**Battery Level BLE Device Using Potentiometer**

**Emre Çifçi**

CONTENTS

**1-Introductıon 2**

1.1-AIM 2

**2-ABSTRACT 2**

2.1-WHAT IS BLE? 2

2.2-BLE HIERARCHY AND GATT 2

2.3-BATTERY LEVEL PROFILE, SERVİCE, CHARACTERISTIC & DESCRIPTOR……………………………......................................3

2.4-MY CIRCUIT & THE CODE 3

**3-CONCLUSION 6**

3.1-WHAT HAVE WE DONE? 6

3.2-WHAT ELSE CAN BE IMPROVED? 6

3.3-REFERENCES 6

1. **Introduction**

**1.1 Aim**

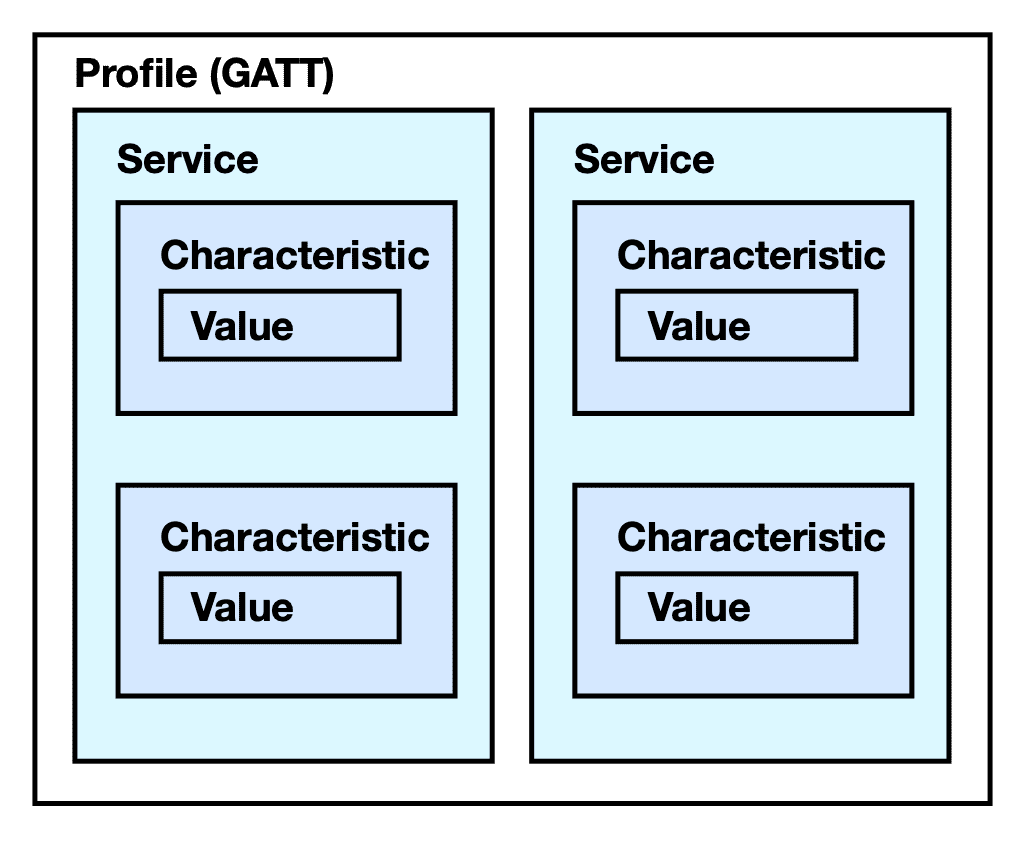
The main aim of this article is to introduce BLE technology to those who are not used to it and add a meaningful usage to our main device which is an intelligent stethoscope. For this, first of all, we need to introduce BLE technology, services, characterizations, descriptors etc. with its differences between classical Bluetooth. After all, we need to specify our Battery Level device and show our Arduino code with potentiometer circuit that will be discussed later in detail.

1. **Abstract**
   1. **What is BLE?**

“BLE (Bluetooth Low Energy) is a wireless personal area network technology designed and marketed by the Bluetooth Special Interest Group (Bluetooth SIG) aimed at novel applications in the healthcare, fitness, beacons, security and home entertainment industries.” (Wikipedia, 2021) The main difference between Classical Bluetooth and BLE is power consumption levels and cost. Even though BLE has lower data rate and distance relative to the classical one, the differences mentioned before make it worthy to use. One example to introduce its lower power consumption can be its sleep mode. “BLE operates only when a communication is activated.” (Raj, 2018)

A BLE device can be central (client) or peripheral (server). In principle, we can say that a client initiates commands and a peripheral receives and returns responds to them. In our design, the BLE device will be the peripheral and the mobile phone will be the client.

* 1. **BLE Hierarchy and GATT**

GATT (Generic Attribute Profile) is to define the transfer between BLE devices. It uses ATT (Attribute Protocol) to store services, characteristics, profiles and descriptors in 16-bit UUIDs. What is more, GATT provides BLE connections to be “exclusive”. This means that only one peripheral can be connected to a central device at the same time.

-Profile: Predefined collection of services.

-Service: Each one has one or more characteristics under it.

-Characteristic: Parameters that we would like to exchange, the main point of BLE peripheral connection. Its functions are “read”, “write”, “notify” and “indicate”. These functions are called the “properties” of characteristics.

-Descriptor: It specifies how to access the characteristic.

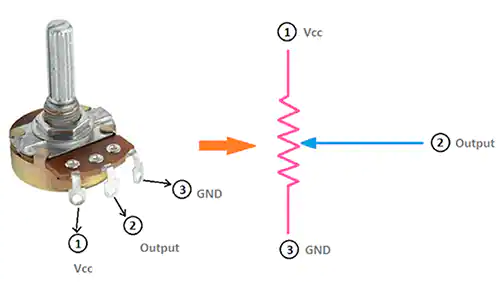
* 1. **Battery Level Profile, Service, Characteristic and Descriptor**

Service UUID: 0x180F > This service displays the battery level from 0% to 100%, represented as an 8 bit number.

Characteristic UUID: 0x2A19 > It has properties/functions ‘read’ and ‘notify’.

Descriptor UUID: 0x290 > As mentioned before, it specifies how to access the characteristic.

**2.4 My Circuit & The Code**



First of all, we have a very basic potentiometer circuit with Vcc 5V, ground and output which can be changed manually by the user. The main purpose is to scale this output voltage value according to 0-100 range to show the level of battery. To achieve this, the value that is read by using analogRead() command is multiplied by a constant. This constant is determined by first printing the unmodulated value. The resulting value was between 0 to 4097, this means it has 16 bits to represent the output of analogRead() function. If we multiply it by 5 and then divide it by 4097, the result corresponds to the 0-5 V range. But since we would like to achieve a 0-100 range, the final value is multiplied by 20. This part explains the void loop() part of the code:

void loop() {

value=analogRead(output);

value=value\*(100.0/4097.0);

percen=value;

if(value > 99.5){

percen=100;}

BatteryLevelCharacteristic.setValue(&percen, 1);

BatteryLevelCharacteristic.notify();

delay(1000);

}

Now, we can start to look at other parts of the code. First we need to add some libraries to use BLE technology in our code. Then to check whether a client is connected to our server or not, a Boolean variable called \_BLEClientConnected is used. Then we need to define battery service, characteristic and descriptor. To do that, predetermined16 bits UUIDs are used. These UUIDs are shown on a list called “assigned-values 16-bit UUID”. The link for this is shared below as a comment.

#include <BLEDevice.h>

#include <BLEUtils.h>

#include <BLEServer.h> //Library to use BLE as server

#include <BLE2902.h>

//Server is ESP32 and the client is our mobile phone

bool \_BLEClientConnected = false; //is not connected yet

#define BatteryService BLEUUID((uint16\_t)0x180F) //define battery service: 0x180F is predetermined ->

//https://btprodspecificationrefs.blob.core.windows.net/assigned-values/16-bit%20UUID%20Numbers%20Document.pdf

BLECharacteristic BatteryLevelCharacteristic(BLEUUID((uint16\_t)0x2A19), BLECharacteristic::PROPERTY\_READ |

BLECharacteristic::PROPERTY\_NOTIFY);

//define battery level characteristic and find its corresponding UUID using the same link above

//GATT Characteristic and Object Type 0x2A19 Battery Level

//this characteristic is represented with two properties: read and notify

BLEDescriptor BatteryLevelDescriptor(BLEUUID((uint16\_t)0x2901));

//using the same link, but we could not enter the description page

//more explanation needed

Next, we need to add some variables and a call back class to ensure the connection between the client and the server is successful:

class MyServerCallbacks : public BLEServerCallbacks {

void onConnect(BLEServer\* pServer) {

\_BLEClientConnected = true;

};

void onDisconnect(BLEServer\* pServer) {

\_BLEClientConnected = false;

}

};

int output=15;

float value;

float value2;

uint8\_t percen;

//smoothing process can be needed later

Finally, we need to explain the setup part of this code. Serial.begin() function tells ESP32 board to get ready to exchange data at a data rate specified as 155200 bits per second in this code. BLEDevice::init() determines the name of the device. After this, our server is created. Then under this server, a service, a characteristic and descriptors are created or added.

void setup() {

Serial.begin(115200);

BLEDevice::init("BLE Battery"); //name of the Bluetooth device

BLEServer \*pServer = BLEDevice::createServer(); //create a server called "pServer"

BLEService \*pBattery= pServer->createService(BatteryService); //create a service called "pBattery" under the same server

pServer->setCallbacks(new MyServerCallbacks()); //set call backs for the server using the class above

pBattery->addCharacteristic(&BatteryLevelCharacteristic); //add characteristic defined at the very beginning of this code to the service

BatteryLevelDescriptor.setValue("Percentage 0 - 100"); //descriptor helps us to notify values as percentage in the range of 0 to 100

BatteryLevelCharacteristic.addDescriptor(&BatteryLevelDescriptor); //add this descriptor to the characteristic

BatteryLevelCharacteristic.addDescriptor(new BLE2902()); //"a piece of software on the client side that informs the server to turn notification On or Off"

pServer->getAdvertising()->addServiceUUID(BatteryService); //add service UUID before advertising is started

pBattery->start(); //start the service

pServer->getAdvertising()->start(); //start advertising and wait for a client to connect our server

}

1. **Conclusion**
   1. **What have we done?**

We have designed a simple circuit using a potentiometer. As we changed the resistance inside it, the resulting output voltage changed. We changed its range and we mapped it between 0-100 % where 100% corresponds to 5V. Then, using embedded code via ESP32, we wrote a BLE Code to transfer percentage data to any client. (example: mobile phone) This whole design can be progressed and then be used for other devices that need to show its live battery level using BLE technology. As mentioned before, for live data transmissions, Bluetooth seems like an admirable way. What is more, as we attach importance to lower the power consumption, BLE can be a better way than a classical Bluetooth.

**3.2 What else can be improved?**

One purpose that we could not achieve yet is to show the battery level of any connected device together with the battery of the client (mobile phone in this example) just like a live notification. There are some applications doing this. Alternatively, there are some ways to show devices as headsets or smartwatches. This is one of the “in progress” points of this design and can be improved later.

As the research about the topic called one of the “in progress” points progresses, some problems with some restrictions came out. Despite the fact that we have tried to show our BLE Battery Level device as a headset, it did not solve the problem, which means that we could not see the battery icon of a headset device on the status bar of the mobile phone. Yet, there were other alternatives that we could have tried on the table of UUIDs. There were lots of GATT services for different usages from cycling power to heart rate. Sadly, we could not get a better solution.

This problem might have occurred because of the capabilities of BLE. We can try different choices for a Serial BT device instead of using BLE just like other headsets. This is the new step that we choose as our new “in progress point” of this project.

* 1. **References**

Raj, A. (2018, October 23). *ESP32 BLE Server - GATT Service for Battery Level Indication*. Circuit Digest: https://circuitdigest.com/microcontroller-projects/esp32-ble-server-how-to-use-gatt-services-for-battery-level-indication

Wikipedia. (2021, July 17). *Bluetooth Low Energy*. Wikipedia: https://en.wikipedia.org/wiki/Bluetooth\_Low\_Energy