Table of Contents

1. Bluetooth Low Energy for STM32 - peripheral read 2
1.1. ButtonLED example - reading the characteristic data 2
1.2. Tasks to do during the laboratory 3

1. Bluetooth Low Energy for STM32 - peripheral read

In the previous example, a simple peripheral device program has been shown. The program allowed the central device to write data to the chosen characteristic and control the LEDs. In this part, we will read the characteristic data. The example available for STM32WB55 in Arduino in addition to controlling LED, allows also to read the state of the button.
The simplest method to get the information from a connected peripheral device is to read the characteristic. BLE specification defines also the notification and indication mechanisms to enable automatic updating of the information in which the central device is interested. These methods of data transmission work similarly to interrupts in microprocessors. The client subscribes to the data by enabling notification or indication. Data is sent only if the peripheral decides to do it, periodically or on update. This saves the network use and energy. The notification is not acknowledged, the indication must be acknowledged by the central device. Both mechanisms are controlled with the characteristic descriptor with UUID 0x2902.

- Notification is controlled with bit 0 (0-disable, 1-enable).
- Indication is controlled with bit 1 (0-disable, 1-enable).

1.1. ButtonLED example - reading the characteristic data

If the characteristic enables data read, the central device can retrieve information from the peripheral. in the ButtonLED example the “buttonCharacteristic” was defined which enables reading and notification.

```
// create button characteristic and allow remote device to get notifications
BLEByteCharacteristic buttonCharacteristic("19B10012-E8F2-537E-4F6C-
D104768A1214", BLERead | BLENotify);
```

Let's consider reading first. The program does not differ much from the “LED” example. The definition of button pin is added at the beginning of the “setup()” function.

```
pinMode(buttonPin, INPUT_PULLUP); // use button pin as an input
```

The program uses two characteristics, one for LED, and one for button. Both are elements of the same service.

```
  // add the characteristics to the service
  ledService.addCharacteristic(ledCharacteristic);
  ledService.addCharacteristic(buttonCharacteristic);

  // add the service
  BLE.addService(ledService);
```

In the main “loop()” the program reads the current state of the button, checks if it differs from the previous read and if so updates LED and button characteristics' values. Updating the value of the LED

characteristic automatically changes the LED state, so you can observe turning the LED on and off with the use of the button.

```
  // read the current button pin state
  char buttonValue = digitalRead(buttonPin);

  // has the value changed since the last read
  bool buttonChanged = (buttonCharacteristic.value() != buttonValue);

  if (buttonChanged) {
    // button state changed, update characteristics
    ledCharacteristic.writeValue(buttonValue);
    buttonCharacteristic.writeValue(buttonValue);
  }
```

While the notification is enabled by the central device, the updated value will be sent to the central automatically. The same occurs if the central device enables indication. This can be done if the characteristic property is properly set.

```
BLEByteCharacteristic buttonCharacteristic("19B10012-E8F2-537E-4F6C-
D104768A1214", BLERead | BLEIndicate);
```

1.2. Tasks to do during the laboratory

1. Import LEDbutton example, compile and upload to Nucleo board.
2. Observe the behaviour of the notification mechanism.
3. Modify the button characteristic's property to enable indication.
4. Add more characteristics, and code to handle other buttons.

From:
https://dokuwiki.zmitac.aei.polsl.pl/ - **RAU6 Dokuwiki**

Permanent link:
https://dokuwiki.zmitac.aei.polsl.pl/en:edu:zmitac:s2:iotn:ble2

Last update: **2023/12/17 20:57**