ANR003 PROTEUS-I

LOW POWER APPLICATION WITH PERIODIC WAKE-UP

VERSION 2.2

JUNE 18, 2019

Revision history

Manual version	HW version	Notes	Date
1.0	2.1	• Initial version	February 2017
2.0	2.1	• New corporate design	June 2018
2.1	2.1	• Updated product name from AMB2621 to Proteus-I	November 2018
2.2	2.1	• Updated file name to new AppNote name structure. Updated important notes, legal notice & license terms chapters.	June 2019

Contents

1 Introduction

The Proteus-I is a Bluetooth module based on Nordic Semiconductors nRF52832 SoC that brings various BLE and low power features.
The SoC has a system-off mode (deep-sleep) that allows to preserve power when the module is sleeping. Leaving this mode can be triggered by pin interrupt, low power comparator or NFC (NFC pins can be accessed on the Proteus-I-EV).
However, in many applications a periodic wake-up from a sleep mode is needed. Therefore the chip offers a system-on mode that wakes on any selected event.
In this application note the realization and test results of a periodic wake-up using the real time clock (RTC) is presented. The tested implementation sets the module to sleep and wakes it periodically.
While awake, the module advertises and waits for incoming connections. Therefore the Nordic "UART Example for peripheral devices" is taken and updated in a few steps such that the periodic wake-up and the low power capabilities of the chip can be demonstrated using messages on the UART.
The test results in chapter 3 show that we can periodically switch between sleep and normal mode. When sleeping it consumes less than 2µA with RTC enabled.

2 Realization

2.1 Prerequisites

- The evaluation board Proteus-I-EV and a Segger flash adapter

- Software provided by Nordic Semiconductor: The BLE stack Softdevice S132 V3.0.0, the software development kit SDK nRF5 V 12.1.0 and the example code "Nordic UART Example for peripheral devices" (ble_app_uart_pca10040_s132)
- Keil µVision installed on your PC (the example base upon version 5.20.0.0)

2.2 Implementation

The goal is to update the Nordic "UART example" such that the module goes to sleep (system on) mode if no connection request was received during advertising for a predefined time. After a sleep period, the module is supposed to wake up after a predefined time and start advertising again to be ready for incoming connections. To realize the automatic wake-up a timer will be implemented that uses the real time clock (RTC) and the internal low frequency oscillator (so no external 32768 Hz watch crystal is needed).

Due to copyright rules of the Nordic SDK we are not allowed to supply you with a zip file containing all needed files for this demonstration. Please install the SDK from Nordic and add or patch the corresponding files and project settings.

To do so, please perform the following steps:

1. Load the Nordic "UART example for peripherals" from the Nordic SDK nRF5 V 12.1.0 and check whether it compiles without errors.
2. Update the board file, such that the code can run on the Proteus-I-EV platform:
   a) You find these changes in the Appendix (Boards.h and AMB2621.h). You need to create and add AMB2621.h and patch the project settings and some files of the SDK's demo project. Following up the needed changes (already contained in the files of the appendix)
   b) Update the pin numbers according to the Proteus-I design.
   c) Set the *RTS* and *CTS* UART pins to 0, since they are not used in this example.
   d) Invert the LEDs. Each LED takes about 3mA, when lighted. Thus we prefer to flash them only for a short time.
   e) Use the internal RC-oscillator as low frequency clock.

```
#define NRF_CLOCK_LFCLKSRC {\
    .source = NRF_CLOCK_LF_SRC_RC,\
    .rc_ctiv = 16,\
    .rc_temp_ctiv = 2,\
}
```

3. Compile the updated code and check for errors.
4. Flash the BLE stack S132 V3.0.0 and the compiled code onto the module. Check if the Nordic UART example still does its job. If so, you have the original Nordic UART Example ported to the Proteus-I.

In case, you haven't loaded the Nordic Softdevice onto the chip, erase the full chip and load the Nordic Softdevice on it.

5. Then start with the modifications to realize the above specifications. First enable the DCDC to save current.

```
err_code = sd_power_dcdc_mode_set(NRF_POWER_DCDC_ENABLE);
APP_ERROR_CHECK(err_code);
```

6. This needs to be done in the `ble_stack_init()` function.
7. Set the `APP_ADV_TIMEOUT_IN_SECONDS` to 5s for example. This is the timeout after which the module goes to sleep mode, when no connection request was received during advertising.
8. Then implement a timer:

```
APP_TIMER_DEF(wakeup_timer_id);
err_code = app_timer_create(&wakeup_timer_id,
    APP_TIMER_MODE_SINGLE_SHOT, wakeup_timer_handler);
APP_ERROR_CHECK(err_code);
```

```
void wakeup_timer_handler(void * p_context)
{
    app_timer_stop(wakeup_timer_id);
}
```

9. In function `on_adv_evt()` change the content of the `BLE_ADV_EVT_IDLE` case. When advertising timeouts, let the LED indicate idle, close the UART and start the timer. Here we also choose 5s as sleep time. This will be the time after which the module will wake-up again.

```
case BLE_ADV_EVT_IDLE:
    err_code = bsp_indication_set(BSP_INDICATE_IDLE);
    APP_ERROR_CHECK(err_code);
    app_timer_start(wakeup_timer_id,
    APP_TIMER_TICKS(5000, APP_TIMER_PRESCALER), NULL);
    app_uart_close();
break;
```

In this case, all peripherals are stopped and the `power_manage()` function in the main loop puts the system to system on mode.

10. Then fill the `wakeup_timer_handler()` function. It has to re-enable the UART, re-initialize the LEDs and restart advertising upon wake-up.

```
void wakeup_timer_handler(void * p_context)
{
    app_timer_stop(wakeup_timer_id);
    uart_init();
    uint32_t err_code = bsp_init(BSP_INIT_LED | BSP_INIT_BUTTONS,
    APP_TIMER_TICKS(100, APP_TIMER_PRESCALER),
    bsp_event_handler);
    APP_ERROR_CHECK(err_code);
    err_code = ble_advertising_start(BLE_ADV_MODE_FAST);
    APP_ERROR_CHECK(err_code);
}
```

11. Re-compile and flash the new code onto the module.

12. Disconnect the flasher and resource the module, such that the chip runs in normal mode.

If you do not disconnect the flasher it is possible that the nrf52 stays in debug mode.

Also check the Jumper JP4 on the EV board that it is set.

13. Now, you can see that the module advertises for 5s and sleeps for 5s. During advertising, the *LED_3* of the Proteus-I-EV flashes periodically. When the module sleeps, this LED is off constantly.

3 Test results

When running the new code, the Proteus-I starts advertising after power-up. In this case the *LED_3* of the Proteus-I-EV is flashing periodically and the chip needs about 1.42mA. When no connection request was received until the advertising timeout (here 5s) was received, the module stops advertising and disables the UART. Since no events occur after switching of these peripherals, the core can go to sleep (system on mode). Until the core is woken up by the timer, the module consumes less than 2µA. When the timer re-enables the UART and starts the advertising again, the module is in normal mode with a current consumption as before (1.42mA). This will be repeated periodically.

	Operation mode with UART on and advertising	Sleep (System on) mode
Power consumption	1.42mA	< 2µA
Next step	Go to sleep mode, when no connection request was received for 5s	Wake-up from sleep after 5s using the RTC

3.1 Power consumption notes

Please note that the power consumption during advertising time can be decreased either. First of all, it depends on the advertising timing settings (how often an advertise packet is sent). Furthermore, switching off the UART yields in a significant saving of power. This can be a solution to realize a lowest power application with periodic wake-up.

4 Appendix

Boards.h Add the additional case for the Proteus-I (AMB2621) board:

```
    #elif defined(BOARD_AMB2621)
     #include "AMB2621.h"
```

And change the project settings, on tab c/c++, Section "Preprocessor Symbols -> Define" of the demo project to use "BOARD_AMB2621" instead of "BOARD_PCA10040".

AMB2621.h

```
#ifndef AMB2621_H
#define AMB2621_H

/* PINS of the nRF52:
 * The pins are named w.r.t their function in the AMB2621 standard firmware
 */

#define NRF_PIN_LED_1    0
#define NRF_PIN_LED_2    1
#define NRF_PIN_UARTTX   2
#define NRF_PIN_UARTRX   3
#define NRF_PIN_UARTRTS  4
#define NRF_PIN_BOOT     5
#define NRF_PIN_6    6
#define NRF_PIN_7    7
#define NRF_PIN_8    8
#define NRF_PIN_CUSTOM_9  9   /* corresponds to AMB2621_PIN_9 */
#define NRF_PIN_OPERATIONMODE 10  /* corresponds to AMB2621_PIN_8 */
#define NRF_PIN_11 11
#define NRF_PIN_12 12
#define NRF_PIN_13 13
#define NRF_PIN_14 14
#define NRF_PIN_15 15
#define NRF_PIN_16 16
#define NRF_PIN_17 17
#define NRF_PIN_18 18
#define NRF_PIN_19 19
#define NRF_PIN_20 20
#define NRF_PIN_RESET 21
#define NRF_PIN_22 22
#define NRF_PIN_23 23
#define NRF_PIN_24 24
#define NRF_PIN_25 25
#define NRF_PIN_26 26
#define NRF_PIN_27 27
#define NRF_PIN_UARTCTS 28
#define NRF_PIN_SLEEP    29
#define NRF_PIN_30 30
#define NRF_PIN_31 31

// LEDs definitions for AMB2621
#define LEDS_NUMBER      2
#define LEDS_LIST {NRF_PIN_LED_1, NRF_PIN_LED_2}
#define BSP_LED_0        NRF_PIN_LED_1
#define BSP_LED_1        NRF_PIN_LED_2
```

```
/* all LEDs are lit when GPIO is low */
#define LEDS_ACTIVE_STATE 1
#define LEDS_INV_MASK  LEDS_MASK

// Buttons definitions for AMB2621
#define BUTTONS_NUMBER 1
#define BUTTONS_LIST {NRF_PIN_SLEEP}
#define BSP_BUTTON_0   NRF_PIN_SLEEP
#define BUTTON_PULL    NRF_GPIO_PIN_PULLUP
#define BUTTONS_ACTIVE_STATE 0

// UART definitions for AMB2621
#define RX_PIN_NUMBER  NRF_PIN_UARTRX
#define TX_PIN_NUMBER  NRF_PIN_UARTTX
#define RTS_PIN_NUMBER NRF_PIN_UARTRTS
#define CTS_PIN_NUMBER NRF_PIN_UARTCTS

// Low frequency clock source to be used by the SoftDevice
#define NRF_CLOCK_LFCLKSRC      {\
        .source = NRF_CLOCK_LF_SRC_RC,\
        .rc_ctiv = 16,\
        .rc_temp_ctiv = 2,\
        }

#endif // AMB2621_H
```

main.c

```
/* Copyright (c) 2014 Nordic Semiconductor. All Rights Reserved.
 *
 * The information contained herein is property of Nordic Semiconductor ASA.
 * Terms and conditions of usage are described in detail in NORDIC
 * SEMICONDUCTOR STANDARD SOFTWARE LICENSE AGREEMENT.
 *
 * Licensees are granted free, non-transferable use of the information. NO
 * WARRANTY of ANY KIND is provided. This heading must NOT be removed from
 * the file.
 *
 */

#define AMBER_AN_VERSION 1.0

/** @file
 *
 * @defgroup ble_sdk_uart_over_ble_main main.c
 * @{
 * @ingroup  ble_sdk_app_nus_eval
 * @brief    UART over BLE application main file.
 *
 * This file contains the source code for a sample application that uses the Nordic UART service.
 * This application uses the @ref srvlib_conn_params module.
 */

#include <stdint.h>
#include <string.h>
#include "nordic_common.h"
#include "nrf.h"
#include "ble_hci.h"
#include "ble_advdata.h"
#include "ble_advertising.h"
#include "ble_conn_params.h"
#include "softdevice_handler.h"
#include "app_timer.h"
#include "app_button.h"
#include "ble_nus.h"
#include "app_uart.h"
#include "app_util_platform.h"
#include "bsp.h"
#include "bsp_btn_ble.h"

#define IS_SRVC_CHANGED_CHARACT_PRESENT 0                                           /**< Include the service_changed
    characteristic. If not enabled, the server's database cannot be changed for the lifetime of the device. */

#if (NRF_SD_BLE_API_VERSION == 3)
#define NRF_BLE_MAX_MTU_SIZE            GATT_MTU_SIZE_DEFAULT                       /**< MTU size used in the softdevice
    enabling and to reply to a BLE_GATTS_EVT_EXCHANGE_MTU_REQUEST event. */
#endif

#define APP_FEATURE_NOT_SUPPORTED       BLE_GATT_STATUS_ATTERR_APP_BEGIN + 2        /**< Reply when unsupported features are
    requested. */
```

```
#define CENTRAL_LINK_COUNT              0                                   /**< Number of central links used by the
    application. When changing this number remember to adjust the RAM settings */
#define PERIPHERAL_LINK_COUNT           1                                   /**< Number of peripheral links used by
    the application. When changing this number remember to adjust the RAM settings */

#define DEVICE_NAME                     "Nordic_UART"                       /**< Name of device. Will be included in
    the advertising data. */
#define NUS_SERVICE_UUID_TYPE           BLE_UUID_TYPE_VENDOR_BEGIN          /**< UUID type for the Nordic UART Service
     (vendor specific). */

#define APP_ADV_INTERVAL                64                                  /**< The advertising interval (in units of
     0.625 ms. This value corresponds to 40 ms). */
#define APP_ADV_TIMEOUT_IN_SECONDS      5                                   /**< The advertising timeout (in units of
    seconds). */

#define APP_TIMER_PRESCALER             0                                   /**< Value of the RTC1 PRESCALER register.
     */
#define APP_TIMER_OP_QUEUE_SIZE         4                                   /**< Size of timer operation queues. */

#define MIN_CONN_INTERVAL               MSEC_TO_UNITS(20, UNIT_1_25_MS)     /**< Minimum acceptable connection
    interval (20 ms), Connection interval uses 1.25 ms units. */
#define MAX_CONN_INTERVAL               MSEC_TO_UNITS(75, UNIT_1_25_MS)     /**< Maximum acceptable connection
    interval (75 ms), Connection interval uses 1.25 ms units. */
#define SLAVE_LATENCY                   0                                   /**< Slave latency. */
#define CONN_SUP_TIMEOUT                MSEC_TO_UNITS(4000, UNIT_10_MS)     /**< Connection supervisory timeout (4
    seconds), Supervision Timeout uses 10 ms units. */
#define FIRST_CONN_PARAMS_UPDATE_DELAY  APP_TIMER_TICKS(5000, APP_TIMER_PRESCALER)  /**< Time from initiating event (connect
    or start of notification) to first time sd_ble_gap_conn_param_update is called (5 seconds). */
#define NEXT_CONN_PARAMS_UPDATE_DELAY   APP_TIMER_TICKS(30000, APP_TIMER_PRESCALER) /**< Time between each call to
    sd_ble_gap_conn_param_update after the first call (30 seconds). */
#define MAX_CONN_PARAMS_UPDATE_COUNT    3                                   /**< Number of attempts before giving up
    the connection parameter negotiation. */

#define DEAD_BEEF                       0xDEADBEEF                          /**< Value used as error code on stack
    dump, can be used to identify stack location on stack unwind. */

#define UART_TX_BUF_SIZE                256                                 /**< UART TX buffer size. */
#define UART_RX_BUF_SIZE                256                                 /**< UART RX buffer size. */

static ble_nus_t                        m_nus;                              /**< Structure to identify the Nordic UART
     Service. */
static uint16_t                         m_conn_handle = BLE_CONN_HANDLE_INVALID;    /**< Handle of the current connection. */

static ble_uuid_t                       m_adv_uuids[] = {{BLE_UUID_NUS_SERVICE, NUS_SERVICE_UUID_TYPE}};  /**< Universally
    unique service identifier. */

APP_TIMER_DEF(wakeup_timer_id);

/**@brief Function for assert macro callback.
 *
 * @details This function will be called in case of an assert in the SoftDevice.
 *
 * @warning This handler is an example only and does not fit a final product. You need to analyse
 *          how your product is supposed to react in case of Assert.
 * @warning On assert from the SoftDevice, the system can only recover on reset.
 *
 * @param[in] line_num    Line number of the failing ASSERT call.
 * @param[in] p_file_name File name of the failing ASSERT call.
 */
void assert_nrf_callback(uint16_t line_num, const uint8_t * p_file_name)
{
    app_error_handler(DEAD_BEEF, line_num, p_file_name);
}


/**@brief Function for the GAP initialization.
 *
 * @details This function will set up all the necessary GAP (Generic Access Profile) parameters of
 *          the device. It also sets the permissions and appearance.
 */
static void gap_params_init(void)
{
    uint32_t                err_code;
    ble_gap_conn_params_t   gap_conn_params;
    ble_gap_conn_sec_mode_t sec_mode;

    BLE_GAP_CONN_SEC_MODE_SET_OPEN(&sec_mode);

    err_code = sd_ble_gap_device_name_set(&sec_mode,
                                          (const uint8_t *) DEVICE_NAME,
                                          strlen(DEVICE_NAME));
    APP_ERROR_CHECK(err_code);

    memset(&gap_conn_params, 0, sizeof(gap_conn_params));

    gap_conn_params.min_conn_interval = MIN_CONN_INTERVAL;
    gap_conn_params.max_conn_interval = MAX_CONN_INTERVAL;
    gap_conn_params.slave_latency     = SLAVE_LATENCY;
    gap_conn_params.conn_sup_timeout  = CONN_SUP_TIMEOUT;

    err_code = sd_ble_gap_ppcp_set(&gap_conn_params);
    APP_ERROR_CHECK(err_code);
}


/**@brief Function for handling the data from the Nordic UART Service.
 *
 * @details This function will process the data received from the Nordic UART BLE Service and send
 *          it to the UART module.
```

```
 *
 * @param[in] p_nus    Nordic UART Service structure.
 * @param[in] p_data   Data to be send to UART module.
 * @param[in] length   Length of the data.
 */
/**@snippet [Handling the data received over BLE] */
static void nus_data_handler(ble_nus_t * p_nus, uint8_t * p_data, uint16_t length)
{
    for (uint32_t i = 0; i < length; i++)
    {
        while (app_uart_put(p_data[i]) != NRF_SUCCESS);
    }
    while (app_uart_put('\r') != NRF_SUCCESS);
    while (app_uart_put('\n') != NRF_SUCCESS);
}
/**@snippet [Handling the data received over BLE] */


/**@brief Function for initializing services that will be used by the application.
 */
static void services_init(void)
{
    uint32_t       err_code;
    ble_nus_init_t nus_init;

    memset(&nus_init, 0, sizeof(nus_init));

    nus_init.data_handler = nus_data_handler;

    err_code = ble_nus_init(&m_nus, &nus_init);
    APP_ERROR_CHECK(err_code);
}


/**@brief Function for handling an event from the Connection Parameters Module.
 *
 * @details This function will be called for all events in the Connection Parameters Module
 *          which are passed to the application.
 *
 * @note All this function does is to disconnect. This could have been done by simply setting
 *       the disconnect_on_fail config parameter, but instead we use the event handler
 *       mechanism to demonstrate its use.
 *
 * @param[in] p_evt  Event received from the Connection Parameters Module.
 */
static void on_conn_params_evt(ble_conn_params_evt_t * p_evt)
{
    uint32_t err_code;

    if (p_evt->evt_type == BLE_CONN_PARAMS_EVT_FAILED)
    {
        err_code = sd_ble_gap_disconnect(m_conn_handle, BLE_HCI_CONN_INTERVAL_UNACCEPTABLE);
        APP_ERROR_CHECK(err_code);
    }
}


/**@brief Function for handling errors from the Connection Parameters module.
 *
 * @param[in] nrf_error  Error code containing information about what went wrong.
 */
static void conn_params_error_handler(uint32_t nrf_error)
{
    APP_ERROR_HANDLER(nrf_error);
}


/**@brief Function for initializing the Connection Parameters module.
 */
static void conn_params_init(void)
{
    uint32_t               err_code;
    ble_conn_params_init_t cp_init;

    memset(&cp_init, 0, sizeof(cp_init));

    cp_init.p_conn_params                  = NULL;
    cp_init.first_conn_params_update_delay = FIRST_CONN_PARAMS_UPDATE_DELAY;
    cp_init.next_conn_params_update_delay  = NEXT_CONN_PARAMS_UPDATE_DELAY;
    cp_init.max_conn_params_update_count   = MAX_CONN_PARAMS_UPDATE_COUNT;
    cp_init.start_on_notify_cccd_handle    = BLE_GATT_HANDLE_INVALID;
    cp_init.disconnect_on_fail             = false;
    cp_init.evt_handler                    = on_conn_params_evt;
    cp_init.error_handler                  = conn_params_error_handler;

    err_code = ble_conn_params_init(&cp_init);
    APP_ERROR_CHECK(err_code);
}


/**@brief Function for putting the chip into sleep mode.
 *
 * @note This function will not return.
 */
static void sleep_mode_enter(void)
{
    uint32_t err_code = bsp_indication_set(BSP_INDICATE_IDLE);
    APP_ERROR_CHECK(err_code);

    // Prepare wakeup buttons.
```

```
    err_code = bsp_btn_ble_sleep_mode_prepare();
    APP_ERROR_CHECK(err_code);

    // Go to system-off mode (this function will not return; wakeup will cause a reset).
    err_code = sd_power_system_off();
    APP_ERROR_CHECK(err_code);
}


/**@brief Function for handling advertising events.
 *
 * @details This function will be called for advertising events which are passed to the application.
 *
 * @param[in] ble_adv_evt  Advertising event.
 */
static void on_adv_evt(ble_adv_evt_t ble_adv_evt)
{
    uint32_t err_code;

    switch (ble_adv_evt)
    {
        case BLE_ADV_EVT_FAST:
            err_code = bsp_indication_set(BSP_INDICATE_ADVERTISING);
            APP_ERROR_CHECK(err_code);
            break;
        case BLE_ADV_EVT_IDLE:
            /* switch UART off and indicate IDLE, we now go to system on mode */
            err_code = bsp_indication_set(BSP_INDICATE_IDLE);
            APP_ERROR_CHECK(err_code);
            app_timer_start(wakeup_timer_id, APP_TIMER_TICKS(5000, APP_TIMER_PRESCALER), NULL);
            app_uart_close();
            break;
        default:
            break;
    }
}


/**@brief Function for the application's SoftDevice event handler.
 *
 * @param[in] p_ble_evt SoftDevice event.
 */
static void on_ble_evt(ble_evt_t * p_ble_evt)
{
    uint32_t                         err_code;

    switch (p_ble_evt->header.evt_id)
    {
        case BLE_GAP_EVT_CONNECTED:
            err_code = bsp_indication_set(BSP_INDICATE_CONNECTED);
            APP_ERROR_CHECK(err_code);
            m_conn_handle = p_ble_evt->evt.gap_evt.conn_handle;
            break; // BLE_GAP_EVT_CONNECTED

        case BLE_GAP_EVT_DISCONNECTED:
            err_code = bsp_indication_set(BSP_INDICATE_IDLE);
            APP_ERROR_CHECK(err_code);
            m_conn_handle = BLE_CONN_HANDLE_INVALID;
            break; // BLE_GAP_EVT_DISCONNECTED

        case BLE_GAP_EVT_SEC_PARAMS_REQUEST:
            // Pairing not supported
            err_code = sd_ble_gap_sec_params_reply(m_conn_handle, BLE_GAP_SEC_STATUS_PAIRING_NOT_SUPP, NULL, NULL);
            APP_ERROR_CHECK(err_code);
            break; // BLE_GAP_EVT_SEC_PARAMS_REQUEST

        case BLE_GATTS_EVT_SYS_ATTR_MISSING:
            // No system attributes have been stored.
            err_code = sd_ble_gatts_sys_attr_set(m_conn_handle, NULL, 0, 0);
            APP_ERROR_CHECK(err_code);
            break; // BLE_GATTS_EVT_SYS_ATTR_MISSING

        case BLE_GATTC_EVT_TIMEOUT:
            // Disconnect on GATT Client timeout event.
            err_code = sd_ble_gap_disconnect(p_ble_evt->evt.gattc_evt.conn_handle,
                                             BLE_HCI_REMOTE_USER_TERMINATED_CONNECTION);
            APP_ERROR_CHECK(err_code);
            break; // BLE_GATTC_EVT_TIMEOUT

        case BLE_GATTS_EVT_TIMEOUT:
            // Disconnect on GATT Server timeout event.
            err_code = sd_ble_gap_disconnect(p_ble_evt->evt.gatts_evt.conn_handle,
                                             BLE_HCI_REMOTE_USER_TERMINATED_CONNECTION);
            APP_ERROR_CHECK(err_code);
            break; // BLE_GATTS_EVT_TIMEOUT

        case BLE_EVT_USER_MEM_REQUEST:
            err_code = sd_ble_user_mem_reply(p_ble_evt->evt.gattc_evt.conn_handle, NULL);
            APP_ERROR_CHECK(err_code);
            break; // BLE_EVT_USER_MEM_REQUEST

        case BLE_GATTS_EVT_RW_AUTHORIZE_REQUEST:
        {
            ble_gatts_evt_rw_authorize_request_t  req;
            ble_gatts_rw_authorize_reply_params_t auth_reply;

            req = p_ble_evt->evt.gatts_evt.params.authorize_request;

            if (req.type != BLE_GATTS_AUTHORIZE_TYPE_INVALID)
            {
```

```
                if ((req.request.write.op == BLE_GATTS_OP_PREP_WRITE_REQ)     ||
                    (req.request.write.op == BLE_GATTS_OP_EXEC_WRITE_REQ_NOW) ||
                    (req.request.write.op == BLE_GATTS_OP_EXEC_WRITE_REQ_CANCEL))
                {
                    if (req.type == BLE_GATTS_AUTHORIZE_TYPE_WRITE)
                    {
                        auth_reply.type = BLE_GATTS_AUTHORIZE_TYPE_WRITE;
                    }
                    else
                    {
                        auth_reply.type = BLE_GATTS_AUTHORIZE_TYPE_READ;
                    }
                    auth_reply.params.write.gatt_status = APP_FEATURE_NOT_SUPPORTED;
                    err_code = sd_ble_gatts_rw_authorize_reply(p_ble_evt->evt.gatts_evt.conn_handle,
                                                               &auth_reply);
                    APP_ERROR_CHECK(err_code);
                }
            }
        } break; // BLE_GATTS_EVT_RW_AUTHORIZE_REQUEST

#if (NRF_SD_BLE_API_VERSION == 3)
        case BLE_GATTS_EVT_EXCHANGE_MTU_REQUEST:
            err_code = sd_ble_gatts_exchange_mtu_reply(p_ble_evt->evt.gatts_evt.conn_handle,
                                                       NRF_BLE_MAX_MTU_SIZE);
            APP_ERROR_CHECK(err_code);
            break; // BLE_GATTS_EVT_EXCHANGE_MTU_REQUEST
#endif

        default:
            // No implementation needed.
            break;
    }
}


/**@brief Function for dispatching a SoftDevice event to all modules with a SoftDevice
 *        event handler.
 *
 * @details This function is called from the SoftDevice event interrupt handler after a
 *          SoftDevice event has been received.
 *
 * @param[in] p_ble_evt  SoftDevice event.
 */
static void ble_evt_dispatch(ble_evt_t * p_ble_evt)
{
    ble_conn_params_on_ble_evt(p_ble_evt);
    ble_nus_on_ble_evt(&m_nus, p_ble_evt);
    on_ble_evt(p_ble_evt);
    ble_advertising_on_ble_evt(p_ble_evt);
    bsp_btn_ble_on_ble_evt(p_ble_evt);

}


/**@brief Function for the SoftDevice initialization.
 *
 * @details This function initializes the SoftDevice and the BLE event interrupt.
 */
static void ble_stack_init(void)
{
    uint32_t err_code;

    nrf_clock_lf_cfg_t clock_lf_cfg = NRF_CLOCK_LFCLKSRC;

    // Initialize SoftDevice.
    SOFTDEVICE_HANDLER_INIT(&clock_lf_cfg, NULL);

    ble_enable_params_t ble_enable_params;
    err_code = softdevice_enable_get_default_config(CENTRAL_LINK_COUNT,
                                                    PERIPHERAL_LINK_COUNT,
                                                    &ble_enable_params);
    APP_ERROR_CHECK(err_code);

    //Check the ram settings against the used number of links
    CHECK_RAM_START_ADDR(CENTRAL_LINK_COUNT,PERIPHERAL_LINK_COUNT);

    // Enable BLE stack.
#if (NRF_SD_BLE_API_VERSION == 3)
    ble_enable_params.gatt_enable_params.att_mtu = NRF_BLE_MAX_MTU_SIZE;
#endif
    err_code = softdevice_enable(&ble_enable_params);
    APP_ERROR_CHECK(err_code);

    // Subscribe for BLE events.
    err_code = softdevice_ble_evt_handler_set(ble_evt_dispatch);
    APP_ERROR_CHECK(err_code);

    /* enable DCDC to save current */
    err_code = sd_power_dcdc_mode_set(NRF_POWER_DCDC_ENABLE);
    APP_ERROR_CHECK(err_code);
}


/**@brief Function for handling events from the BSP module.
 *
 * @param[in]   event   Event generated by button press.
 */
void bsp_event_handler(bsp_event_t event)
{
    uint32_t err_code;
```

```
    switch (event)
    {
        case BSP_EVENT_SLEEP:
            sleep_mode_enter();
            break;

        case BSP_EVENT_DISCONNECT:
            err_code = sd_ble_gap_disconnect(m_conn_handle, BLE_HCI_REMOTE_USER_TERMINATED_CONNECTION);
            if (err_code != NRF_ERROR_INVALID_STATE)
            {
                APP_ERROR_CHECK(err_code);
            }
            break;

        case BSP_EVENT_WHITELIST_OFF:
            if (m_conn_handle == BLE_CONN_HANDLE_INVALID)
            {
                err_code = ble_advertising_restart_without_whitelist();
                if (err_code != NRF_ERROR_INVALID_STATE)
                {
                    APP_ERROR_CHECK(err_code);
                }
            }
            break;

        default:
            break;
    }
}


/**@brief   Function for handling app_uart events.
 *
 * @details This function will receive a single character from the app_uart module and append it to
 *          a string. The string will be be sent over BLE when the last character received was a
 *          'new line' i.e '\r\n' (hex 0x0D) or if the string has reached a length of
 *          @ref NUS_MAX_DATA_LENGTH.
 */
/**@snippet [Handling the data received over UART] */
void uart_event_handle(app_uart_evt_t * p_event)
{
    static uint8_t data_array[BLE_NUS_MAX_DATA_LEN];
    static uint8_t index = 0;
    uint32_t       err_code;

    switch (p_event->evt_type)
    {
        case APP_UART_DATA_READY:
            UNUSED_VARIABLE(app_uart_get(&data_array[index]));
            index++;

            if ((data_array[index - 1] == '\n') || (index >= (BLE_NUS_MAX_DATA_LEN)))
            {
                err_code = ble_nus_string_send(&m_nus, data_array, index);
                if (err_code != NRF_ERROR_INVALID_STATE)
                {
                    APP_ERROR_CHECK(err_code);
                }

                index = 0;
            }
            break;

        case APP_UART_COMMUNICATION_ERROR:
            APP_ERROR_HANDLER(p_event->data.error_communication);
            break;

        case APP_UART_FIFO_ERROR:
            APP_ERROR_HANDLER(p_event->data.error_code);
            break;

        default:
            break;
    }
}
/**@snippet [Handling the data received over UART] */


/**@brief  Function for initializing the UART module.
 */
/**@snippet [UART Initialization] */
static void uart_init(void)
{
    uint32_t                     err_code;
    const app_uart_comm_params_t comm_params =
    {
        RX_PIN_NUMBER,
        TX_PIN_NUMBER,
        0,
        0,
        APP_UART_FLOW_CONTROL_DISABLED,
        false,
        UART_BAUDRATE_BAUDRATE_Baud115200
    };

    APP_UART_FIFO_INIT( &comm_params,
                        UART_RX_BUF_SIZE,
                        UART_TX_BUF_SIZE,
                        uart_event_handle,
                        APP_IRQ_PRIORITY_LOW,
```

```
                        err_code);
    APP_ERROR_CHECK(err_code);
}
/**@snippet [UART Initialization] */


/**@brief Function for initializing the Advertising functionality.
 */
static void advertising_init(void)
{
    uint32_t               err_code;
    ble_advdata_t          advdata;
    ble_advdata_t          scanrsp;
    ble_adv_modes_config_t options;

    // Build advertising data struct to pass into @ref ble_advertising_init.
    memset(&advdata, 0, sizeof(advdata));
    advdata.name_type          = BLE_ADVDATA_FULL_NAME;
    advdata.include_appearance = false;
    advdata.flags              = BLE_GAP_ADV_FLAGS_LE_ONLY_LIMITED_DISC_MODE;

    memset(&scanrsp, 0, sizeof(scanrsp));
    scanrsp.uuids_complete.uuid_cnt = sizeof(m_adv_uuids) / sizeof(m_adv_uuids[0]);
    scanrsp.uuids_complete.p_uuids  = m_adv_uuids;

    memset(&options, 0, sizeof(options));
    options.ble_adv_fast_enabled  = true;
    options.ble_adv_fast_interval = APP_ADV_INTERVAL;
    options.ble_adv_fast_timeout  = APP_ADV_TIMEOUT_IN_SECONDS;

    err_code = ble_advertising_init(&advdata, &scanrsp, &options, on_adv_evt, NULL);
    APP_ERROR_CHECK(err_code);
}


/**@brief Function for initializing buttons and leds.
 *
 * @param[out] p_erase_bonds  Will be true if the clear bonding button was pressed to wake the application up.
 */
static void buttons_leds_init(bool * p_erase_bonds)
{
    bsp_event_t startup_event;

    uint32_t err_code = bsp_init(BSP_INIT_LED | BSP_INIT_BUTTONS,
                                 APP_TIMER_TICKS(100, APP_TIMER_PRESCALER),
                                 bsp_event_handler);
    APP_ERROR_CHECK(err_code);

    err_code = bsp_btn_ble_init(NULL, &startup_event);
    APP_ERROR_CHECK(err_code);

    *p_erase_bonds = (startup_event == BSP_EVENT_CLEAR_BONDING_DATA);
}


/**@brief Function for placing the application in low power state while waiting for events.
 */
static void power_manage(void)
{
    uint32_t err_code = sd_app_evt_wait();
    APP_ERROR_CHECK(err_code);
}

/**@brief Undo the changes that we did when advertising has the timeout.
 */
void wakeup_timer_handler(void * p_context)
{
    app_timer_stop(wakeup_timer_id);
    uart_init();
    uint32_t err_code = bsp_init(BSP_INIT_LED | BSP_INIT_BUTTONS,
                                 APP_TIMER_TICKS(100, APP_TIMER_PRESCALER),
                                 bsp_event_handler);
    APP_ERROR_CHECK(err_code);
    err_code = ble_advertising_start(BLE_ADV_MODE_FAST);
            APP_ERROR_CHECK(err_code);
}

/**@brief Application main function.
 */
int main(void)
{
    uint32_t err_code;
    bool erase_bonds;

    // Initialize.
    APP_TIMER_INIT(APP_TIMER_PRESCALER, APP_TIMER_OP_QUEUE_SIZE, false);
    uart_init();

    err_code = app_timer_create(&wakeup_timer_id, APP_TIMER_MODE_SINGLE_SHOT, wakeup_timer_handler);
    APP_ERROR_CHECK(err_code);

    buttons_leds_init(&erase_bonds);
    ble_stack_init();
    gap_params_init();
    services_init();
    advertising_init();
    conn_params_init();

    //printf("\r\nUART Start!\r\n");
    err_code = ble_advertising_start(BLE_ADV_MODE_FAST);
```

```
    APP_ERROR_CHECK(err_code);

    // Enter main loop.
    for (;;)
    {
        power_manage();
    }
}


/**
 * @}
 */
```

5 Important notes

The following conditions apply to all goods within the wireless connectivity product range of Würth Elektronik eiSos GmbH & Co. KG:

5.1 General customer responsibility

Some goods within the product range of Würth Elektronik eiSos GmbH & Co. KG contain statements regarding general suitability for certain application areas. These statements about suitability are based on our knowledge and experience of typical requirements concerning the areas, serve as general guidance and cannot be estimated as binding statements about the suitability for a customer application. The responsibility for the applicability and use 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, where appropriate to investigate and to decide whether the device with the specific product characteristics described in the product specification is valid and suitable for the respective customer application or not. Accordingly, the customer is cautioned to verify that the documentation is current before placing orders.

5.2 Customer responsibility related to specific, in particular safety-relevant applications

It has to be clearly pointed out that the possibility of a malfunction of electronic components or failure before the end of the usual lifetime cannot be completely eliminated in the current state of the art, even if the products are operated within the range of the specifications. The same statement is valid for all software sourcecode and firmware parts contained in or used with or for products in the wireless connectivity and sensor product range of Würth Elektronik eiSos GmbH & Co. KG. In certain customer applications requiring a high level of safety and especially in customer applications in which the malfunction or failure of an electronic component could endanger human life or health, it must be ensured by most advanced technological aid of suitable design of the customer application that no injury or damage is caused to third parties in the event of malfunction or failure of an electronic component.

5.3 Best care and attention

Any product-specific data sheets, manuals, application notes, PCN's, warnings and cautions must be strictly observed in the most recent versions and matching to the products firmware revisions. This documents can be downloaded from the product specific sections on the wireless connectivity homepage.

5.4 Customer support for product specifications

Some products within the product range may contain substances, which are subject to restrictions in certain jurisdictions in order to serve specific technical requirements. Necessary information is available on request. In this case, the field sales engineer or the internal sales person in charge should be contacted who will be happy to support in this matter.

5.5 Product improvements

Due to constant product improvement, product specifications may change from time to time. As a standard reporting procedure of the Product Change Notification (PCN) according to the JEDEC-Standard, we inform about major changes. In case of further queries regarding the PCN, the field sales engineer, the internal sales person or the technical support team in charge should be contacted. The basic responsibility of the customer as per section 5.1 and 5.2 remains unaffected. All wireless connectivity module driver software ¨wireless connectivity SDK¨ and it's source codes as well as all PC software tools are not subject to the Product Change Notification information process.

5.6 Product life cycle

Due to technical progress and economical evaluation we also reserve the right to discontinue production and delivery of products. As a standard reporting procedure of the Product Termination Notification (PTN) according to the JEDEC-Standard we will inform at an early stage about inevitable product discontinuance. According to this, we cannot ensure that all products within our product range will always be available. Therefore, it needs to be verified with the field sales engineer or the internal sales person in charge about the current product availability expectancy before or when the product for application design-in disposal is considered. The approach named above does not apply in the case of individual agreements deviating from the foregoing for customer-specific products.

5.7 Property rights

All the rights for contractual products produced by Würth Elektronik eiSos GmbH & Co. KG on the basis of ideas, development contracts as well as models or templates that are subject to copyright, patent or commercial protection supplied to the customer will remain with Würth Elektronik eiSos GmbH & Co. KG. Würth Elektronik eiSos GmbH & Co. KG does not warrant or represent that any license, either expressed or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right relating to any combination, application, or process in which Würth Elektronik eiSos GmbH & Co. KG components or services are used.

5.8 General terms and conditions

Unless otherwise agreed in individual contracts, all orders are subject to the current version of the "General Terms and Conditions of Würth Elektronik eiSos Group", last version available at *www.we-online.com*.

6 Legal notice

6.1 Exclusion of liability

Würth Elektronik eiSos GmbH & Co. KG considers the information in this document to be correct at the time of publication. However, Würth Elektronik eiSos GmbH & Co. KG reserves the right to modify the information such as technical specifications or functions of its products or discontinue the production of these products or the support of one of these products without any written announcement or notification to customers. The customer must make sure that the information used corresponds to the latest published information. Würth Elektronik eiSos GmbH & Co. KG does not assume any liability for the use of its products. Würth Elektronik eiSos GmbH & Co. KG does not grant licenses for its patent rights or for any other of its intellectual property rights or third-party rights.

Notwithstanding anything above, Würth Elektronik eiSos GmbH & Co. KG makes no representations and/or warranties of any kind for the provided information related to their accuracy, correctness, completeness, usage of the products and/or usability for customer applications. Information published by Würth Elektronik eiSos GmbH & Co. KG regarding third-party products or services does not constitute a license to use such products or services or a warranty or endorsement thereof.

6.2 Suitability in customer applications

The customer bears the responsibility for compliance of systems or units, in which Würth Elektronik eiSos GmbH & Co. KG products are integrated, with applicable legal regulations. Customer acknowledges and agrees that it is solely responsible for compliance with all legal, regulatory and safety-related requirements concerning its products, and any use of Würth Elektronik eiSos GmbH & Co. KG components in its applications, notwithstanding any applications-related in-formation or support that may be provided by Würth Elektronik eiSos GmbH & Co. KG. Customer represents and agrees that it has all the necessary expertise to create and implement safeguards which anticipate dangerous consequences of failures, monitor failures and their consequences lessen the likelihood of failures that might cause harm and take appropriate remedial actions. The customer will fully indemnify Würth Elektronik eiSos GmbH & Co. KGand its representatives against any damages arising out of the use of any Würth Elektronik eiSos GmbH & Co. KG components in safety-critical applications.

6.3 Trademarks

AMBER wireless is a registered trademark of Würth Elektronik eiSos GmbH & Co. KG. All other trademarks, registered trademarks, and product names are the exclusive property of the respective owners.

6.4 Usage restriction

Würth Elektronik eiSos GmbH & Co. KG products have been designed and developed for usage in general electronic equipment only. This product is not authorized for use in equipment where a higher safety standard and reliability standard is especially required or where

a failure of the product is reasonably expected to cause severe personal injury or death, unless the parties have executed an agreement specifically governing such use. Moreover, Würth Elektronik eiSos GmbH & Co. KG products are neither designed nor intended for use in areas such as military, aerospace, aviation, nuclear control, submarine, transportation (automotive control, train control, ship control), transportation signal, disaster prevention, medical, public information network etc. Würth Elektronik eiSos GmbH & Co. KG 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 electronic component, which is used in electrical circuits that require high safety and reliability function or performance. By using Würth Elektronik eiSos GmbH & Co. KG products, the customer agrees to these terms and conditions.

7 License terms

This License Terms will take effect upon the purchase and usage of the Würth Elektronik eiSos GmbH & Co. KG wireless connectivity products. You hereby agree that this license terms is applicable to the product and the incorporated software, firmware and source codes (collectively, "Software") made available by Würth Elektronik eiSos in any form, including but not limited to binary, executable or source code form.
The software included in any Würth Elektronik eiSos wireless connectivity product is purchased to you on the condition that you accept the terms and conditions of this license terms. You agree to comply with all provisions under this license terms.

7.1 Limited license

Würth Elektronik eiSos hereby grants you a limited, non-exclusive, non-transferable and royalty-free license to use the software and under the conditions that will be set forth in this license terms. You are free to use the provided Software only in connection with one of the products from Würth Elektronik eiSos to the extent described in this license terms. You are entitled to change or alter the source code for the sole purpose of creating an application embedding the Würth Elektronik eiSos wireless connectivity product. The transfer of the source code to third parties is allowed to the sole extent that the source code is used by such third parties in connection with our product or another hardware provided by Würth Elektronik eiSos under strict adherence of this license terms. Würth Elektronik eiSos will not assume any liability for the usage of the incorporated software and the source code. You are not entitled to transfer the source code in any form to third parties without prior written consent of Würth Elektronik eiSos.
You are not allowed to reproduce, translate, reverse engineer, decompile, disassemble or create derivative works of the incorporated Software and the source code in whole or in part. No more extensive rights to use and exploit the products are granted to you.

7.2 Usage and obligations

The responsibility for the applicability and use of the Würth Elektronik eiSos wireless connectivity product with the incorporated Firmware in a particular customer design is always solely within the authority of the customer. Due to this fact, it is up to you to evaluate and investigate, where appropriate, and to decide whether the device with the specific product characteristics described in the product specification is valid and suitable for your respective application or not.
You are responsible for using the Würth Elektronik eiSos wireless connectivity product with the incorporated Firmware in compliance with all applicable product liability and product safety laws. You acknowledge to minimize the risk of loss and harm to individuals and bear the risk for failure leading to personal injury or death due to your usage of the product.
Würth Elektronik eiSos' products with the incorporated Firmware are not authorized for use in safety-critical applications, or where a failure of the product is reasonably expected to cause severe personal injury or death. Moreover, Würth Elektronik eiSos' products with the incorporated Firmware are neither designed nor intended for use in areas such as military, aerospace, aviation, nuclear control, submarine, transportation (automotive control, train control, ship control), transportation signal, disaster prevention, medical, public information network etc. You shall inform Würth Elektronik eiSos about the intent of such usage before

design-in stage. In certain customer applications requiring a very high level of safety and in which the malfunction or failure of an electronic component could endanger human life or health, you must ensure to have all necessary expertise in the safety and regulatory ramifications of your applications. You acknowledge and agree that you are solely responsible for all legal, regulatory and safety-related requirements concerning your products and any use of Würth Elektronik eiSos' products with the incorporated Firmware in such safety-critical applications, notwithstanding any applications-related information or support that may be provided by Würth Elektronik eiSos. YOU SHALL INDEMNIFY WÜRTH ELEKTRONIK EISOS AGAINST ANY DAMAGES ARISING OUT OF THE USE OF WÜRTH ELEKTRONIK EISOS' PRODUCTS WITH THE INCORPORATED FIRMWARE IN SUCH SAFETY-CRITICAL APPLICATIONS.

7.3 Ownership

The incorporated Firmware created by Würth Elektronik eiSos is and will remain the exclusive property of Würth Elektronik eiSos.

7.4 Firmware update(s)

You have the opportunity to request the current and actual Firmware for a bought wireless connectivity Product within the time of warranty. However, Würth Elektronik eiSos has no obligation to update a modules firmware in their production facilities, but can offer this as a service on request. The upload of firmware updates falls within your responsibility, e.g. via ACC or another software for firmware updates. Firmware updates will not be communicated automatically. It is within your responsibility to check the current version of a firmware in the latest version of the product manual on our website. The revision table in the product manual provides all necessary information about firmware updates. There is no right to be provided with binary files, so called "Firmware images", those could be flashed through JTAG, SWD, Spi-Bi-Wire, SPI or similar interfaces.

7.5 Disclaimer of warranty

THE FIRMWARE IS PROVIDED "AS IS". YOU ACKNOWLEDGE THAT WÜRTH ELEKTRONIK 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 YOUR INTENDED PURPOSE OR USAGE. WÜRTH ELEKTRONIK 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 THE WÜRTH ELEKTRONIK EISOS' PRODUCT WITH THE INCORPORATED FIRMWARE IS USED. INFORMATION PUBLISHED BY WÜRTH ELEKTRONIK EISOS REGARDING THIRD-PARTY PRODUCTS OR SERVICES DOES NOT CONSTITUTE A LICENSE FROM WÜRTH ELEKTRONIK EISOS TO USE SUCH PRODUCTS OR SERVICES OR A WARRANTY OR ENDORSEMENT THEREOF.

7.6 Limitation of liability

Any liability not expressly provided by Würth Elektronik eiSos shall be disclaimed.
You agree to hold us harmless from any third-party claims related to your usage of the Würth Elektronik eiSos' products with the incorporated Firmware, software and source code. Würth Elektronik eiSos disclaims any liability for any alteration, development created by you or your customers as well as for any combination with other products.

7.7 Applicable law and jurisdiction

Applicable law to this license terms shall be the laws of the Federal Republic of Germany. Any dispute, claim or controversy arising out of or relating to this license terms shall be resolved and finally settled by the court competent for the location of Würth Elektronik eiSos' registered office.

7.8 Severability clause

If a provision of this license terms is or becomes invalid, unenforceable or null and void, this shall not affect the remaining provisions of the terms. The parties shall replace any such provisions with new valid provisions that most closely approximate the purpose of the terms.

7.9 Miscellaneous

Würth Elektronik eiSos reserves the right at any time to change this terms at its own discretion. It is your responsibility to check at Würth Elektronik eiSos homepage for any updates. Your continued usage of the products will be deemed as the acceptance of the change.
We recommend you to be updated about the status of new firmware and software, which is available on our website or in our data sheet and manual, and to implement new software in your device where appropriate.
By ordering a wireless connectivity product, you accept this license terms in all terms.

List of Figures

List of Tables

WÜRTH ELEKTRONIK

more than you expect

Internet of Things

Monitoring & Control

Automated Meter Reading

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

Rudi-Schillings-Str. 31
54296 Trier
Germany

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