Table of Contents

1. ESP32 BLE in Arduino. Part 3: BLE Server with notify/indicate 2
1.1. Prerequisites 2
1.2. Suggested Readings and Knowledge Resources 2
1.3. Hands-on Lab Scenario 2
1.3.1. Task to be implemented 2
1.3.2. Start 2
1.3.3. Steps 3
1.3.3.1. Step 1 3
1.3.3.2. Step 2 4
1.3.4. Result validation 4
1.4. FAQ 4
1.5. Project information 5

1. ESP32 BLE in Arduino. Part 3: BLE Server with notify/indicate

This scenario presents how to extend the Bluetooth Low Energy server device with a notification/indication mechanism for sending data. If enabled notifications or indications are sent at any time while the data in the server is updated. A difference between them is that a notification is an unacknowledged message while an indication is an acknowledged message. While one of them is enabled by the client, the server decides on the time of the message sent.

1.1. Prerequisites

It is necessary to understand the principles of the Bluetooth Low Energy protocol with concepts of services, characteristics and descriptors. Notification and indication methods of data transmission should be known. We will use in this scenario the knowledge of the services and characteristics so making the part 2 exercise is recommended.

1.2. Suggested Readings and Knowledge Resources

None at the time.

1.3. Hands-on Lab Scenario

This scenario is intended to be implemented using actual equipment on-site. Although it is possible to make it remotely, with the use of two BLE laboratory nodes, to observe all the details you need to have your mobile phone or similar device with an application which allows you to scan and explore your Bluetooth Low Energy devices and communicate with them. One of the best applications is nRF Connect created by Nordic Semiconductors.

1.3.1. Task to be implemented

Implement a program that operates as the BLE server which allows us to read the data with a notification/indication mechanism.

1.3.2. Start

A similar program is available as the software example for the BLE server device in the ESP32 BLE Arduino package. You can use this example or the server program from part 2 as the starting point.

1.3.3. Steps

We will pass through the lab in a few steps. We will add the second characteristic to the example from lab part 2. We will configure this characteristic as notify/indicate capable. It allows us to establish a connection and, if successfully connected, enable the indication or notification feature. With this feature enabled peripheral device initiates data transmission. It can be used for the periodic measurement reading or updating information on value change.

1.3.3.1. Step 1

We start with the program written during the laboratory part 2 by including the BLE2902.h library which handles the Client Characteristic Configuration Descriptor (CCCD). The descriptor of this type is used to enable or disable the notification and indication feature.

Descriptor UUID	Bits 1 and 0	CCCD value	Function
0x2902	00	0	notify/indicate disabled
	01	1	enable notification
	10	2	enable indication

```
#include "BLE2902.h"
```

Next, we add another characteristic to the service and variable required to hold the pointer to the characteristic class. We need to define the UUID for this characteristic.

```
BLECharacteristic  *pNotifyCharacteristic; //class for the characteristic
#define NOTIFY_CHARACTERISTIC_UUID "6e9b7b28-ca96-4774-b056-8ec5b759fd86"
```

We create an additional characteristic with reading, writing, notify and indicate enabled, and the initial text “00” as a placeholder for the data. We also add the descriptor to the characteristic.

```
// Create a BLE Characteristic
pNotifyCharacteristic = pService->createCharacteristic(
                      NOTIFY_CHARACTERISTIC_UUID,
                      BLECharacteristic::PROPERTY_READ   |
                      BLECharacteristic::PROPERTY_WRITE  |
                      BLECharacteristic::PROPERTY_NOTIFY |
                      BLECharacteristic::PROPERTY_INDICATE
                    );
pNotifyCharacteristic->setValue("0");
// Create a BLE Descriptor
pNotifyCharacteristic->addDescriptor(new BLE2902());
```



Any characteristic can be configured as notify or indicate enabled. We can enable this mechanism for more than one characteristic.

After these modifications, it would be possible to observe an additional characteristic in the service with the possibility of reading, writing, and enabling notification and indication.

1.3.3.2. Step 2

At this step, we implement periodical data sending. We add two variables, an integer for holding the value incremented every loop pass and a text value converted from an integer to be sent within the indication/notification packet.

```
int int_value=0;
char text_value[] = "000000";
```

In the loop() function we modify the part executed if the connection was established. We'll add conversion from integer to text with atoi(), an update of characteristic value, and an incrementation of integer value.

```
void loop(){
    if (!deviceConnected && !advStarted) {
        pServer->startAdvertising(); // restart advertising
        advStarted = true;
    }

    if (deviceConnected){
        advStarted = false;
        itoa(int_value,text_value,10);
        pNotifyCharacteristic->setValue(text_value);
        pNotifyCharacteristic->notify();
        int_value++;
    }

    delay(500);
};
```

1.3.4. Result validation

You should be able to find the device in the BLE neighbourhood. It should allow us to establish a connection, read the characteristic data and write a new value. After enabling the notification we will observe periodic incrementation of the value sent with data packets.

1.4. FAQ

What is the difference between notification and indication?: As it was mentioned in the beginning the notification is an unacknowledged message while an indication is an acknowledged message. It means that if the indication message remains unacknowledged, the peripheral device won't send another message for this characteristic.

Is the notification/indication mechanism similar to interrupts?: You are right. They work in a similar manner as interrupts in the microprocessor. They allow to sending of the data asynchronously, and the peripheral decides about the time of the message sent like the peripheral decides on the time of signalling interrupt. Data transmission when the central device sends the packet with a

characteristic read command is similar to the polling method used in microprocessors.

<WRAP noprint>

1.5. Project information

This Intellectual Output was implemented under the Erasmus+ KA2.
Project IOT-OPEN.EU Reloaded – Education-based strengthening of the European universities, companies and labour force in the global IoT market.
Project number: 2022-1-PL01-KA220-HED-000085090.

Erasmus+ Disclaimer
This project has been funded with support from the European Commission.
This publication reflects the views of only the author, and the Commission cannot be held responsible for any use that may be made of the information contained therein.

Copyright Notice
This content was created by the IOT-OPEN.EU Reloaded consortium, 2022,2024.
The content is Copyrighted and distributed under CC BY-NC Creative Commons Licence, free for Non-Commercial use.

</WRAP>

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

Permanent link:
https://dokuwiki.zmitac.aei.polsl.pl/en:edu:zmitac:s2:iotn:esp32ble3

Last update: **2024/04/09 09:01**