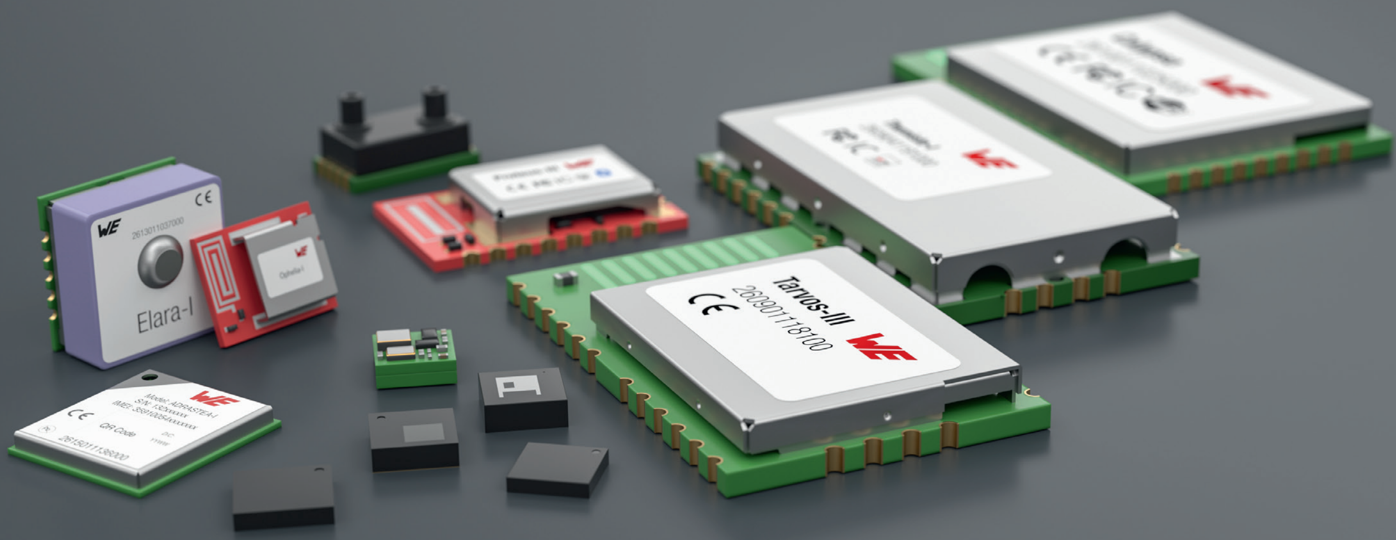


# ANR034

## How to use Zephyr WE sensor drivers

Version 1.0

July 19, 2023

WÜRTH ELEKTRONIK MORE THAN YOU EXPECT

# Revision history

| Manual version | Notes | Date |
|---|---|---|
| 1.0 | • Initial version | July 2023 |

# Abbreviations

| Abbreviation | Description |
|---|---|
| CS | Chip Select line of the SPI interface |
| INT | Interrupt line of the sensor |
| MISO | Master In Slave Out line of the SPI interface |
| MOSI | Master Out Slave In line of the SPI interface |
| ODR | Output Data Rate |
| OS | Operating System |
| SCL | Clock line of the $I^2C$ interface |
| SCLK | Clock line of the SPI interface |
| SDA | Data line of the $I^2C$ interface |
| *WE* | Würth Elektronik eiSos |

# Contents

# 1. Introduction

Zephyr OS [6, 7] is an operating system for micro-controllers distributed by the Linux Foundation and used in many end devices. This operating system not only contains features related to the operating system, like multi-threading and dynamic memory allocation, but also offers many functions for the operation of external electronic components, such as sensors, radio modules and displays.

For that reason Würth Elektronik eiSos integrated drivers of the *WE* sensors to the Zephyr OS[1], such that Zephyr OS natively brings these functions to the source code of the user application. In other words, the software integration of the provided *WE* sensors into the user's application can be done in a few steps.

This application note demonstrates how to integrate and use a *WE* sensor driver in the user's source code.

# 2. Provided drivers

The following sensor drivers are integrated in Zephyr OS:

| Short name | Article number | Function | Location in source code | Min. Zephyr version |
|---|---|---|---|---|
| WSEN_HIDS[2] [1] | 2525020210001 | Humidity and temperature | *./zephyr/drivers/sensor/wsen_hids* | 3.2.0 |
| WSEN_ITDS[2] [2] | 2533020201601 | 3 Axis Acceleration | *./zephyr/drivers/sensor/wsen_itds* | 2.4.0 |
| WSEN_PADS[2] [3] | 2511020213301 | Absolute pressure | *./zephyr/drivers/sensor/wsen_pads* | 3.4.0 |
| WSEN_PDUS[2] [4] | 2513130810001, 2513130810101, 2513130810201, 2513130810301, 2513130810401, 2513130815401 | Differential pressure | *./zephyr/drivers/sensor/wsen_pdus* | 3.4.0 |
| WSEN_TIDS[2] [5] | 2521020222501 | Temperature | *./zephyr/drivers/sensor/wsen_tids* | 3.4.0 |

Table 1: *WE* sensor drivers

[1]Besides sensor drivers, also board files of Würth Elektronik eiSos radio modules have been added to Zephyr OS. These allow to develop firmware for the radio chip integrated in *WE* radio modules.
[2]Please see our website for updates to the product status.

# 3. Driver integration

To integrate a specific sensor to the user's application code several steps need to be performed. Please find the steps in the subsequent sub chapters.

In addition to the description, there will be example codes to demonstrate the implementation. For that reason the nRF52840 development board (PCA10056 / nrf52840dk_nrf52840) from Nordic Semiconductor is used here. For any other board, please use the name of the corresponding board file instead, for example "we_proteus3ev_nrf52840".

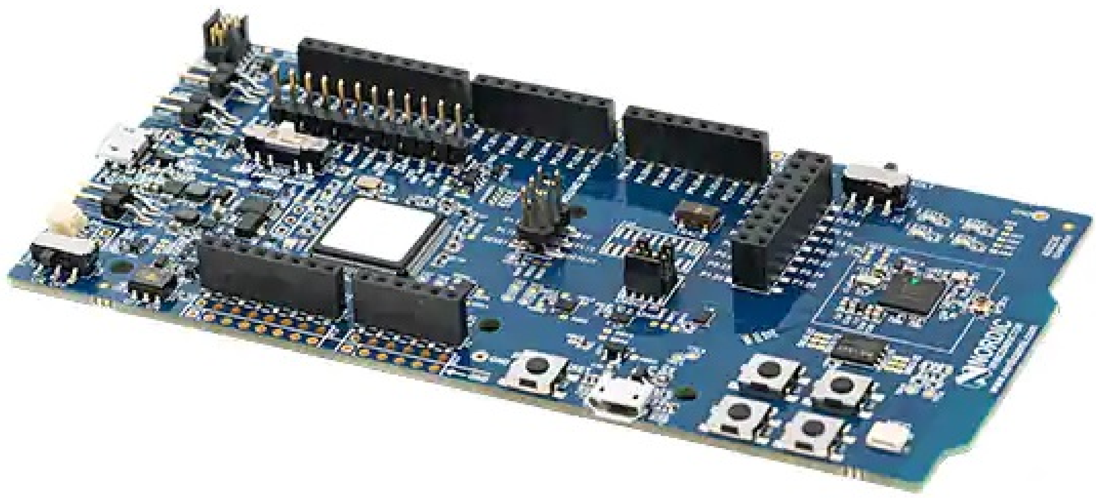

Figure 1: PCA10056 / nrf52840dk_nrf52840 used as board for demonstration reasons

## 3.1. Update device tree

The sensor must be loaded as "device" in the application's source code. To be able to do that, the sensor's device definition must be added to the device tree of the board file of the underlying board.
The description and device tree definition of all supported boards can be found in the sub directories of *./zephyr/boards/*. In case of the nrf52840dk_nrf52840 board, the device tree is in the file:
*./zephyr/boards/arm/nrf52840dk_nrf52840/nrf52840dk_nrf52840.dts*

It contains the definition of all peripherals available on the chosen board. The corresponding controller pins are defined in a separate file:
*./zephyr/boards/arm/nrf52840dk_nrf52840/nrf52840dk_nrf52840-pinctrl.dtsi*

For sensor integration, we are especially interested in the SPI or I²C interfaces.

```
...
&i2c0 {
    compatible = "nordic,nrf-twi";
    status = "okay";
    pinctrl-0 = <&i2c0_default>;
    pinctrl-1 = <&i2c0_sleep>;
    pinctrl-names = "default", "sleep";
};
...
&spi0 {
   compatible = "nordic,nrf-spi";
   /* Cannot be used together with i2c0. */
   /* status = "okay"; */
   pinctrl-0 = <&spi0_default>;
   pinctrl-1 = <&spi0_sleep>;
   pinctrl-names = "default", "sleep";
};
...
```

Code 1: Example: Device tree of nrf52840dk_nrf52840 (nrf52840dk_nrf52840.dts)

```
...
i2c0_default: i2c0_default {
    group1 {
        psels = <NRF_PSEL(TWIM_SDA, 0, 26)>,
        <NRF_PSEL(TWIM_SCL, 0, 27)>;
    };
};

i2c0_sleep: i2c0_sleep {
    group1 {
        psels = <NRF_PSEL(TWIM_SDA, 0, 26)>,
        <NRF_PSEL(TWIM_SCL, 0, 27)>;
        low-power-enable;
    };
};
...
spi0_default: spi0_default {
    group1 {
        psels = <NRF_PSEL(SPIM_SCK, 0, 27)>,
        <NRF_PSEL(SPIM_MOSI, 0, 26)>,
        <NRF_PSEL(SPIM_MISO, 0, 29)>;
    };
};

spi0_sleep: spi0_sleep {
    group1 {
        psels = <NRF_PSEL(SPIM_SCK, 0, 27)>,
        <NRF_PSEL(SPIM_MOSI, 0, 26)>,
        <NRF_PSEL(SPIM_MISO, 0, 29)>;
        low-power-enable;
    };
};
...
```

Code 2: Example: Pin definition of nrf52840dk_nrf52840 (nrf52840dk_nrf52840-pinctrl.dtsi)

To use our sensor, the interface definition of the underlying device tree must be extended. This can be done using so called "overlay files". Overlay files contain the code that is patched into the standard device tree file. It must have the same name as the device tree file, but with a ".overlay" file name extention. After creation, it must be place in the sub directory "boards" of the project directory.

```
project
├── proj.conf
├── CMakeLists.txt
├── src
│   ├── main.c
│   └── ...
└── boards
    └── nrf52840dk_nrf52840.overlay
```

The overlay adds the sensor to the respective interface. Besides the sensor name, sensor related information must be added, such as the sensor address in case of I²C communication. In the example (see Code 3) the sensor WSEN_TIDS is added. Its 7-bit I²C address is 0x38, the output data rate (ODR) is index 25 and the "data ready interrupt gpio" is gpio number 24 on port 0.

```
&i2c0 {
    tids@38 {
        compatible = "we,wsen-tids";
        reg = <0x38>;
        int-gpios = <&gpio0 24 GPIO_ACTIVE_LOW>;
        odr = <25>;
        temp-high-threshold = <27>;
        temp-low-threshold = <10>;
    };
};
```

Code 3: Example: "./project/boards/nrf52840dk_nrf52840.overlay"

For more examples on overlay files, please refer to chapter A.1.

> **!** Depending on the hardware design either µC internal pull-ups or external pull-ups are mandatory for I²C communication. Here external pull-ups are selected.

## 3.2. Update the project configuration file

After the device tree has been extended, the next step is to update the project configuration file (proj.conf). Here the sensor drivers and the SPI or I²C interface must be enabled for the project.

This is done by adding the macros CONFIG_SENSOR=y and CONFIG_I2C=y (or CONFIG_SPI=y) to the project configuration file.

```
...
CONFIG_I2C=y
CONFIG_SENSOR=y
...
```

Code 4: Example: Enable I²C interface and sensor drivers in the proj.conf

```
...
CONFIG_SPI=y
CONFIG_SENSOR=y
...
```

Code 5: Example: Enable SPI interface and sensor drivers in the proj.conf

Here additional sensor related configurations can be added. Please check the sensor's Kconfig file for information about sensor related options.

## 3.3. Use the sensor in the application source code

After implementing the previous steps the sensor can be loaded as device in the application's source code. This is done using the `DEVICE_DT_GET_ONE` command and the sensor's name. The function `device_is_ready` checks whether then sensor device has been loaded and returns `true` in case of success.

```
    ...
    const struct device *dev = DEVICE_DT_GET_ONE(we_wsen_tids);

    if (!device_is_ready(dev)) {
        LOG_ERR("sensor: device not ready.\n");
        return;
    }
    ...
```

Code 6: Example: Load WSEN-TIDS sensor

The sensor and communication interface initialisation is done Zephyr internally. A default configuration is applied to it, which is sufficient to immediately request the first sensor values. In case non-default configurations shall be applied, please call the respective configuration functions, now.

Otherwise the Zephyr-functions `sensor_sample_fetch` and `sensor_channel_get` can be used to read the first sensor values.

```
static void process_sample(const struct device *dev)
{
    struct sensor_value temperature;

    if (sensor_sample_fetch(dev) < 0) {
        LOG_ERR("Failed to fetch TIDS sensor sample.");
        return;
    }

    if (sensor_channel_get(dev, SENSOR_CHAN_AMBIENT_TEMP, &temperature) < 0) {
        LOG_ERR("Failed to read TIDS temperature channel.");
        return;
    }

    /* Display temperature */
    LOG_INF("Temperature: %.1f C", sensor_value_to_double(&temperature));
}
```

Code 7: Example: Read TIDS sensor sample

With the implementation of these steps, the sensor integration was succeeded. To configure the sensor with respect to special needs, please refer to the Zephyr sensor functions that are documented in the *./zephyr/include/zephyr/drivers/sensor.h* file.

# 4. Hardware setup

The previous chapter has shown how to integrate the driver of the respective sensor into the application code.
But how to connect the sensor to the host processor in terms of hardware? To do that the respective I²C or SPI pins must be connected, as shown in the following drawings.

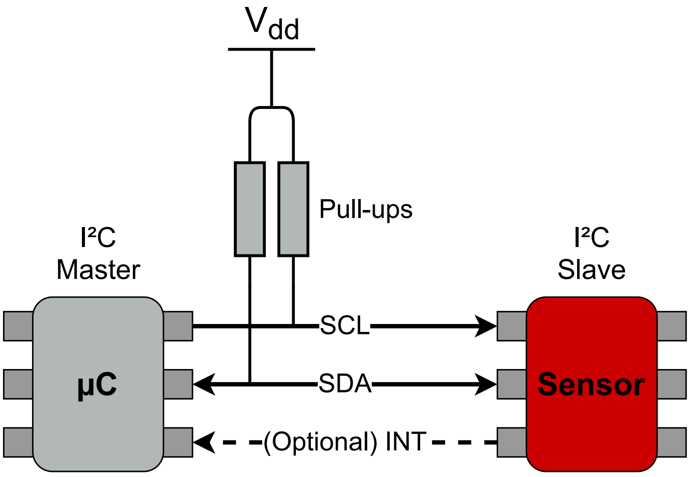


Figure 2: I²C connection between the host and the sensor

For I²C connection the *SCL* (I²C clock) and *SDA* (I²C data) lines must be connected. If the micro controller does not enable internal pull-up resistors, external ones must be applied to that lines. In case the host uses interrupt based functions of the sensor, the interrupt pins *INT* must be connected in addition.

> Please check the overlay files in chapter `A.1` if internal pull-up resistors have been enabled.

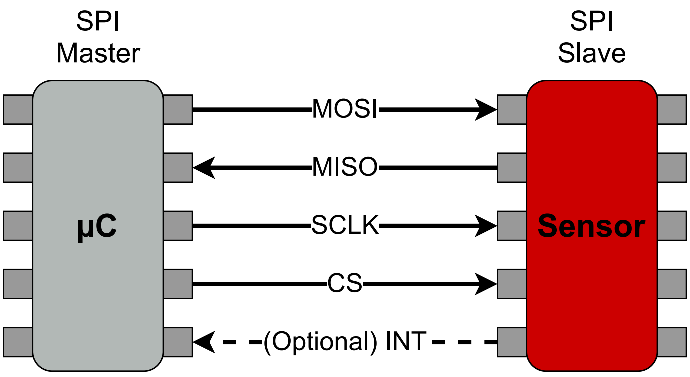


Figure 3: SPI connection between the host and the sensor

For SPI connection the *SCLK* (SPI clock), *CS* (SPI chip select), *MOSI* and *MISO* (SPI data) lines must be connected. In case the host uses interrupt based functions of the sensor, the interrupt pins *INT* must be connected in addition.

For the reference design and pin numbers of the respective sensor, please refer to its user manual. The pin numbers of the SPI or I²C interface of the host controller can be found in its device tree (see section 3.1).

## 4.1. Connect to an Würth Elektronik eiSos radio module

As mentioned in a previous chapter, the hardware description of some Würth Elektronik eiSos radio modules is natively available in Zephyr OS as well. Using this, an application firmware that is running on the radio module using *WE* sensor can be developed simply by selecting the corresponding software components.

Würth Elektronik eiSos provides this service of a custom firmware development as well. A customer specific firmware may include "Custom configuration of standard firmware" plus additional options or functions and tasks that are customer specific and not part of the standard firmware.
Further scheduled firmware updates of the standard firmware will not be applied to this variant automatically. Applying updates or further functions require a customer request and release procedure.
This also results in a customer exclusive module with a unique ordering number.
An example for this level of customization are functions like host-less operation where the module will perform data generation (e.g. by reading a SPI or I²C sensor) and cyclic transmission of this data to a data collector, while sleeping or being passive most of the time.
Also replacing UART with SPI as host communication interface is classified such a custom specific option.
Certification critical changes need to be re-evaluated by an external qualified measurement laboratory. These critical changes may occur when e.g. changing radio parameters, the channel access method, the duty-cycle or in case of various other functions and options possibly used or changed by a customer specific firmware.

Please contact your local field sales engineer (FSE) or *WCS@we-online.com* for quotes regarding these topics.

# 5. References

[1] Würth Elektronik. Web page: WSEN-HIDS Humidity and temperature sensor. https://www.we-online.de/katalog/de/article/2525020210001.

[2] Würth Elektronik. Web page: WSEN-ITDS 3 axis acceleration sensor. https://www.we-online.de/katalog/de/article/2533020201601.

[3] Würth Elektronik. Web page: WSEN-PADS Absolute pressure sensor. https://www.we-online.de/katalog/de/article/2511020213301.

[4] Würth Elektronik. Web page: WSEN-PDUS Differential pressure sensor. https://www.we-online.de/katalog/de/article/2513130810001.

[5] Würth Elektronik. Web page: WSEN-TIDS Temperature sensor. https://www.we-online.de/katalog/de/article/2521020222501.

[6] Zephyr OS on GitHub. https://github.com/zephyrproject-rtos/zephyr.

[7] Zephyr Organisation. https://zephyrproject.org/.

# 6. Important notes

The Application Note and its containing information ("Information") is based on Würth Elektronik eiSos GmbH & Co. KG and its subsidiaries and affiliates ("WE eiSos") knowledge and experience of typical requirements concerning these areas. It serves as general guidance and shall not be construed as a commitment for the suitability for customer applications by WE eiSos. While WE eiSos has used reasonable efforts to ensure the accuracy of the Information, WE eiSos does not guarantee that the Information is error-free, nor makes any other representation, warranty or guarantee that the Information is completely accurate or up-to-date. The Information is subject to change without notice. To the extent permitted by law, the Information shall not be reproduced or copied without WE eiSos' prior written permission. In any case, the Information, in full or in parts, may not be altered, falsified or distorted nor be used for any unauthorized purpose.

WE eiSos is not liable for application assistance of any kind. Customer may use WE eiSos' assistance and product recommendations for customer's applications and design. No oral or written Information given by WE eiSos or its distributors, agents or employees will operate to create any warranty or guarantee or vary any official documentation of the product e.g. data sheets and user manuals towards customer and customer shall not rely on any provided Information. THE INFORMATION IS PROVIDED "AS IS". CUSTOMER ACKNOWLEDGES THAT WE EISOS MAKES NO REPRESENTATIONS AND WARRANTIES OF ANY KIND RELATED TO, BUT NOT LIMITED TO THE NON-INFRINGEMENT OF THIRD PARTIES' INTELLECTUAL PROPERTY RIGHTS OR THE MERCHANTABILITY OR FITNESS FOR A PURPOSE OR USAGE. WE EISOS DOES NOT WARRANT OR REPRESENT THAT ANY LICENSE, EITHER EXPRESS OR IMPLIED, IS GRANTED UNDER ANY PATENT RIGHT, COPYRIGHT, MASK WORK RIGHT, OR OTHER INTELLECTUAL PROPERTY RIGHT RELATING TO ANY COMBINATION, MACHINE, OR PROCESS IN WHICH WE EISOS INFORMATION IS USED. INFORMATION PUBLISHED BY WE EISOS REGARDING THIRD-PARTY PRODUCTS OR SERVICES DOES NOT CONSTITUTE A LICENSE FROM WE eiSos TO USE SUCH PRODUCTS OR SERVICES OR A WARRANTY OR ENDORSEMENT THEREOF.

The responsibility for the applicability and use of WE eiSos' components in a particular customer design is always solely within the authority of the customer. Due to this fact it is up to the customer to evaluate and investigate, where appropriate, and decide whether the device with the specific characteristics described in the specification is valid and suitable for the respective customer application or not. The technical specifications are stated in the current data sheet and user manual of the component. Therefore the customers shall use the data sheets and user manuals and are cautioned to verify that they are current. The data sheets and user manuals can be downloaded at *www.we-online.com*. Customers shall strictly observe any product-specific notes, cautions and warnings. WE eiSos reserves the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time without notice.

WE eiSos will in no case be liable for customer's use, or the results of the use, of the components or any accompanying written materials. IT IS CUSTOMER'S RESPONSIBILITY TO VERIFY THE RESULTS OF THE USE OF THIS INFORMATION IN IT'S OWN PARTICULAR ENGINEERING AND PRODUCT ENVIRONMENT AND CUSTOMER ASSUMES THE ENTIRE RISK OF DOING SO OR FAILING TO DO SO. IN NO CASE WILL WE EISOS BE LIABLE FOR

CUSTOMER'S USE, OR THE RESULTS OF IT'S USE OF THE COMPONENTS OR ANY ACCOMPANYING WRITTEN MATERIAL IF CUSTOMER TRANSLATES, ALTERS, ARRANGES, TRANSFORMS, OR OTHERWISE MODIFIES THE INFORMATION IN ANY WAY, SHAPE OR FORM.

If customer determines that the components are valid and suitable for a particular design and wants to order the corresponding components, customer acknowledges to minimize the risk of loss and harm to individuals and bears the risk for failure leading to personal injury or death due to customers usage of the components. The components have been designed and developed for usage in general electronic equipment only. The components are not authorized for use in equipment where a higher safety standard and reliability standard is especially required or where a failure of the components is reasonably expected to cause severe personal injury or death, unless WE eiSos and customer have executed an agreement specifically governing such use. Moreover WE eiSos components are neither designed nor intended for use in areas such as military, aerospace, aviation, nuclear control, submarine, transportation, transportation signal, disaster prevention, medical, public information network etc. WE eiSos must be informed about the intent of such usage before the design-in stage. In addition, sufficient reliability evaluation checks for safety must be performed on every component which is used in electrical circuits that require high safety and reliability functions or performance. COSTUMER SHALL INDEMNIFY WE EISOS AGAINST ANY DAMAGES ARISING OUT OF THE USE OF THE COMPONENTS IN SUCH SAFETY-CRITICAL APPLICATIONS.

# List of Figures



# List of Tables

# A. Appendix

## A.1. Overlay file content

```
&i2c0 {
    hids@5f {
        compatible = "we,wsen-hids";
        reg = <0x5f>;
        drdy-gpios = <&gpio0 24 GPIO_ACTIVE_HIGH>;
        odr = "1";
    };
};

&pinctrl {
    i2c0_default: i2c0_default {
        group1 {
            psels = <NRF_PSEL(TWIM_SDA, 0, 26)>,
            <NRF_PSEL(TWIM_SCL, 0, 27)>;
            bias-pull-up;
        };
    };

    i2c0_sleep: i2c0_sleep {
        group1 {
            psels = <NRF_PSEL(TWIM_SDA, 0, 26)>,
            <NRF_PSEL(TWIM_SCL, 0, 27)>;
            low-power-enable;
        };
    };
};
```

Code 8: HIDS $I^2C$ overlay including pins and pull-up

```
&i2c0 {
    tids@38 {
        compatible = "we,wsen-tids";
        reg = <0x38>;
        int-gpios = <&gpio0 24 GPIO_ACTIVE_LOW>;
        odr = <25>;
        temp-high-threshold = <27>;
        temp-low-threshold = <10>;
    };
};

&pinctrl {
    i2c0_default: i2c0_default {
        group1 {
            psels = <NRF_PSEL(TWIM_SDA, 0, 26)>,
            <NRF_PSEL(TWIM_SCL, 0, 27)>;
            bias-pull-up;
        };
    };

    i2c0_sleep: i2c0_sleep {
        group1 {
            psels = <NRF_PSEL(TWIM_SDA, 0, 26)>,
            <NRF_PSEL(TWIM_SCL, 0, 27)>;
            low-power-enable;
        };
    };
};
```

Code 9: TIDS I²C overlay including pins and pull-up

```
&i2c0 {
    pads@5d {
        compatible = "we,wsen-pads";
        reg = <0x5d>;
        drdy-gpios = <&gpio0 24 GPIO_ACTIVE_HIGH>;
        odr = <1>;
    };
};

&pinctrl {
    i2c0_default: i2c0_default {
        group1 {
            psels = <NRF_PSEL(TWIM_SDA, 0, 26)>,
            <NRF_PSEL(TWIM_SCL, 0, 27)>;
            bias-pull-up;
        };
    };

    i2c0_sleep: i2c0_sleep {
        group1 {
            psels = <NRF_PSEL(TWIM_SDA, 0, 26)>,
            <NRF_PSEL(TWIM_SCL, 0, 27)>;
            low-power-enable;
        };
    };
};
```

Code 10: PADS $I^2C$ overlay including pins and pull-up

```
&spi0 {
    compatible = "nordic,nrf-spi";
    status = "okay";
    pinctrl-0 = <&spi0_default>;
    pinctrl-1 = <&spi0_sleep>;
    pinctrl-names = "default", "sleep";
    cs-gpios = <&gpio0 24 GPIO_ACTIVE_LOW>;
    pads@0 {
        compatible = "we,wsen-pads";
        reg = <0>;
        drdy-gpios = <&gpio0 22 GPIO_ACTIVE_HIGH>;
        odr = <1>;
        spi-max-frequency = <4000000>;
    };
};

&pinctrl {
    spi0_default: spi0_default {
        group1 {
            psels = <NRF_PSEL(SPIM_SCK, 0, 27)>,
            <NRF_PSEL(SPIM_MOSI, 0, 26)>,
            <NRF_PSEL(SPIM_MISO, 1, 8)>;
        };
    };
    spi0_sleep: spi0_sleep {
        group1 {
            psels = <NRF_PSEL(SPIM_SCK, 0, 27)>,
            <NRF_PSEL(SPIM_MOSI, 0, 26)>,
            <NRF_PSEL(SPIM_MISO, 1, 8)>;
            low-power-enable;
        };
    };
};
```

Code 11: PADS SPI overlay including new pins

```
&i2c0 {
    pdus@78 {
        compatible = "we,wsen-pdus";
        reg = <0x78>;
        sensor-type = <3>;
    };
};

&pinctrl {
    i2c0_default: i2c0_default {
        group1 {
            psels = <NRF_PSEL(TWIM_SDA, 0, 26)>,
            <NRF_PSEL(TWIM_SCL, 0, 27)>;
            bias-pull-up;
        };
    };

    i2c0_sleep: i2c0_sleep {
        group1 {
            psels = <NRF_PSEL(TWIM_SDA, 0, 26)>,
            <NRF_PSEL(TWIM_SCL, 0, 27)>;
            low-power-enable;
        };
    };
};
```

Code 12: PDUS $I^2C$ overlay including pins and pull-up

**Contact**
Würth Elektronik eiSos GmbH & Co. KG
Division Wireless Connectivity & Sensors

Max-Eyth-Straße 1
74638 Waldenburg
Germany

Tel.: +49 651 99355-0
Fax.: +49 651 99355-69
www.we-online.com/wireless-connectivity

**WÜRTH ELEKTRONIK** MORE THAN YOU EXPECT