Bachelor's Thesis

Bachelor’s degree in Industrial Technology Engineering

Bluetooth Low Energy based on the nRF52840 USB dongle

REPORT

Author: Jordi Ribas Sobreviela
Director: Manuel Moreno Eguílaz
Call: September 2019

Escola Tècnica Superior d'Enginyeria Industrial de Barcelona
Abstract

The objective of this report is to explain the process of creating a system that receives data from a sensor, which is connected to a power source, and sends it to a computer without using wires. The technologies chosen to carry out the project have been the wireless protocol Bluetooth Low Energy and the nRF52840 USB Dongle from Nordic Semiconductor. Bluetooth Low Energy, shortened BLE, has the advantage of having very low energy consumption. For this reason, devices using BLE can run on simple batteries for very long periods of time.

At the end of this project, the Dongle, that will be attached to the sensor and powered by a power bank, will send the data coming from the sensor to a Raspberry Pi using BLE. To do this, two applications have been created and installed into the Dongle. The first application allows the Dongle to collect the data coming from the sensor using an analogical to digital converter. Then, this data is sent in packets and the Raspberry Pi displays it in a screen. The second application is an improvement of the first one. It allows the Dongle to perform multitasking and do some calculations on the data. The Raspberry displays the results and writes them in text files for their analysis.

The Dongle is programmed using C language and with the help of the Segger Embedded Studio software and the nRF Connect for Desktop application. The applications that the Dongle runs are a modification of one of the examples that are included in the nRF5 Software Development Kit, which can be downloaded directly from Nordic Semiconductor’s webpage. This example is a heart rate application that sends data via BLE. The programming on the Raspberry has been made using Python.

To test the applications, simulated data has been used in order to control if the data that the Dongle sent was the same that the Raspberry Pi received. When the applications passed the tests with the simulated data, the Dongle was connected to a power source that created a wave whose parameters were known. Using the analogical to digital converter, the Dongle acquired data from the wave and calculated its parameters. The results that the Raspberry Pi showed were not exactly the same as the expected ones, but the error was small enough to consider them a good approximation.
Summary

ABSTRACT 3

SUMMARY 4

1. GLOSSARY 7

2. INTRODUCTION 9
   2.1. Objectives of the project 9
   2.2. Scope of the project 9

3. BLUETOOTH LOW ENERGY 11
   3.1. History 11
   3.2. Compatibility with Bluetooth Classic 12
   3.3. Technical specifications 13
   3.4. Architecture 14
      3.4.1. Physical Layer (PHY) 14
      3.4.2. Link Layer 15
      3.4.3. Generic Access Profile (GAP) 16
      3.4.4. Generic Attribute Profile (GATT) 17

4. HARDWARE SETUP 20
   4.1. nRF52840 USB Dongle 21
   4.2. nRF52840 DK 22
   4.3. Raspberry Pi 2B 23

5. GENERIC PROCESS FOR CREATING A BLE APPLICATION 26
   5.1. nRF5 SDK 26
   5.2. Segger Embedded Studio (SES) 28
   5.3. nRF Connect for Desktop 30
      5.3.1. Bluetooth Low Energy application 30
      5.3.2. Programmer application 32

6. THE BLUETOOTH LOW ENERGY APPLICATIONS 35
   6.1. First application: sending raw data 36
      6.1.1. Analogical to digital conversion 37
      6.1.2. Software development 38
      6.1.3. Results 41
   6.2. Second application: parameter calculation 43
      6.2.1. Calculated parameters 43
6.2.2. Software development ................................................................. 44
6.2.3. Results .................................................................................... 46

7. ECONOMIC STUDY .................................................................. 52
  7.1. Cost of the equipment .............................................................. 52
  7.2. Cost of the development time .................................................. 54

8. ENVIRONMENTAL IMPACT ..................................................... 55
  8.1. Impact on people’s health ......................................................... 55
  8.2. Impact on the planet ............................................................... 55

CONCLUSIONS ............................................................................. 57

ACKNOWLEDGEMENTS .............................................................. 59

BIBLIOGRAPHY ............................................................................. 60
  Bibliographic references .............................................................. 60
  Consulted bibliography ............................................................... 63
1. Glossary

- **API**: Application Programming Interface

- **BC**: Bluetooth Classic

- **BLE**: Bluetooth Low Energy

- **Bluetooth SIG**: Bluetooth Special Interest Group is the standards organisation that oversees the development of Bluetooth standards and the licensing of the Bluetooth technologies and trademarks to manufacturers [1].

- **Bootloader**: A bootloader is a type of program that loads and starts the boot time tasks and processes of an operating system. It enables loading the operating system within the device memory when a device is started or booted up [2].

- **DFU**: The Device Firmware Upgrade is a vendor and device independent mechanism for upgrading the firmware of USB devices with improved versions provided by their manufacturers [3].

- **GPIO**: General Purpose Input/Output

- **IDE**: Integrated Development Environment

- **IEEE**: Institute of Electrical and Electronics Engineers

- **IoT**: Internet of Things

- **MBR**: The Master Boot Record provides an interface to allow in-system updates of the application, the SoftDevice, and bootloader firmware [4].

- **SDK**: Software Development Kit

- **SoftDevice**: A SoftDevice is a precompiled and linked binary software implementing a wireless protocol developed by Nordic Semiconductor [5].

- **SWD**: Serial Wire Debug
- **UUID:** The Universally Unique Identifier is a 128-bit number that is used to identify information, in the case of this project it identifies an Attribute type. The Attributes that are adopted by the Bluetooth SIG share all but 16 bits of a special 128-bit Base UUID. On the other hand, UUIDs that are defined by the developer can be any 128-bit numbers.
2. Introduction

The motivation for doing this project is the lack of a system to transfer the data from a sensor to a computer wireless with a low power consumption. In order to solve this problem, this project has consisted of the creation of several Bluetooth Low Energy applications that allow this data transmission.

2.1. Objectives of the project

- Develop a way to collect data continuously from the sensor using the nRF52840 USB Dongle device from Nordic Semiconductor.

- Be able to send the data acquired by the Dongle using BLE to a Raspberry Pi. This process has to be continually repeated at the maximum frequency possible in order to get a good approximation of the data from the sensor.

- Receive the data coming from the Dongle using a Raspberry Pi and display it on a screen. This data must also be saved in text files.

- Include the calculation of parameters in the information that is sent from the Dongle to the Raspberry Pi.

- Implement a method for allowing the Dongle to do the data acquisition process at the same time it performs the parameters calculation and sends the information.

2.2. Scope of the project

This project intends to create a system to make the data reading of a sensor easier for its further analysis. To do this, two software applications have been created using various development tools. The functionality of these applications is specifically adapted to the device nRF52840 USB Dongle. The code that is incorporated in the Raspberry Pi works only for the two applications developed. As soon as the Dongle is powered, the developed application will start, but the connection between the Dongle and the Raspberry Pi will not happen until the Raspberry Pi accepts it. The applications will periodically send raw data or parameters that will
represent an approximation of the behaviour of the sensor signal.
3. Bluetooth Low Energy

Bluetooth Low Energy is a low power wireless technology used for connecting devices with each other. It is focused on low power consumption in order to be able to maintain a connection between devices for long periods of time without needing to replace or charge any battery.

Even though this technology uses the Bluetooth brand it should not be considered as just an improvement of the Bluetooth Classic everyone knows. It is a technology with different design goals and its marketing purposes differ from the Bluetooth ones. While Bluetooth Classic aims to be a wireless link between devices for large data transfer (e.g. audio streaming, exchanging large documents), Bluetooth Low Energy pretends to be a tool to exchange small amounts of data between devices that need to run on batteries for days, months or even years.

BLE has a lot of possible applications in the emerging field of IoT. That’s why it is thought that it will gain a lot of importance in the next years to come. Some interesting examples of the use of BLE are: wireless charging, beacons, home automation systems or smart sensors.

3.1. History

In 2006, a group of companies released to the public a new wireless protocol with the objective of providing a technology that could run on devices for long periods of time minimizing the power consumption. This technology was named Wibree and it was based on the already existing Bluetooth.

In 2010, the Bluetooth SIG completed the integration of the technology into its latest release, the Bluetooth Core Specification Version 4.0 and it was renamed to Bluetooth Smart.

One year later, in 2011, the Bluetooth Smart logo scheme was announced in order to establish the compatibility between devices that used the technology. It differentiated between dual mode devices, which can use both Bluetooth Classic and Bluetooth Low Energy and single mode devices, which are only capable of using Low Energy technology or Bluetooth Classic. The first type of devices was tagged with the Bluetooth Smart Ready logo and the latter with the Bluetooth Smart logo for BLE devices and the Bluetooth logo for Bluetooth Classic devices. In this year it was also released the first smartphone to implement Bluetooth 4.0 and therefore, to be able to support Bluetooth Low Energy.
In 2013, the 4.1 specification was released and it improved the interoperability among devices with different Bluetooth versions including support for multiple roles simultaneously.

In 2014, the Bluetooth 4.2 specification improved the security, increased the speed and included IP connectivity.

In 2016, the latest version of Bluetooth was released. Bluetooth 5.0 included some relevant improvements, the most important changes were: two times the speed, four times the range and eight times the broadcast message capacity.

### 3.2. Compatibility with Bluetooth Classic

Bluetooth Classic (BC) and Bluetooth Low Energy (BLE) are very different technologies with different approaches. For this reason, a device that only supports Bluetooth Classic cannot establish a connection with another device that only supports BLE. In order to be able to communicate these two incompatible systems, a Dual Mode device can be used.

A Dual Mode device is able to support both protocols and can communicate with devices that work only with BLE and devices that work only with Bluetooth Classic. Examples of this kind of instruments are: a smartphone, a modern laptop or a tablet.

**Table 4.1. Compatibility between Bluetooth technologies according to their logo [7].**
Table 4.2. summarizes how the different types of Bluetooth devices communicate with each other. It is interesting to highlight that Dual Mode devices communicate with each other using Bluetooth Classic. It is also important to remember that Bluetooth specifications are backwards compatible with each other, but the communication features are limited to the older version.

<table>
<thead>
<tr>
<th>Type of device</th>
<th>BLE only</th>
<th>Dual Mode</th>
<th>BC only</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLE only</td>
<td>LE</td>
<td>LE</td>
<td>None</td>
</tr>
<tr>
<td>Dual Mode</td>
<td>LE</td>
<td>Classic</td>
<td>Classic</td>
</tr>
<tr>
<td>BC only</td>
<td>None</td>
<td>Classic</td>
<td>Classic</td>
</tr>
</tbody>
</table>

Table 4.2. Communication between types of device. Source: own

3.3. Technical specifications

Bluetooth Low Energy operates in the 2.4 GHz ISM band and has a range between 30 and 400 meters. The range is affected by various factors, some of them can be: the interference of physical obstacles in the communication, the design of the antenna of the BLE device or the device orientation. Even though BLE can work in pretty long distances the typical operating range is about 2 or 5 meters because it reduces the power consumption and guarantees a better data transmission.

The data throughput is limited by the rate at which the physical antenna from the BLE device is able to transmit data. For Bluetooth versions prior to Bluetooth 5 the data throughput is fixed at 1 Mbps, but for Bluetooth 5 and later this rate can be increased to 2 Mbps when using the high-speed feature. If the long-range feature is utilised, the data throughput drops to 500 or 125 Kbps.

Figure 3.2. Bluetooth Low Energy technical data [8].
It is very important to have in mind that gaps in between packets exist and this means that there is a time where no data is being exchanged. This time loss has a value of 150 microseconds. Another important fact is that the maximum amount of data that a BLE packet can have is 33 bytes, but only 20 bytes are available for the attribute data.

3.4. Architecture

The architecture of Bluetooth Low Energy is composed of three main blocks called: Application, Host and Controller. This chapter will be focused on the upper layers of the BLE protocol stack (Application block and GATT and GAP layers) because they are the ones that users usually directly interface with, but it is first needed to have some knowledge of the lower layers.

![Bluetooth Low Energy stack architecture](image)

3.4.1. Physical Layer (PHY)

The physical layer refers to the physical radio capable of modulating and demodulating the analogical signals and transforming them to digital signals. This radio uses the 2.4 GHz ISM band mentioned before. It also divides this frequency into 40 parts of 2 MHz each, where 37 of these 40 frequencies are used to exchange data during a connection with a device and the other 3 are used to establish the connection.
3.4.2. Link Layer

The link layer is the element that interfaces directly with the physical layer and is responsible for meeting all the timing requirements and handling the state of the radio. Handling the state of the radio means that it has to deal with the way the device connects to other devices. There exist three principal states a BLE device can work in:

- **Advertising state**: the device sends packets to inform other devices that it is available for a connection.

- **Scanning state**: the device is looking for advertising packets from other devices.

- **Connected state**: the device has established a link with another device and it interchanges data with it. A device in the Connected state can come from either the Advertising state or the Scanning state. If the previous state was Advertising, now the device is called the Slave and it was Scanning, the device is called Master. The difference between the Master and the Slave is that the Master manages the connection and the Slave follows the Master’s timing.

There are two types of packets in BLE: advertising packets and data packets. Advertising packets are utilized to discover Slaves and connect to them and to broadcast data without creating a connection. On the other hand, data packets transmit information between connected devices.

In the case of this project, the Dongle acts as the Slave during the connection because it was previously sending Advertising packets and the Raspberry Pi acts as the Master.
**Bluetooth Device Address**

The Bluetooth Device Address is an identifier for Bluetooth devices (similar to a MAC address) formed by a 48-bit (6-byte) number. There are two types of addresses:

- **Public Address**: It is factory-programmed and must be registered with the IEEE Registration Authority. This type of address is permanent; thus, it will not change during the device lifetime.

- **Random Address**: It can be pre-programmed on the device or generated during its lifetime. It is the most common type of address because it does not need to be registered with the IEEE.

**3.4.3. Generic Access Profile (GAP)**

The Generic Access Profile (GAP) defines how BLE devices interact with each other. It is key to determine the way two or more devices exchange information and communicate. GAP covers the following aspects: modes and roles of BLE devices, advertisements, connection establishment and security. In this report only the aspect of the modes and roles of BLE devices will be covered because it gives enough information to understand the project. Further information about this topic can be found on the complementary bibliography section of this document.

**Roles**

- **Broadcaster**: The Broadcaster uses the Advertising state from the Link Layer to send advertising packets without establishing a connection with any devices. The data is available for all the devices that are listening. An example of a Broadcaster would be a public thermometer that informs about the temperature to all the interested devices.

- **Observer**: The Observer uses the Scanning state from the Link Layer to look for advertising data coming from a Broadcaster. This role does not establish a connection with other devices and does not send data. A device with a display is a common example of this role.
- **Peripheral**: The Peripheral sends advertising packets in order to establish a connection with another device. It uses the Connected state from the Link Layer and it is a Slave. Peripherals are optimized to consume the least amount of processing power and energy. For this reason, they are low-cost gadgets. The Dongle will use this role in the applications developed in this project.

- **Central**: The Central is the Master of the Connected state from the Link Layer. Central devices look for advertising packets from Peripherals and initiate a connection with the ones they choose. A Central can be connected to multiple devices creating a single network. The Central role is usually performed by devices with powerful CPUs.

### 3.4.4. Generic Attribute Profile (GATT)

The Generic Attribute Profile defines the way data is organized and exchanged during a Bluetooth Low Energy connection. To fully understand GATT, it is first needed to learn about the Attribute Protocol (ATT) since they share the same roles. These two roles are the Server and the Client:

![Figure 3.5. Broadcaster/Observer network topology [11].](image)

![Figure 3.6. Peripheral/Central network topology [11].](image)
- **Server**: The Server is responsible for containing the data that is going to be exchanged. It receives requests from the Client and answers sending responses back. The data that the Server has is organized in the form of Attributes, which have the structure represented below:

![Attribute structure](image)

*Figure 3.7. Attribute structure [12].*

- **Client**: The Client sends requests to the Server in order to read its data or control its behaviour. Firstly, the Client does not know about the Attributes located in the Server, for this reason it performs a service discovery before sending requests. With the service discovery the Client is able to find out about the presence and nature of the Server’s Attributes.

The GATT defines a hierarchy for organizing Attributes. These Attributes are organized as GATT Server Profiles, which at the same time are formed by Services, which contain Characteristics, which contain Attributes. A Characteristic’s Attribute can be a Declaration, a Value or a Descriptor.

![Hierarchy for organizing Attributes](image)

*Figure 3.8. Hierarchy for organizing Attributes [13].*
The GATT Profile that has been modified in this project to adapt it to the objective of each application is the Heart Rate Service, which is a public Service defined by the Bluetooth SIG. The structure and hierarchy of this Profile will be used as an example to clarify the concepts recently explained.

![Figure 3.9. Structure and hierarchy of the GATT Heart Rate Service [13].](image)

It can be observed that the Service contains eight Attribute data structures (underlined in red) and three Characteristics (circled in green). The "Heart Rate Measurement" Characteristic contains three Attributes, including a Descriptor Attribute. This Characteristic is a Server-initiated update characteristic, which means that the value will be sent to the Client whenever the value Attribute is updated.
4. Hardware setup

In this section, it is described an explanation about the hardware devices that have been used to program the applications on the Dongle and the way these devices connect and interact with each other. The most important devices in this setup are the nRF52840 USB dongle, the Raspberry Pi 2 and a computer.

The Dongle is connected to a laptop via USB, which is its energy source. In the laboratory the energy source will be a battery with a USB port, but the computer is used because it allows to program the Dongle. Soldered to the castellated edges of the Dongle there will be the pins, which will make the connection between the Dongle and the sensors available in the laboratory.

The Raspberry Pi is connected to a keyboard and a mouse through USB and to a display via HDMI. It may be needed to use a VGA to HDMI adapter cable to make the display connection. The power source of the Raspberry will also be the laptop and they will be connected using a micro-USB to USB cable. If it is needed to have Internet on the Raspberry, an Ethernet cable will also be required, which will be connecting the Raspberry with the laptop. Finally, to communicate the Raspberry with the Dongle via Bluetooth Low Energy, a Bluetooth 4.0 USB adapter is necessary. It is important to remember that the Raspberry needs a micro SD card to store all the data. In this project the micro SD card that has been employed has a capacity of 16 GB, but an 8 GB card should be enough to handle the developed applications or similar ones.

To test the code written for the applications and be able to perform debugging there is another gadget that has been used. This device is the nRF52840 DK from Nordic Semiconductor. This board is more sophisticated than the Dongle, but it is also bigger and more expensive; that’s the reason why it will only be used to make the coding tasks easier. First the applications will be programmed on the nRF52840 DK and then, once it is checked that they work properly, they will be programmed on the Dongle. The nRF52840 DK replaces the Dongle in the setup, but its connection with the computer is made via a micro-USB to USB cable instead of connecting it directly to the computer USB port.
4.1. nRF52840 USB Dongle

The nRF52840 USB Dongle (board number PCA10059) is a device from Nordic Semiconductor that has the advantages of being small and low-cost while being able to use most of the short-range wireless standards including Bluetooth Low Energy. It can be used for product and software development with the help of the nRF5 SDK and the application nRF Connect for Desktop, which is why it is a great tool to develop wireless applications.

The Dongle features a green LED (LD1), a multicolour RGB LED (LD2), a user configurable button (SW1), and a reset button (SW2). It also has a USB-A-type connector printed on the circuit board and 15 GPIOs in addition to the ground, power and SWD connections along the castellated edges [21].
The Dongle can be powered using the USB interface or using an external regulated 1.8V – 3.6V source connected to the VDD OUT point. As it has been explained, in this project the power source will be the USB interface. The reset button (SW2) is used to put the Dongle in the DFU mode, which allows the user to program it. LED 2 will be pulsing red to indicate that the device has entered the DFU mode.

**Figure 4.2. nRF52840 Dongle’s buttons and LEDs drawing (front view)** [21].

The nRF52840 DK (board number PCA10056) will be used as a development platform for testing Bluetooth Low Energy applications. One its main benefits is that it features an onboard programming and debugging solution. This is a sophisticated board which includes a lot of features and offers a wide variety of options for developers. However, in this report only the required information for understanding the procedure of debugging the project will be explained.

The device has four buttons and four LEDs and can be powered using various methods, but in this case its energy source will be the computer. The board is linked with the laptop using the USB connector (J2). When connecting the board to the computer it is important to verify that the power switch (SW8) is set to ON, the nRF power source switch (SW9) is set to VDD and the nRF only switch (SW6) is set to DEFAULT.

**Figure 4.3. The USB Dongle with soldered pins that has been used. Source: own.**

### 4.2. nRF52840 DK

The nRF52840 DK (board number PCA10056) will be used as a development platform for testing Bluetooth Low Energy applications. One its main benefits is that it features an onboard programming and debugging solution. This is a sophisticated board which includes a lot of features and offers a wide variety of options for developers. However, in this report only the required information for understanding the procedure of debugging the project will be explained.

The device has four buttons and four LEDs and can be powered using various methods, but in this case its energy source will be the computer. The board is linked with the laptop using the USB connector (J2). When connecting the board to the computer it is important to verify that the power switch (SW8) is set to ON, the nRF power source switch (SW9) is set to VDD and the nRF only switch (SW6) is set to DEFAULT.
Figure 4.4. The nRF52840 DK board that has been used. Source: own.

Figure 4.5. nRF52840 DK drawing (front view) [22].

4.3. Raspberry Pi 2B

A Raspberry Pi is a single board computer which allows the user to connect peripherals such as a keyboard, a mouse or a display and it is capable of doing the tasks a normal desktop computer can do. Due to its small size and reduced price it is very used in projects where some computational power is needed to carry out simple tasks. In this project its function will be the reception and representation of information coming from the USB Dongle or the nRF52840DK.
In order to be able to use the Raspberry Pi, the first thing that has to be done is to download Raspbian [23], which is the operating system that runs on the Raspberry Pi. This system software can be easily downloaded from the official page of the Foundation: https://www.raspberrypi.org/downloads/raspbian/ There are three different versions of Raspbian, depending on the amount of features and software that they include. The chosen version for this project has been the “Raspbian Stretch with desktop and recommended software”, which is the most sophisticated one. Nowadays, this version cannot be found on the website mentioned before because a new version of the operating system (Raspbian Buster) has been released.

The next step is to format the micro SD card and then write the Raspbian image file into it. The micro SD card has been formatted using Windows 10, but for the image writing the software Win 32 Disk Imager has been needed. Once this is done, the Raspberry Pi can be started by just plugging the card into the Raspberry Pi and connecting it to a power source (in this case the laptop). With this done, the device is ready to be configured and be used. But without some peripherals it will not be possible to interact with the Raspberry Pi, so the next step is connecting the external devices.

At first, the Raspberry Pi connection was intended to be made using the laptop screen, keyboard and mouse. This way less devices would be required and the working environment would be more organized. This configuration is called headless setup and there is plenty of information about it on the Internet.

The headless setup consists in creating a file named “ssh”, that has no extension and placing it onto the boot partition of the micro SD card. This way SSH is enabled and this allows the remote access of the Raspberry Pi from a computer. Then, after enabling other devices to connect to the computer’s Internet on the Windows networks configuration, the Internet can be shared with the Raspberry Pi using the Ethernet cable. Once this is done, using the software Putty it will be possible to access to the command prompt of the Raspberry Pi. But, in order to be able to use the desktop and all its functionalities, it will be necessary to enable VNC (which will have to be downloaded) in the Raspberry Pi’s configuration app.

The described steps were carefully followed and initially it was possible to use the Raspberry Pi with only the laptop, but when the Raspberry Pi shut down and then turned on there were problems with recognizing the Raspberry Pi’s IP address. Before spending a lot of time finding and solving the problem it was decided that the easiest option would be to use the standard
connection that uses an external display, keyboard and mouse.

Figure 4.6. Raspberry Pi setup. Source: own.
5. Generic process for creating a BLE application

In this section there is an explanation regarding the tools and procedures that have been used during the development of the project. These tools include the software nRF Connect for Desktop and Segger Embedded Studio (SES) as well as the nRF5 SDK, which is a group of files organized in folders that provide the environment for developing applications with all the nRF5 Series devices from Nordic Semiconductor. The nRF5 Series include both the Dongle and the nRF52840 DK.

5.1. nRF5 SDK

The nRF5 SDK includes a broad selection of drivers, libraries, examples for peripherals, Soft Devices, and proprietary radio protocols [24]. It can be downloaded for free from the Nordic Semiconductor website (https://www.nordicsemi.com/Software-and-Tools/Software/nRF5-SDK) [25]. Besides, on the same site, all the documentation regarding the files included in the SDK is fully available. Once the nRF5 SDK has been downloaded, it is important to create a copy as a backup because the examples from the SDK will be modified to adapt them to the needs of the applications that are intended to be built.

When the SDK is unzipped and opened it can be observed that there are various folders in it. The most interesting folder for creating a BLE application is the examples one, which contains models of different types of applications that can be created with the nRF5 Series devices. Among all the classes of applications existent in the examples folder, it can be observed that there are only three folders that contain BLE examples. The one that this project will use is the ble_peripheral because as it has been said, the Dongle will act as a peripheral in the BLE connections that the applications here presented will establish.

Inside the blePeripheral folder, a wide variety of potentially useful examples of applications can be found. Each example has a main.c file, which contains the code, written in C language, that the application uses to perform its functions. The modification of this file will be done using the Segger Embedded Studio, which will be presented below. The examples also have folders with the name of different boards from the nRF5 Series. These are the boards that can handle the code of the application example.
Inside every folder with a board name there is one or more folders with the name of the Soft Device that is supported by the board chip. In order to select the correct Soft Device for a BLE application, a table with the usage scenarios for each Soft Device is used.

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Role</th>
<th>Chip</th>
<th>SoftDevice</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bluetooth Low Energy</td>
<td>Peripheral</td>
<td>nRF51422</td>
<td>$110</td>
</tr>
<tr>
<td></td>
<td></td>
<td>nRF51822</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>nRF52810</td>
<td>$112</td>
</tr>
<tr>
<td></td>
<td></td>
<td>nRF52832</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Central or Peripheral</td>
<td>nRF51422</td>
<td>$120</td>
</tr>
<tr>
<td></td>
<td></td>
<td>nRF51822</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Central and Peripheral</td>
<td>nRF51422</td>
<td>$130</td>
</tr>
<tr>
<td></td>
<td></td>
<td>nRF51822</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>nRF52832</td>
<td>$132</td>
</tr>
<tr>
<td></td>
<td></td>
<td>nRF52840</td>
<td>$140</td>
</tr>
<tr>
<td>ANT</td>
<td></td>
<td>nRF51422</td>
<td>$210</td>
</tr>
<tr>
<td></td>
<td></td>
<td>nRF52832</td>
<td>$212</td>
</tr>
<tr>
<td>Bluetooth Low Energy and ANT</td>
<td>Peripheral</td>
<td>nRF51422</td>
<td>$310</td>
</tr>
<tr>
<td></td>
<td></td>
<td>nRF52832</td>
<td>$332</td>
</tr>
</tbody>
</table>

Table 5.1. Usage scenarios for nRF5 Series devices and Soft Devices [26].

When the Soft Device has been selected, the window shows different IDE options to develop the application. In this project the IDE that has been used is the Segger Embedded Studio (SES) [27], which corresponds to the folder named ses. In the ses folder there is a file whose format is .emProject; if this file is opened, a SES session will be opened with the project of the selected example.

The example that has been selected to be modified is the ble_app_hrs. The path that needs to be followed in the SDK to get to open the SES session with the Dongle is:

C:\...\nRF5_SDK\examples\ble_peripheral\ble_app_hrs\pca10059\s140\ses

Using the device nRF52840 DK the path is different as another board needs to be selected:

C:\...\nRF5_SDK\examples\ble_peripheral\ble_app_hrs\pca10056\s140\ses
5.2. Segger Embedded Studio (SES)

Segger Embedded Studio [27], from now on, SES, is a cross-platform IDE for microcontrollers that provides a workspace for embedded C programming and development. One of the reasons why SES is a good IDE choice for this particular project is that it has an agreement with Nordic Semiconductor which allows nRF users to get a free license with no limitations. This free license is activated the first time SES is used; the program asks for a registration and then sends an activation key to the registered email.

The software is available for Windows, Linux and Mac OS and it can be easily downloaded from Segger’s official website: https://www.segger.com/downloads/embedded-studio/

The downloaded version has been the Embedded Studio for ARM, Windows, 64-bit. From the same website, the J-Link Software and Documentation Pack for Windows have also been downloaded.

A Solution in SES is a grouping of one or more Projects. A Project is a singular application that can be compiled and flashed to your device, and they contain the source files that are used within the application. Solutions, on the other hand, are used to group related Projects together while still being able to switch between them within the same view. A Solution is stored as a .emProject file in your filesystem. Projects, on the other hand, do not have separate files. Instead, they are contained within the Solution file [28].

Opening a Solution can be done searching the .emProject file in the SDK and clicking on it or using the File tab on the SES toolbar. If SES is used, the option that needs to be selected in the File tab is Open Solution (Ctrl+Shift+O). Then, a File Explorer window will pop up and the desired .emProject file will have to be selected. When the Solution is opened, the SES window will present the screen in Figure 5.1. At this point, the modifications on the application example can get started.

By default, the file main.c is opened, but all the files on the Project can be modified. To open a file, it is only needed to double click on the selected in the Project Explorer window and it will be available on the Dashboard. Once all the coding is finished, the Project has to be built. This is done clicking with the right button on the name of the Project in the Project Explorer window and selecting the option Build. This will compile all the code and inform if an error has occurred because the code is incorrect. These messages are shown in the Output window.
If the board that is being used is the nRF52840 DK there is the possibility of debugging using SES. The debugging can be started clicking on the debugging icon from the toolbar (see Figure 5.2). Breakpoints are set on the on the desired line to stop the execution. To put a breakpoint in a line the user has to click on the grey part of the Dashboard (left side) at the height of the selected line. While debugging, when the program runs into a breakpoint it stops the execution. To continue with the execution, the Play button has to be clicked (see Figure 5.3).

There is an important file that exists in all the examples from the SDK that deserves special attention. This file is called *sdk_config.h* and it is placed under the *main.c* file in the Project Explorer. What makes this file interesting is that includes different configurations and macro definitions that allow you to customize the SDK and application to enable various features [28]. The file can be modified opening it on the Dashboard and editing the code, but there is another option, which is makes the task easier. This tool is called CMSIS Configuration Wizard and it is a user-friendly interface that allows the user to make changes in the *sdk_config.h* file. To enable the CMSIS Configuration Wizard some code has to be written in a file called *tools.xml*.
The details of this procedure are not relevant enough to be described here, but they can be found on the link [28].

Figure 5.3. Debugging window showing the Play button (red circle) and a breakpoint (blue circle).
Source: own.

5.3. nRF Connect for Desktop

nRF Connect for Desktop is a software package that can run on Windows, Linux and MacOS. The program is available without any cost on Nordic Semiconductor’s webpage (https://www.nordicsemi.com/Software-and-Tools/Development-Tools/nRF-Connect-for-desktop) [29]. It can be used for various purposes as it has a set of different applications available that can be downloaded and added to the software framework at any moment. Among all the applications, only two are useful for this project, they are called Bluetooth Low Energy and Programmer. The Bluetooth Low Energy application has been used to understand the concepts of the BLE protocol and the Programmer application allows flashing the firmware to the Dongle.

5.3.1. Bluetooth Low Energy application

This application has been useful for getting familiar with, developing, and testing Bluetooth Low Energy devices. It allows the user to set up a local device, connect it to advertising devices and discover their services, maintain the connection and the connection parameters, pair the devices, and change the server setup for the user’s local device. It also offers a detailed log for troubleshooting purposes [30].

To work with this application a BLE device must be connected to the computer, otherwise the
application will not be able to recognize it. In the case of the Dongle, the device has to be in DFU mode for the application to detect it. When the application is launched, the device must be selected from the Select Device drop-down menu. Then the device appears on the screen and it is available for a BLE connection as either a central or peripheral. As the Dongle will be acting as a peripheral in the final setup, when practising with this program, the connections that were established were using the Dongle in the peripheral role.

The other device that was connected to the Dongle was a mobile smartphone with support for BLE. The connection was established using two apps: nRF Connect for Mobile and LightBlue. Both apps do the same, but with different graphical user interface. To establish the connection the Dongle has to start advertising. Clicking on the gear button a menu will appear where there is the option of start advertising. When the Dongle is advertising and using the phone app, scanning can be performed and in the list of available devices the name nRF Connect should appear. Selecting nRF Connect the connection will be established.

![nRF Connect v3.1.0 - Bluetooth Low Energy](image)

*Figure 5.4. Screenshot of the nRF Connect for Desktop when a connection is established between the Dongle (nRF5x) and the phone (Redmi). Source: own.*
At this point there are a lot of things that can be done, but to test the connection, a new service with a characteristic was created. The characteristic had the Read and Notify properties and it was given an initial value. Then a descriptor called Client Characteristic Configuration was added to the characteristic. With the connection established, the characteristic’s value was displayed on the phone’s app and if the value was changed in the nRF Connect, the value also changed in the phone.

![Image](image.png)

Figure 5.5. Battery Level characteristic on the nRF Connect for Desktop (left) and value read on the LighBlue phone app (right). Source: own.

### 5.3.2. Programmer application

nRF Connect Programmer can be used to program firmware to Nordic devices. The application allows the user to see the memory layout for both J-Link and Nordic USB devices. It also allows the user to display content for the HEX files and write it to devices [31]. In this project this application has been used to program the applications on the Dongle.

The process of connecting the Dongle to the application is the same as in the Bluetooth Low Energy application. When the device has been selected the screen shows the device with the contents of its memory represented. In Figure 6.6. the names referring to the different memory
areas can be appreciated. The application has also a log section, where the messages about
the processes that have been carried out appear, and the right section has the tools to write
files into the device and also erase them.

![Screenshot of the Programmer application with the memory parts of the Dongle.](image)

Figure 5.6. Screenshot of the Programmer application with the memory parts of the Dongle.
Source: own.

To flash the Dongle, two HEX files need to be loaded:

- **Application HEX file**: This file is created when a SES Project is built correctly. The file is
  located in the SDK, in a folder called hex. The path to the file is: `C:\...\examples\ble_peripheral\ble_app_hrs\pca10059\s140\ses\Output\Release\Exe`

- **Soft Device HEX file**: This file can be found in the downloaded SDK; it is not created
  afterwards. Its path is: `C:\...\components\softdevice\s140\hex`

To load the files, the menu on the right of the nRF Connect screen will be used (see Figure
5.7). Clicking on the Add HEX file tab it will be able to browse the HEX files and select them.
When a HEX file is selected, it will appear in the File Memory Layout. The Soft Device HEX
file includes the blue region and the orange one. The blue region corresponds to the soft device
and the orange region corresponds to the MBR. When the Application HEX file is also added,
a green region appears, which is the application.
To write the files into the Dongle the only thing that needs to be done is to click on the *Write* tab on the right menu. Then, the Dongle flashing process will start showing various messages in the log section. When this process finishes, the Dongle will automatically exit DFU mode, it will be disconnected from the program as it will not appear in the device list, and it will start the application that has just been written on it.

![File Memory Layout](image)

*Figure 5.7. Programmer application screenshot showing the tools menu on the right and the File Memory Layout with the loaded HEX files on the left. Source: own.*

It is important to remark that the Programmer application is not used when working with the nRF52840 DK because the application is automatically flashed into the device clicking on the option “Download name of the application” in the “Target” tab in SES. Before doing this, the Project has to be built.
6. The Bluetooth Low Energy applications

This section consists in the explanation of which BLE applications have been developed and their use, how they have been written and which the result of their performance is when they carry out their tasks. The objective of the applications is to transmit information using BLE, from the Dongle to the Raspberry Pi. The data that is transferred is a measurement from a sensor that is connected with the Dongle. For this reason, the transmission needs to have certain characteristics because otherwise the information received in the Raspberry Pi will not be useful.

The sensor gathers data from a power source that has a frequency of 50 Hz, this is the same as saying that its period is 0.02 seconds or 20 milliseconds. In order to be able to reconstruct the original signal with the data that is sent to the Raspberry Pi using BLE, it is crucial that the process of sampling respects the Nyquist–Shannon sampling theorem. This theorem states that the sampling frequency has to be at least two times the frequency of the signal that is being sampled. To maximize the precision of the signal reconstruction, the sampling frequency has been set to 1 kHz, which means that the sampling period is 1 ms.

The design of these applications is based on an example from the SDK called ble_app hrs. This example consists of a heart rate sensor that sends out data including the battery level, the detection of the sensor and the heart rate measurement. The code creates simulators to obtain the data and uses timers to send it via BLE. When a timer expires, the code sends the data and the timer is restarted. This example was chosen because it has support for both the Dongle and the nRF52840 DK and its function is similar to the purpose of the applications that will be created.

The process of coding and testing the applications has been possible with the help of the setup that has been described. The first task of the process is the modification of the code using SES. Then, once the SES Project is built, the application is flashed to the Dongle using the nRF Connect for Desktop. The Dongle sends the data to the Raspberry Pi and using a Python code the data can be represented on the monitor. If the sent data is correct and the connection characteristics between the Dongle and the Raspberry Pi are the expected ones, the application is ready. If this is not the case, then the code has to be modified again and repeat the whole process. To make the coding task easier, the nRF52840 DK is used to be able to debug when using SES. This board can also transmit data to the Raspberry Pi using BLE.
At the end of the project, the Dongle will acquire the data from the sensor using an analagical to digital converter (ADC), but to test the applications easily it uses simulated data. This data is generated using an Excel file that emulates the signal from the power source in the laboratory. To generate the data, the Excel file takes the frequency, the amplitude and the offset of the signal. With these parameters and the step size of the time, a sine wave function is created. The step size is defined as 1 millisecond because it is the sampling period of the applications. This way the Excel file generates the value of the sine wave every 1 millisecond.

![Excel file used to generate simulated data. Source: own.](image)

6.1. First application: sending raw data

The purpose of this application is to obtain the data from the sensor every 1 millisecond using an analagical to digital converter (ADC), and to send BLE packets with this data to the Raspberry Pi. At first it was thought that sending a packet of 20 values every 20 milliseconds would be the best option because that is the signal period. But in practice this generated problems because it exceeded the maximum length value of 20 bytes that a BLE packet can have. For this reason, the application was programmed to send 10 values every 10 milliseconds (half of a period).
6.1.1. Analogical to digital conversion

The first and most important modification that had to be made to the hrs example was to add an ADC to replace the simulators in the data collection. The ADC needs to define a point where the data will be collected. This point has to be one of the connection points in the castellated edges, where the pins have been soldered. To determine which pin is the ideal, the Dongle’s connections schematic has been consulted [32]. The connection points in the schematic that have the nomenclature “AIN#”, where the character “#” is a number, are the points that allow an analogical input. Among them, the ones that have a square at the end of their line in the schematic, are the ones that allow a connection. The chosen pin has been the P0.02, whose connection point is the AIN0.

![Figure 6.2. Part of the Dongle’s schematic showing the chosen pin for the ADC connection.][32]

The ADC default configuration has a resolution of 10 bits; this involves that the Dongle will be able to send $2^{10} = 1024$ different values. The voltage range accepted by the Dongle is 3.6 V, therefore, the voltage resolution of the ADC is $3.6/1024 \approx 0.00352$ V. The data that the ADC collects is stored in 16 bits (2 bytes), meaning that it will only be possible to send 10 values (20 bytes) in each BLE packet.
6.1.2. Software development

In this section there is the explanation of how does the original hrs example works and the modifications that have been made to it in order to accomplish the desired behaviour. Not all the details about the example will be analysed, only the important parts that allow the comprehension of this project. How the Raspberry Pi receives the data from the Dongle and represents it using a Python code is also explained here. This is the part that has required more effort due to the fact that SES uses C language to develop the applications, so a new programming language had to be learnt.

The hrs example from the SDK acts as a sensor that is continually sending values of heart rate measurements to the device it is connected with. When the application is initialized it automatically starts advertising so that other BLE devices can discover the sensor. When a device decides to establish a connection with the sensor, the application starts sending heart rate measurements to the device every 1 second. The sensor continues sending BLE packets until the connection between the two devices is stopped.

To do this, the application uses a heart rate timer, which is originally set to 1000 milliseconds (1 second). Every time this timer expires, a function is called and the timer is restarted. The called function creates the measurement with a simulator and then calls another function that encodes the data and sends it to the client. The measurement is a 16-bit value, but the data that is sent is an array of 8-bit values. Now that the original functioning of the application is known, let’s explain the modifications.

To implement the analogical to digital conversion in the code some drivers need to be enabled and the file containing the ADC configuration code needs to be loaded to the SES project. The CMSIS Configuration Wizard can be used to enable the corresponding drivers. Writing some code, the ADC is configured to collect data from the input point AIN0 and it substitutes the simulator in the data acquisition process. Now the heart rate measurements that the application sends are actually the values that the ADC receives and then stores in a variable called adc0.

As the sending channel uses 8-bit data and the values from the ADC have 16 bits, the adc0 values are split in two 8-bit values and stored in an array. When the length of the array is 20 (it takes 10 milliseconds because every time the timer expires two values are added to the array), the sending function is called. This function copies the previous array to the array that transmits
the data, which is called *encoded_hrm* and sends it via BLE to the Raspberry Pi.

![Diagram for schematizing the developed code](image)

**Figure 6.3. Diagram for schematizing the developed code. Source: own.**

The Raspberry Pi also needs to be programmed to establish the BLE connection with the Dongle and to receive and interpret the data. The programming language used for this task has been Python. The Python program has to respond to the advertising notifications that the BLE is sending and accept the data that is being sent. Once the data is received, the Raspberry Pi has to regroup the values that were previously separated, convert them to Volts units and print them. To make the data analysis easier, a text file is generated with all the data that is also printed in the terminal.

This Python code is the result of changing an already existing code and adapting it to the application objective. The original code is called *bluepy* and it is available on GitHub ([https://github.com/IanHarvey/bluepy](https://github.com/IanHarvey/bluepy)). *Bluepy* provides an application programming interface (API) to handle BLE devices using Python. As *bluepy* is a generic code, some adaptions have been performed to make the code recognize the Dongle and receive the notifications from the heart rate example.

To recognize the Attributes from the heart rate application, the code uses their UUIDs. These UUIDs can be found in the official Bluetooth SIG website. The next table summarizes the
UUIDs that have been used and their corresponding Attribute.

<table>
<thead>
<tr>
<th>UUID</th>
<th>Attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x180D</td>
<td>Heart Rate Service</td>
</tr>
<tr>
<td>0x2A37</td>
<td>Heart Rate Measurement Characteristic</td>
</tr>
<tr>
<td>0x2902</td>
<td>Client Characteristic Configuration</td>
</tr>
</tbody>
</table>

*Table 6.1. UUIDs used and their corresponding Attribute. Source: own.*

With the Heart Rate Service UUID the data from the application is exposed and ready to be accepted. The Heart Rate Measurement Characteristic controls the transmission of data from heart rate measurements and it has a notifying property and a Descriptor. This Descriptor is the Client Characteristic Configuration. Using these UUIDs the code is able to continuously accept the data coming from the Dongle.

Now the Raspberry Pi has to display the data in a way that it is easily understandable. To do this, a function for handling notifications includes a series of orders to organize the data. Firstly, the function opens an empty text file where the data will be added. Then, it regroups the separated data contained in the received array and uses a conversion factor to transform the values to Volts units. This conversion factor consists in multiplying the value by the accepted voltage range of the Dongle (3.6V) and dividing it for 1023 (base 2 exponential to the number of bits of the ADC minus 1). After this, the function prints the converted data in the terminal and writes it in the text file. When all the data from the received array has been written to the text file, the function closes the file. The data is organized in a single column in the text file.

To run the Python program in the Raspberry it is necessary to access the directory where the file containing the program is placed using the terminal. Then, the program has to be called including the Device Address of the Dongle and its type. The Dongle address can be found using various tools, but maybe the nRF Connect for Desktop application is the easiest one. When the Dongle is connected to the application the address appears under its name (see *Figure 5.4*). The address type is random.
6.1.3. Results

The first tests to see if the Dongle was able to send data were done using the nRF Connect application to establish a connection between the phone and the Dongle. Once the phone was able to receive notifications, the tests were done using the Raspberry Pi setup. To be able to run tests without a power source, the Excel file with simulated data has been used. In this file a new column is added with the ADC values (from 0 to 1023) obtained when converting the simulated ones. In the C code from SES, the data collection from the ADC is replaced with the array of these simulated values. This involves that every time the ADC used to take a sample and store it in adc0, now adc0 takes a value from the array. With this method, if the application works properly the text file and the terminal will show the same values from the Excel file. Once the application passes the test with the simulated data, the final test with the power source can be performed.

The final test has been done in the laboratory because a power source and an oscilloscope are needed. An analogical waveform with a frequency of 50 Hz and a range of Volts anywhere between 0 and 3.6 is set using the power source. The oscilloscope represents the waveform and displays the maximum and minimum value, the average value and the rms value.

![Oscilloscope screen showing a wave used for testing. Source: own.](image)
The Dongle is connected with the power source using its pins and a pair of wires. One of the wires is connected to a ground pin (GND) from the Dongle and the other one is connected to the pin P0.02 because it is from where the ADC collects the data.

*Figure 6.6. Connection between the power source and the Dongle. The green wire is connected to the ground (GND) and the red one is connected to the pin P0.02. Source: own.*

*Figure 6.7. Laboratory setup for testing with the power source. Source: own.*
The obtained results when the power source generates the waveform shown in Figure 6.5 are represented in Figure 6.8. The picture is a snippet of the column of values that appear on the terminal window. It can be observed that every 10 values the word “correcte” appears. This was done to differentiate the place where a BLE packet finishes. Every time the word “correcte” appears, the following values belong to the same packet. In the picture there is a total of 20 values, which is equivalent to 2 BLE packets, which also represents a period of the 50 Hz waveform from the power source. Just watching at the extreme values and comparing them to the maximum and minimum that the oscilloscope displays, it can be stated that the values belong to the waveform.

![Figure 6.8. Terminal output. Source: own.](image)

### 6.2. Second application: parameter calculation

This application collects data using an ADC the same way as the first application, but when 20 milliseconds have passed, instead of directly sending the data packet, it calculates some parameters of the wave function and then sends these parameters using BLE. The process of calculation and data transmitting is being carried out at the same time new data is acquired by the ADC, unlike the previous application which did not acquire data during the sending. The sampling time remains at the minimum possible for the application (1 millisecond), but now the data is sent every 20 milliseconds because the parameters do not exceed the 20 bytes length. This application uses a variation of the code from the first application because a lot of functions that control the connection parameters do not have to be changed.

#### 6.2.1. Calculated parameters

The wave parameters that have been calculated are: maximum, minimum, average value and root mean square (rms) value. These parameters are often used when working with signals. For this reason, it is interesting to have a system that provides their value every time the signal completes a period. To calculate the average value the next formula has been used:
\[
\text{avg} = \frac{\text{sample } 1 + \text{sample } 2 + \text{sample } 3 + \cdots + \text{sample } 20}{20}
\]  
(Eq. 7.1.)

To calculate the rms value, the next formula has been used:

\[
\text{rms} = \sqrt{\frac{\text{sample } 1^2 + \text{sample } 2^2 + \text{sample } 3^2 + \cdots + \text{sample } 20^2}{20}}
\]  
(Eq. 7.2.)

The maximum and the minimum are defined in the code as 16-bit (2 bytes) integers, but the average and the rms value are defined as floats. The float type has a length of 32 bits (4 bytes), which means that all four parameters have a length of 12 bytes. For this reason, the four parameters can be sent together. Every time the waveform finishes a period, the parameters corresponding to that period are calculated and sent via BLE.

### 6.2.2. Software development

Like in the first application, the heart rate timer remains at 1 millisecond and when it expires a function is called and the timer is restarted. The called function makes the ADC collect a sample and it is stored in a variable called \textit{adc0}. Then, \textit{adc0} is divided in two bytes and they are stored in an array. So far, there are no changes from the previous application, but when the length of the array is 40 the code functioning is different. When the array is full (has 40 elements) the ADC does not stop sampling, it continues its action, but the data is stored in a different array. This way the program never stops taking samples and meanwhile other orders are being carried out.

When the length of an array is 40 (any of the two), the parameters calculation takes place without stopping the sampling. To do the calculations, the array that is full is selected and its data is regrouped and stored into another array. This new array will contain half the elements of the other array, that is 20 elements which will have 2 bytes each. This array new array will be used to do the calculations of the parameters. When the calculations are finished, the sending function is called.

It is important to remember that this function sends an array of bytes, therefore, the maximum and the minimum, that are 16-bit integers, are again divided into 2 bytes and stored to the array that will be sent, which is called \textit{encoded_hrm}. With regard to the average and rms values, which are floats, they are divided into 4 bytes and also stored in the \textit{encoded_hrm}
array. Finally, this array is sent to the Raspberry Pi. To know when an array is full and the samples have to be stored into the other array, the code uses a flag that changes its value when an array is full. Another flag is used to know when the calculations need to be started.

Figure 6.9. Diagram to schematise the code functioning of the second application. Source: own.
The Python code from the Raspberry Pi has also been modified. The program regroups the data from the maximum and minimum values the same way as it did with the data from the first application and also converts it to Volts units. But for the average and rms values the data regrouping needs to be done using a different method. This method uses an array of bytes to unpack the data.

6.2.3. Results

The Python code prints the name of the parameter followed by its value on the terminal and it also creates three text files. One of them contains the maximum and minimum values organized in a single column. The values on the column are alternately maximum and minimum. Another file contains the average values and the last one contains the rms values.

![Image](image.png)

*Figure 6.10. Snippet of the terminal out. Source: own.*

*Figure 6.10* shows the values of the parameters from one period of the waveform generated with the power source. It is the same waveform that is represented in *Figure 6.5*. Comparing these values with the ones that are shown on the oscilloscope it can be appreciated that the error is very small. This error is caused because the range of Volts in the conversion factor is not exactly 3.6 V; that is just a nominal value used as a reference. It also depends on the values that the ADC takes from the power source, as this one does not provide the same values every period.

Using the information in the text files it is possible to represent the evolution of these parameters through time. Analysing *Figure 6.11* and *Figure 6.12*, it can be observed that the average and the rms value that have been calculated present the same evolution, as they correspond to the same wave. But the rms value presents a better accuracy of the value that is represented in the oscilloscope than the average value.
To make sure that the application works properly, it has been tested using two more waveforms: one with a bigger amplitude and the other one with a different offset. The frequency of the waveforms has remained at 50 Hz. Using the data received with the Raspberry Pi, the average and rms value evolution has been represented. This way, the different graphics can be compared and the performance of the Dongle when working with different inputs can be analysed.
Figure 6.13. Waveform with bigger amplitude. Source: own.

Figure 6.14. Evolution of the average value through time. Source: own.
Figure 6.15. Evolution of the rms through time. Source: own.

Figure 6.14 and Figure 6.15 show the results from representing the data obtained when the waveform from Figure 6.13 is connected to the Dongle. Both graphics represent a good approximation of the parameters as the error in the average and the rms value is in the order of 20 mV (millivolts).

Figure 6.16. Waveform with a different offset. Source: own.
Figure 6.17. Evolution of the average value through time. Source: own.

Figure 6.18. Evolution of the rms value through time. Source: own.
Figure 6.17 and Figure 6.18 show the results obtained when the waveform from Figure 6.16 is generated using the power source. As it can be seen, the results are very similar to the previous ones. Analysing all the figures, it can be observed that the value given by the oscilloscope is usually a few millivolts greater than the calculated value, but in the case of Figure 6.14 it is smaller. Despite this, it is risky to claim that the calculated value is always inferior than the one given by the oscilloscope. An error this small can be caused by the existent noise in the laboratory or the little variations of the waveform values.
7. Economic study

In this section there is the analysis of the budget needed to carry out this project. This budget includes the cost of the materials, tools, software and hardware devices that have been used, as well as the cost of the working hours that have been necessary to finish the project.

7.1. Cost of the equipment

The equipment required to do this project includes: the wires used to connect the devices, the software licenses, the documentation used to write this report, all the devices that have been bought during the passing of the project and the amortization of the instruments that have been used. The software tools that have been used did not represent any cost, as they were all license free and could be easily downloaded from their brand’s official website. The book that has been used to understand all the BLE basic theory and the first steps of the functioning of the nRF5 Nordic devices, is available on the author’s website (https://www.novelbits.io/bluetooth-5-developers-e-book/#ebook-packages) for a price of 99$. This value has been converted to euros for this analysis.

All the hardware devices needed for this project can be ordered online, so it is easy to estimate their price. Both the BLE book and the hardware devices used in this project did not represent an expense for the author because all the material was lent by the supervisor of the project. During the course of the project two nRF52840 DK boards were used because one did not work properly and had to be replaced. The following tables summarize all the equipment cost for this project and the cost of amortization due to their usage.

To calculate the cost of amortization of the equipment, the next information is needed: lifetime of the equipment, purchase value and amount of the equipment’s working hours during this project. Dividing the purchase value by the lifetime, the depreciation fee is obtained. Then the fee is multiplied for the hours of functioning and the cost of amortization is obtained. The amortization cost of the oscilloscope and the power source hasn’t been calculated because their usage time has been very small. The only objects that have been considered for the amortization cost are the laptop and the Raspberry Pi because they are the devices that have been working for more hours. Despite this, observing Table 7.2, it is obvious that this cost is insignificant compared with the acquisition price of the equipment.
Bluetooth Low Energy based on the nRF52840 USB dongle

### Table 7.1. Cost of the equipment used in this project. Source: own.

<table>
<thead>
<tr>
<th>Equipment</th>
<th>Unit price (€)</th>
<th>Quantity</th>
<th>Total price (€)</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLE theory book</td>
<td>89,20</td>
<td>1</td>
<td>89,20</td>
</tr>
<tr>
<td>Raspberry Pi 2B</td>
<td>42,37</td>
<td>1</td>
<td>42,37</td>
</tr>
<tr>
<td>nRF52840 USB Dongle</td>
<td>9,40</td>
<td>1</td>
<td>9,40</td>
</tr>
<tr>
<td>nRF52840 DK</td>
<td>43,29</td>
<td>2</td>
<td>86,58</td>
</tr>
<tr>
<td>Bluetooth 4.0 USB adapter</td>
<td>15,99</td>
<td>1</td>
<td>15,99</td>
</tr>
<tr>
<td>USB – microUSB wire</td>
<td>1,70</td>
<td>2</td>
<td>3,40</td>
</tr>
<tr>
<td>VGA – HDMI adapter</td>
<td>8,99</td>
<td>1</td>
<td>8,99</td>
</tr>
<tr>
<td>Ethernet wire</td>
<td>2,95</td>
<td>1</td>
<td>2,95</td>
</tr>
<tr>
<td>16 GB microSD card</td>
<td>6,99</td>
<td>1</td>
<td>6,99</td>
</tr>
<tr>
<td><strong>TOTAL</strong></td>
<td><strong>265,87</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Table 7.2. Cost of amortization of the equipment used in this project. Source: own.

<table>
<thead>
<tr>
<th>Equipment</th>
<th>Purchase value (€)</th>
<th>Estimated lifetime (years)</th>
<th>Estimated working time (days)</th>
<th>Amortization cost (€)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Laptop</td>
<td>800</td>
<td>7</td>
<td>10</td>
<td>3,13</td>
</tr>
<tr>
<td>Raspberry Pi 2B</td>
<td>42,37</td>
<td>7</td>
<td>5</td>
<td>0,08</td>
</tr>
<tr>
<td><strong>TOTAL</strong></td>
<td></td>
<td></td>
<td></td>
<td><strong>3,21</strong></td>
</tr>
</tbody>
</table>
7.2. Cost of the development time

The cost of the time that has been spent working on this project is showed in this section. The working has been divided in three sections:

- Research and previous study: contains the hours spent learning the basics of C programming, learning what is BLE and how it works, which are the characteristics and capabilities of the Dongle and how do the different software function.

- Software developing: time spent programming the two applications on the Dongle or the nRF52840 DK using SES and the examples from the SDK. The programming tasks that have been done using Python to allow the Raspberry Pi to receive and display data are also considered here.

- Testing and writing the report: contains the time dedicated to test the application, whether it is with the power source or the simulated data. This section also contains the hours committed to writing this report.

<table>
<thead>
<tr>
<th>Task</th>
<th>Time spent (hours)</th>
<th>Cost (€/hour)</th>
<th>Total cost (€)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Research and previous study</td>
<td>50</td>
<td>40</td>
<td>2000</td>
</tr>
<tr>
<td>Software developing</td>
<td>200</td>
<td>40</td>
<td>8000</td>
</tr>
<tr>
<td>Testing and writing the report</td>
<td>50</td>
<td>40</td>
<td>2000</td>
</tr>
<tr>
<td><strong>TOTAL</strong></td>
<td></td>
<td></td>
<td><strong>12000</strong></td>
</tr>
</tbody>
</table>

Table 7.3. Summary of the cost of the working hours. Source: own.
8. Environmental impact

The environmental impact of this project is small because most of it consists in software development. Therefore, the only part of the project that will make an impact on the environment will be the hardware devices. It is also important to consider that these devices communicate with each other using radio frequencies and they contain electrical and electronic components that generate electromagnetic waves.

8.1. Impact on people’s health

The radiofrequency of 2.4 GHz, which is the one BLE and other wireless protocols use such as Wi-Fi use, has been subject of numerous studies because is present in most of people’s everyday life. These studies have concluded that a prolonged exposure to these electromagnetic waves do not have any effect to people’s health, not even to children. The hardware setup in the laboratory also should not represent any danger for the people working there. The equipment uses very low voltages, so there is not any risk of electrocution.

8.2. Impact on the planet

The part of the project that creates a bigger impact on the planet comes when the lifetime of the devices ends. When this happens, they need to be properly recycled because these devices contain a wide variety of materials and some of them are toxic. For this reason, the recycling process of this kind of products is complicated.

During their use, the devices consume a certain amount of energy. Even though this energy consumption is low because that is just one of the BLE technology goals, it has to be considered. When the Dongle is powered using the USB interface, it only consumes 5 V and some of its functions can work with only 3 V. As for the Raspberry Pi, it uses an average of 295 mA when working. Its maximum consumption is 400 mA, which means 2.1 W approximately; this value is reached when it uses the 400% of its CPU, so probably it will not be reached when working with the Dongle.
Conclusions

The goal of the project was to develop BLE applications with the USB Dongle that could allow the wireless transmission of the data coming from a sensor. It has been proved using different tests that the applications here presented can create a BLE connection with another device and send data.

The first application enables the data collection using an ADC and then successfully transmits this data in BLE packets. The tests have shown that the same data that has been collected by the Dongle is then received and displayed using the Raspberry Pi.

The second application is an improvement of the first one. It makes possible for the Dongle to continue with the data acquisition while performing other tasks. It also incorporates the calculation of four parameters from the signal that is being sampled. Using the data from the text files that are generated when the parameters are sent via BLE, it is possible to study the time evolution of these parameters.

To enhance the performance of the Dongle, a good idea would be to create another application to put the Dongle in a sleep mode. This way, if data needed to be collected, the Dongle would run the application, but otherwise it would remain in sleep mode. With this application the energy consumption would be reduced even more.

To conclude, this project has proved that the nRF52850 USB Dongle from Nordic Semiconductor is a great option for running simple BLE applications due its low cost. The applications that have been created in this project using the Dongle have achieved the desired results. This project has also demonstrated that Bluetooth Low Energy is a technology with a lot of possibilities in the actual era. Its characteristics make it an ideal solution to establish simple wireless communications between devices and it allows the creation of applications that bring new functionalities to the devices, as in the case of this project.
Acknowledgements

I would like to thank the supervisor of the thesis, Manuel Moreno Egilaz for his help throughout this project, especially with the process of software development. He has guided me to structure the project and his advice with some of the C coding has allowed me to understand better this language. I also appreciate the fact that he has lent me all the devices and electronic material that has been used in the project.

I would also like to take advantage of the opportunity to thank my family for giving me support during the time I have been doing this thesis and the whole degree in general. Their encouragement has allowed me to keep going in the difficult moments.
Bibliography

Bibliographic references


Bluetooth Low Energy based on the nRF52840 USB dongle


[22] Nordic Semiconductor. *nRF52840 Development Kit User Guide* [online]. [Consulted: August 5, 2019]. Available at:


Consulted bibliography

This section contains the resources that have been used to gain knowledge about the theory of Bluetooth Low Energy, the basics of C programming or another process that has been used in this project. These documents have been consulted during the course of the project.

Tom’s hardware. *How to Set Up a Headless Raspberry Pi, Without Ever Attaching a Monitor* [online]. Available at: <https://www.tomshardware.com/reviews/raspberry-pi-headless-setup-how-to,6028.html>

Nordic Semiconductor. *nRF5 SDK v15.3.0* [online]. Available at: <https://infocenter.nordicsemi.com/index.jsp?topic=%2Fstruct_sdk%2Fstruct%2Fsdk_nrf5_latest.html&cp=5_1>

Tutorials point. *C programming* [online]. Available at: <https://www.tutorialspoint.com/cprogramming/index.htm>


Medium. *How To Use Android BLE to Communicate with Bluetooth Devices - An Overview & Code examples* [online]. Available at: <https://medium.com/@avigezerit/bluetooth-low-energy-on-android-22bc7310387a>