CEC Continuing Education Center

DesignNews

Developing IoT Applications with Nordic nRF Modules

Day 3:
Basic BLE Data Exchange

Sponsored by

DigiKey

informa markets

Webinar Logistics

- Turn on your system sound to hear the streaming presentation.
- If you have technical problems, click “Help” or submit a question asking for assistance.
- Participate in ‘Attendee Chat’ by maximizing the chat widget in your dock.

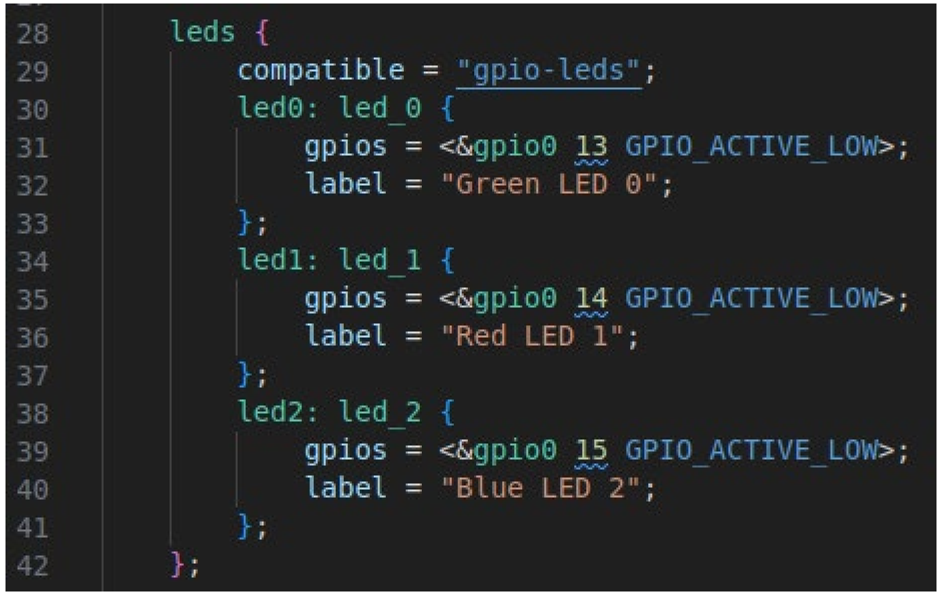
Fred Eady

Visit ‘Lecturer Profile’ in your console for more details.

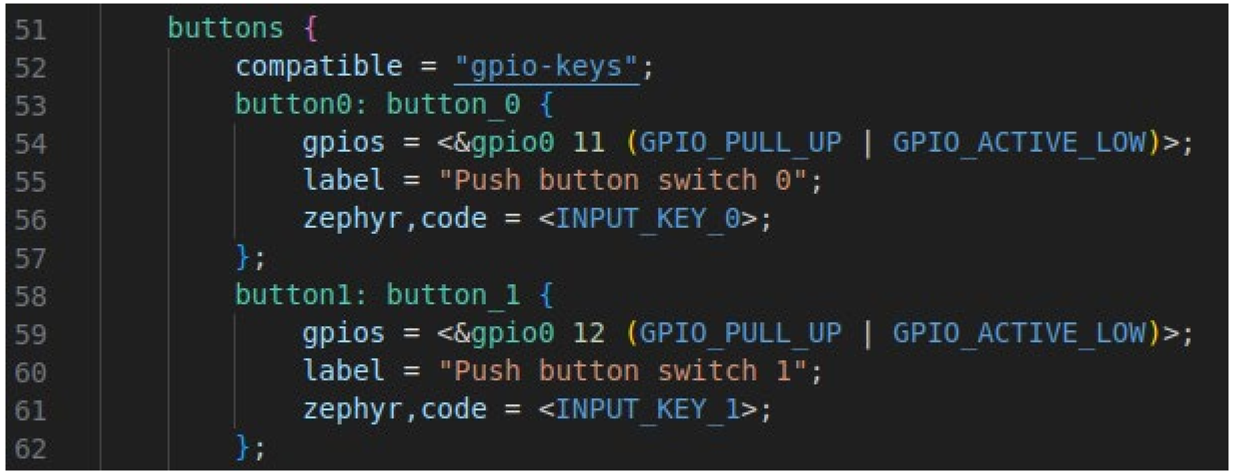
AGENDA

- **Nordic BLE Building Blocks**

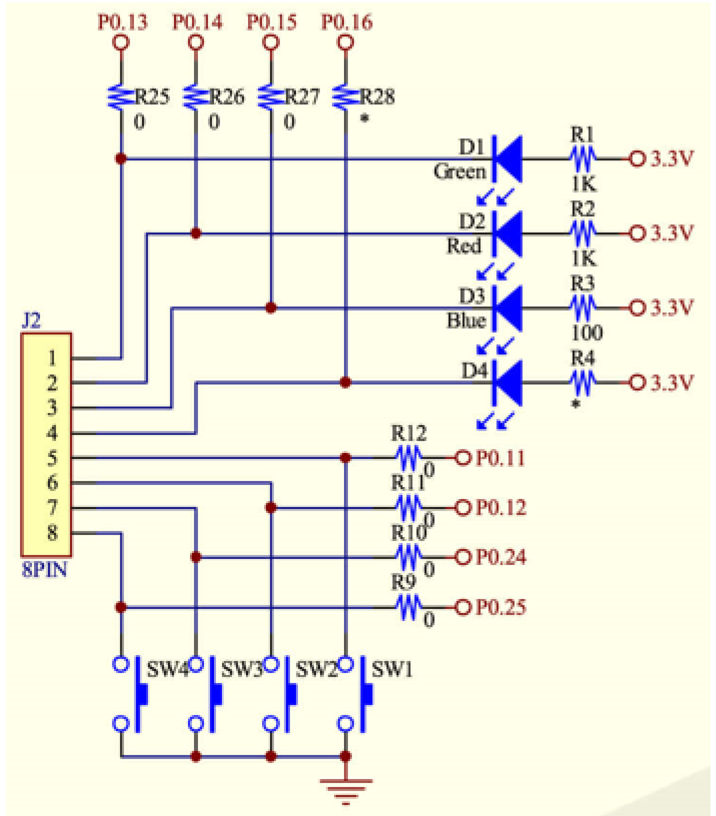
Populate the Devicetree



```
        leds {
            compatible = "gpio-leds";
            led0: led_0 {
                gpios = <&gpio0 13 GPIO_ACTIVE_LOW>;
                label = "Green LED 0";
            };
            led1: led_1 {
                gpios = <&gpio0 14 GPIO_ACTIVE_LOW>;
                label = "Red LED 1";
            };
            led2: led_2 {
                gpios = <&gpio0 15 GPIO_ACTIVE_LOW>;
                label = "Blue LED 2";
            };
        };
```



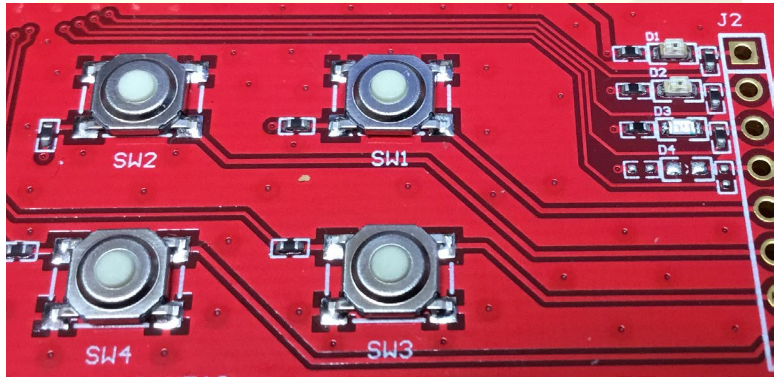
```
        buttons {
            compatible = "gpio-keys";
            button0: button_0 {
                gpios = <&gpio0 11 (GPIO_PULL_UP | GPIO_ACTIVE_LOW)>;
                label = "Push button switch 0";
                zephyr,code = <INPUT_KEY_0>;
            };
            button1: button_1 {
                gpios = <&gpio0 12 (GPIO_PULL_UP | GPIO_ACTIVE_LOW)>;
                label = "Push button switch 1";
                zephyr,code = <INPUT_KEY_1>;
            };
```



led0

led1

led2

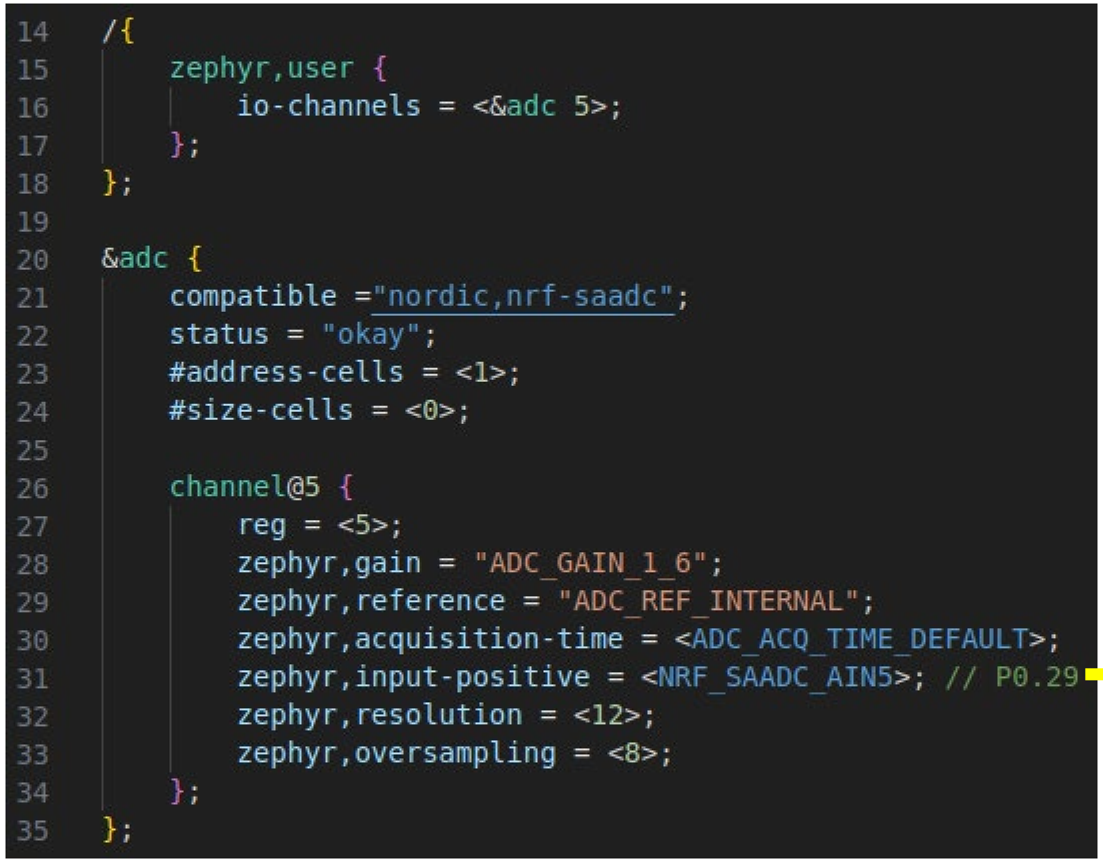


Developing IoT Applications with Nordic nRF Modules

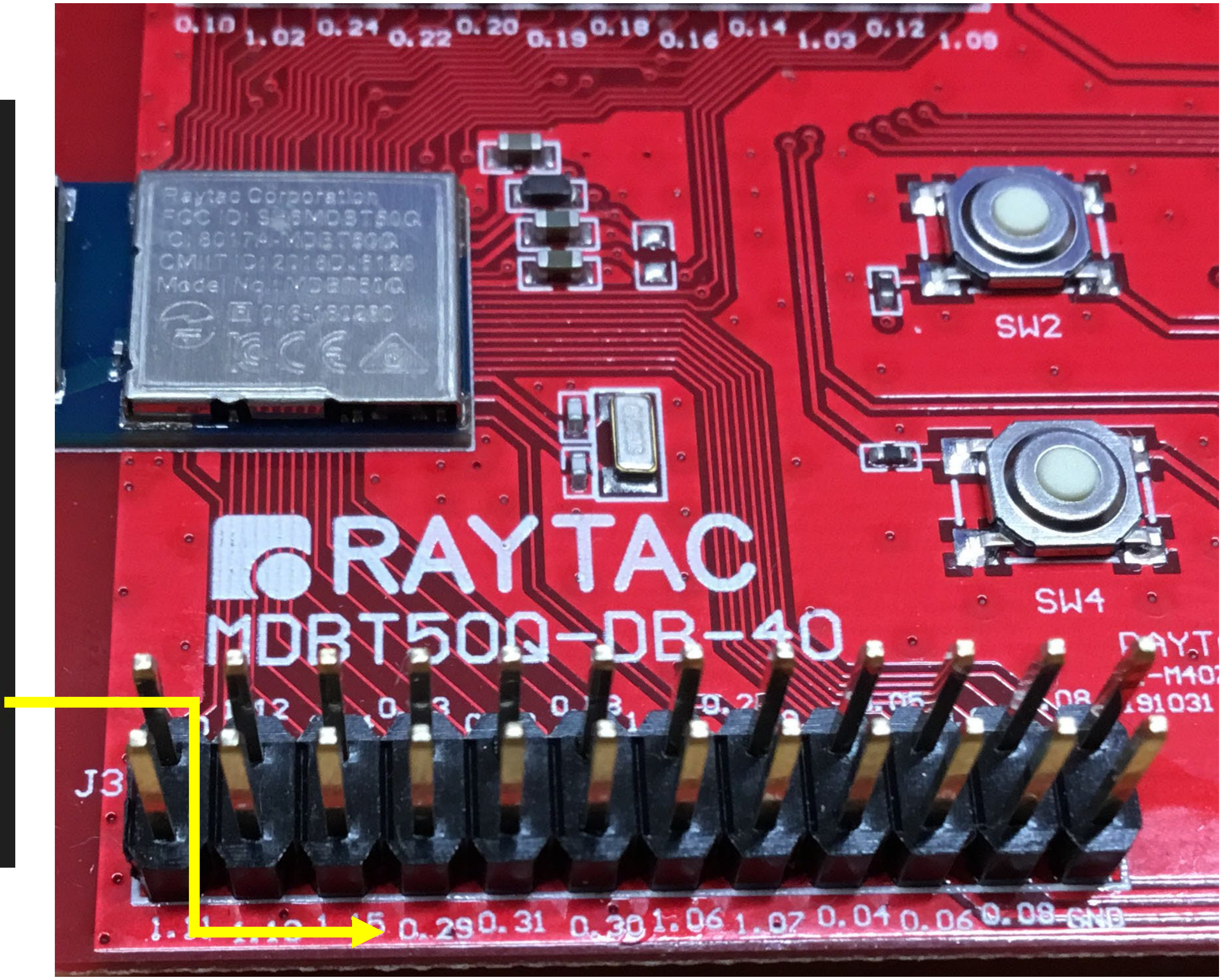
Basic BLE Data Exchange

Nordic BLE Building Blocks – raytac_mdbt50q_db_40_nrf52840.overlay

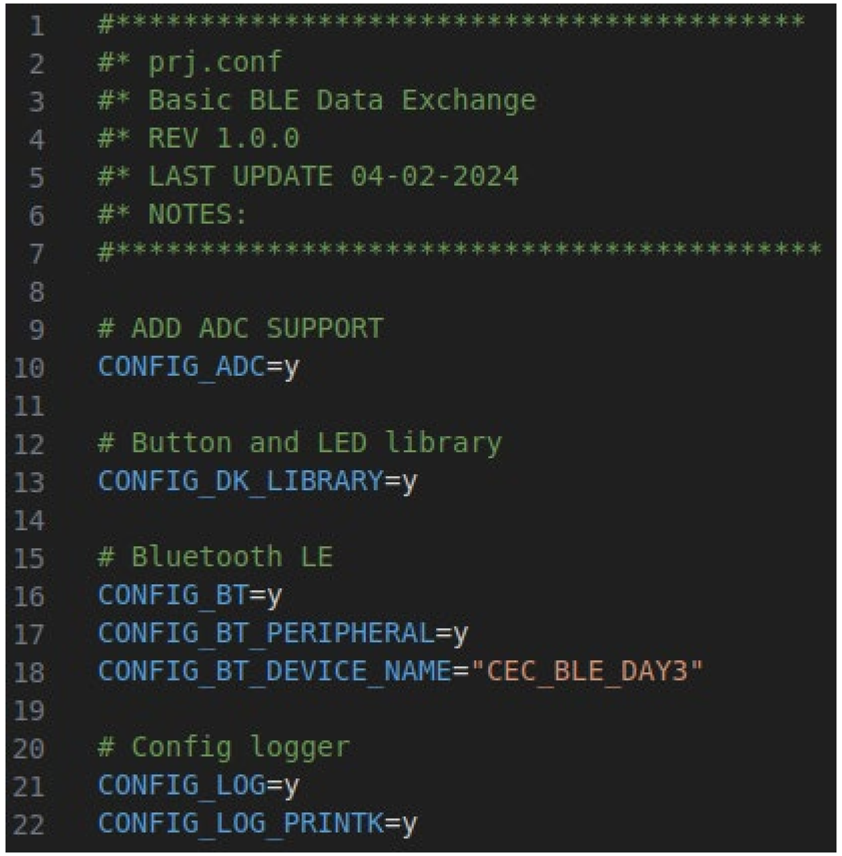
Create a Devicetree Overlay – Analog-to-Digital Pin



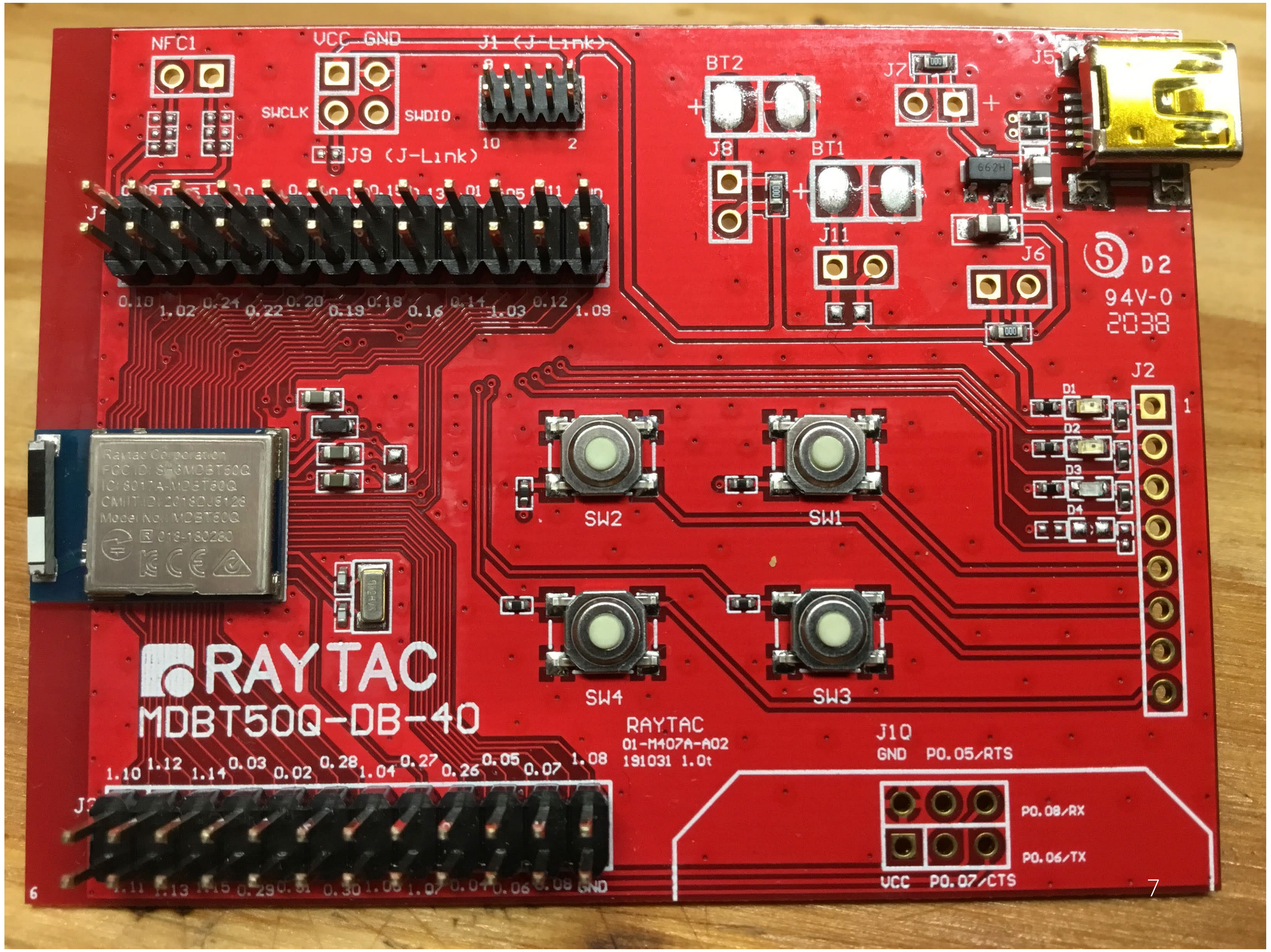
```
/{
    zephyr,user {
        io-channels = <&adc 5>;
    };
};

&adc {
    compatible ="nordic,nrf-saadc";
    status = "okay";
    #address-cells = <1>;
    #size-cells = <0>;

    channel@5 {
        reg = <5>;
        zephyr,gain = "ADC_GAIN_1_6";
        zephyr,reference = "ADC_REF_INTERNAL";
        zephyr,acquisition-time = <ADC_ACQ_TIME_DEFAULT>;
        zephyr,input-positive = <NRF_SAADC_AIN5>; // P0.29
        zephyr,resolution = <12>;
        zephyr,oversampling = <8>;
    };
};
```



Define and Populate the prj.conf File



```
#*****************************************
#* prj.conf
#* Basic BLE Data Exchange
#* REV 1.0.0
#* LAST UPDATE 04-02-2024
#* NOTES:
#******************************************

# ADD ADC SUPPORT
CONFIG_ADC=y

# Button and LED library
CONFIG_DK_LIBRARY=y

# Bluetooth LE
CONFIG_BT=y
CONFIG_BT_PERIPHERAL=y
CONFIG_BT_DEVICE_NAME="CEC_BLE_DAY3"

# Config logger
CONFIG_LOG=y
CONFIG_LOG_PRINTK=y
```



Continuing Education Center

Developing IoT Applications with Nordic nRF Modules

Basic BLE Data Exchange

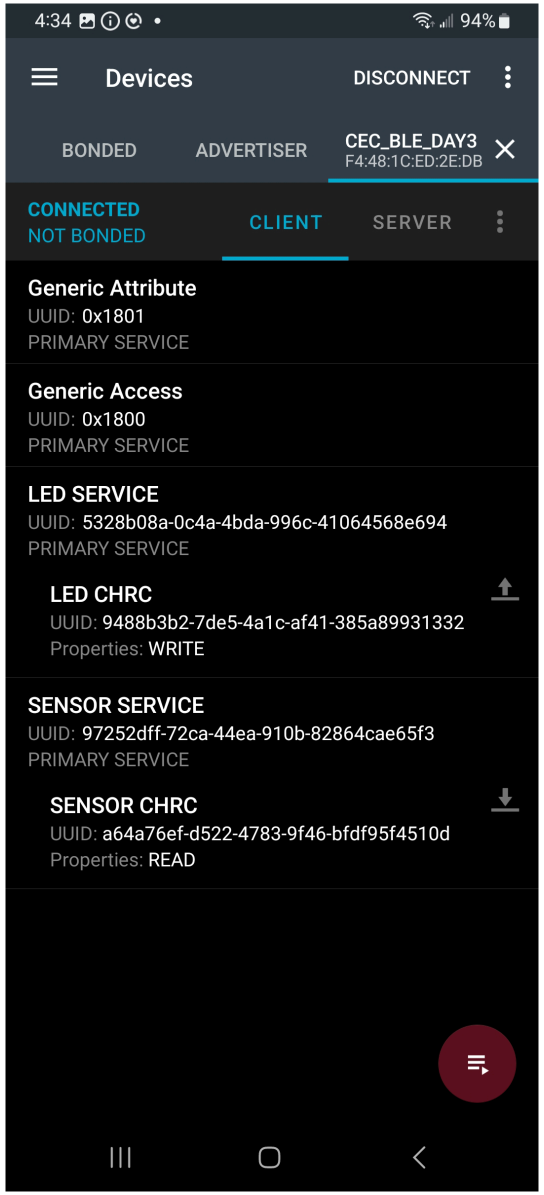
Nordic BLE Building Blocks – cecsvc.h

Define the Service and Characteristic UUIDs - Sensor

DigiKey

```
// Define the 128 bit UUIDs for the GATT service and its characteristics
// @brief SENSOR Service UUID.
#define BT_UUID_SENSORSVC_VAL     \
    BT_UUID_128_ENCODE(0x97252dff,0x72ca,0x44ea,0x910b,0xa82864cae65f3)

// @brief SENSOR Characteristic.
#define BT_UUID_SENSORCHR_VAL     \
    BT_UUID_128_ENCODE(0xa64a76ef,0xd522,0x4783,0x9f46,0xbfdf95f4510d)

#define BT_UUID_SENSORSVC    BT_UUID_DECLARE_128(BT_UUID_SENSORSVC_VAL)
#define BT_UUID_SENSORCHR    BT_UUID_DECLARE_128(BT_UUID_SENSORCHR_VAL)

// @brief Callback type for when SENSOR value change is received.
typedef uint16_t (*sensorchrc_cb_t)(void);

// @brief Callback struct used by the SENSOR Service.
struct sensorsvc_cb {
    // sensorchrc callback.
    sensorchrc_cb_t sensorchrc_cb;
};
```

```
// @brief Initialize the SENSOR Service.
//
// This function registers application callback functions with the SENSOR
// Service
//
// @param[in] sensor_callback Struct containing pointers to callback functions
//            used by the service. This pointer can be NULL
//            if no callback functions are defined.
int sensorsvc_init(struct sensorsvc_cb *sensor_callback);
```



Generic Attribute
UUID: 0x1801
PRIMARY SERVICE

Generic Access
UUID: 0x1800
PRIMARY SERVICE

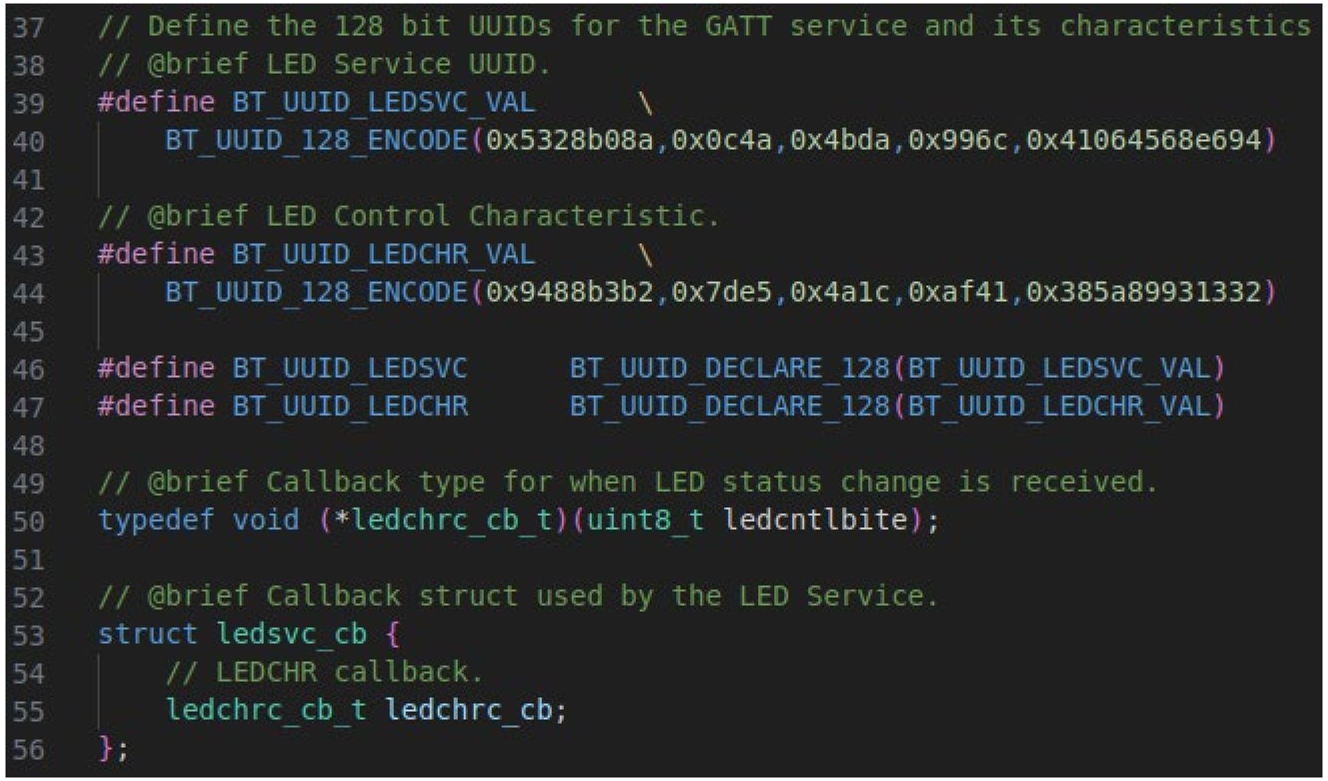
LED SERVICE
UUID: 5328b08a-0c4a-4bda-996c-41064568e694
PRIMARY SERVICE

LED CHRC
UUID: 9488b3b2-7de5-4a1c-af41-385a89931332
Properties: WRITE

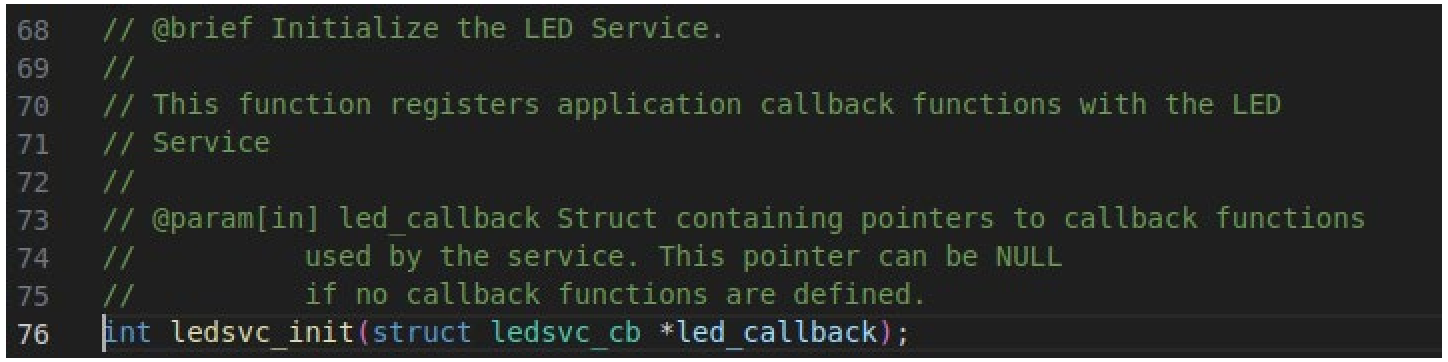
SENSOR SERVICE
UUID: 97252dff-72ca-44ea-910b-82864cae65f3
PRIMARY SERVICE

SENSOR CHRC
UUID: a64a76ef-d522-4783-9f46-bfdf95f4510d
Properties: READ

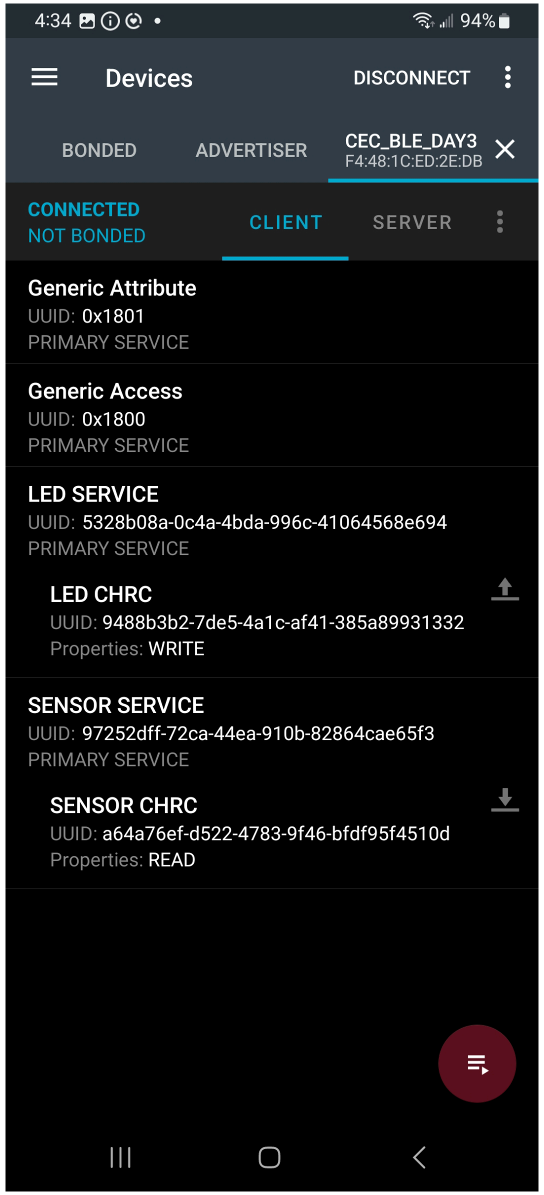
Define the Service and Characteristic UUIDs - LED



```
// Define the 128 bit UUIDs for the GATT service and its characteristics
// @brief LED Service UUID.
#define BT_UUID_LEDSVC_VAL      \
    BT_UUID_128_ENCODE(0x5328b08a,0x0c4a,0x4bda,0x996c,0x41064568e694)

// @brief LED Control Characteristic.
#define BT_UUID_LEDCHR_VAL      \
    BT_UUID_128_ENCODE(0x9488b3b2,0x7de5,0x4a1c,0xaf41,0x385a89931332)

#define BT_UUID_LEDSVC      BT_UUID_DECLARE_128(BT_UUID_LEDSVC_VAL)
#define BT_UUID_LEDCHR      BT_UUID_DECLARE_128(BT_UUID_LEDCHR_VAL)

// @brief Callback type for when LED status change is received.
typedef void (*ledchrc_cb_t)(uint8_t ledcntlbite);

// @brief Callback struct used by the LED Service.
struct ledsvc_cb {
    // LEDCHR callback.
    ledchrc_cb_t ledchrc_cb;
};
```



```
// @brief Initialize the LED Service.
//
// This function registers application callback functions with the LED
// Service
//
// @param[in] led_callback Struct containing pointers to callback functions
//            used by the service. This pointer can be NULL
//            if no callback functions are defined.
int ledsvc_init(struct ledsvc_cb *led_callback);
```



4:34 94%

Devices DISCONNECT

BONDED ADVERTISER CEC_BLE_DAY3 F4:48:1C:ED:2E:DB

CONNECTED
NOT BONDED
CLIENT SERVER

Generic Attribute
UUID: 0x1801
PRIMARY SERVICE

Generic Access
UUID: 0x1800
PRIMARY SERVICE

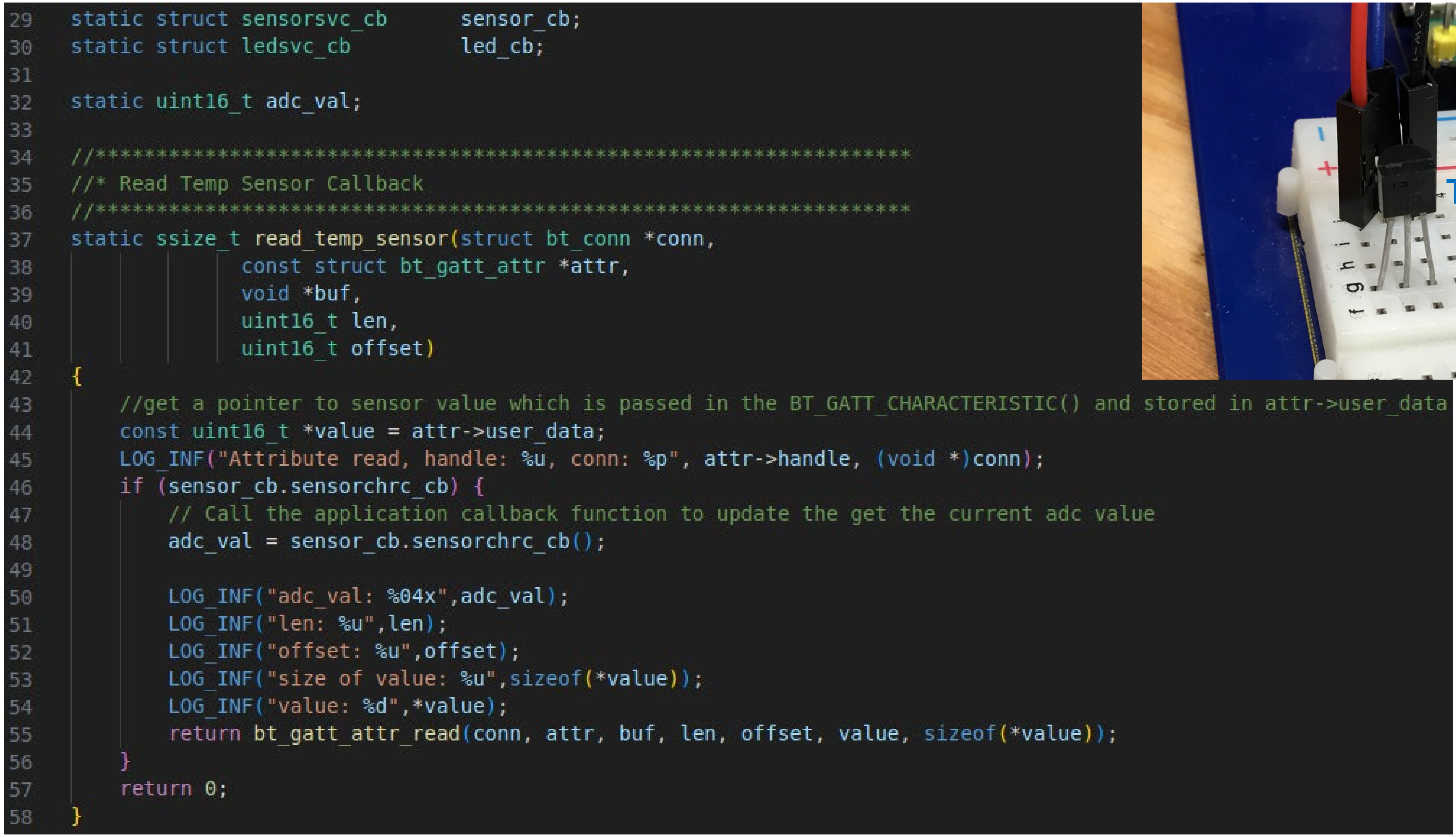
LED SERVICE
UUID: 5328b08a-0c4a-4bda-996c-41064568e694
PRIMARY SERVICE

LED CHRC
UUID: 9488b3b2-7de5-4a1c-af41-385a89931332
Properties: WRITE

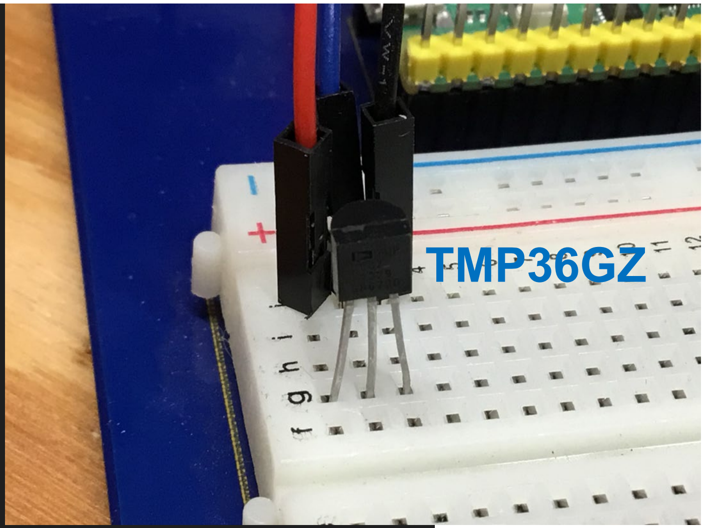
SENSOR SERVICE
UUID: 97252dff-72ca-44ea-910b-82864cae65f3
PRIMARY SERVICE

SENSOR CHRC
UUID: a64a76ef-d522-4783-9f46-bfdf95f4510d
Properties: READ

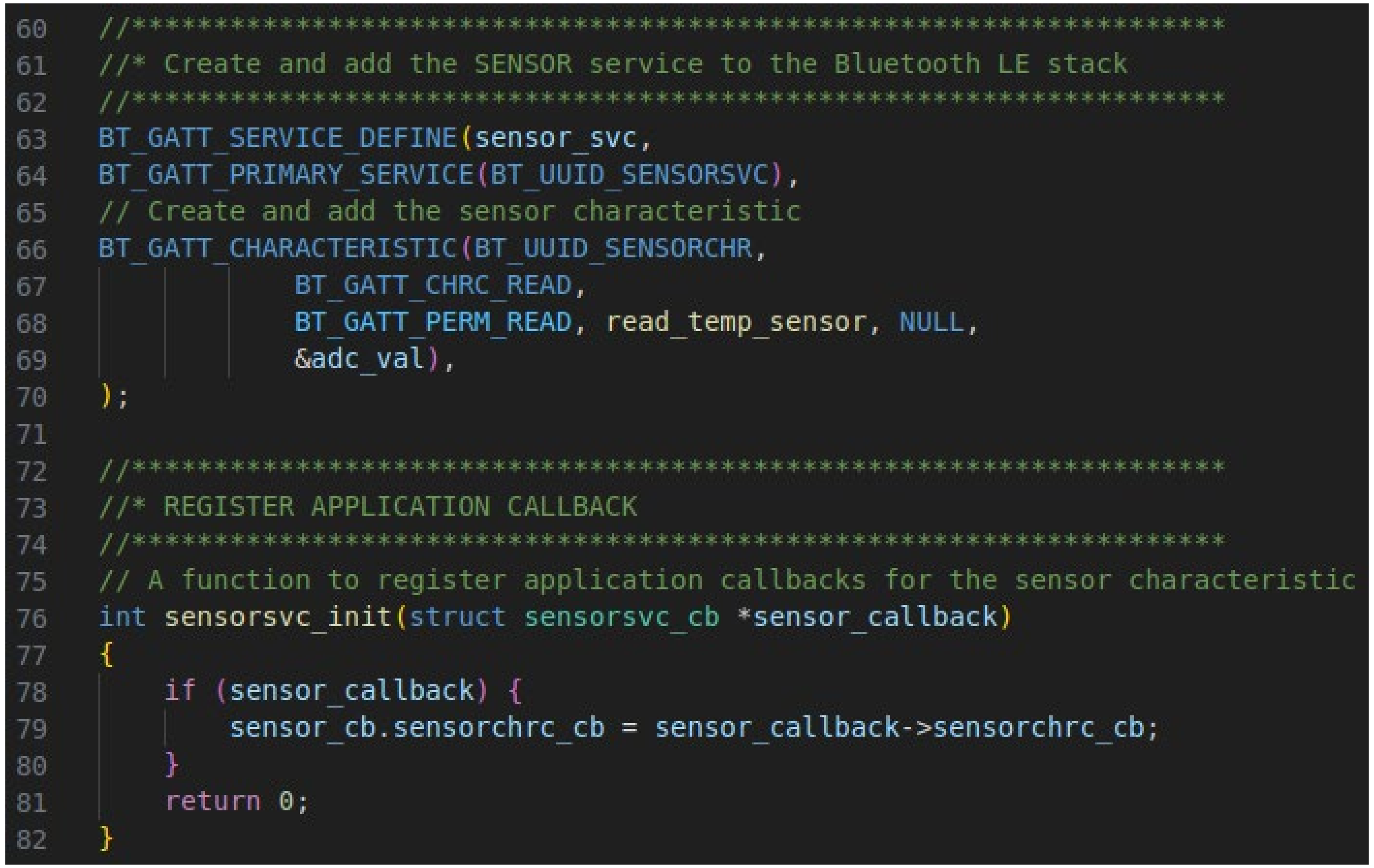
Callbacks – Read Temp Sensor Callback

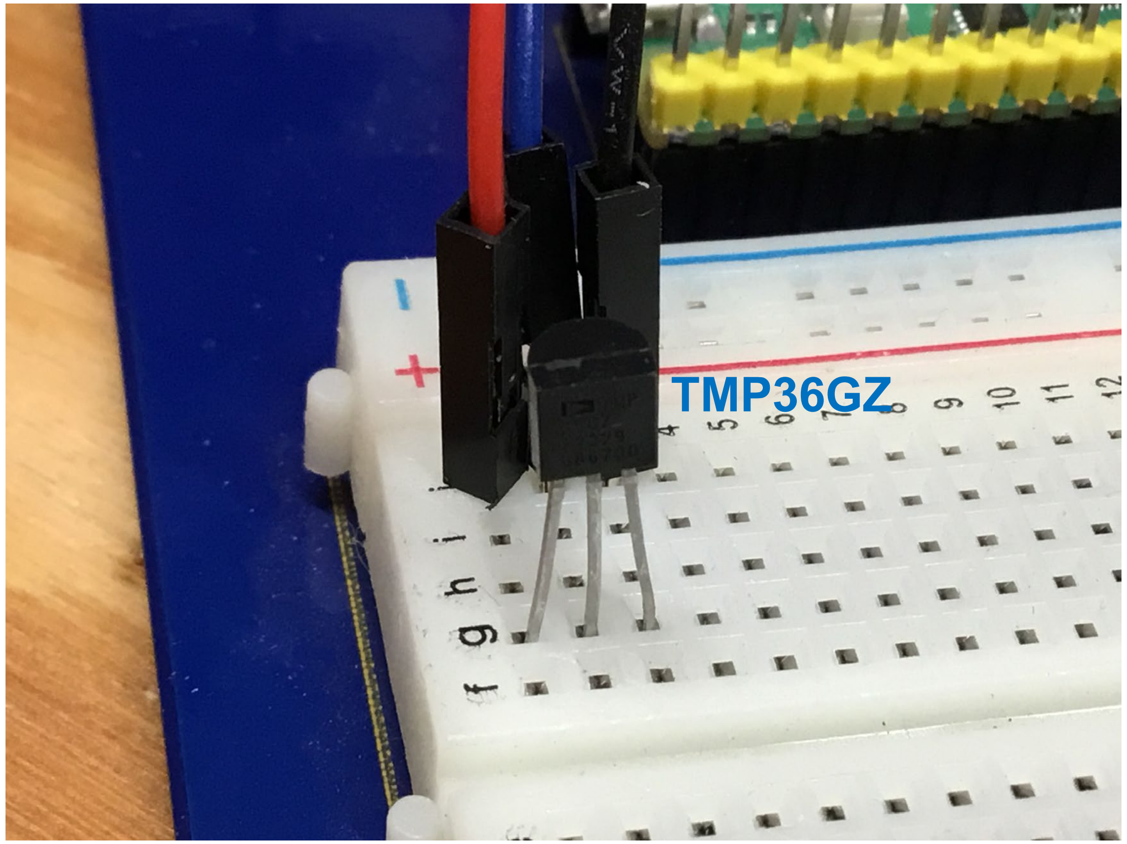


```c
static struct sensorsvc_cb      sensor_cb;
static struct ledsvc_cb         led_cb;

static uint16_t adc_val;

//*******************************************************************
//* Read Temp Sensor Callback
//*******************************************************************
static ssize_t read_temp_sensor(struct bt_conn *conn,
                                const struct bt_gatt_attr *attr,
                                void *buf,
                                uint16_t len,
                                uint16_t offset)
{
    //get a pointer to sensor value which is passed in the BT_GATT_CHARACTERISTIC() and stored in attr->user_data
    const uint16_t *value = attr->user_data;
    LOG_INF("Attribute read, handle: %u, conn: %p", attr->handle, (void *)conn);
    if (sensor_cb.sensorchrc_cb) {
        // Call the application callback function to update the get the current adc value
        adc_val = sensor_cb.sensorchrc_cb();

        LOG_INF("adc_val: %04x",adc_val);
        LOG_INF("len: %u",len);
        LOG_INF("offset: %u",offset);
        LOG_INF("size of value: %u",sizeof(*value));
        LOG_INF("value: %d",*value);
        return bt_gatt_attr_read(conn, attr, buf, len, offset, value, sizeof(*value));
    }
    return 0;
}
```

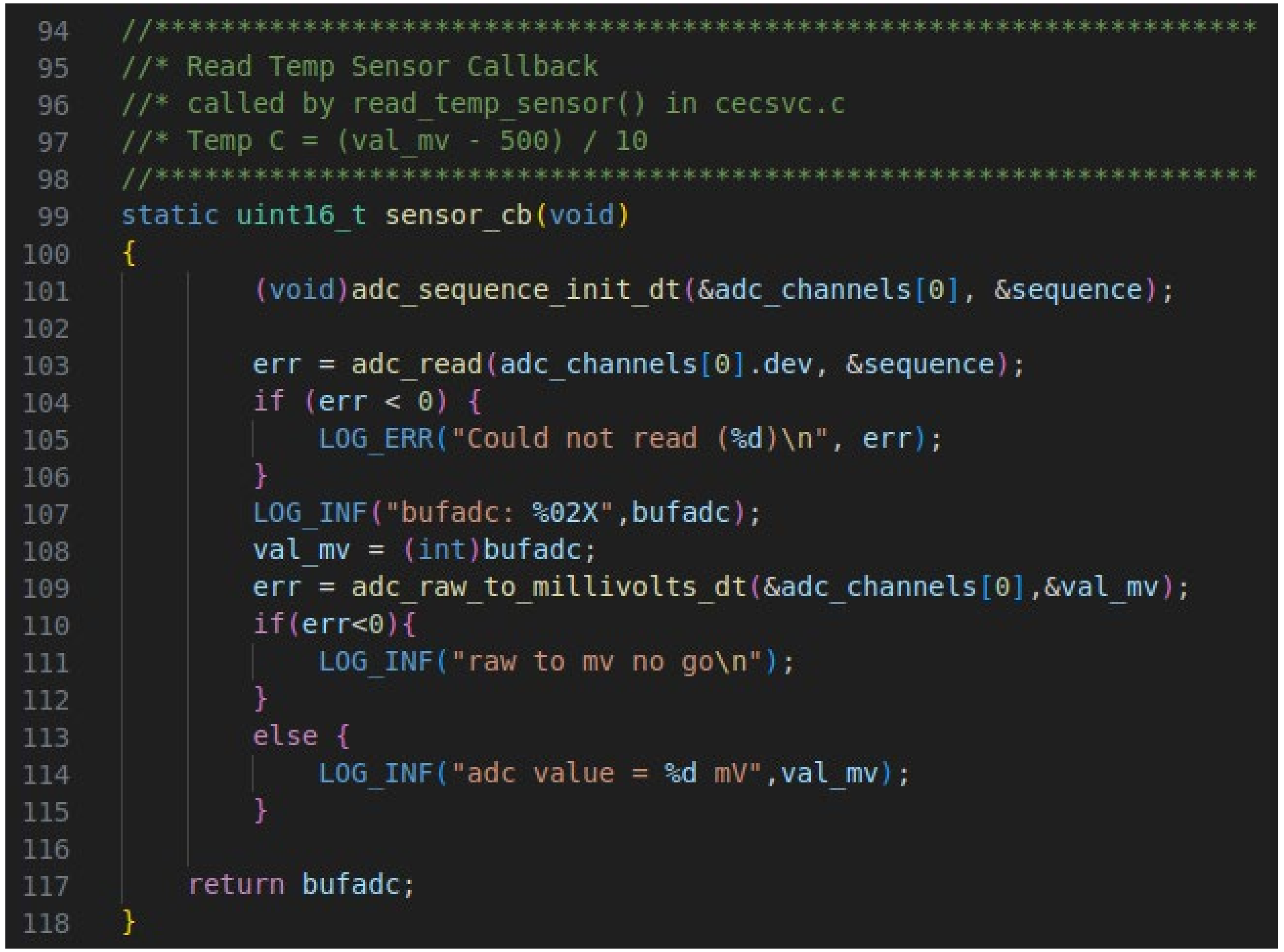


Callbacks – Read Temp Sensor Callback

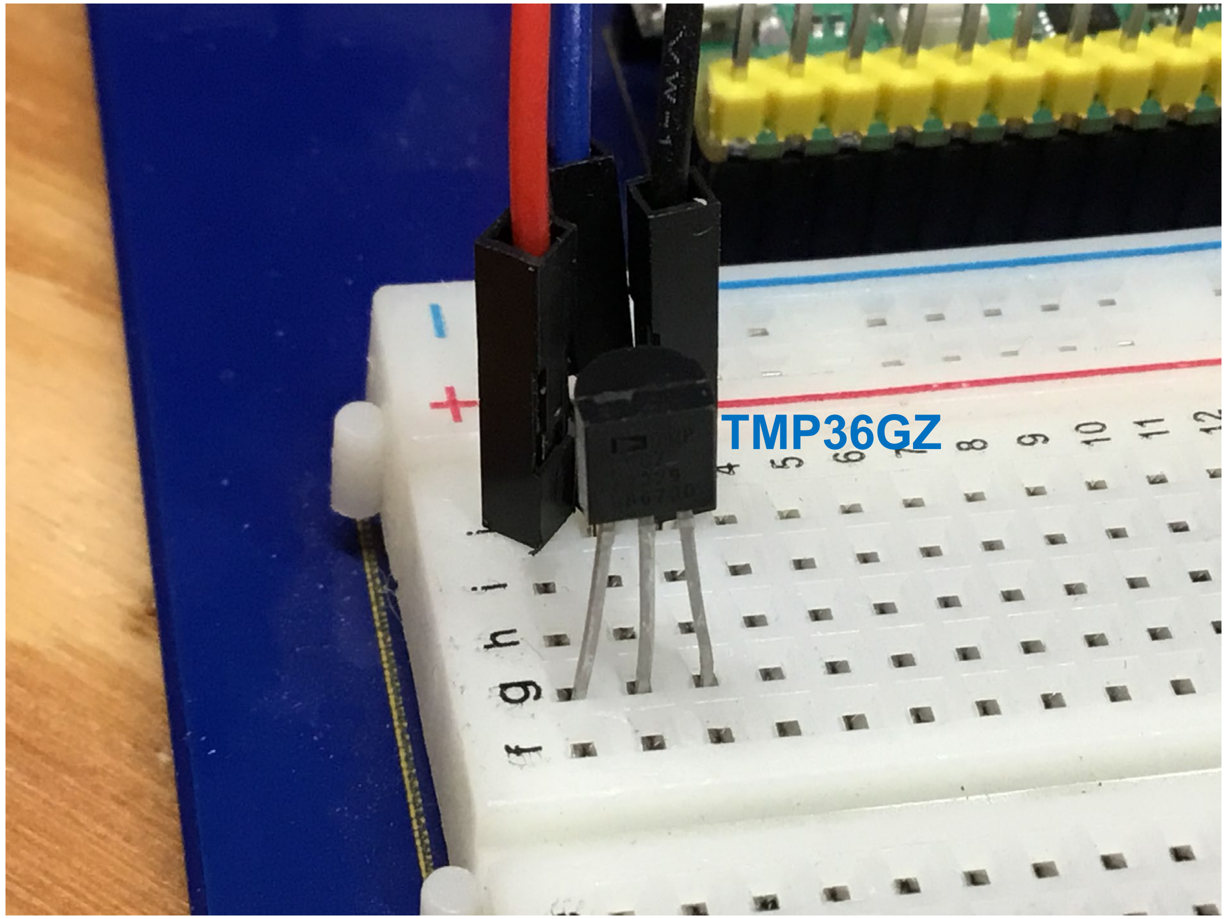

```
//**********************************************************************
//* Create and add the SENSOR service to the Bluetooth LE stack
//**********************************************************************
BT_GATT_SERVICE_DEFINE(sensor_svc,
BT_GATT_PRIMARY_SERVICE(BT_UUID_SENSORSVC),
// Create and add the sensor characteristic
BT_GATT_CHARACTERISTIC(BT_UUID_SENSORCHR,
            BT_GATT_CHRC_READ,
            BT_GATT_PERM_READ, read_temp_sensor, NULL,
            &adc_val),
);

//**********************************************************************
//* REGISTER APPLICATION CALLBACK
//**********************************************************************
// A function to register application callbacks for the sensor characteristic
int sensorsvc_init(struct sensorsvc_cb *sensor_callback)
{
    if (sensor_callback) {
        sensor_cb.sensorchrc_cb = sensor_callback->sensorchrc_cb;
    }
    return 0;
}
```



Callbacks – Read Temp Sensor Callback

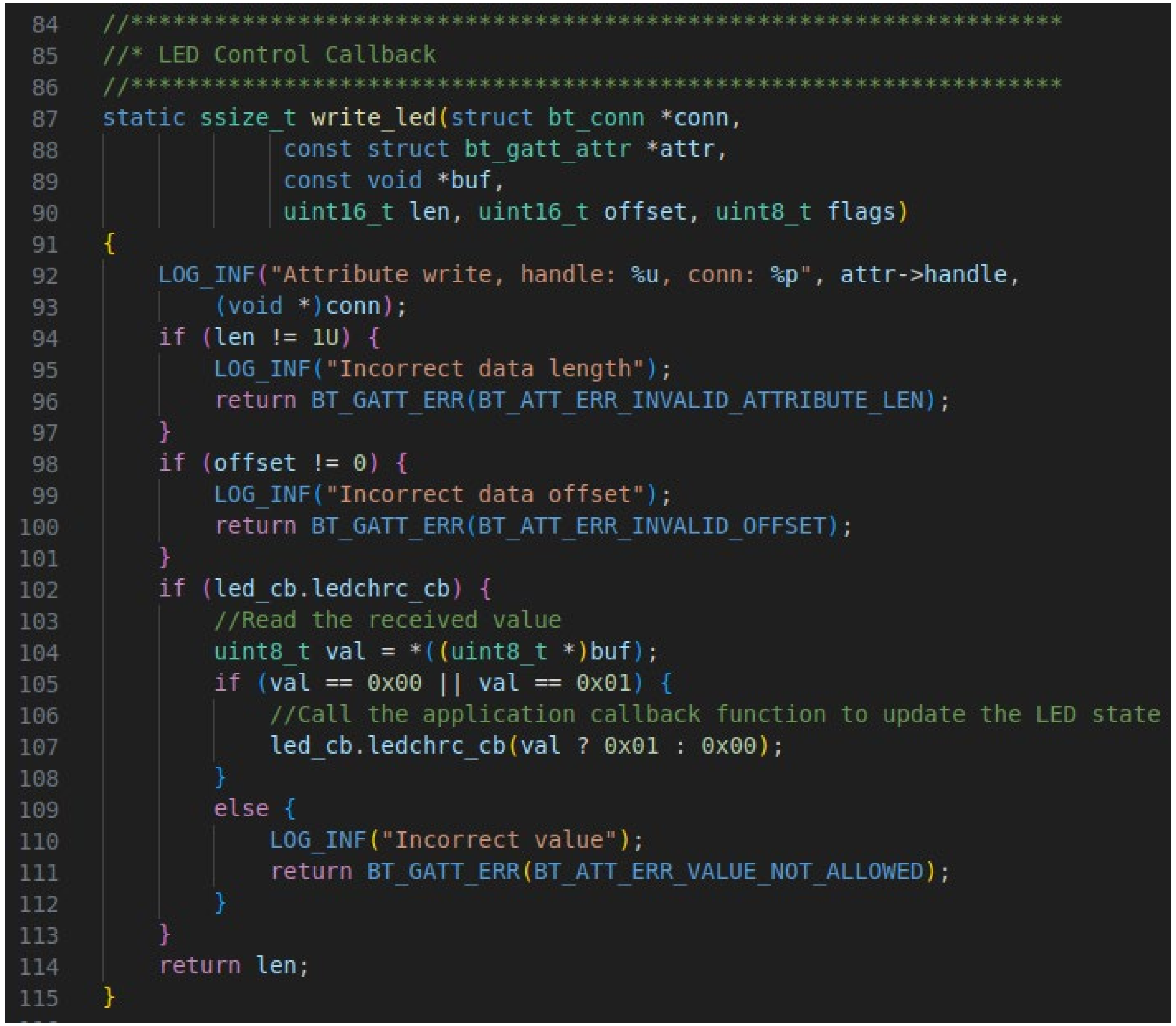


```
//*******************************************************************
//* Read Temp Sensor Callback
//* called by read_temp_sensor() in cecsvc.c
//* Temp C = (val_mv - 500) / 10
//*******************************************************************
static uint16_t sensor_cb(void)
{
        (void)adc_sequence_init_dt(&adc_channels[0], &sequence);

        err = adc_read(adc_channels[0].dev, &sequence);
        if (err < 0) {
            LOG_ERR("Could not read (%d)\n", err);
        }
        LOG_INF("bufadc: %02X",bufadc);
        val_mv = (int)bufadc;
        err = adc_raw_to_millivolts_dt(&adc_channels[0],&val_mv);
        if(err<0){
            LOG_INF("raw to mv no go\n");
        }
        else {
            LOG_INF("adc value = %d mV",val_mv);
        }

    return bufadc;
}
```

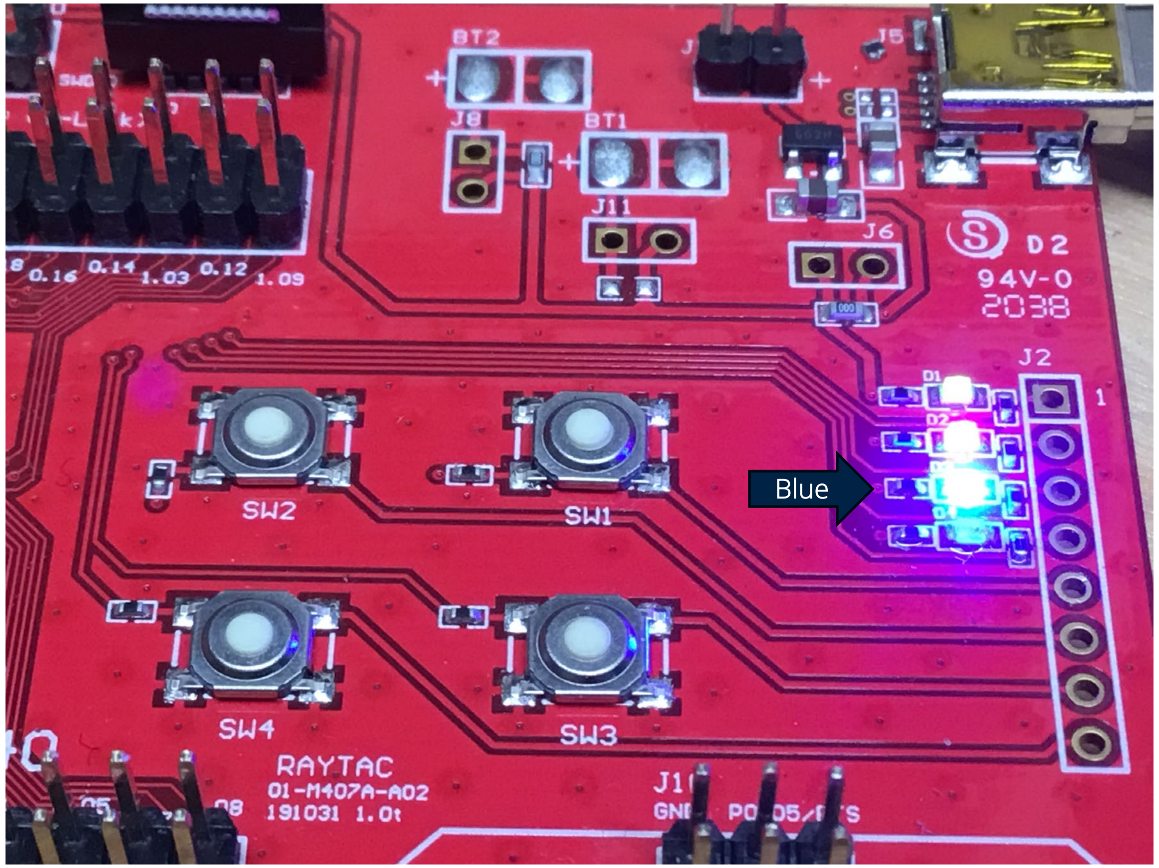


Developing IoT Applications with Nordic nRF Modules
Basic BLE Data Exchange
Nordic BLE Building Blocks – cecsvc.c

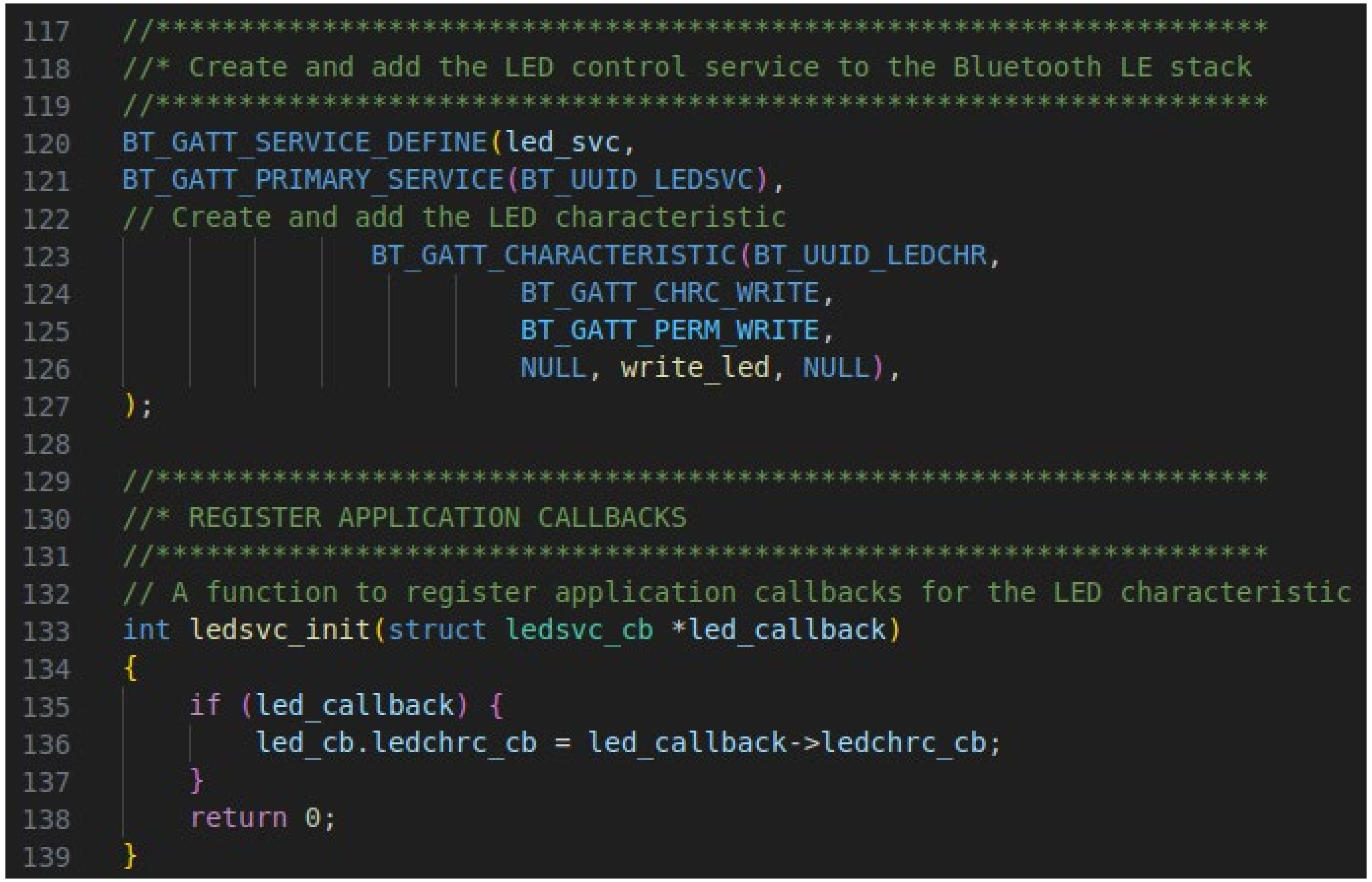
Callbacks – LED Control Callback



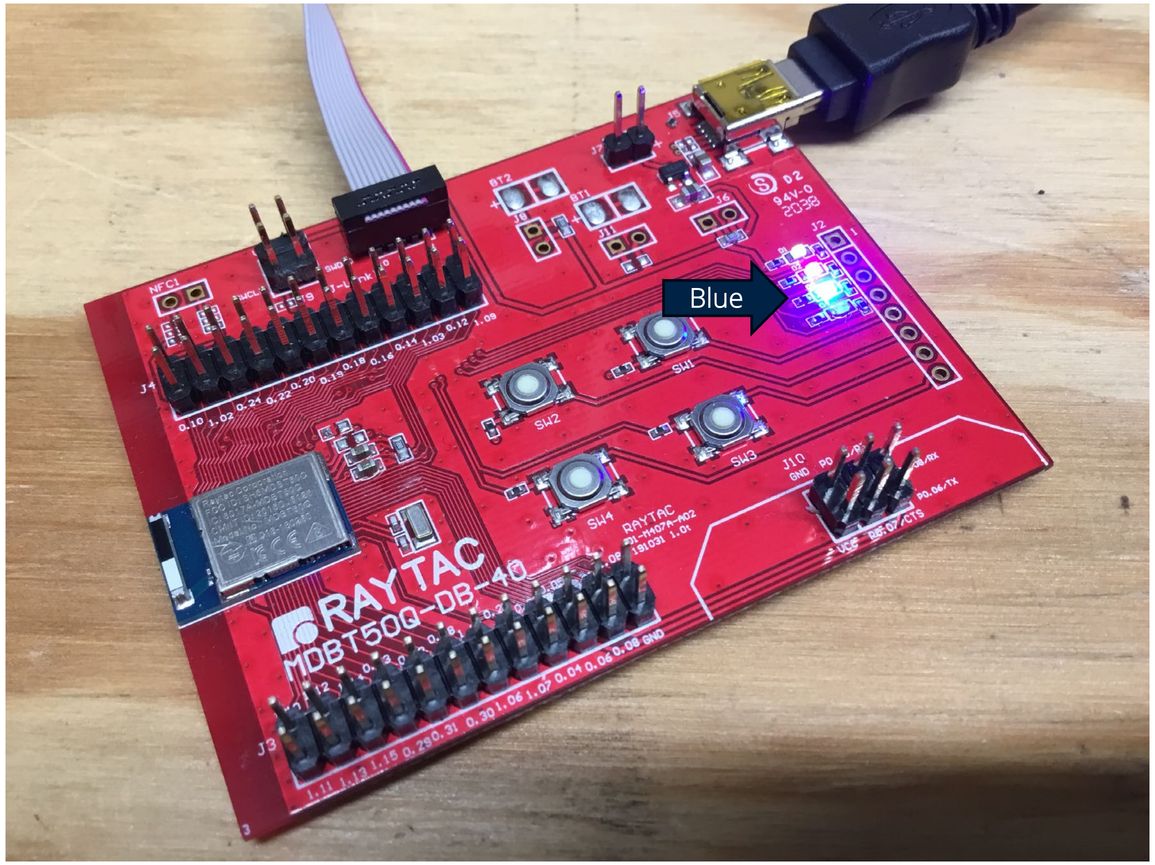
```c
//*******************************************************************
//* LED Control Callback
//*******************************************************************
static ssize_t write_led(struct bt_conn *conn,
                         const struct bt_gatt_attr *attr,
                         const void *buf,
                         uint16_t len, uint16_t offset, uint8_t flags)
{
    LOG_INF("Attribute write, handle: %u, conn: %p", attr->handle,
        (void *)conn);
    if (len != 1U) {
        LOG_INF("Incorrect data length");
        return BT_GATT_ERR(BT_ATT_ERR_INVALID_ATTRIBUTE_LEN);
    }
    if (offset != 0) {
        LOG_INF("Incorrect data offset");
        return BT_GATT_ERR(BT_ATT_ERR_INVALID_OFFSET);
    }
    if (led_cb.ledchrc_cb) {
        //Read the received value
        uint8_t val = *((uint8_t *)buf);
        if (val == 0x00 || val == 0x01) {
            //Call the application callback function to update the LED state
            led_cb.ledchrc_cb(val ? 0x01 : 0x00);
        }
        else {
            LOG_INF("Incorrect value");
            return BT_GATT_ERR(BT_ATT_ERR_VALUE_NOT_ALLOWED);
        }
    }
    return len;
}
```



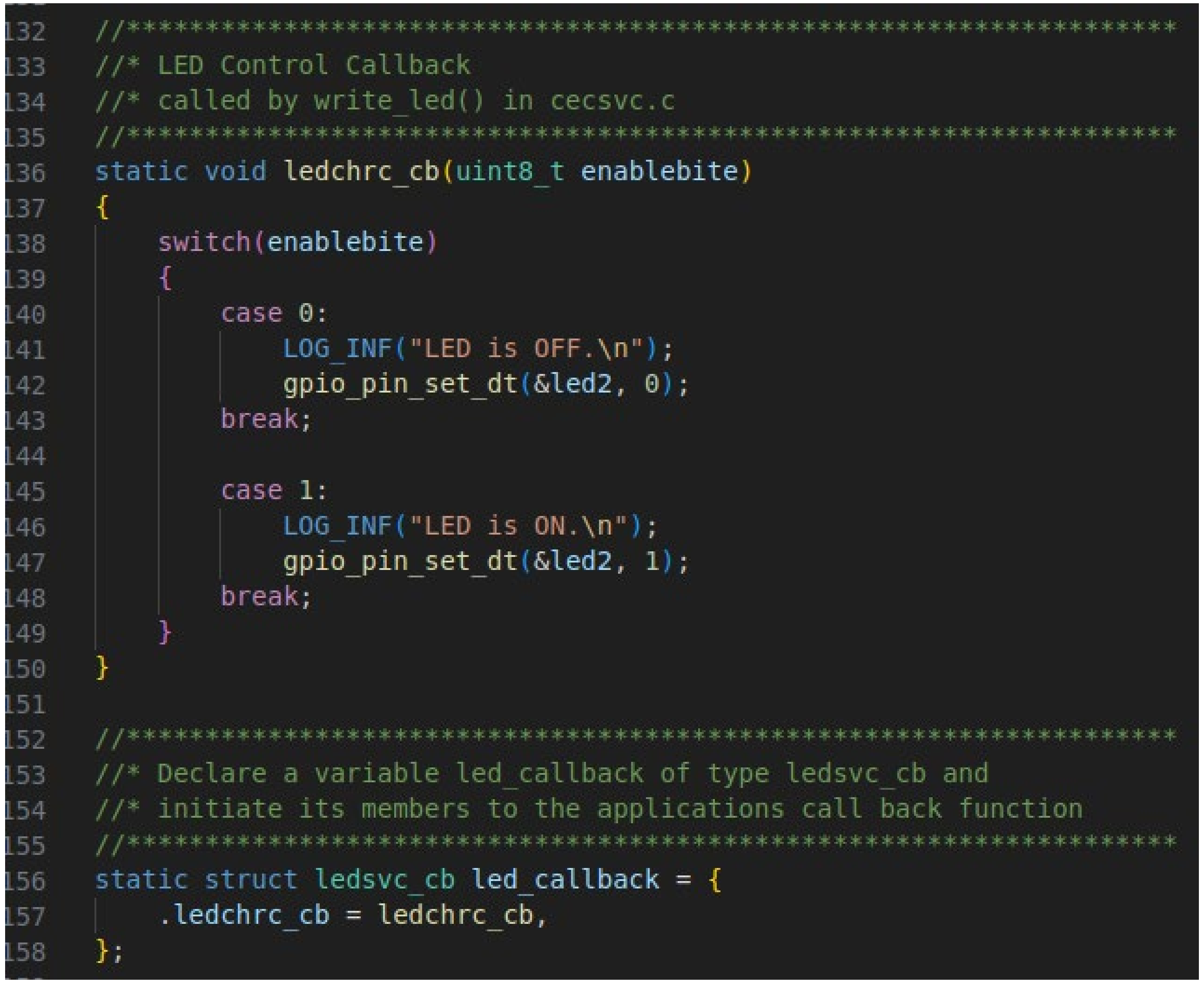
Callbacks – LED Control Callback



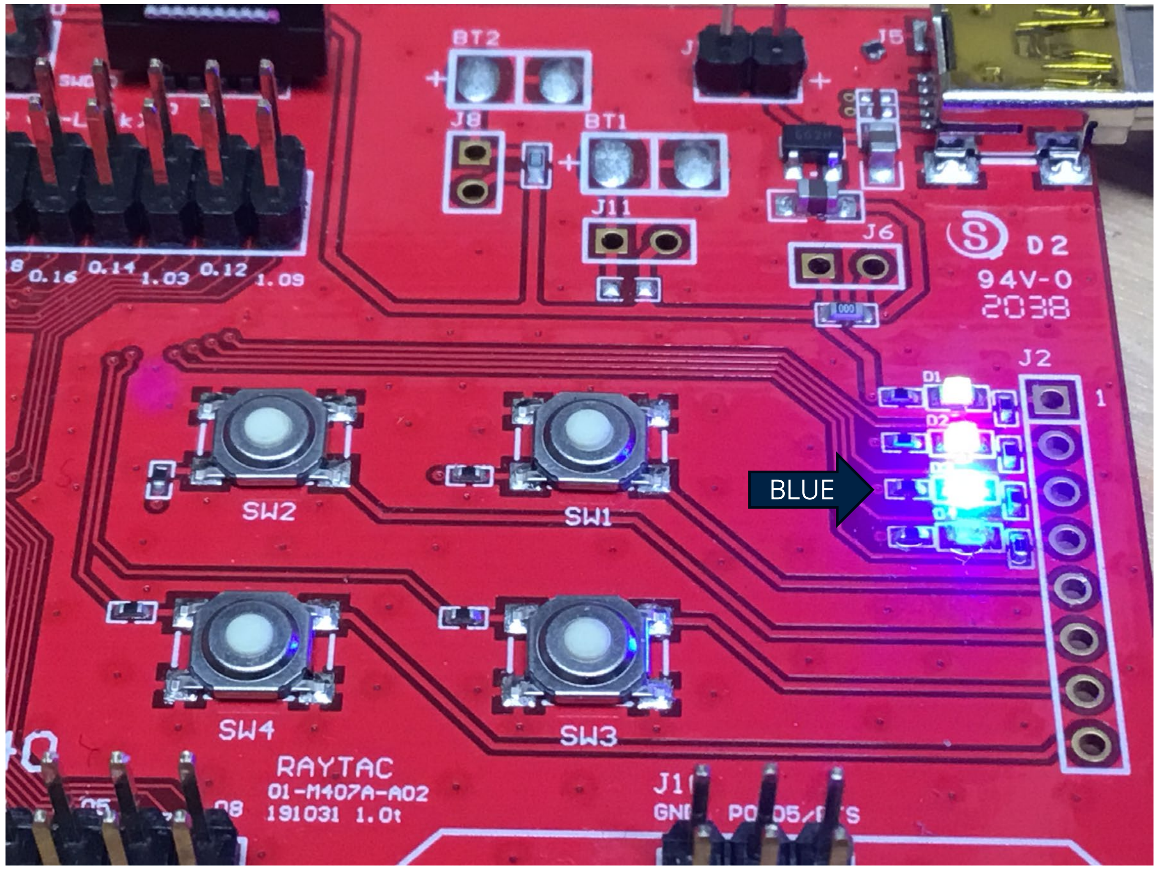
```c
//*****************************************************************
//* Create and add the LED control service to the Bluetooth LE stack
//*****************************************************************
BT_GATT_SERVICE_DEFINE(led_svc,
BT_GATT_PRIMARY_SERVICE(BT_UUID_LEDSVC),
// Create and add the LED characteristic
                BT_GATT_CHARACTERISTIC(BT_UUID_LEDCHR,
                         BT_GATT_CHRC_WRITE,
                         BT_GATT_PERM_WRITE,
                         NULL, write_led, NULL),
);

//*****************************************************************
//* REGISTER APPLICATION CALLBACKS
//*****************************************************************
// A function to register application callbacks for the LED characteristic
int ledsvc_init(struct ledsvc_cb *led_callback)
{
    if (led_callback) {
        led_cb.ledchrc_cb = led_callback->ledchrc_cb;
    }
    return 0;
}
```



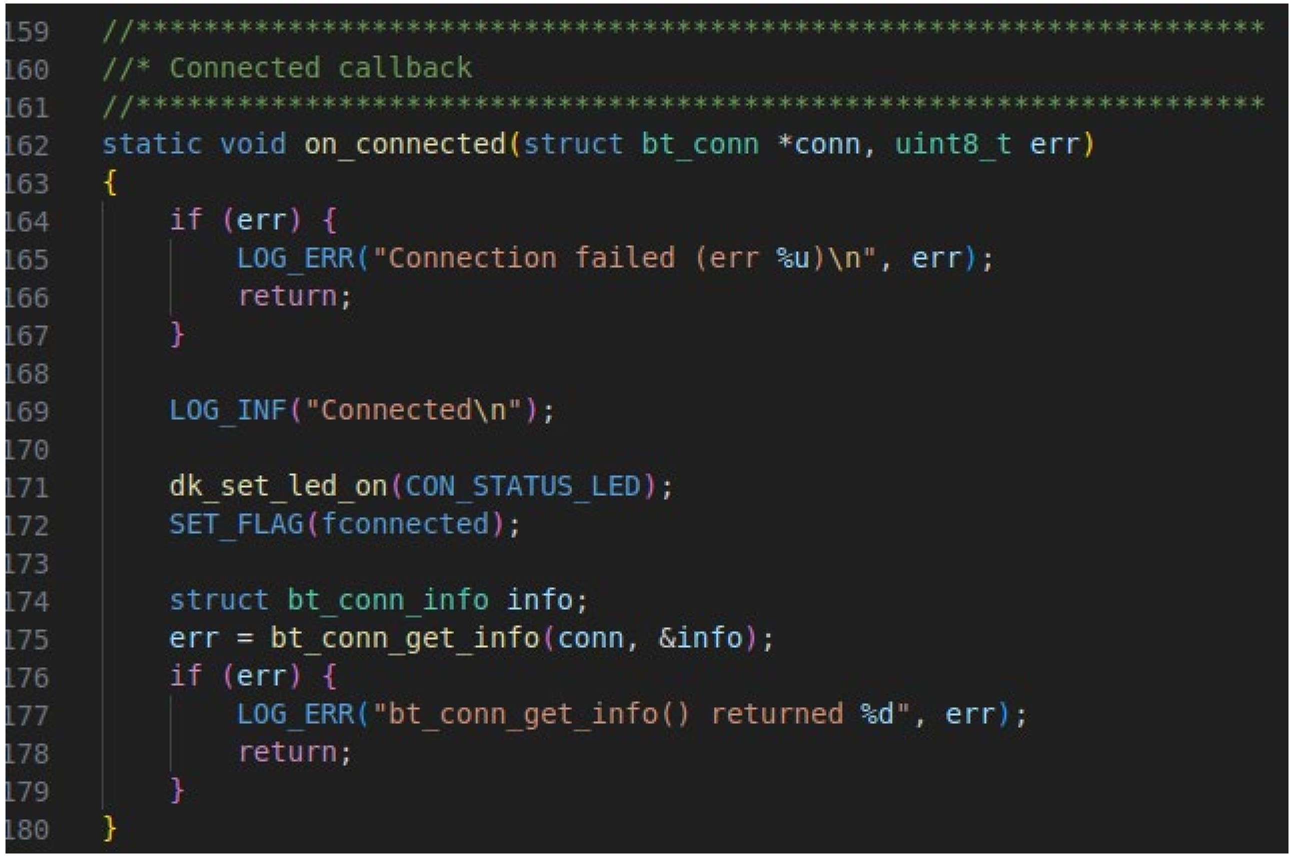
Callbacks – LED Control Callback



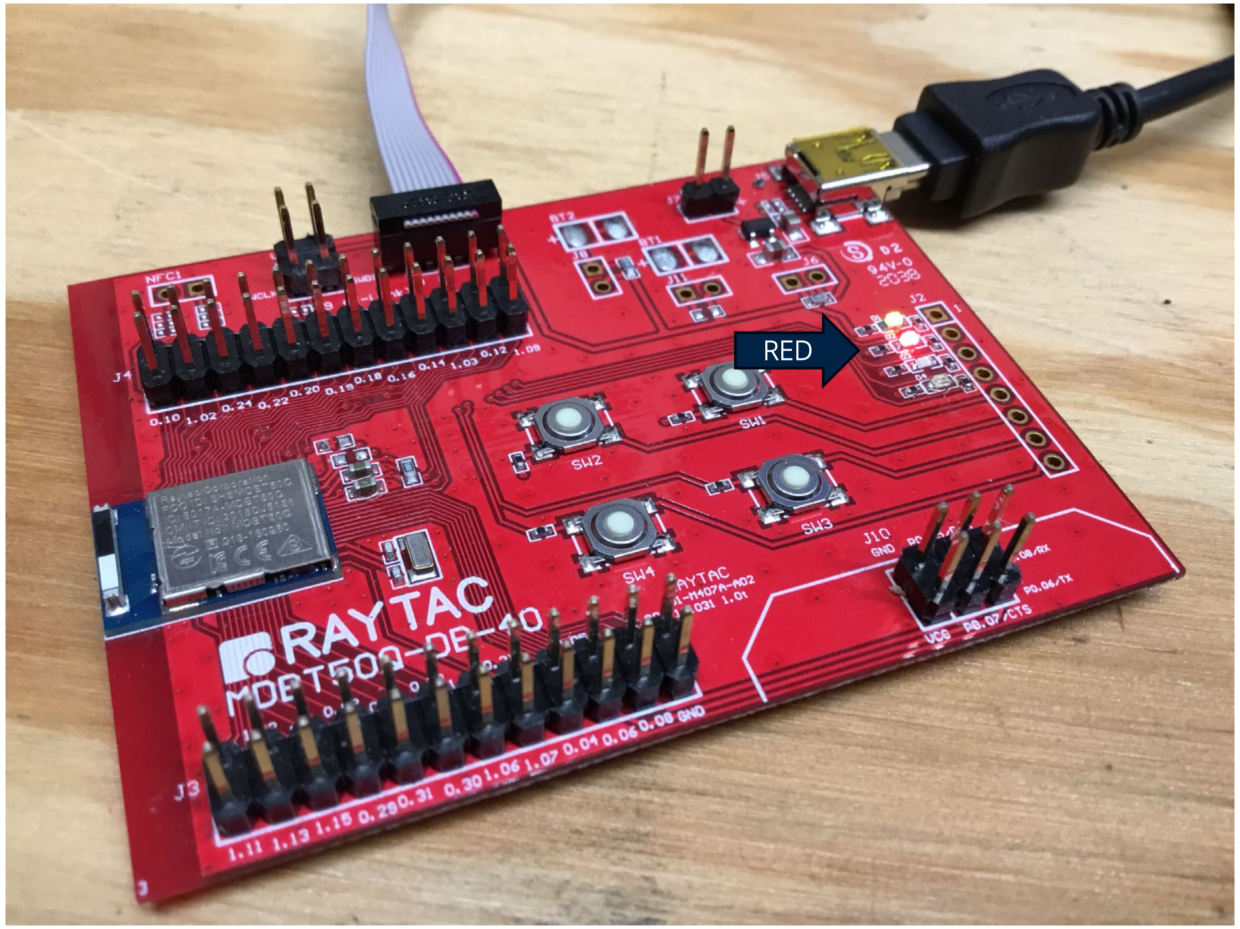
```
//**********************************************************************
//* LED Control Callback
//* called by write_led() in cecsvc.c
//**********************************************************************
static void ledchrc_cb(uint8_t enablebite)
{
    switch(enablebite)
    {
        case 0:
            LOG_INF("LED is OFF.\n");
            gpio_pin_set_dt(&led2, 0);
        break;

        case 1:
            LOG_INF("LED is ON.\n");
            gpio_pin_set_dt(&led2, 1);
        break;
    }
}

//**********************************************************************
//* Declare a variable led_callback of type ledsvc_cb and
//* initiate its members to the applications call back function
//**********************************************************************
static struct ledsvc_cb led_callback = {
    .ledchrc_cb = ledchrc_cb,
};
```



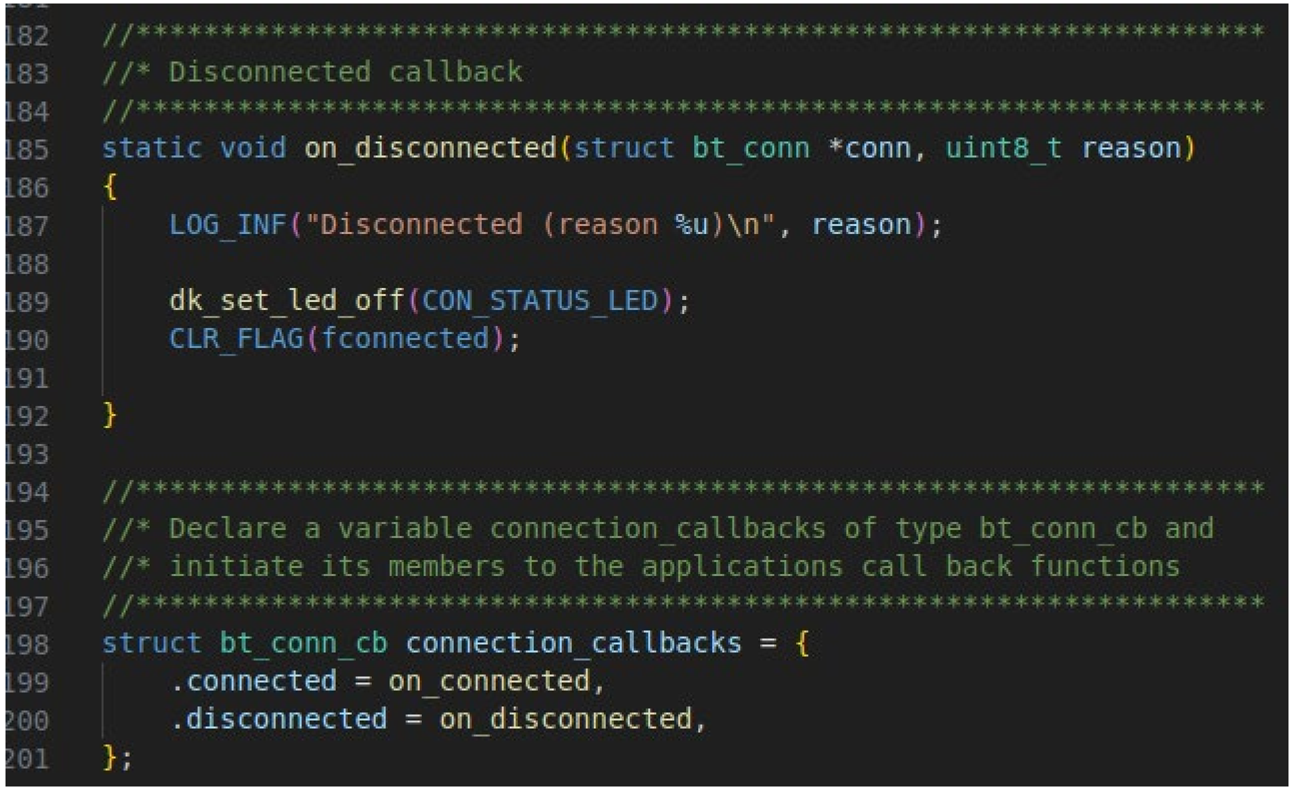
Callbacks – Connected



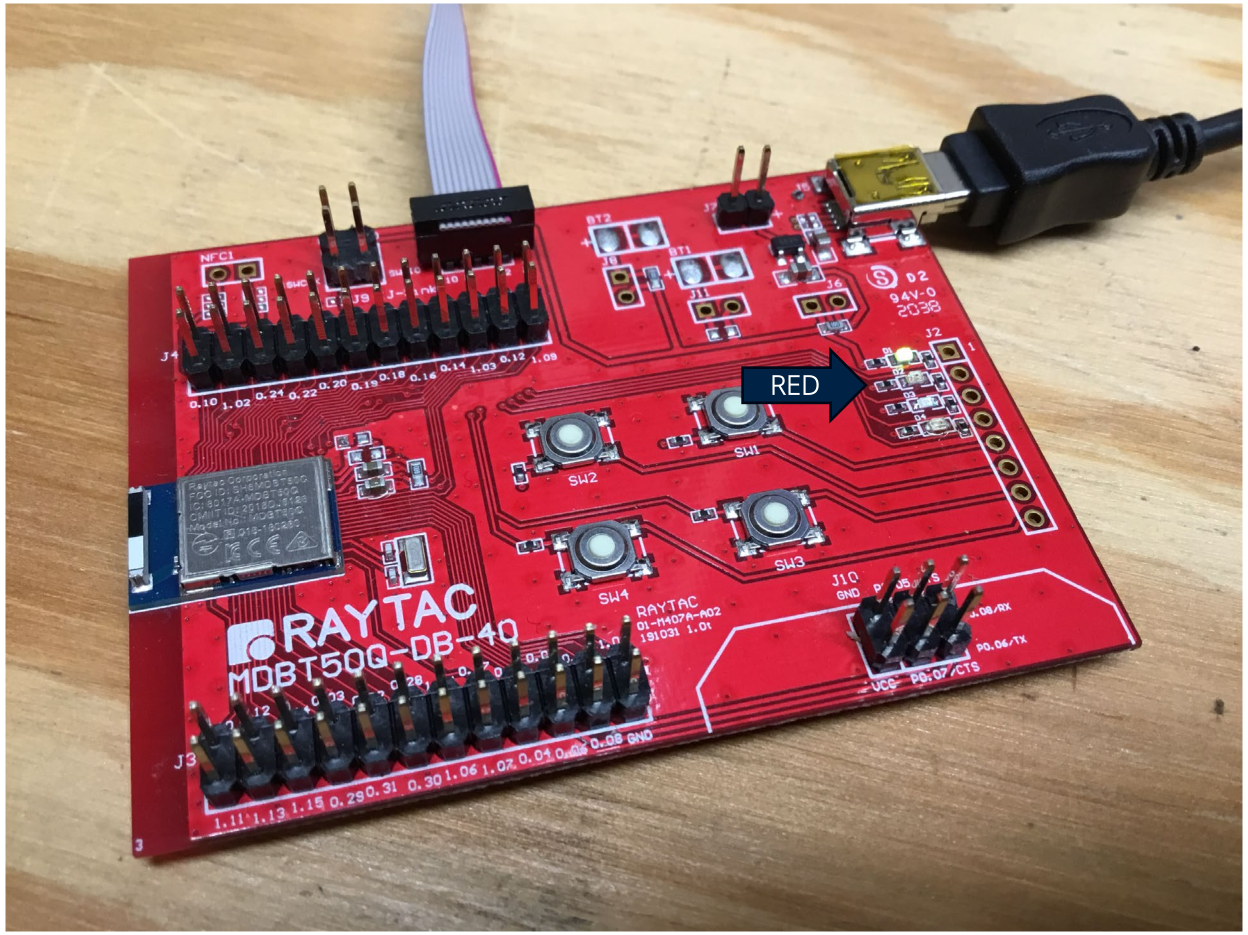
```
//**********************************************************************
//* Connected callback
//**********************************************************************
static void on_connected(struct bt_conn *conn, uint8_t err)
{
    if (err) {
        LOG_ERR("Connection failed (err %u)\n", err);
        return;
    }

    LOG_INF("Connected\n");

    dk_set_led_on(CON_STATUS_LED);
    SET_FLAG(fconnected);

    struct bt_conn_info info;
    err = bt_conn_get_info(conn, &info);
    if (err) {
        LOG_ERR("bt_conn_get_info() returned %d", err);
        return;
    }
}
```



Callbacks – Disconnected



```
//**********************************************************************
//* Disconnected callback
//**********************************************************************
static void on_disconnected(struct bt_conn *conn, uint8_t reason)
{
    LOG_INF("Disconnected (reason %u)\n", reason);

    dk_set_led_off(CON_STATUS_LED);
    CLR_FLAG(fconnected);

}

//**********************************************************************
//* Declare a variable connection_callbacks of type bt_conn_cb and
//* initiate its members to the applications call back functions
//**********************************************************************
struct bt_conn_cb connection_callbacks = {
    .connected = on_connected,
    .disconnected = on_disconnected,
};
```

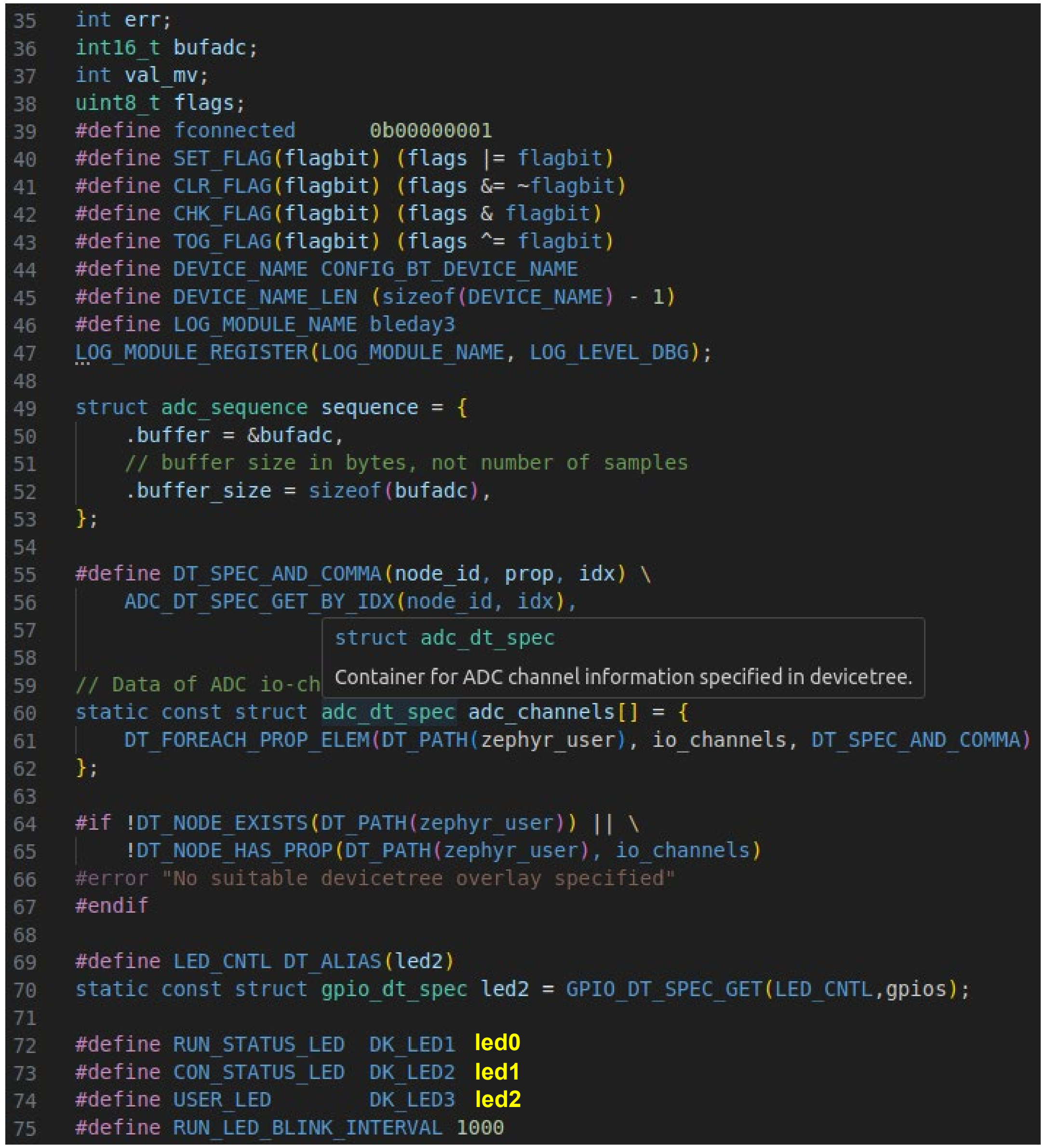


Developing IoT Applications with Nordic nRF Modules

Basic BLE Data Exchange

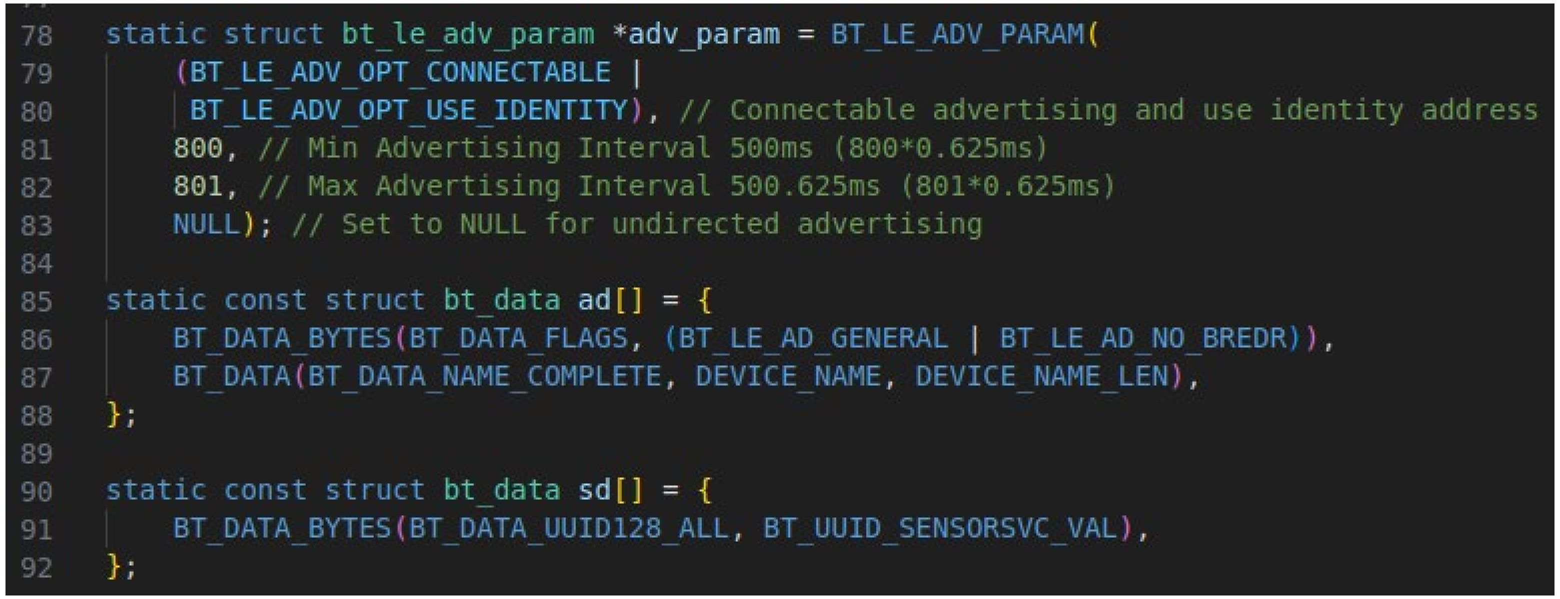
Nordic BLE Building Blocks – main.c

Housekeeping

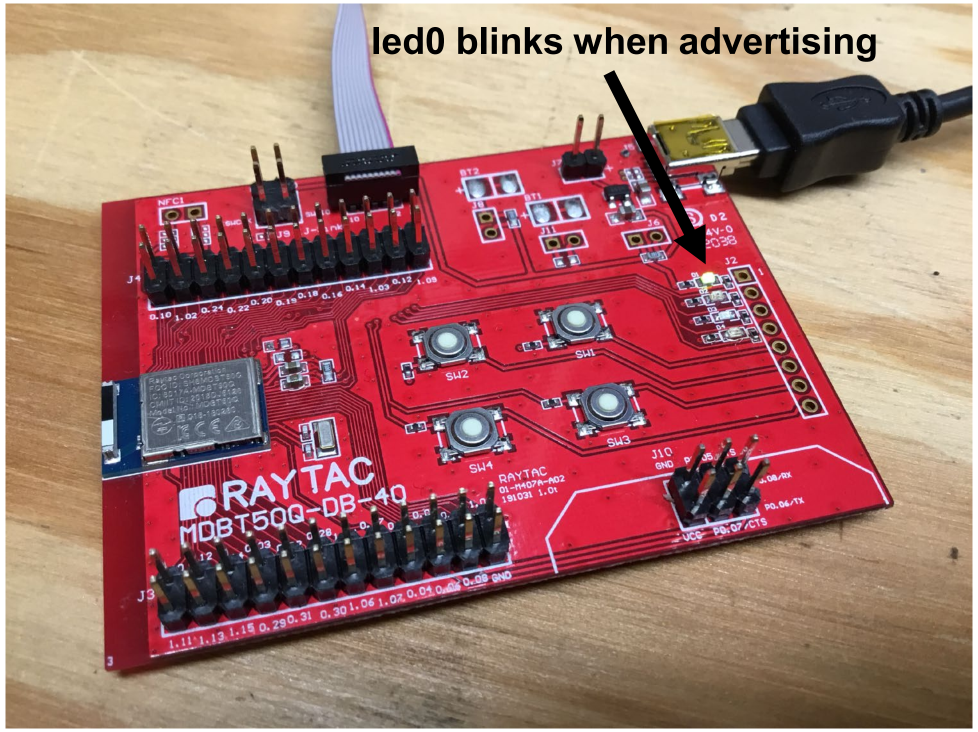


```c
int err;
int16_t bufadc;
int val_mv;
uint8_t flags;
#define fconnected      0b00000001
#define SET_FLAG(flagbit) (flags |= flagbit)
#define CLR_FLAG(flagbit) (flags &= ~flagbit)
#define CHK_FLAG(flagbit) (flags & flagbit)
#define TOG_FLAG(flagbit) (flags ^= flagbit)
#define DEVICE_NAME CONFIG_BT_DEVICE_NAME
#define DEVICE_NAME_LEN (sizeof(DEVICE_NAME) - 1)
#define LOG_MODULE_NAME bleday3
LOG_MODULE_REGISTER(LOG_MODULE_NAME, LOG_LEVEL_DBG);

struct adc_sequence sequence = {
    .buffer = &bufadc,
    // buffer size in bytes, not number of samples
    .buffer_size = sizeof(bufadc),
};

#define DT_SPEC_AND_COMMA(node_id, prop, idx) \
    ADC_DT_SPEC_GET_BY_IDX(node_id, idx),
                        struct adc_dt_spec
                        Container for ADC channel information specified in devicetree.
// Data of ADC io-ch
static const struct adc_dt_spec adc_channels[] = {
    DT_FOREACH_PROP_ELEM(DT_PATH(zephyr_user), io_channels, DT_SPEC_AND_COMMA)
};

#if !DT_NODE_EXISTS(DT_PATH(zephyr_user)) || \
    !DT_NODE_HAS_PROP(DT_PATH(zephyr_user), io_channels)
#error "No suitable devicetree overlay specified"
#endif

#define LED_CNTL DT_ALIAS(led2)
static const struct gpio_dt_spec led2 = GPIO_DT_SPEC_GET(LED_CNTL,gpios);

#define RUN_STATUS_LED  DK_LED1  led0
#define CON_STATUS_LED  DK_LED2  led1
#define USER_LED        DK_LED3  led2
#define RUN_LED_BLINK_INTERVAL 1000
```

Advertising Parameters



```c
static struct bt_le_adv_param *adv_param = BT_LE_ADV_PARAM(
      (BT_LE_ADV_OPT_CONNECTABLE |
       BT_LE_ADV_OPT_USE_IDENTITY), // Connectable advertising and use identity address
      800, // Min Advertising Interval 500ms (800*0.625ms)
      801, // Max Advertising Interval 500.625ms (801*0.625ms)
      NULL); // Set to NULL for undirected advertising

static const struct bt_data ad[] = {
      BT_DATA_BYTES(BT_DATA_FLAGS, (BT_LE_AD_GENERAL | BT_LE_AD_NO_BREDR)),
      BT_DATA(BT_DATA_NAME_COMPLETE, DEVICE_NAME, DEVICE_NAME_LEN),
};

static const struct bt_data sd[] = {
      BT_DATA_BYTES(BT_DATA_UUID128_ALL, BT_UUID_SENSORSVC_VAL),
};
```

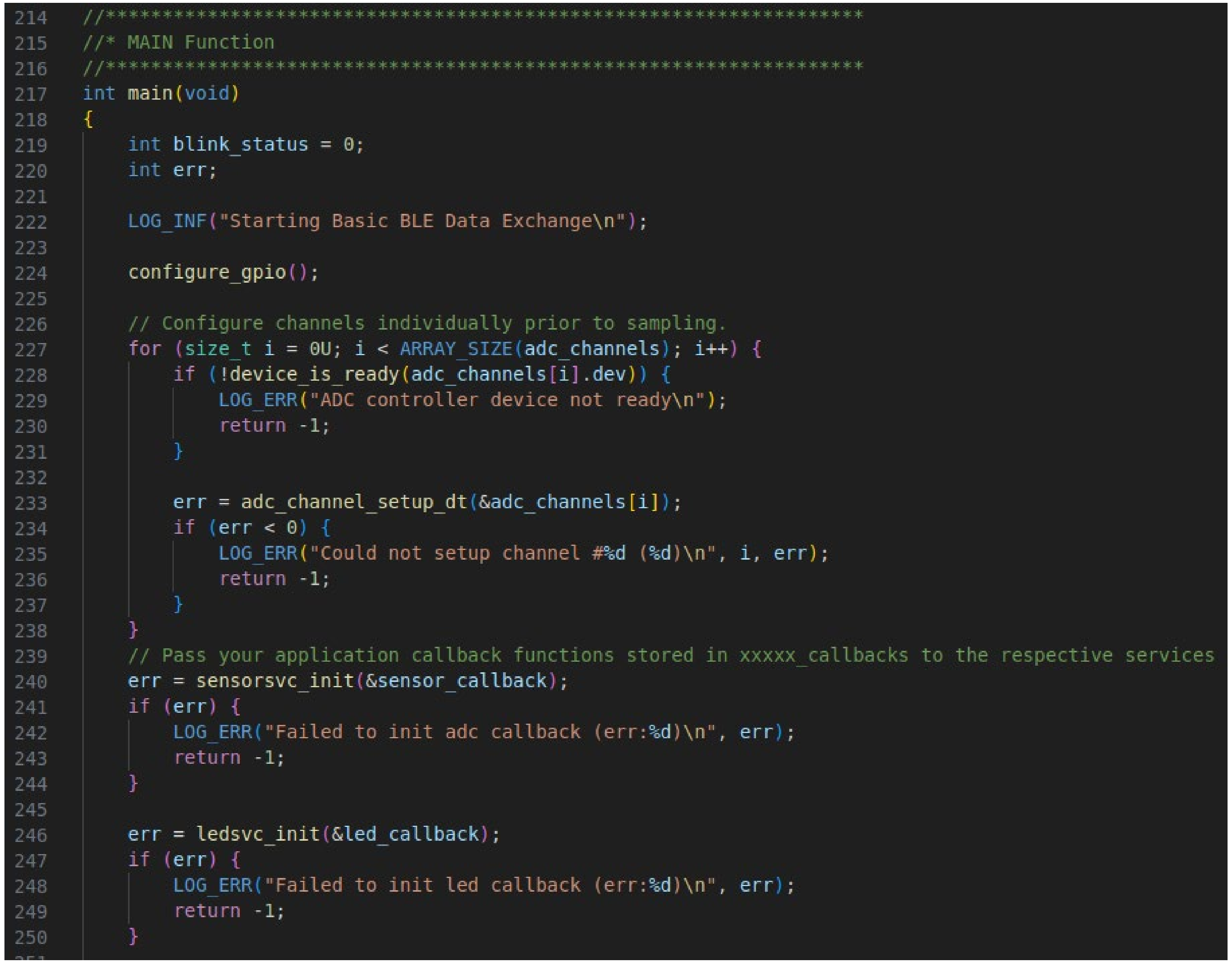


Developing IoT Applications with Nordic nRF Modules

Basic BLE Data Exchange

Nordic BLE Building Blocks – main.c

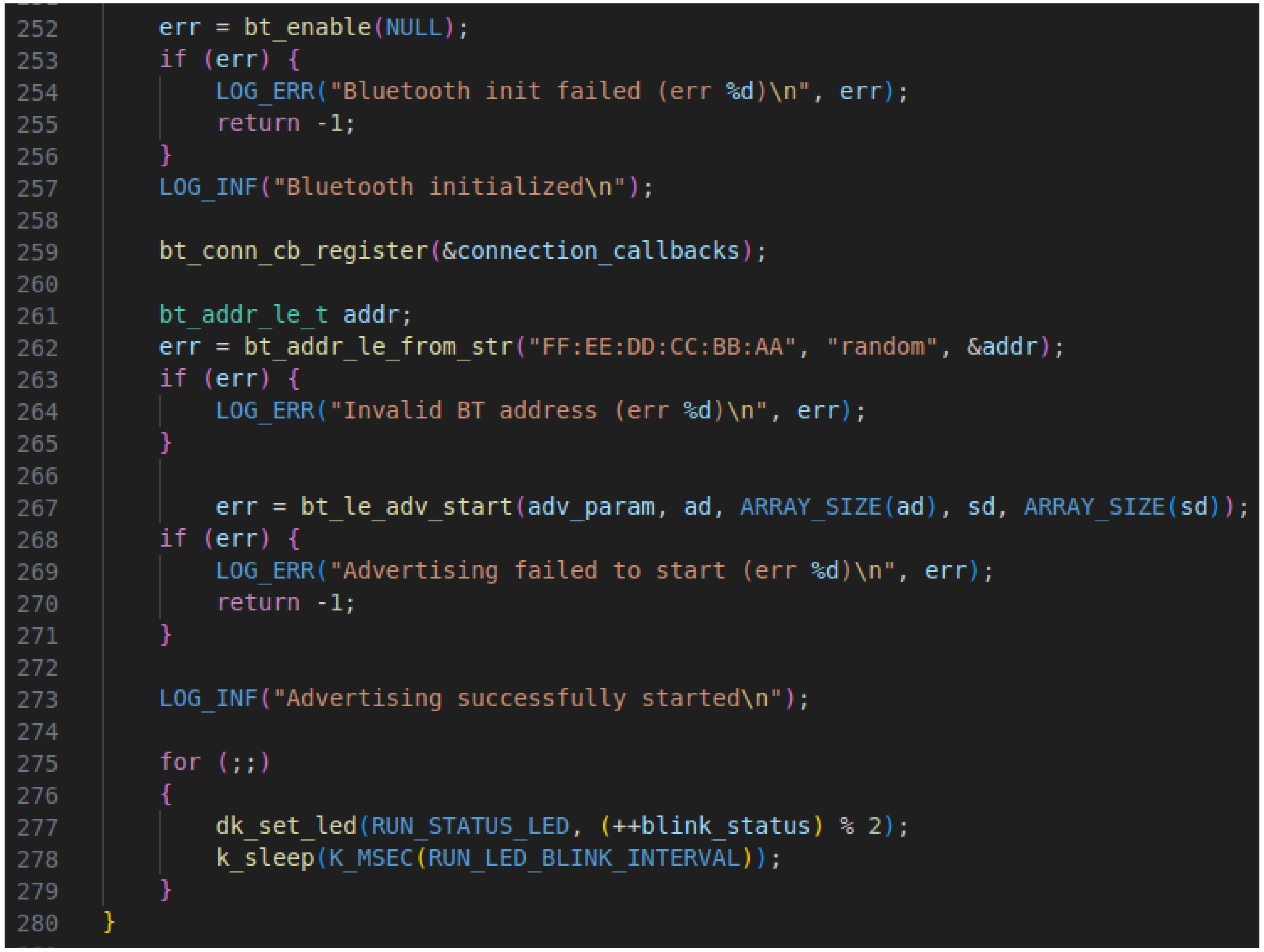
Meat and Potatoes – Finish the Application Setup



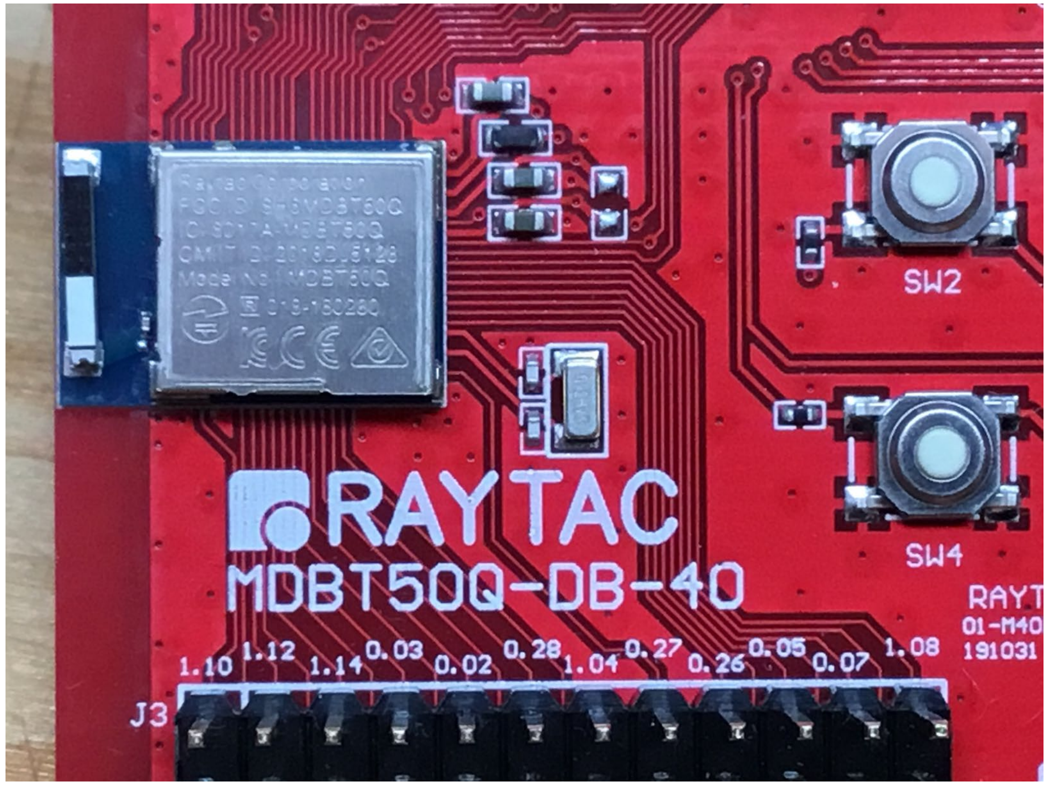
```c
//**********************************************************************
//* MAIN Function
//**********************************************************************
int main(void)
{
    int blink_status = 0;
    int err;

    LOG_INF("Starting Basic BLE Data Exchange\n");

    configure_gpio();

    // Configure channels individually prior to sampling.
    for (size_t i = 0U; i < ARRAY_SIZE(adc_channels); i++) {
        if (!device_is_ready(adc_channels[i].dev)) {
            LOG_ERR("ADC controller device not ready\n");
            return -1;
        }

        err = adc_channel_setup_dt(&adc_channels[i]);
        if (err < 0) {
            LOG_ERR("Could not setup channel #%d (%d)\n", i, err);
            return -1;
        }
    }
    // Pass your application callback functions stored in xxxxx_callbacks to the respective services
    err = sensorsvc_init(&sensor_callback);
    if (err) {
        LOG_ERR("Failed to init adc callback (err:%d)\n", err);
        return -1;
    }

    err = ledsvc_init(&led_callback);
    if (err) {
        LOG_ERR("Failed to init led callback (err:%d)\n", err);
        return -1;
    }
```

Developing IoT Applications with Nordic nRF Modules
Basic BLE Data Exchange
Nordic BLE Building Blocks – main.c

Meat and Potatoes – Enable the BLE Radio

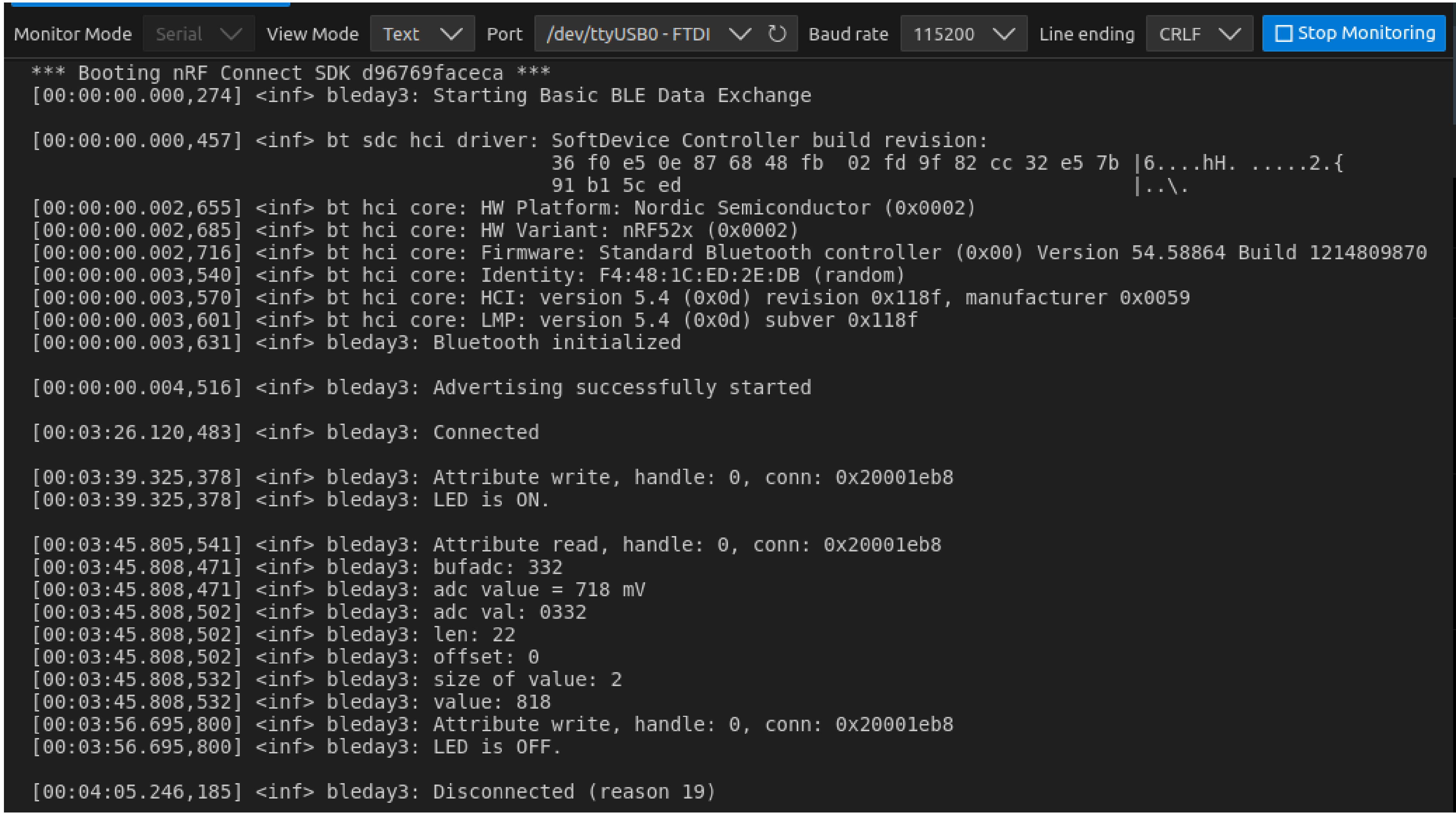


```
    err = bt_enable(NULL);
    if (err) {
        LOG_ERR("Bluetooth init failed (err %d)\n", err);
        return -1;
    }
    LOG_INF("Bluetooth initialized\n");

    bt_conn_cb_register(&connection_callbacks);

    bt_addr_le_t addr;
    err = bt_addr_le_from_str("FF:EE:DD:CC:BB:AA", "random", &addr);
    if (err) {
        LOG_ERR("Invalid BT address (err %d)\n", err);
    }

        err = bt_le_adv_start(adv_param, ad, ARRAY_SIZE(ad), sd, ARRAY_SIZE(sd));
    if (err) {
        LOG_ERR("Advertising failed to start (err %d)\n", err);
        return -1;
    }

    LOG_INF("Advertising successfully started\n");

    for (;;)
    {
        dk_set_led(RUN_STATUS_LED, (++blink_status) % 2);
        k_sleep(K_MSEC(RUN_LED_BLINK_INTERVAL));
    }
}
```

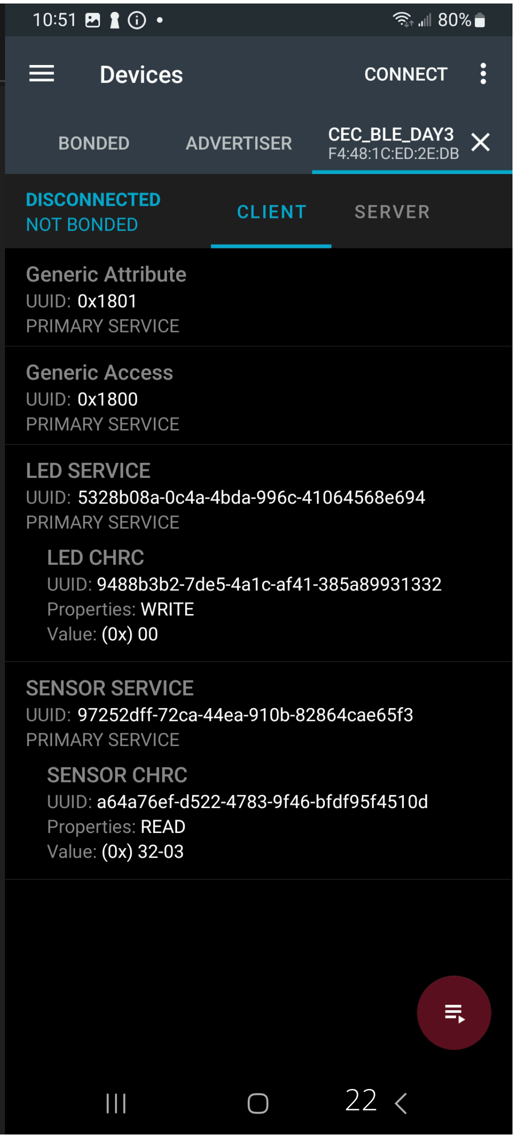


Developing IoT Applications with Nordic nRF Modules
Basic BLE Data Exchange
Nordic BLE Building Blocks – main.c

Let Her Rip or, as we say in the South, Let 'Er Rip



```
 *** Booting nRF Connect SDK d96769faceca ***
[00:00:00.000,274] <inf> bleday3: Starting Basic BLE Data Exchange

[00:00:00.000,457] <inf> bt sdc hci driver: SoftDevice Controller build revision:
                                            36 f0 e5 0e 87 68 48 fb  02 fd 9f 82 cc 32 e5 7b |6....hH. .....2.{
                                            91 b1 5c ed                                      |..\.
[00:00:00.002,655] <inf> bt hci core: HW Platform: Nordic Semiconductor (0x0002)
[00:00:00.002,685] <inf> bt hci core: HW Variant: nRF52x (0x0002)
[00:00:00.002,716] <inf> bt hci core: Firmware: Standard Bluetooth controller (0x00) Version 54.58864 Build 1214809870
[00:00:00.003,540] <inf> bt hci core: Identity: F4:48:1C:ED:2E:DB (random)
[00:00:00.003,570] <inf> bt hci core: HCI: version 5.4 (0x0d) revision 0x118f, manufacturer 0x0059
[00:00:00.003,601] <inf> bt hci core: LMP: version 5.4 (0x0d) subver 0x118f
[00:00:00.003,631] <inf> bleday3: Bluetooth initialized

[00:00:00.004,516] <inf> bleday3: Advertising successfully started

[00:03:26.120,483] <inf> bleday3: Connected

[00:03:39.325,378] <inf> bleday3: Attribute write, handle: 0, conn: 0x20001eb8
[00:03:39.325,378] <inf> bleday3: LED is ON.

[00:03:45.805,541] <inf> bleday3: Attribute read, handle: 0, conn: 0x20001eb8
[00:03:45.808,471] <inf> bleday3: bufadc: 332
[00:03:45.808,471] <inf> bleday3: adc value = 718 mV
[00:03:45.808,502] <inf> bleday3: adc val: 0332
[00:03:45.808,502] <inf> bleday3: len: 22
[00:03:45.808,502] <inf> bleday3: offset: 0
[00:03:45.808,532] <inf> bleday3: size of value: 2
[00:03:45.808,532] <inf> bleday3: value: 818
[00:03:56.695,800] <inf> bleday3: Attribute write, handle: 0, conn: 0x20001eb8
[00:03:56.695,800] <inf> bleday3: LED is OFF.

[00:04:05.246,185] <inf> bleday3: Disconnected (reason 19)
```



CEC_BLE_DAY3
F4:48:1C:ED:2E:DB

DISCONNECTED
NOT BONDED

Generic Attribute
UUID: 0x1801
PRIMARY SERVICE

Generic Access
UUID: 0x1800
PRIMARY SERVICE

LED SERVICE
UUID: 5328b08a-0c4a-4bda-996c-41064568e694
PRIMARY SERVICE

LED CHRC
UUID: 9488b3b2-7de5-4a1c-af41-385a89931332
Properties: WRITE
Value: (0x) 00

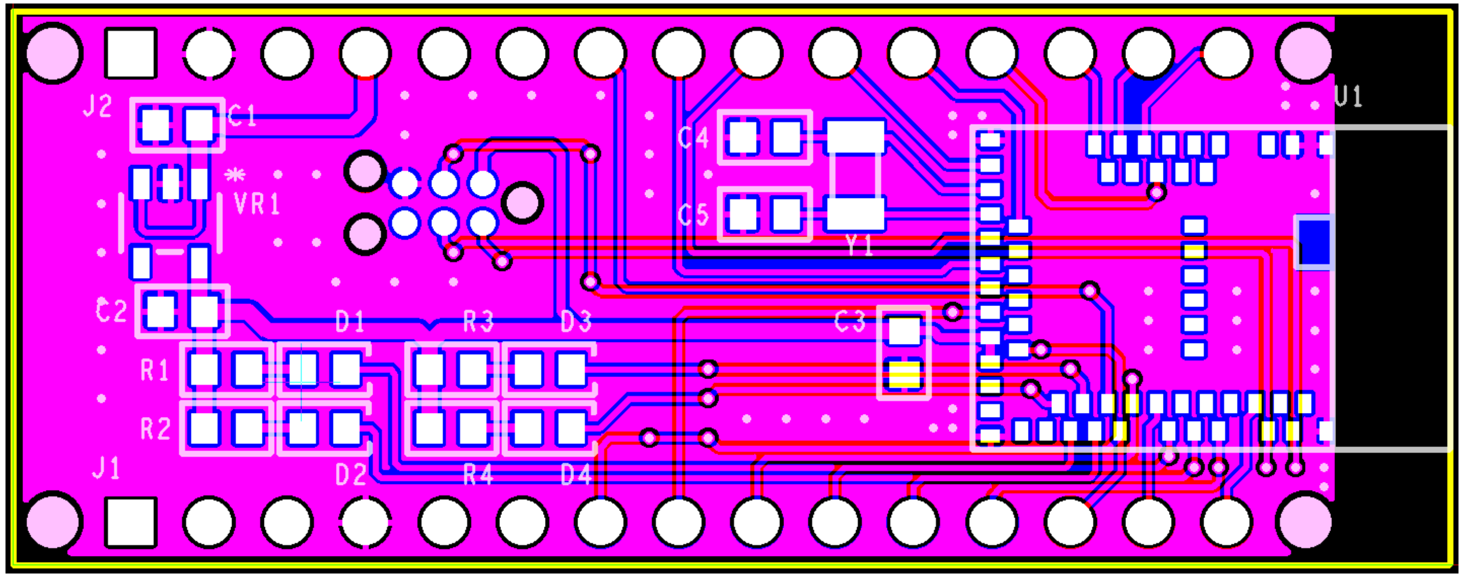
SENSOR SERVICE
UUID: 97252dff-72ca-44ea-910b-82864cae65f3
PRIMARY SERVICE

SENSOR CHRC
UUID: a64a76ef-d522-4783-9f46-bfdf95f4510d
Properties: READ
Value: (0x) 32-03

Next Time…

MORE TO COME..

Thank you for attending!!!



Please consider the resources below:

- Today's Download Package
- nordicsemi.com
- nRF52840 User Guide
- raytac.com

CEC Continuing Education Center

DesignNews

Thank You

Sponsored by

DigiKey

informa markets