Table of Contents

1. ESP32 BLE in Arduino. Part1: Setting-up BLE Beacon 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
1.3.3.3. Step 3 4
1.3.3.4. Step 4 5
1.3.4. Result validation 6
1.3.5. Further work 6
1.4. FAQ 6
1.5. Project information 7

1. ESP32 BLE in Arduino. Part1: Setting-up BLE Beacon

This scenario presents how to create the Bluetooth Low Energy beacon device, which broadcasts periodically a small amount of information. Beacons are usually used for sending useful information (eg. the web address of the owner, a link to the page with tourist information). In some cases, they simply send the identification number recognised by a dedicated mobile application allowing the users to localise themselves.



BLE Beacons just broadcast the information, they are not capable of receiving any information from the users nearby.

1.1. Prerequisites

We will use the advertising process in this scenario so an understanding of the principles of the Bluetooth Low Energy protocol is necessary. It's good to understand the Eddystone data format used to encode URLs in advertising frames.

1.2. Suggested Readings and Knowledge Resources

- Interesting article on Beacons with description of Eddystone, and iBeacon packets[1)]

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 beacon which advertises itself with the link to the SUT website. This can be done with the Eddystone beacon format.

1.3.2. Start

The example software for the beacon device available in the software package differs between

versions of the environment, that's why we will write the program from scratch. But even in such a situation, it is much easier to start by using some simple example. We can use the “arduino-blink” example to have the project structure ready with basic source and configuration files.
The platformio.ini file can look like in the following example.

platformio.ini

```
[env:esp32]
platform = espressif32
board = esp32-s3-devkitc-1
board_build.mcu = esp32s3
board_build.f_cpu = 240000000L
framework = arduino
platform_packages =
    toolchain-riscv32-esp @ 8.4.0+2021r2-patch5
lib_ldf_mode = deep+
```

1.3.3. Steps

We can go through the lab in a few steps. In the beginning, we will write a simple program which advertises the device with the default parameters set in the software package. After a successful first attempt, we will add our information to the advertising frame, and lastly, we will use the Eddystone format to send the chosen URL address.

1.3.3.1. Step 1

Let's begin with a very simple program using default advertising. The code should start with including Arduino and BLE libraries.

```
#include <Arduino.h>
#include "BLEDevice.h"
#include "BLEAdvertising.h"
```

We need one variable only which holds the pointer to the advertising class.

```
BLEAdvertising *pAdvertising;
```

The setup function creates and initialises the BLE device instance with a default name “ESP32”. While the device is ready we can get the pointer to the class and use it to start advertising with a default set of parameters.

```
void setup(){
BLEDevice::init("");
pAdvertising = BLEDevice::getAdvertising();
pAdvertising->start();
};
```

Note that the loop function can remain empty.

```
void loop(){
};
```

1.3.3.2. Step 2

Let us now introduce some of our information. Sure, if you create the device you would like it to present with the name of your choice. If we add our text in the BLEDevice::init() function we will see this text as the device name in the advertising frame. It will also appear as the value of the Device Name characteristic in the Generic Access service.

```
BLEDevice::init("SUT BLE device");
```

The advertisement frame is composed of fields. Every field starts with its length (1 byte). The second byte of the field encodes the field type. The selected field codes used in the default advertising frame defined in the ESP32 BLE library are presented in the table below.

Field length	Field code	Field type	Default value
0x02	0x01	Flags	0x06
0x05	0x12	Peripheral Connection Interval Range	0x20004000
0x06	0x09	Complete Local Name	“ESP32”
0x02	0x0A	Tx Power Level	0x09

1.3.3.3. Step 3

Except for the text information as the device name, it is possible to send the web address which directs us to additional information. Google invented the Eddystone format, used in the advertisement frames simplifying the URL encoding. It requires the presence of the field (code 0x03) which shows the list of 16-bit UUIDs with 0xAAFE value, and the service data field (code 0x16) compatible with Eddystone format. Additional fields are shown in the table below.

Field length	Field code	Field type	Default value
3	0x03	Complete list of 16-bit service UUIDs	0xAAFE
14	0x16	Service data	Eddystone format

The Eddystone field starts with service UUID (0xAAFE), next specifies the content type (0x10 for URL), then the transmitting power in [dBm], and then the compressed URL. To compress URLs some default prefixes and suffixes were defined as non-ASCII characters. They are presented in the tables below.

Decimal	Hex	Prefix
0	0x00	http://www.
1	0x01	https://www.
2	0x02	http://
3	0x03	https://

Decimal	Hex	Suffix
0	0x00	.com/
1	0x01	.org/

Decimal	Hex	Suffix
2	0x02	.edu/
3	0x03	.net/
4	0x04	.info/
5	0x05	.biz/
6	0x06	.gov/
7	0x07	.com
8	0x08	.org
9	0x09	.edu
10	0x0a	.net
11	0x0b	.info
12	0x0c	.biz
13	0x0d	.gov

The Eddystone payload can be created as the std::string class object as required by the function addData() from the BLEAdvertisementData class. So first we need to declare the BLEAdvertismentData class object, and string object with the placeholder for the payload.

```
BLEAdvertisementData oAdvertisementData = BLEAdvertisementData();
std::string eddystone_content ("              polsl.pl"); //14 spaces for other fields
```

In the code, we need to fill in the empty spaces at the beginning of the advertising packet.

```
eddystone_content[0]  = 2;      //Length of flags field
eddystone_content[1]  = 0x01;   //Flags ID
eddystone_content[2]  = 0x06;   //Flags value
eddystone_content[3]  = 3;      //Length of service field
eddystone_content[4]  = 0x03;   //List of 16-bit UUIDs
eddystone_content[5]  = 0xAA;   //UUID of Google Eddystone
eddystone_content[6]  = 0xFE;
eddystone_content[7]  = 14;     //length of Eddystone field
eddystone_content[8]  = 0x16;   //Service data
eddystone_content[9]  = 0xAA;   //Eddystone UUID
eddystone_content[10] = 0xFE;
eddystone_content[11] = 0x10;   //URL Eddystone frame type
eddystone_content[12] = 0xF4;   //Tx power [dBm] (-12dBm)
eddystone_content[13] = 0x00;   //prefix: "http://www."
```

The created payload can be used as the payload data of the advertising packet.

```
oAdvertisementData.addData(eddystone_content);
pAdvertising->setAdvertisementData(oAdvertisementData);
```

1.3.3.4. Step 4

The advertisement frame is quite short so normally additional information is sent upon Scan Request with Scan Response frame. It can contain the name of the device. Notice that it can be a different

name than the device name which we can read as the value of the Generic Access Service (as described in step 2). To specify different names in advertising frame and service characteristics we can use the setScanResponseData() function from BLEAdvertising class.

```
oScanResponseData.setName("SUT Beacon");
pAdvertising->setScanResponseData(oScanResponseData);
```



If the text used as an advertisement (or scan response) name is too long the BLE library automatically empties the text. If the device name is too long it is replaced with the “ESP32” name.

1.3.4. Result validation

You should be able to observe the content of the advertising frame in nRF Connect or a similar application. You should be able to navigate to the URL sent in the advertising.

1.3.5. Further work

You can try the example software for the beacon device which is available in the examples at the path:

```
C:\..\.platformio\packages\framework-
arduinoespressif32\libraries\BLE\examples\BLE_EddystoneURL_Beacon
```

It is more complex but enables automatic determination of packet length depending on the URL provided.
You can also try to implement the beacon device compatible with iBeacon. The description can be found on the website [2)].

1.4. FAQ

What is the maximum length of the advertising frame?: It depends on the BLE version. It used to be 31 bytes only, but in newer versions, even 255 bytes are allowed.
How often should the device send advertising frames?: The delay between advertising frames can vary depending on the device. Beacon devices should send them every 100-200 ms. Other devices usually advertise every 2 seconds. Notice that advertising consumes energy, so the battery-powered devices advertise with lower frequency.
What if I need to send some info other than in Eddystone or iBeacon format?: You can send the text or even some data as part of the advertising frame. Some devices use the advertising frames to send simple 1-2 bytes of measurement data. More data can be sent after establishing the connection between the primary and secondary devices with the use of characteristics.
What is the purpose of the transmitting power data presence in the advertising frame?: The value sent as the transmitting power should represent real power at a 0-meter distance from the beacon. It can be used to estimate the actual distance of the user from the beacon. The distance

needs to be calculated by application on the mobile phone with knowledge of the transmitting power of the beacon and actual measurement of received signal strength (known as RSSI).

<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>

[1)] [2)]
https://www.silabs.com/whitepapers/developing-beacons-with-bluetooth-low-energy-technology

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

Permanent link:
https://dokuwiki.zmitac.aei.polsl.pl/en:edu:zmitac:s2:iotn:esp32ble1

Last update: **2024/04/03 12:51**