

Mobile Computing Bluetooth Low Energy on Microcontrollers

CC BY-SA, T. Amberg, FHNW

Slides: tmb.gr/mc-per

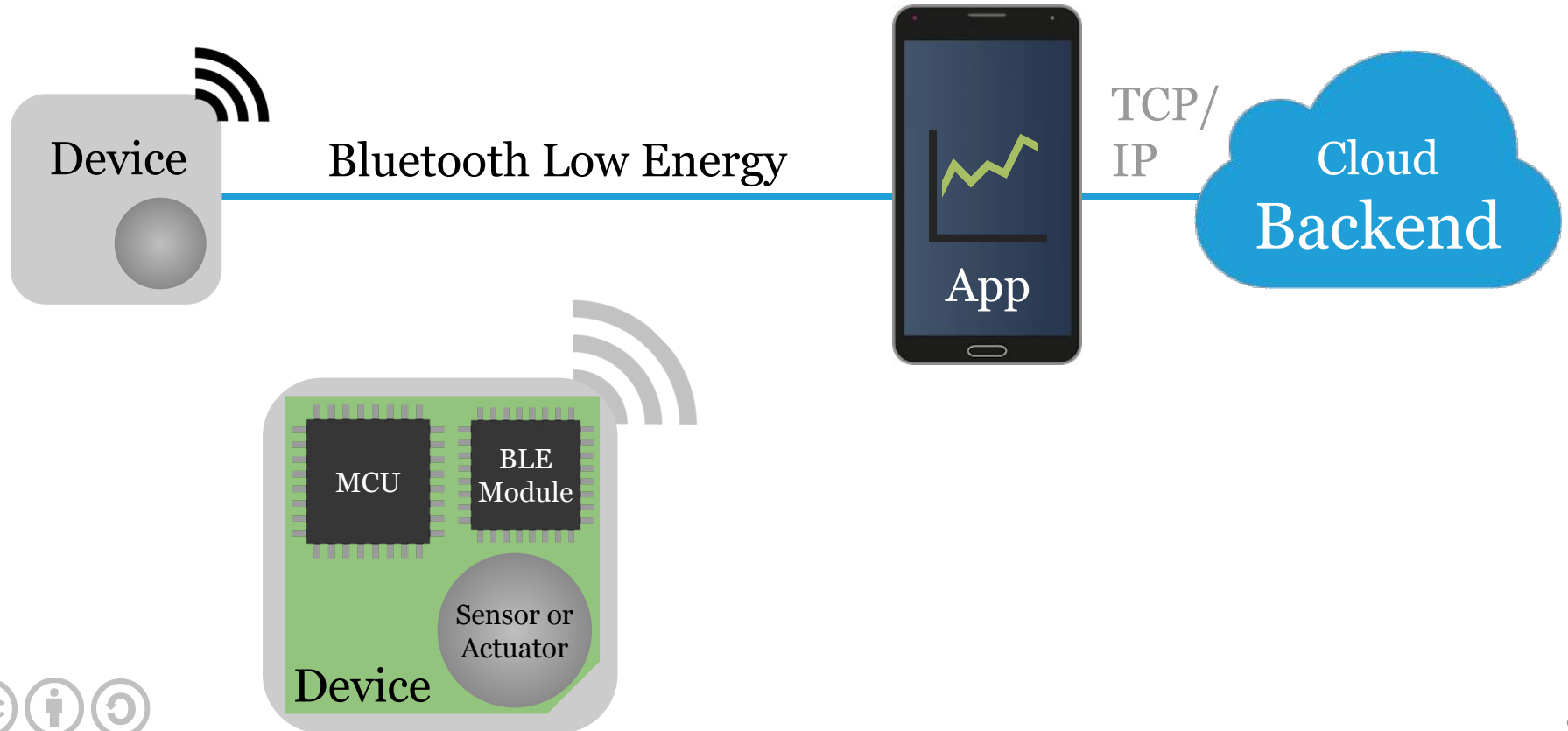
Overview

These slides introduce *BLE on microcontrollers*.

Examples for the peripheral and central roles.

Designing BLE services and characteristics.

Reference model



Prerequisites

Install the Arduino IDE and set up the nRF52840:

Check the Wiki entry on [Installing the Arduino IDE](#).

[Set up the Feather nRF52840 Sense](#) with Arduino.

Setting up the board also installs this [BLE library](#).

For testing, a smartphone with BLE is required.

BLE on the nRF52840

The nRF52840 can take the peripheral or central role.

The [Adafruit BLE library source code](#) and [examples](#) provide some "documentation" and a starting point.

To implement or use a peripheral, we have to know its API, consisting of service and characteristic UUIDs.

Heart rate service

This service is intended for fitness heart rate sensors:

Heart Rate Service UUID (16-bit): 0x**180D**

This service includes the following characteristics:

Heart Rate Measurement UUID: 0x**2A37** [N]

Body Sensor Location UUID: 0x**2A38** [R]

Heart Rate Control Point UUID: 0x**2A39** [W]*

*See also **Heart Rate Service** specification.

nRF52840 HRM BLE peripheral [.ino](#)

```
hrmSvc = BLEService(0x180D); // See HRM spec
hrmChr = BLECharacteristic(0x2A37); // See spec

hrmSvc.begin(); // to add characteristics
hrmChr.setProperties(CHR_PROPS_NOTIFY); ...
hrmChr.begin(); // adds characteristic

uint8_t hrmData[2] = { 0b00000110, value };
hrmChr.notify(hrmData, sizeof(hrmData));
```

Hands-on, 10': HRM BLE peripheral

Build and run the previous nRF52840 BLE example.

Use the *.ino* link on the page to get the example code.

Explore the HRM example using a smartphone app*.

Try to enable notifications to get value updates.

*Try [nRF Connect for Android](#) or [iOS](#).

nRF52840 HRM BLE central

.ino

```
BLEClientService hrmSvc(UUID16_SVC_HEART_RATE);  
BLEClientCharacteristic hrmChr(UUID16_CHR_...);  
  
// part of setup()  
Bluefruit.begin(0, 1); // 1 central connection  
hrmSvc.begin();  
hrmChr.setNotifyCallback(notifyCbck);  
hrmChr.begin(); // implicitly added to service  
Bluefruit.Central.setConnectCallback(connCbck);
```

nRF52840 HRM BLE central (ff.)

.ino

```
void connCbck(uint16_t connHandle) {  
    if (hrmSvc.discover(connHandle)) {  
        if (hrmChr.discover()) {  
            hrmChr.enableNotify();  
        } else { ... }  
    } else {  
        Bluefruit.disconnect(connHandle);  
    }  
}
```

nRF52840 HRM BLE central (ff.) [.ino](#)

```
Bluefruit.Scanner.setRxCallback(scanCbck);  
Bluefruit.Scanner.filterUuid(hrmSvc.uuid);  
Bluefruit.Scanner.restartOnDisconnect(true);  
Bluefruit.Scanner.start(0); // non-stop  
  
void scanCbck(ble_gap_..._report_t* report) {  
    // optional: check for device address  
    Bluefruit.Central.connect(report);  
}
```

Hands-on, 10': HRM BLE central

Build and run the previous nRF52840 BLE example.

Use the *.ino* link on the page to get the example code.

Open the Arduino serial monitor to enter a message.

Use a second nRF52840* as a HRM peripheral.

*Or your smartphone as a [peripheral simulator](#).

Nordic UART service

This service provides a serial connection over BLE:

Nordic UART Service custom (128-bit) UUID:
0x6E400001-B5A3-F393-E0A9-E50E24DCCA9E

This service includes the following characteristics:

RX (device receives data) UUID: 0x**0002** [W]

TX (device transmits data) UUID: 0x**0003** [N]

This service is becoming a de facto standard.

nRF52840 UART BLE peripheral [.ino](#)

```
// UUID: 6E400001-B5A3-F393-E0A9-E50E24DCCA9E
uint8_t const uartSvcUuid[] = { 0x9E, 0xCA, ...,
0xB5, 0x01, 0x00, 0x40, 0x6E }; // lsb first

uartSvc = BLEService(uartSvcUuid); // 128-bit
rxChr = BLECharacteristic(rxChrUuid); // 128-b.
txChr = BLECharacteristic(txChrUuid); // 128-b.

txChar.setProperties(CHR_PROPS_NOTIFY);
rxChar.setProperties(CHR_PROPS_WRITE);
```

Hands-on, 10': UART BLE peripheral

Build and run the previous nRF52840 BLE example.

Use the *.ino* link on the page to get the example code.

Write bytes to *RX* with a generic BLE explorer app.

Check the serial monitor to see the received bytes*.

*Why do some bytes not show up?

nRF52840 UART BLE central

.ino

```
Bluefruit.begin(0, 1); // 1 central connection
uartSvcClient.begin();
uartSvcClient.setRxCallback(rxCbck); // read
Bluefruit.Central.setConnectCallback(connCbck);

void connCbck(uint16_t connHandle) {
    if (uartSvcClient.discover(connHandle)) {
        uartSvcClient.enableTXD(); // enable notify
        uartServiceClient.print(...); // write data
        ... } }
```


nRF52840 UART BLE central (ff.) [.ino](#)

```
Bluefruit.Scanner.setRxCallback(found);

void found(ble_gap_evt_adv_report_t* report) {
    if (...Scanner.checkReportForService(
        report, uartServiceClient)) {
        Bluefruit.Central.connect(report);
    } else {
        Bluefruit.Scanner.resume();
    }
}
```

Hands-on, 10': UART BLE central

Build and run the previous nRF52840 BLE example.

Use the *.ino* link on the page to get the example code.

Open the Arduino serial monitor to enter a message.

Use a second nRF52840 as a UART peripheral.

nRF52840 beacon BLE observable `.ino`

```
BLEBeacon beacon(  
    beaconUuid, // AirLocate UUID  
    beaconMajorVersion,  
    beaconMinorVersion,  
    rssAtOneMeter);  
beacon.setManufacturer(0x004C); // Apple  
startAdvertising();  
  
suspendLoop(); // save power
```

nRF52840 scanner BLE central

.ino

```
Bluefruit.begin(0, 1); // Central
Bluefruit.Scanner.setRxCallback(found);
Bluefruit.Scanner.start(0);

void found(ble_gap_evt_adv_report_t* report) {
    Serial.printBufferReverse( // little endian
        report->peer_addr.addr, 6, ':');
    if (Bluefruit.Scanner.checkReportForUuid(...))...
    Bluefruit.Scanner.resume();
}
```

Hands-on, 10': Scanner BLE central

Build and run the previous nRF52840 BLE examples.

Use the *.ino* link on the page to get the example code.

Test the scanner with a (simulated) HRM peripheral.

Adapt the scanner to scan for the beacon observable.

Bonus: Scan for Covid-19 apps as described [here](#).

Summary

The nRF52840 can take the peripheral or central role.

To build/use a service we need its 16-/128-bit UUID.

Peripherals set up services, update characteristics.

Centrals connect to read, write or get notifications.

The specific value format depends on the service.

Challenge, 1h+

Design and implement an API for the **SHT30 sensor**.

Create UUIDs for your service and its characteristics.

Chose a data format that fits the sensor value range.

Consider to allow reading, writing or notifications.

Test your peripheral with a generic BLE explorer.

Feedback or questions?

Join us on [MSE TSM MobCom](#) in MS Teams

Or email thomas.amberg@fhnw.ch

Thanks for your time.