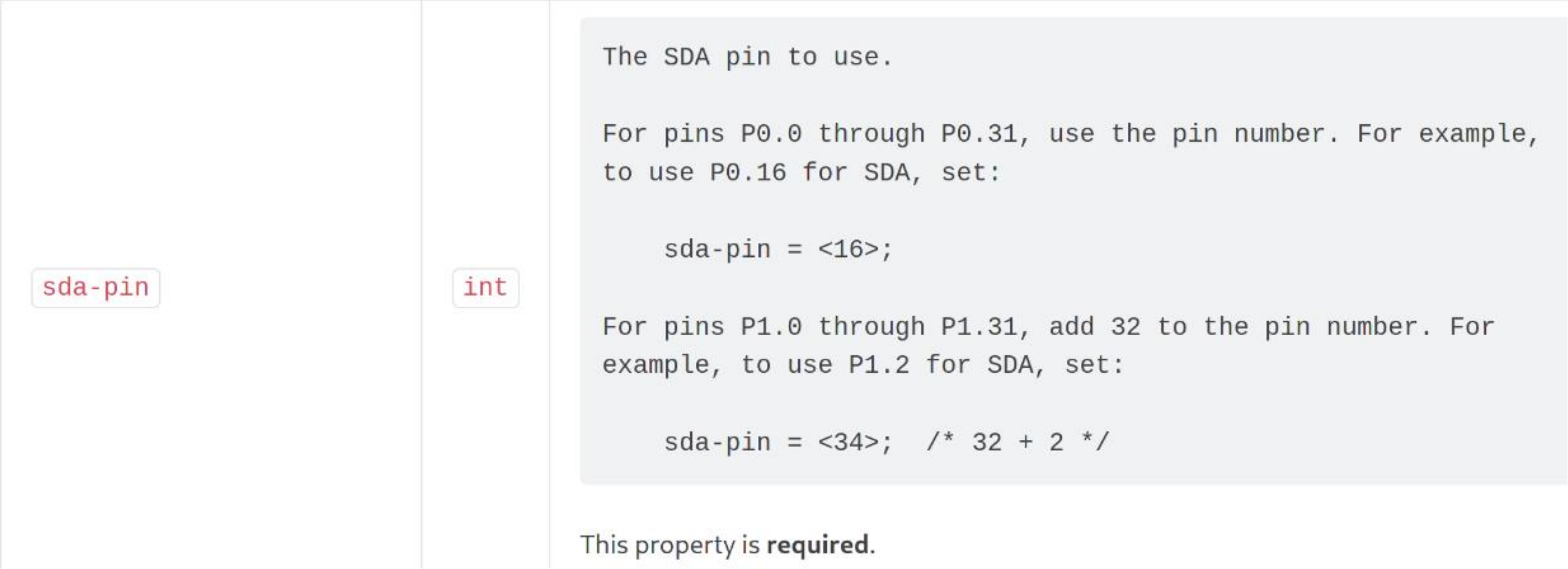
```
&i2c0 {
	compatible = "nordic,nrf-twim";
	status = "okay";
	[illegible]
	scl-pin = <27>;
	mysensor: bme280@77 {
		compatible = "bosch,bme280";
		reg = <0x77>;
		label = "BME280";
	};
};
```

Configure bus pins



```
&i2c0 {
        compatible = "nordic,nrf-twim";
        [illegible]
        sda-pin = <26>;
        scl-pin = <27>;
        [illegible]
                compatible = "bosch,bme280";
                reg = <0x77>;
                label = "BME280";
        };
};
```

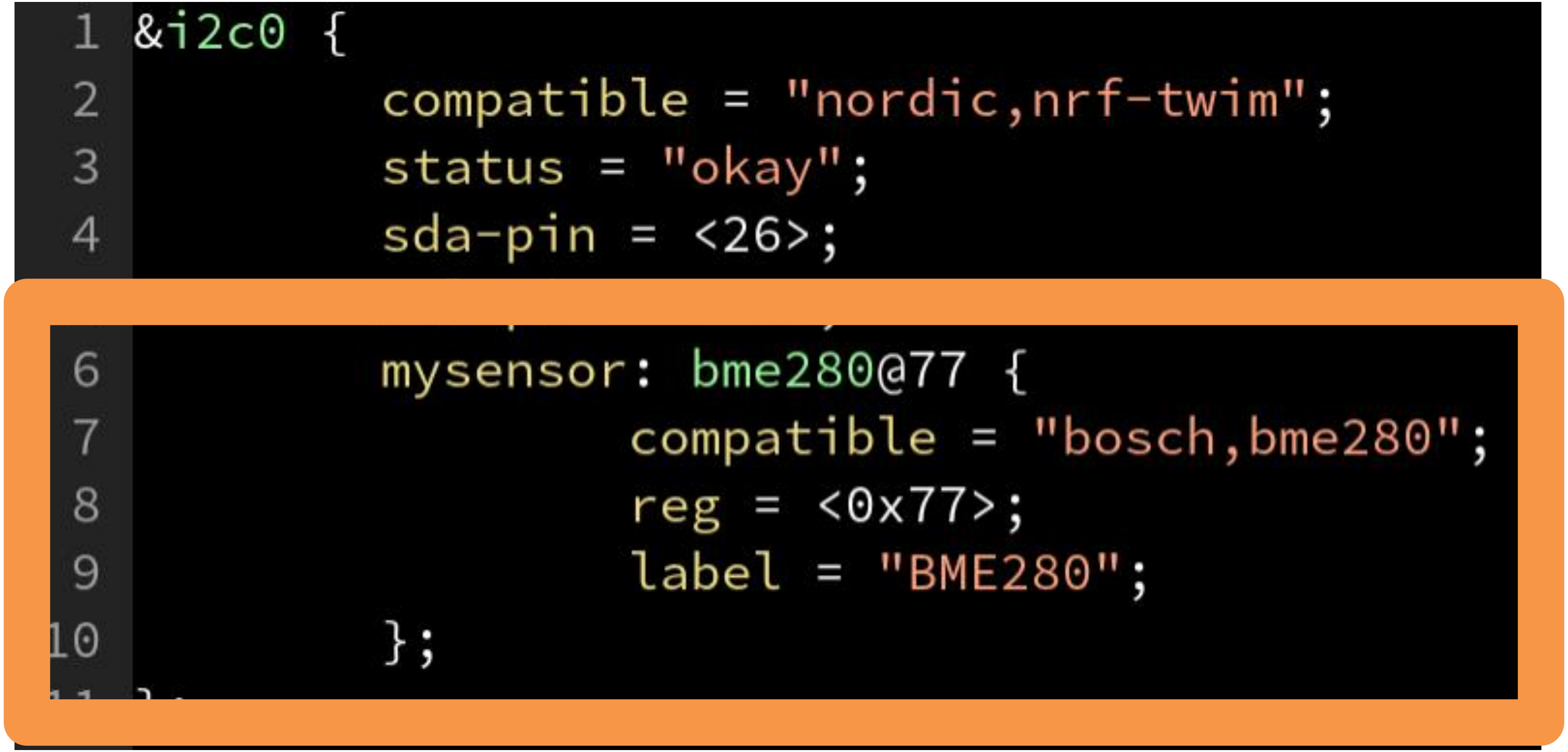
Use the bindings index!



`sda-pin`	`int`	The SDA pin to use.
		For pins P0.0 through P0.31, use the pin number. For example, to use P0.16 for SDA, set:
		`    sda-pin = <16>;`
		For pins P1.0 through P1.31, add 32 to the pin number. For example, to use P1.2 for SDA, set:
		`    sda-pin = <34>;  /* 32 + 2 */`
		This property is **required**.

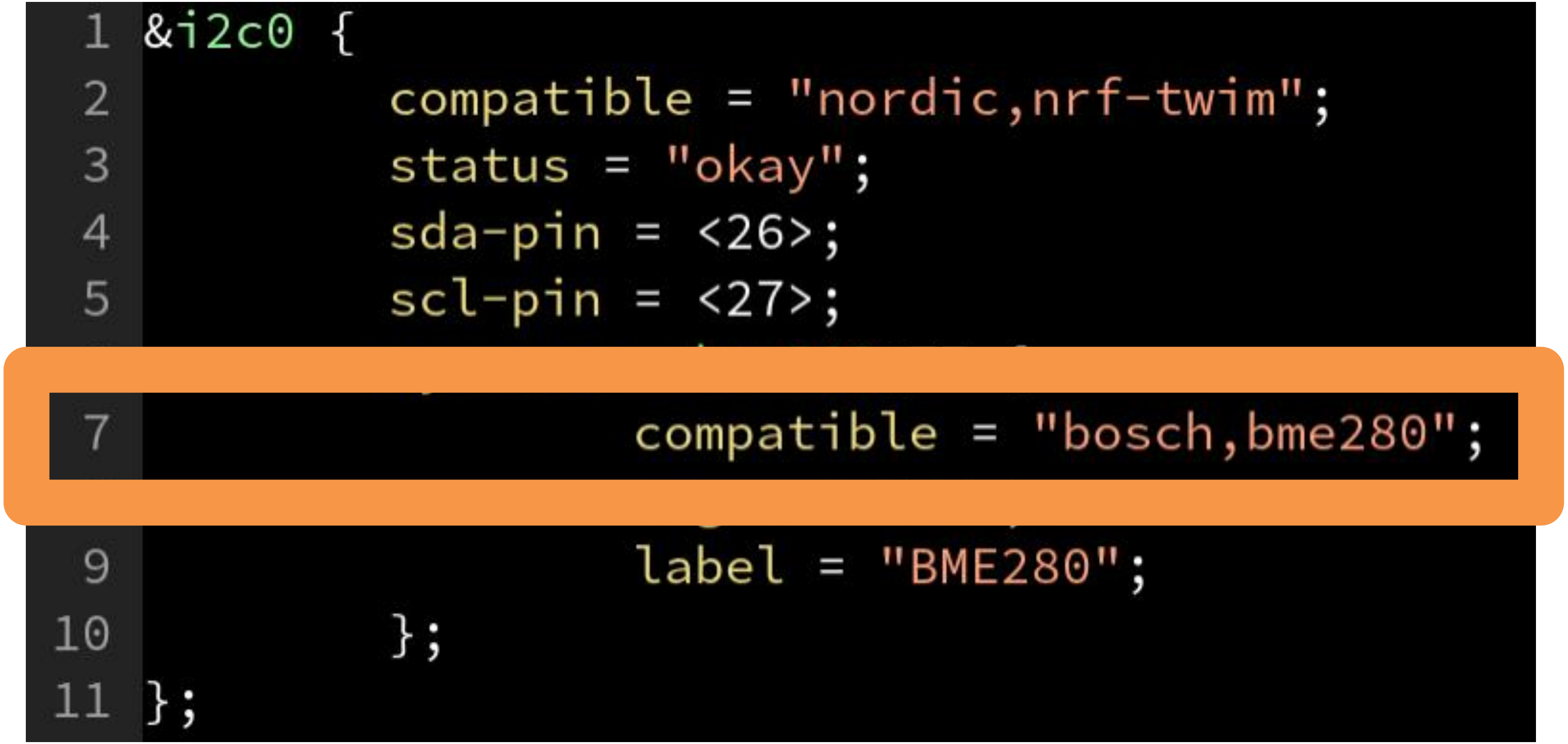
https://docs.zephyrproject.org/2.6.0/reference/devicetree/bindings/i2c/nordic%2Cnrf-twim.html

Define sensor



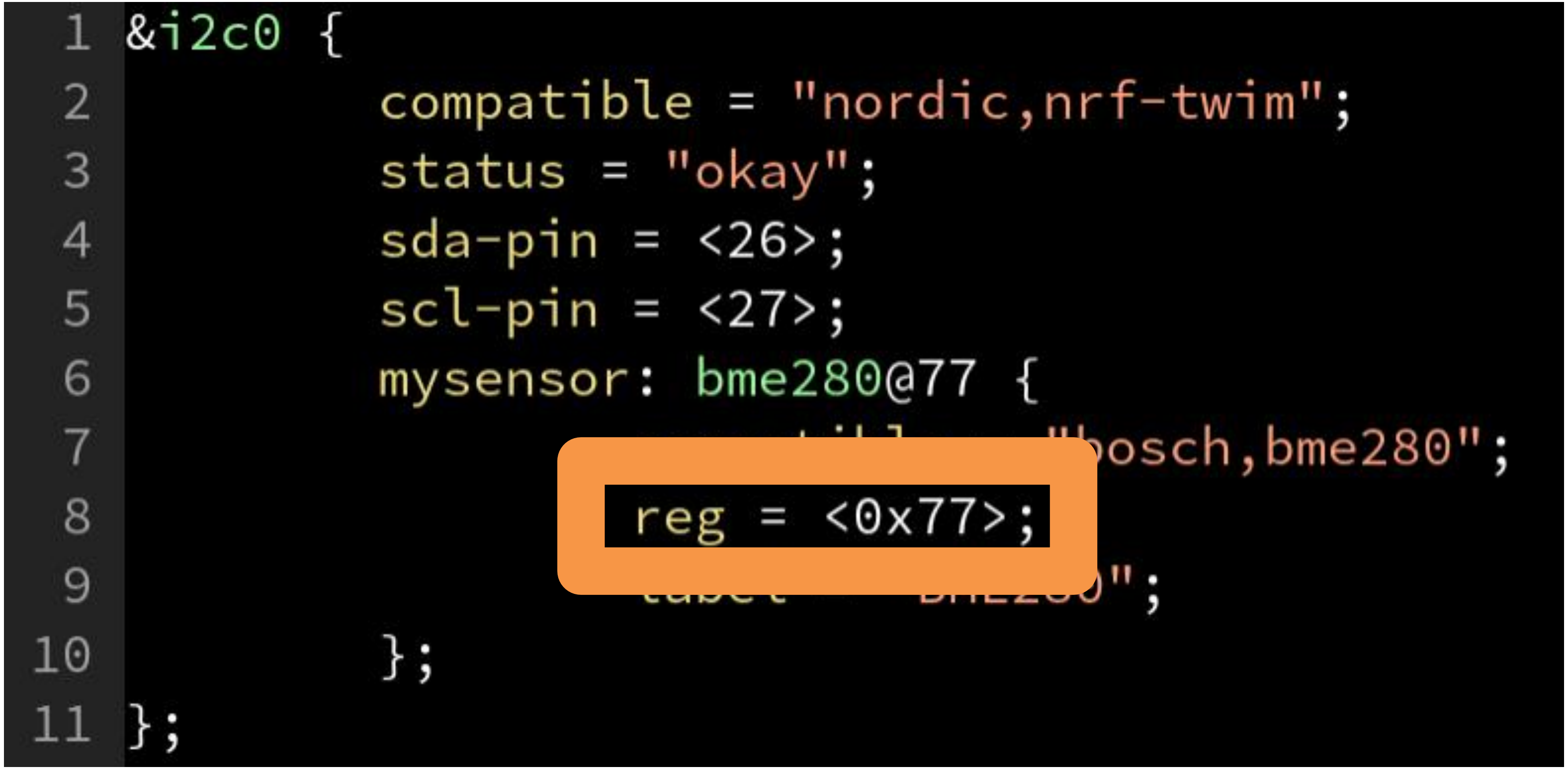
```
&i2c0 {
        compatible = "nordic,nrf-twim";
        status = "okay";
        sda-pin = <26>;

        mysensor: bme280@77 {
                compatible = "bosch,bme280";
                reg = <0x77>;
                label = "BME280";
        };
};
```

Same compatible!



```
&i2c0 {
        compatible = "nordic,nrf-twim";
        status = "okay";
        sda-pin = <26>;
        scl-pin = <27>;
                compatible = "bosch,bme280";
                label = "BME280";
        };
};
```

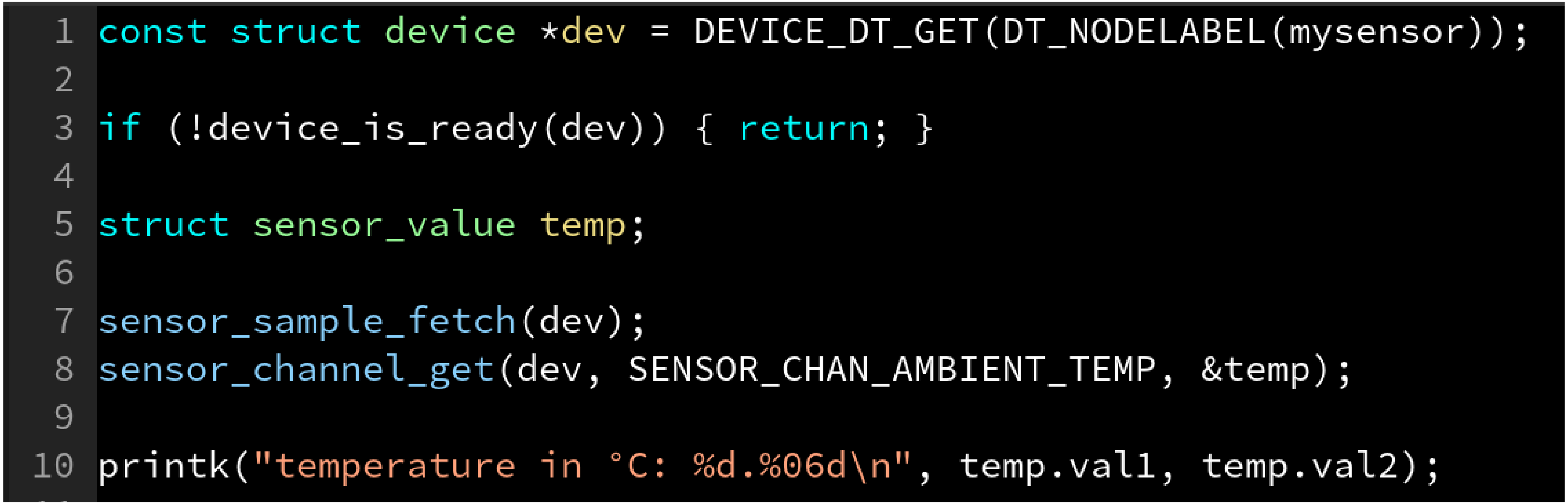
reg is an I2C address now



```
&i2c0 {
        compatible = "nordic,nrf-twim";
        status = "okay";
        sda-pin = <26>;
        scl-pin = <27>;
        mysensor: bme280@77 {
                compatible = "bosch,bme280";
                reg = <0x77>;
                label = "BME280";
        };
};
```

And in C?

No changes!

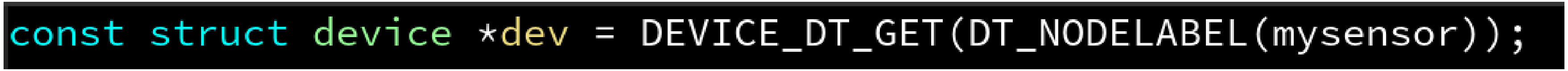


```c
const struct device *dev = DEVICE_DT_GET(DT_NODELABEL(mysensor));

if (!device_is_ready(dev)) { return; }

struct sensor_value temp;

sensor_sample_fetch(dev);
sensor_channel_get(dev, SENSOR_CHAN_AMBIENT_TEMP, &temp);

printk("temperature in °C: %d.%06d\n", temp.val1, temp.val2);
```

Zephyr™ Project
Developer Summit

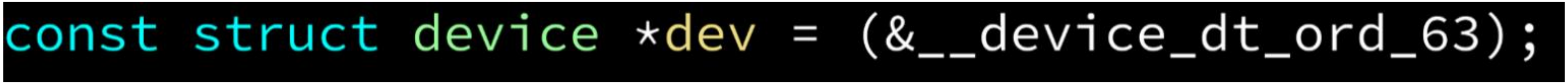
No overhead

App macro expansion



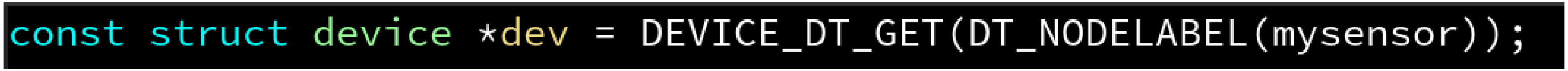
```
const struct device *dev = DEVICE_DT_GET(DT_NODELABEL(mysensor));
```

→



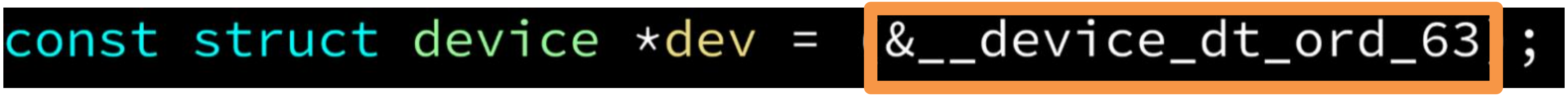
```
const struct device *dev = (&__device_dt_ord_63);
```

What’s going on?



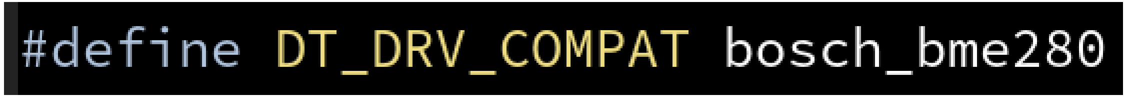
```
const struct device *dev = DEVICE_DT_GET(DT_NODELABEL(mysensor));
```

→



```
const struct device *dev = (&__device_dt_ord_63);
```

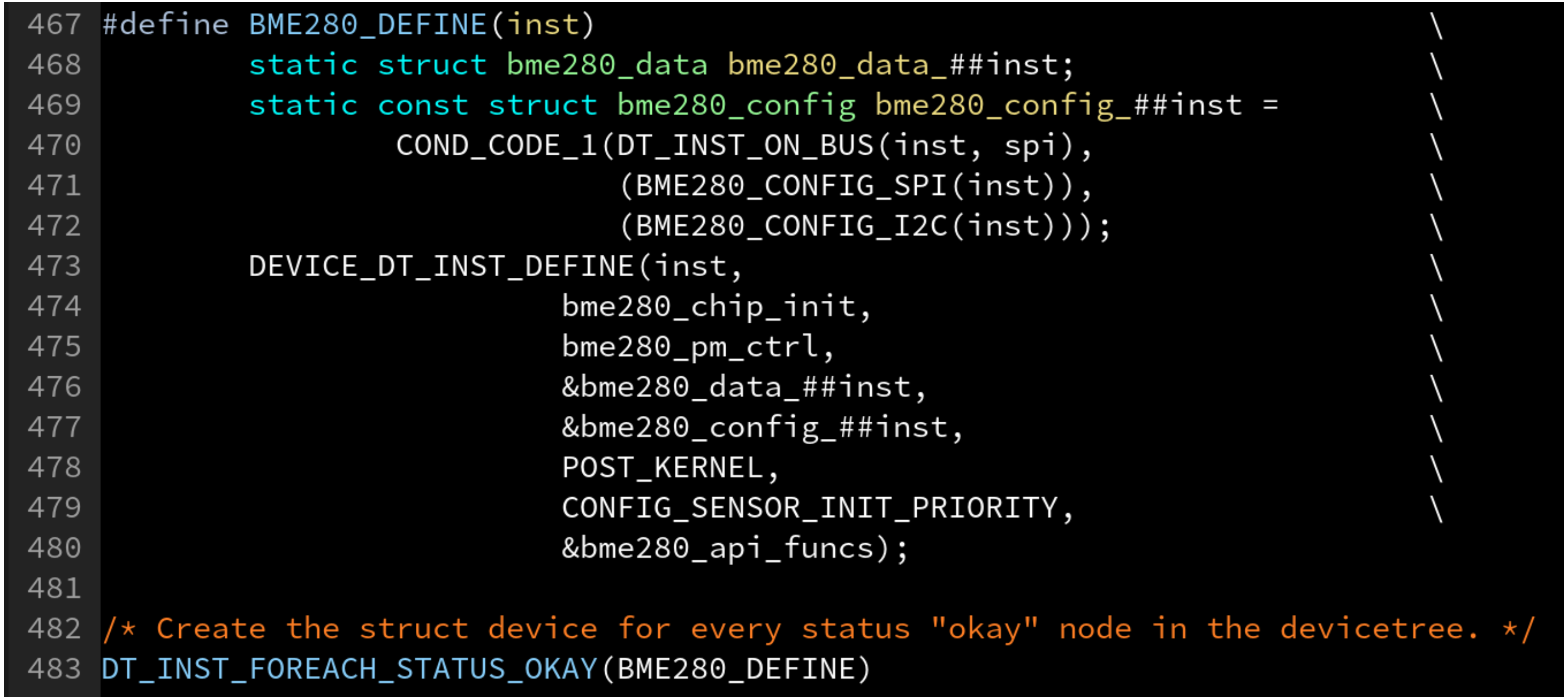
Instance numbers are per-compatible



```
#define DT_DRV_COMPAT bosch_bme280
```

https://github.com/zephyrproject-rtos/zephyr/blob/zephyr-v2.6.0/drivers/sensor/bme280/bme280.h

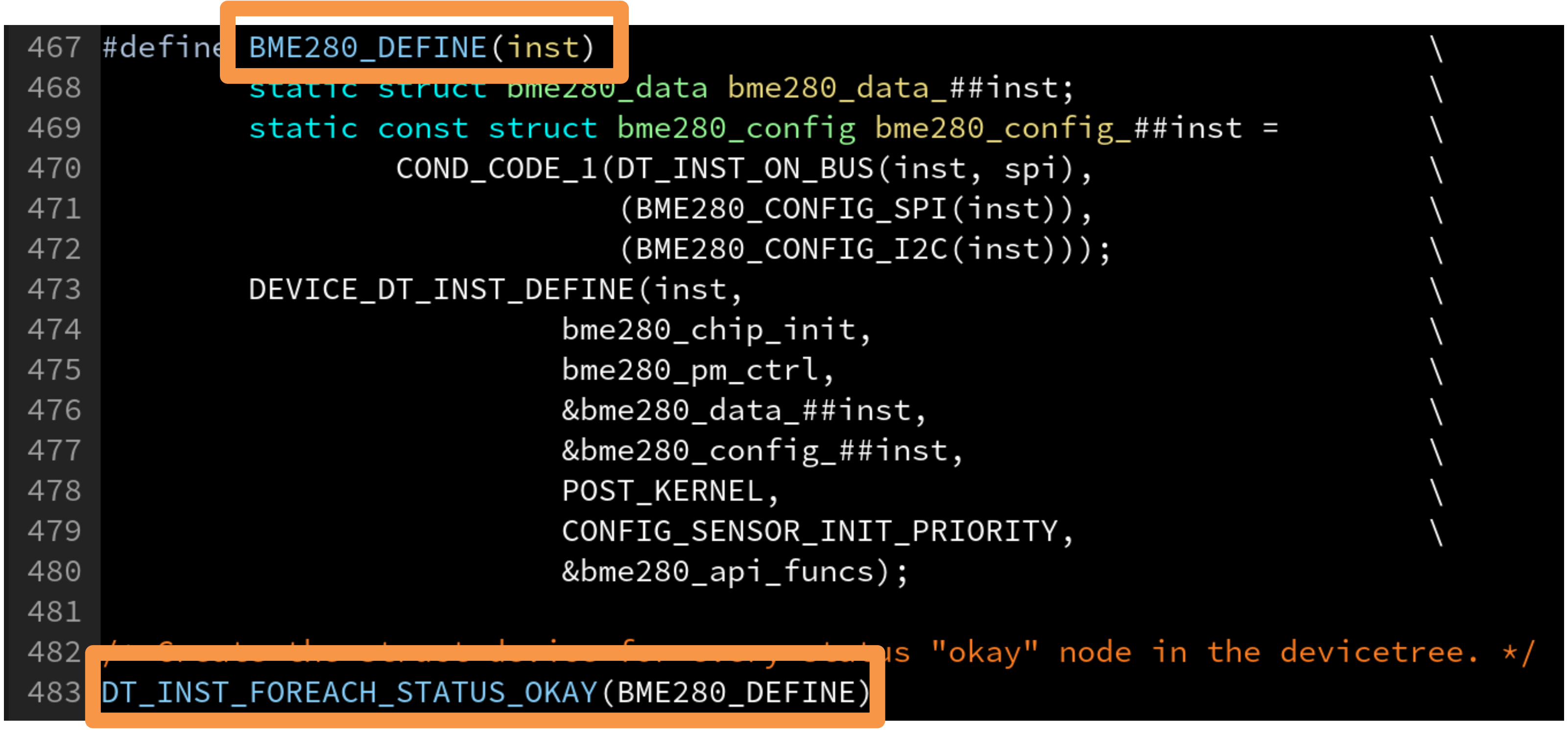
Driver (bme280.c)



```
#define BME280_DEFINE(inst)                                             \
        static struct bme280_data bme280_data_##inst;                   \
        static const struct bme280_config bme280_config_##inst =        \
                COND_CODE_1(DT_INST_ON_BUS(inst, spi),                  \
                            (BME280_CONFIG_SPI(inst)),                  \
                            (BME280_CONFIG_I2C(inst)));                 \
        DEVICE_DT_INST_DEFINE(inst,                                     \
                         bme280_chip_init,                              \
                         bme280_pm_ctrl,                                \
                         &bme280_data_##inst,                           \
                         &bme280_config_##inst,                         \
                         POST_KERNEL,                                   \
                         CONFIG_SENSOR_INIT_PRIORITY,                   \
                         &bme280_api_funcs);

/* Create the struct device for every status "okay" node in the devicetree. */
DT_INST_FOREACH_STATUS_OKAY(BME280_DEFINE)
```

https://github.com/zephyrproject-rtos/zephyr/blob/zephyr-v2.6.0/drivers/sensor/bme280/bme280.c

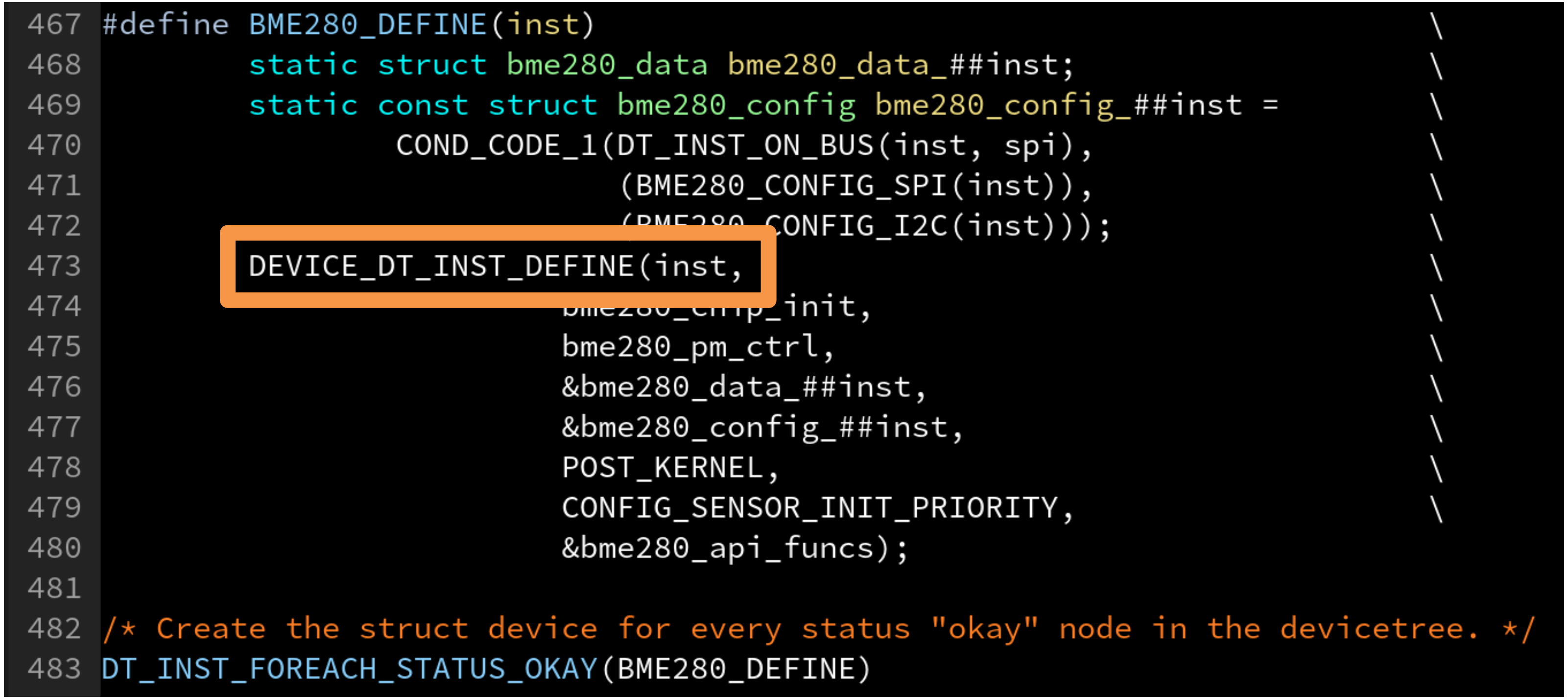
One device per instance of the compatible



```
#define BME280_DEFINE(inst)                                             \
        static struct bme280_data bme280_data_##inst;                   \
        static const struct bme280_config bme280_config_##inst =        \
                COND_CODE_1(DT_INST_ON_BUS(inst, spi),                  \
                            (BME280_CONFIG_SPI(inst)),                  \
                            (BME280_CONFIG_I2C(inst)));                 \
        DEVICE_DT_INST_DEFINE(inst,                                     \
                         bme280_chip_init,                              \
                         bme280_pm_ctrl,                                \
                         &bme280_data_##inst,                           \
                         &bme280_config_##inst,                         \
                         POST_KERNEL,                                   \
                         CONFIG_SENSOR_INIT_PRIORITY,                   \
                         &bme280_api_funcs);

/* Create the struct device for every status "okay" node in the devicetree. */
DT_INST_FOREACH_STATUS_OKAY(BME280_DEFINE)
```

https://github.com/zephyrproject-rtos/zephyr/blob/zephyr-v2.6.0/drivers/sensor/bme280/bme280.c

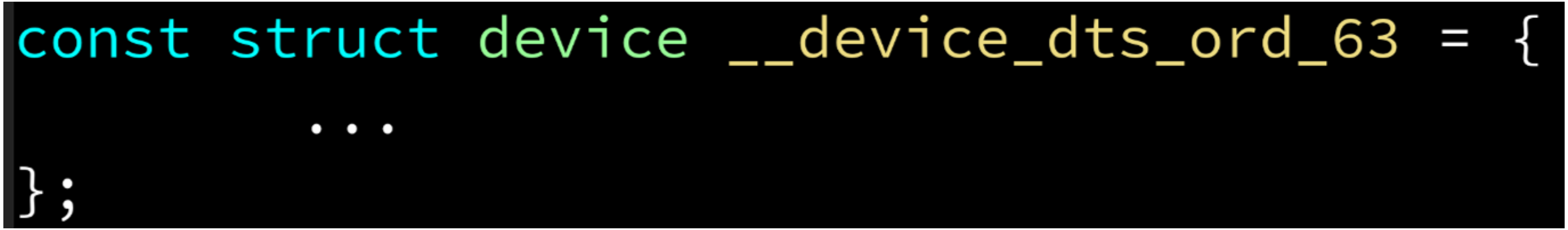
Added in v2.5



```c
#define BME280_DEFINE(inst)                                             \
        static struct bme280_data bme280_data_##inst;                   \
        static const struct bme280_config bme280_config_##inst =        \
                COND_CODE_1(DT_INST_ON_BUS(inst, spi),                  \
                            (BME280_CONFIG_SPI(inst)),                  \
                            (BME280_CONFIG_I2C(inst)));                 \
        DEVICE_DT_INST_DEFINE(inst,                                     \
                         bme280_chip_init,                              \
                         bme280_pm_ctrl,                                \
                         &bme280_data_##inst,                           \
                         &bme280_config_##inst,                         \
                         POST_KERNEL,                                   \
                         CONFIG_SENSOR_INIT_PRIORITY,                   \
                         &bme280_api_funcs);

/* Create the struct device for every status "okay" node in the devicetree. */
DT_INST_FOREACH_STATUS_OKAY(BME280_DEFINE)
```

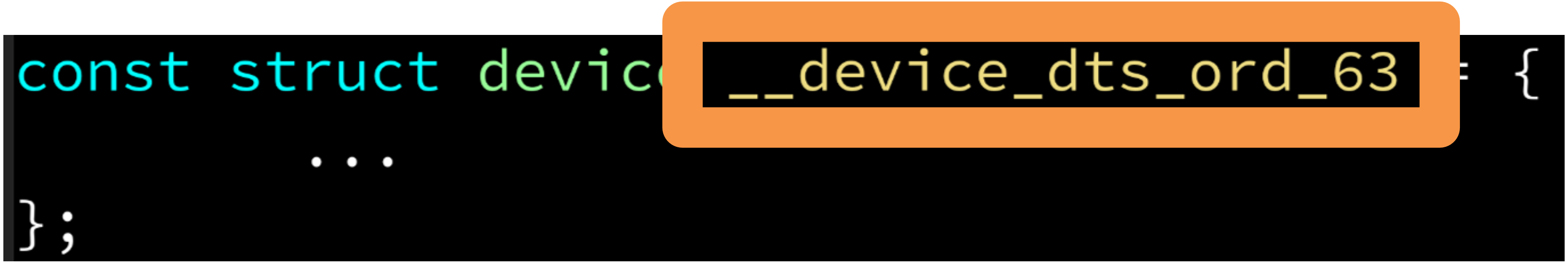
https://github.com/zephyrproject-rtos/zephyr/blob/zephyr-v2.6.0/drivers/sensor/bme280/bme280.c

Driver macro expansion



```
const struct device __device_dts_ord_63 = {
        ...
};
```

Driver macro expansion



```
const struct device __device_dts_ord_63 = {
        ...
};
```

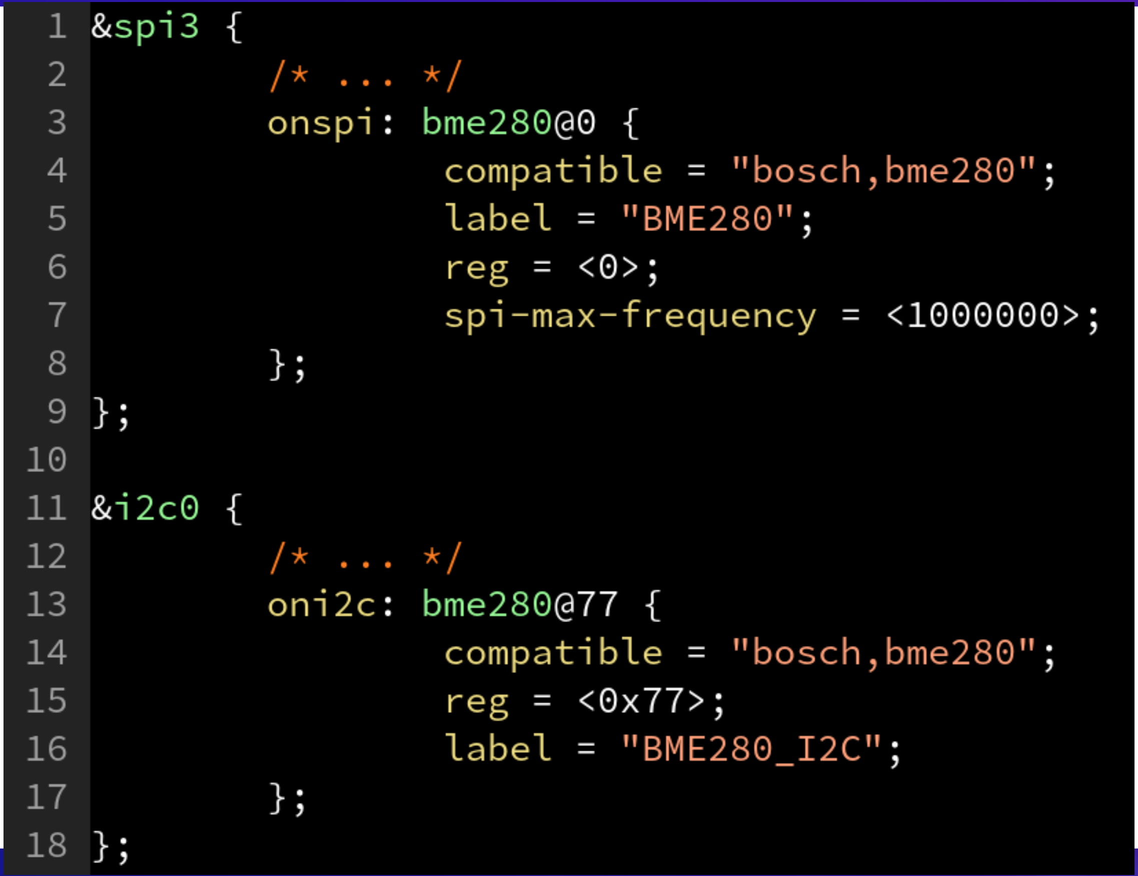
Global variables

“Ordinal” numbers

Zephyr™ Project
Developer Summit

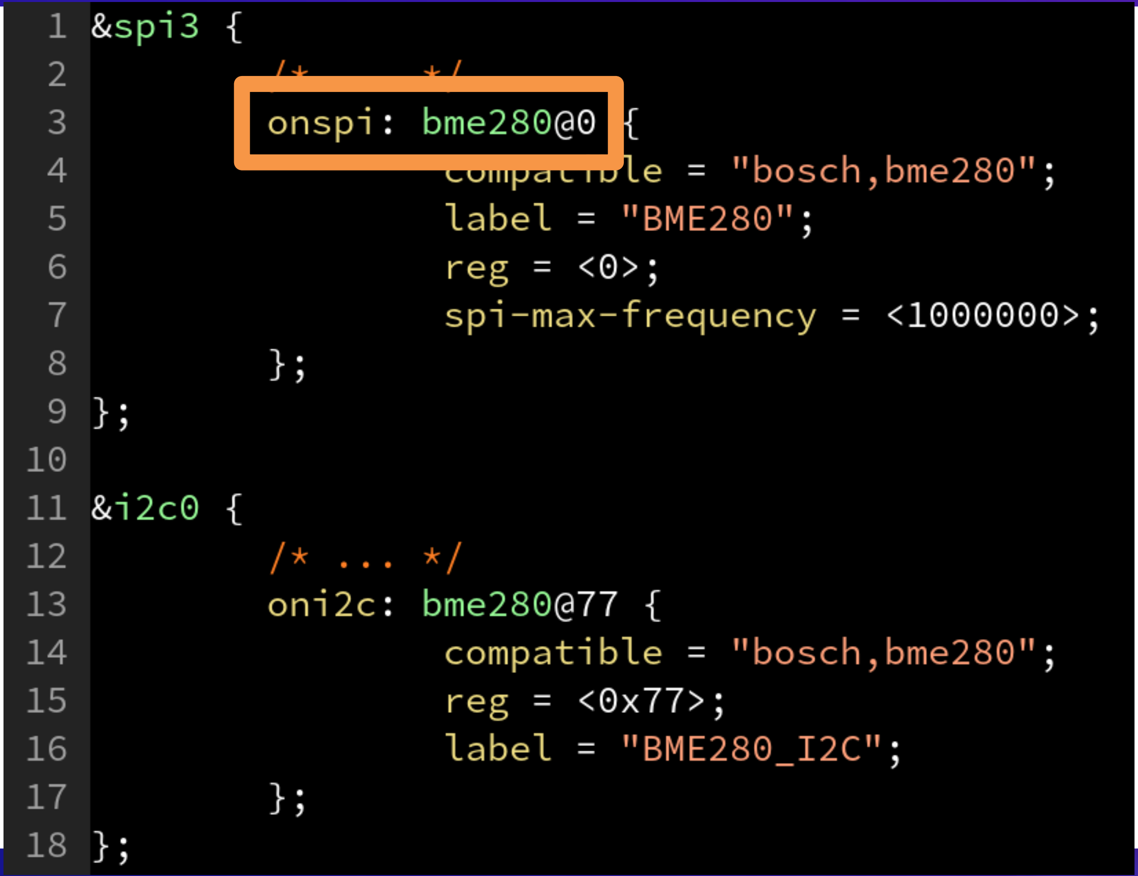
Multiple instances

Two BME280 instances



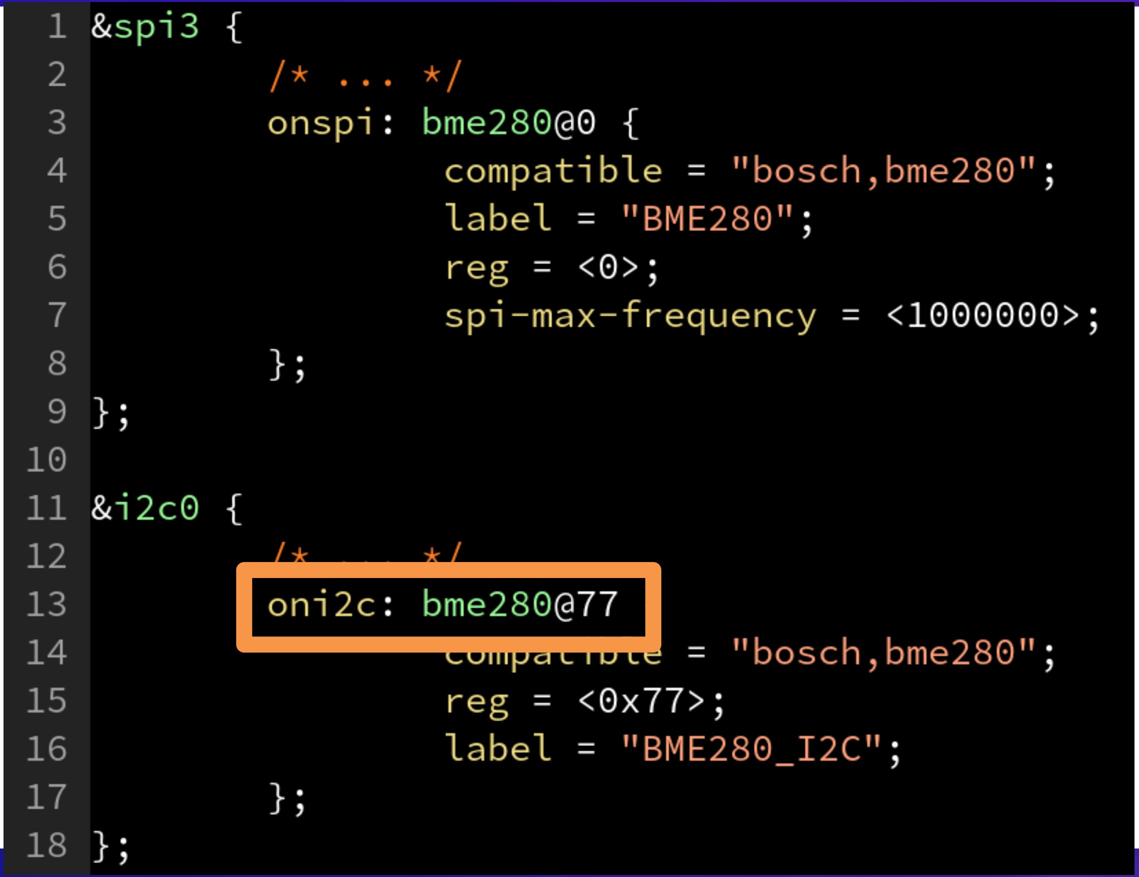
```
&spi3 {
        /* ... */
        onspi: bme280@0 {
                compatible = "bosch,bme280";
                label = "BME280";
                reg = <0>;
                spi-max-frequency = <1000000>;
        };
};

&i2c0 {
        /* ... */
        oni2c: bme280@77 {
                compatible = "bosch,bme280";
                reg = <0x77>;
                label = "BME280_I2C";
        };
};
```

One on SPI



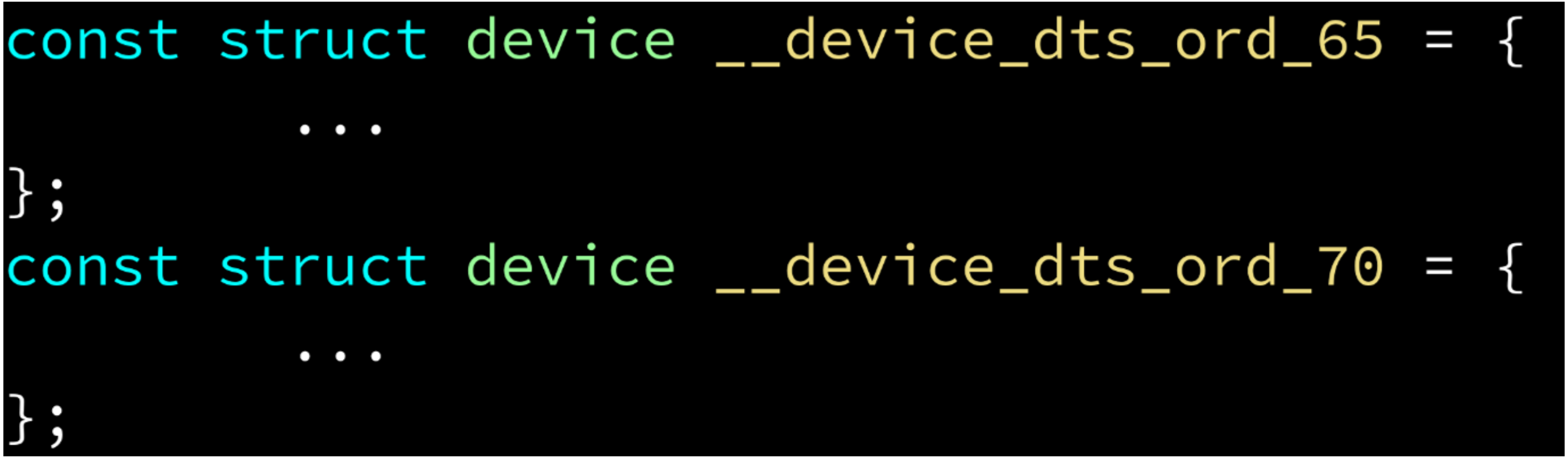
```
&spi3 {
        /* ... */
        onspi: bme280@0 {
                compatible = "bosch,bme280";
                label = "BME280";
                reg = <0>;
                spi-max-frequency = <1000000>;
        };
};

&i2c0 {
        /* ... */
        oni2c: bme280@77 {
                compatible = "bosch,bme280";
                reg = <0x77>;
                label = "BME280_I2C";
        };
};
```

One on I2C



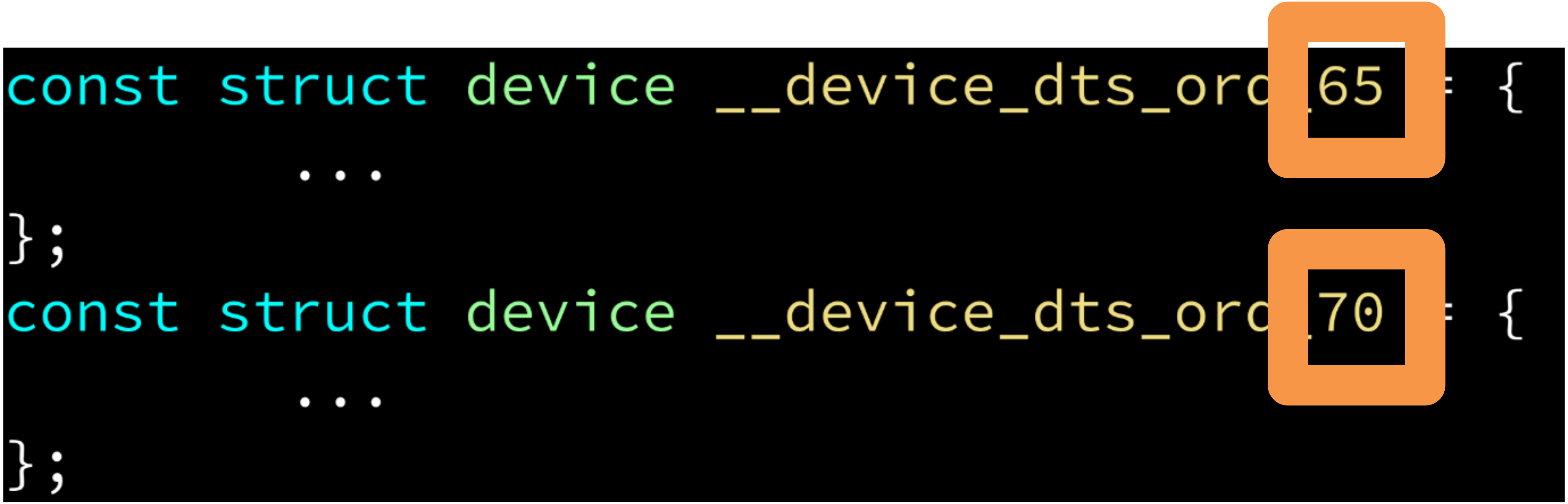
```
&spi3 {
        /* ... */
        onspi: bme280@0 {
                compatible = "bosch,bme280";
                label = "BME280";
                reg = <0>;
                spi-max-frequency = <1000000>;
        };
};

&i2c0 {
        /* ... */
        oni2c: bme280@77
                compatible = "bosch,bme280";
                reg = <0x77>;
                label = "BME280_I2C";
        };
};
```

Driver macro expansion



```
const struct device __device_dts_ord_65 = {
        ...
};
const struct device __device_dts_ord_70 = {
        ...
};
```

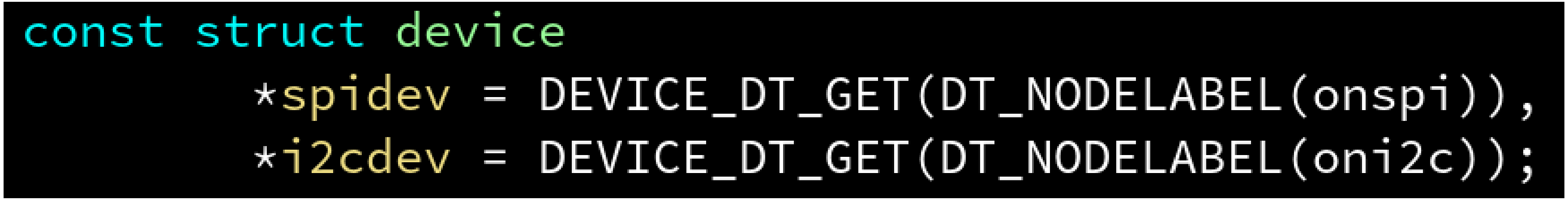
Instance numbers != ordinals



```
const struct device __device_dts_ord_65 = {
        ...
};
const struct device __device_dts_ord_70 = {
        ...
};
```

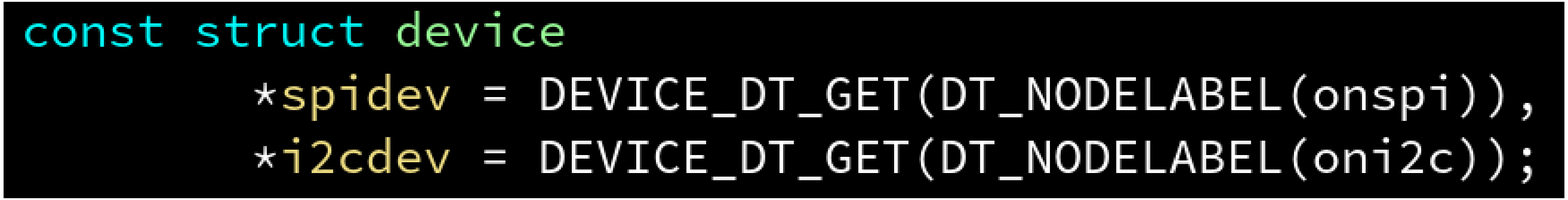
Instances 0 and 1;
ordinals 65 and 70 (here)

Two nodes, two devices



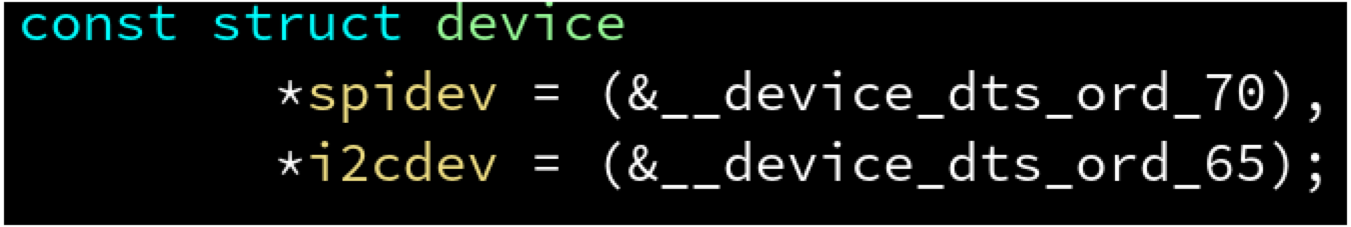
```
const struct device
        *spidev = DEVICE_DT_GET(DT_NODELABEL(onspi)),
        *i2cdev = DEVICE_DT_GET(DT_NODELABEL(oni2c));
```

App macro expansion



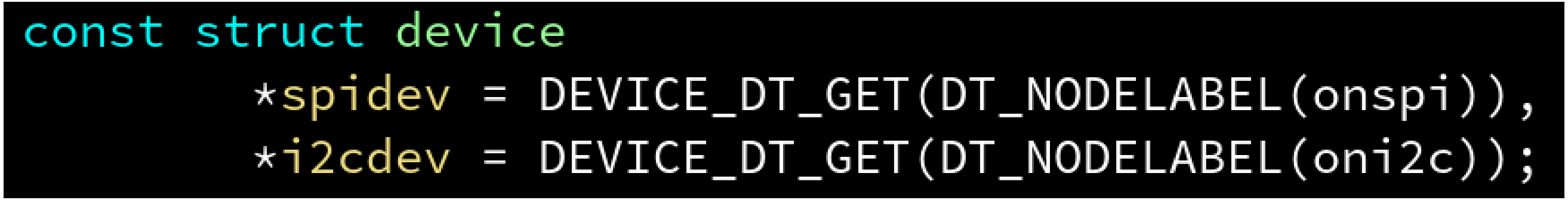
```
const struct device
        *spidev = DEVICE_DT_GET(DT_NODELABEL(onspi)),
        *i2cdev = DEVICE_DT_GET(DT_NODELABEL(oni2c));
```

→



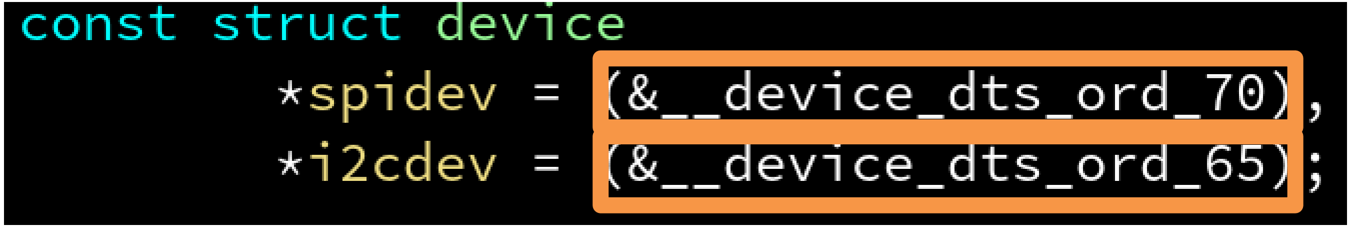
```
const struct device
        *spidev = (&__device_dts_ord_70),
        *i2cdev = (&__device_dts_ord_65);
```

No overhead



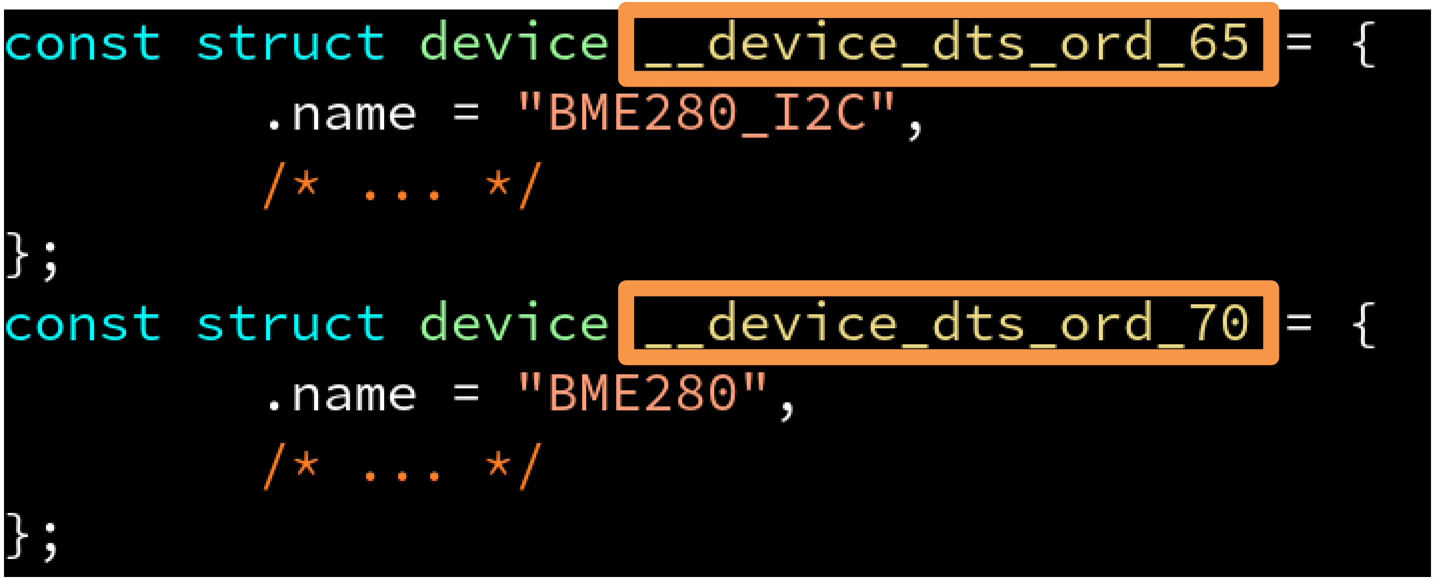
```
const struct device
            *spidev = DEVICE_DT_GET(DT_NODELABEL(onspi)),
            *i2cdev = DEVICE_DT_GET(DT_NODELABEL(oni2c));
```

→

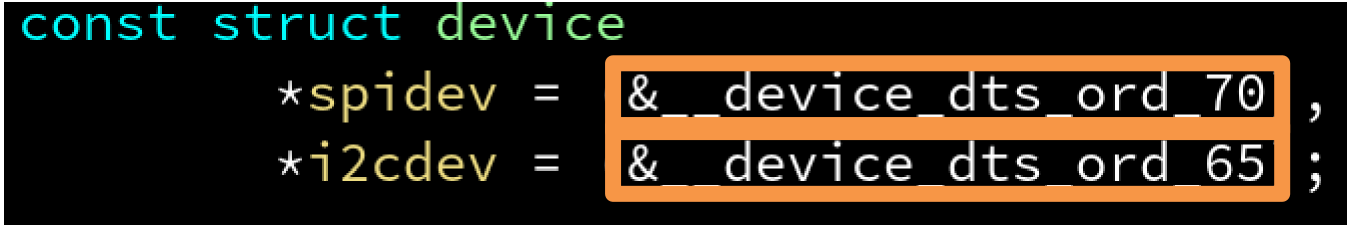


```
const struct device
            *spidev = (&__device_dts_ord_70),
            *i2cdev = (&__device_dts_ord_65);
```

The devicetree vanishes



```
const struct device __device_dts_ord_65 = {
        .name = "BME280_I2C",
        /* ... */
};
const struct device __device_dts_ord_70 = {
        .name = "BME280",
        /* ... */
};
```



```
const struct device
        *spidev = &__device_dts_ord_70,
        *i2cdev = &__device_dts_ord_65;
```

Zephyr™ Project
Developer Summit

But wait!
There’s more!

DT captures dependencies

Ordinals capture this!

Dependencies at build time!

https://docs.zephyrproject.org/2.6.0/reference/devicetree/api.html#inter-node-dependencies

Profound implications

Initialization order

Device power management

Wrapping up

Devicetree → devices → APIs

Global build-time devices

Hierarchy

No overhead

As many devices as you want...

… even zero

Use the bindings index;
read the source

Zephyr™ Project
Developer Summit

Thanks!

Zephyr™ Project

Developer Summit

June 8-10, 2021 ▪ @ZephyrIoT