Table of Contents

1. Bluetooth Low Energy for STM32 - peripheral 2
1.1. Installing the library and importing the example 2
1.2. Description of the most important elements of the code 3
1.2.1. Variables and objects definition 3
1.2.2. Initialization 4
1.2.3. Loop function 5
1.3. Tasks to do during the laboratory 6

1. Bluetooth Low Energy for STM32 - peripheral

This laboratory shows the details of the example provided as part of the STM32duinoBLE library by Arduino.

1.1. Installing the library and importing the example

As the first step, we need to install the library to the current project (how to download and import the first STM32WB55 project is described here. We can choose the simplest blink project as the base for our BLE peripheral).

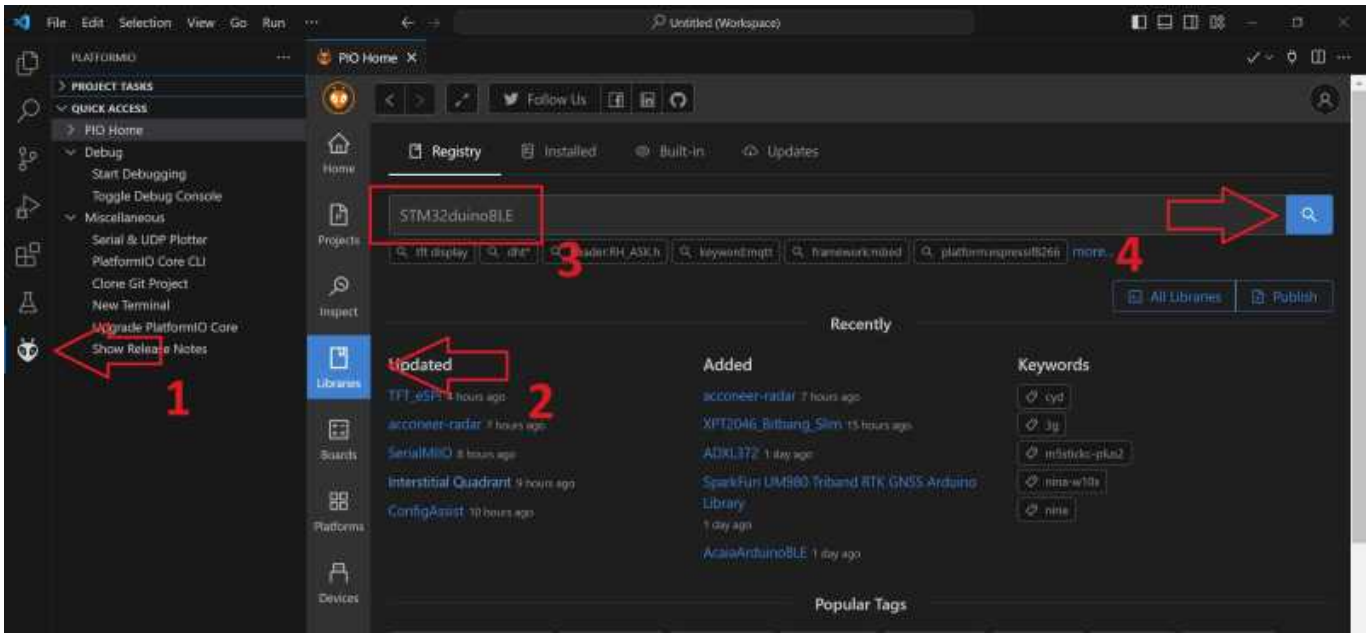


Fig. 1: Search for the BLE library for STM32 in VSCode/PlatformIO

After finding the library in the repository we can add it to our project. By clicking the “Add to Project” button you can select the already existing “arduino blink” project to add the library entry to the platformio.ini file automatically. It will create the dependency of your project build with the specific version of the STP32duinoBLE library. To use the code provided as the example, select the code of the “LED” example and paste it as the content of the “Blink.cpp” program chosen from the project tree.

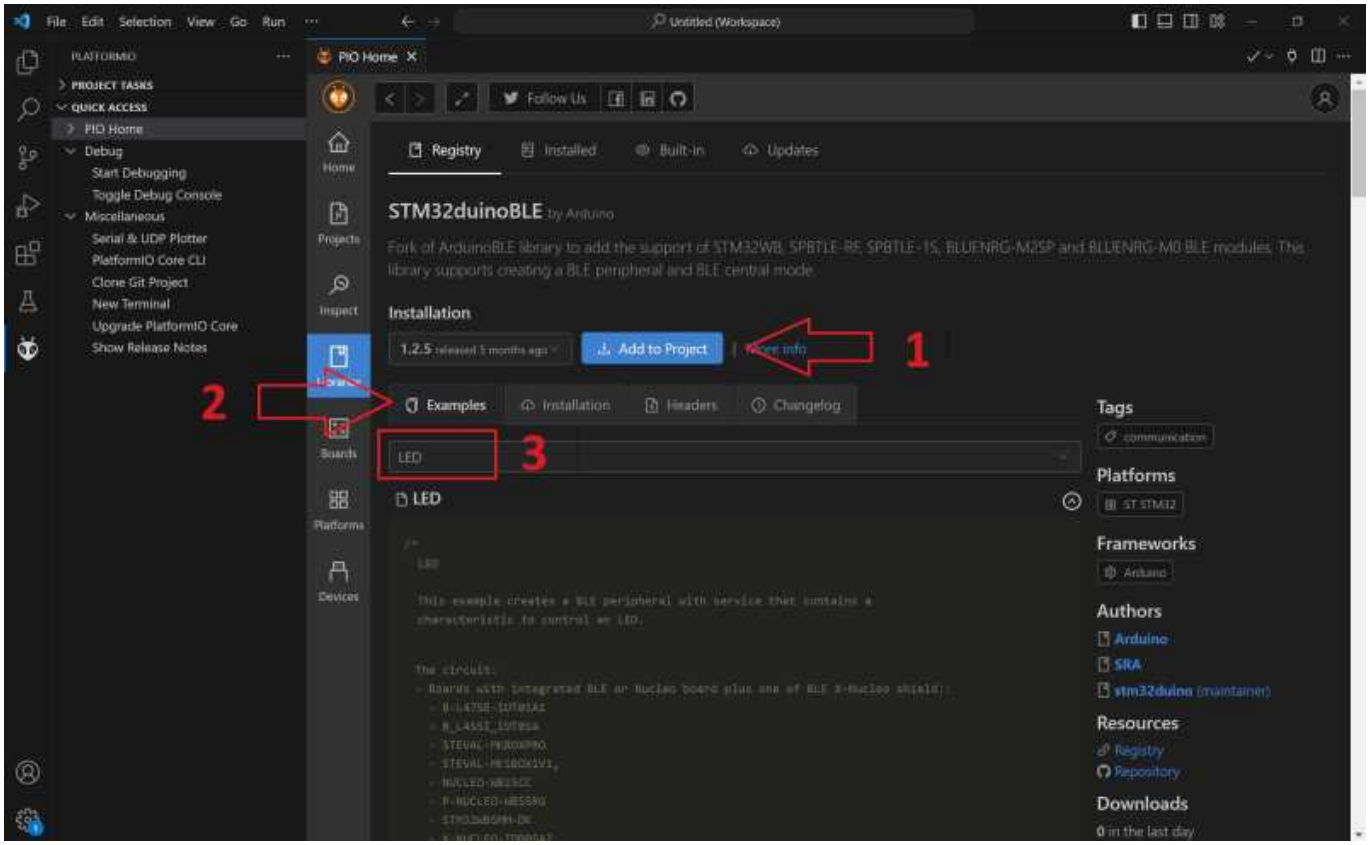


Fig. 2: Install the BLE library in the existing project

1.2. Description of the most important elements of the code

1.2.1. Variables and objects definition

To use the BLE device you need to create the object of the BLELocalDevice class. This object can work using the SPI-connected Bluetooth module or the built-in unit in SoC. In the case of STM32WB55, the internal unit is used, so the HCISharedMemTransportClass is chosen as the connection to the BLE module.

```
HCISharedMemTransportClass HCISharedMemTransport;
BLELocalDevice BLEObj(&HCISharedMemTransport);
BLELocalDevice& BLE = BLEObj;
```

After creating the BLE object, we can configure services, characteristics and advertising process. Let's define the service first. To use the service you define, you specify the 128-bit version of the UUID. You can also use predefined 16-bit services as listed in Bluetooth SIG documents.

```
BLEService ledService("19B10000-E8F2-537E-4F6C-D104768A1214"); // BLE LED
Service
```

Services usually contain characteristics, also identified by 128-bit or 16-bit UUIDs.

```
BLEByteCharacteristic switchCharacteristic("19B10001-E8F2-537E-4F6C-
D104768A1214", BLERead | BLEWrite);
```

The characteristic defined in the example allows remote devices to read or write byte values. It is also possible to define characteristics which contain the text string.

```
BLECharacteristic textCharacteristic("19B10001-E8F2-537E-4F6C-D104768A1215",
BLERead | BLEWrite, "sample text");
```

In the BLE specification, the maximal length of the characteristic's value is 512 bytes, the number of characteristics in one service is also limited to 512. You can experiment on how many characteristics can be used in Arduino implementation.
The example program allows the remote device to control the state of the LED, so there is also a declaration of the “ledPin” variable.

```
const int ledPin = LED_BUILTIN; // pin to use for the LED
```

1.2.2. Initialization

The BLE subsystem must be initialized before use. It is possible to check if initialisation is successful. This can be done once in the Arduino “setup()” function.

```
// begin initialization
 if (!BLE.begin()) {
   Serial.println("starting BLE failed!");

   while (1);
 }
```

After successful initialisation, you can specify the name of your device which appears while scanning the BLE neighbourhood. It is included in the advertising frames. Also, the service you defined can be added to the advertising information.

```
// set advertised local name and service UUID:
 BLE.setLocalName("LED");
 BLE.setAdvertisedService(ledService);
```



Arduino BLE implementation limits the number of advertised services to one.

The next step is to create a list of services and characteristics.

```
 // add the characteristic to the service
 ledService.addCharacteristic(switchCharacteristic);

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

The order is important, first, you need to add a characteristic (or more characteristics) to the service, next, add the service to the list of services.



The number and list of services available after establishing the connection can differ from those advertised.

Before starting the advertising process it is possible to set the initial value of the characteristic.

```
  // set the initial value for the characeristic:
  switchCharacteristic.writeValue(0);

  // start advertising
  BLE.advertise();
```

Also LED output is initialized in the setup() function.

```
  // set LED pin to output mode
  pinMode(ledPin, OUTPUT);
```

1.2.3. Loop function

As in all Arduino programs, after the initialization, the program starts to execute an infinite “loop()” function. First, it calls the “BLE.central()” function that specifies the role of our device as peripheral (server), and checks if there is any central (client) connected.

```
  BLEDevice central = BLE.central();
```



BLE.central() function is non-blocking so you can execute other code if the connection is not established.

To check if there is an active connection the state of “central” variable can be checked. While it is not NULL the program prints the MAC address of the remote device and jumps into the while loop to handle write events requested by the central device. The while loop is executed until the remote device decides to close the connection. If the remote device writes a new value to the characteristic, the “written” property signals this situation. While it is TRUE we can read the value. In the example value “0” turns the LED off, while the other value sets the LED on.

```
// if a central is connected to peripheral:
  if (central) {
    Serial.print("Connected to central: ");
    // print the central's MAC address:
    Serial.println(central.address());

    // while the central is still connected to peripheral:
    while (central.connected()) {
      // if the remote device wrote to the characteristic,
      // use the value to control the LED:
      if (switchCharacteristic.written()) {
        if (switchCharacteristic.value()) {   // any value other than 0
```

```
          Serial.println("LED on");
          digitalWrite(ledPin, HIGH);          // will turn the LED on
        } else {                               // a 0 value
          Serial.println(F("LED off"));
          digitalWrite(ledPin, LOW);           // will turn the LED off
        }
      }
    }

    // when the central disconnects, print it out:
    Serial.print(F("Disconnected from central: "));
    Serial.println(central.address());
  }
```



F(“text”) used in the Serial.print() function means that the text will be always taken directly from Flash memory. It frees some of the RAM during the program execution.

1.3. Tasks to do during the laboratory

1. Import the Arduino blink example.
2. Add STM32duinoBLE library to the project.
3. Replace the original blink code with the LED example, compile and execute.
4. Add characteristics to existing service for controlling other LEDs on the Nucleo board.
5. Add service and the characteristic to display texts (using serial port).

Reading text value can be done with memcpy() function.

```
char buffer[10];
memcpy(buffer, textcharacteristic.value(),
textcharacteristic.valueLength());
```



Nucleo board has three LEDs: red, green, and blue named LED_RED, LED_GREEN, and LED_BLUE. The blue LED is also named LED_BUILTIN or LED1, the green LED is also named LED2, and the red LED is also named LED3.

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

Permanent link:
https://dokuwiki.zmitac.aei.polsl.pl/en:edu:zmitac:s2:iotn:ble1?rev=1702751141

Last update: **2023/12/16 19:25**