POLYTECHNIC OF TURIN

Department of Electronics and Telecommunications

Master of Science in Electronic engineering

Master Degree Thesis

Portable Embedded System for Continuous Monitoring of Anaesthetics

Supervisor
prof. Maurizio Martina
dott. Motto Ros Paolo
dott. Simone Aiassa

Candidate
Davide Tunzi
mat.: 243593

Academic year 2018-2019

Acknowledgements

First of all, I would like to express my gratitude to my thesis supervisor Prof. Maurizio Martina for the opportunity to work at this project, for his willingness and kindness shown me during all the work time. I would like to thank Dr Paolo Motto Ros for his willingness and suggestions. Finally, I would like to express my gratitude to my supervisor Dr Simone Aiassa for all the support provided, for the advice and for his patience, he always has encouraged me and helped me in all the moment representing an expert and trusty point of reference.
I would like to express my gratitude to my family to be always on my side in all the decisions taken along these years. Thank them for the emotional support in all the moments. Thanks to Ale for her infinite patience, support, the advice in all the moments academic and not, above all she is always ready to make me happy. Thanks to my friends and "bombers" Pappina, Pier, Fra, Simone, Marco, Bombone to be always on my side and for the best moments together and thanks to Terry, Penny, Gio, Dory for their super-friendship. Thanks to my friends (and colleagues) Gianni, Luca, Marco and Giuseppe, for their friendship and to have shared all the experience together, for the aristocracy housemate adventure and anxiety, for the big Sicilian smile and the sinusitis "mocevo" and finally thanks to Giuseppe for the right advice at the right moment. Thanks to all the friends not mentioned but always present.

Summary

This thesis project presents the development of an electronic system to enhance the success in the procedure of anaesthesia, the fundamental point during surgery. The anaesthesia is the result of the sum of a mix of various drugs as a hypnotic, analgesic, and muscle relaxant, to administer in the correct order, route and quantity. The most common cause of problems in anaesthesia is the "wrong dose", usually linked to difficulties in the real-time evaluation of the dose. Indeed, at the state of art, the most widespread method of evaluation is based on the Bispectral Index (BIS), which is the long-time estimation weighted sum of several electroencephalographic parameters. This approach, not always accurate, presents some limitations due to the high inter-patient variability. In the aforementioned scenario, the thesis work proposes a novel electronic device based on Therapeutic Drug Monitoring (TDM), an innovative approach grounded on the evaluation of drugs concentration directly by the blood vessel, with not invasive and easy routine.
The objective is to design a small size, low-cost and low power system to achieve a portable device, with syringe shape to be compliant with future medical application. The evaluation of anaesthetics drugs concentrations is based on the chemical RedOx reaction. At the variation of analyte concentration corresponds to the changing of the electron flux. The electrochemical analysis, for the investigation of the reaction mechanisms, requires the employment of external electronic hardware called potentiostat to control the electrode cell. The Screen-Printed Electrode (SPE) composed of Working, Counter and Reference electrodes, is properly connected to the circuit. In the implemented circuit solution, a potential is provided to the Working (with respect to the Reference), and placing the voltage reference at the middle of the dynamic, both the Reduction and Oxidation are evaluated without the usage of dual supply voltage, finally the potentiostat circuit is closed through the Counter and the current flow collected. Therefore the potentiostat has the double roles to drive the electrochemical cell and the ReadOut of the input current. In the proposed potentiostat, to avoid the use of AD-DA converter, replaced by adopting a low power solution based on quasi-digital waveform. Both driver and reader are quasi-digital: the driver converts the Pulse Width Modulation (PWM), in the ramp voltage signal for the electrochemical cell. On the other side the potentiostat ReadOut receives as input the sensor current response converted in a voltage signal, and through the integrator analog circuit, the information is coded in the temporal distance between

the consecutive edge of a digital signal. The event signal is measured directly by the microcontroller: the time gap between two rising edges of the wave results to be proportional to Faradaic current produced by the reduction of the drug.
The presented system is implemented by a custom Printed Circuit Board (PCB), completely dedicated to the application, in which the microcontroller and the circuit are integrated into a unique compact board. The device's board realization from the schematic layouts, the components selection, the placement and the routing, has carried out minimizing the final size, the cost and to provide all the functionalities, compliant for a portable low power device. The PCB is divided into the front-end and the microcontroller areas, the two sections continuously interface and communicate with each other. In the former, the sensor connector is placed on one side of the board, and the response signal travels through the dedicated circuits. The event-based waveform represents the input of the processor interface. Whereas, the opposite route is covered by the PWM signal addressed to the voltage reference control. In the same side of the board, the mechanical ON-OFF switch is mounted directly connected with the circuit for the voltage regulation. In the latter region, the need components for the microcontroller and the user are allocated, such as LEDs, control-button and reset mechanical switch, finally, exactly on the opposite side of the sensor, the printed antenna is placed. The firmware dedicated to The ARM® Cortex®-M4 32-bit processor mounted on the board is dedicated to coping all the needed tasks, to drive the cell and thus the precise PWM generation sequence, and to correctly measuring the event-based square waveform exploiting coded interrupt routine. Moreover, to respect the idea to obtain a portable device, the Bluetooth® 5 Low Energy SIG (BLE) protocol is exploited and the associate firmware developed. The innovative BLE allows for wireless communication and low power consumption adapted to the low amount of data transmission. The chosen solution has covered all the main required features, and the printed onboard antenna allows the communication at radio frequency at 2.4 GHz. Finally, the receiver firmware and terminal graphic user interface developed permits easy data management and elaboration. The firmware and the PCB are tested electrically and functionally to ensure the correct behaviour and the measured data elaborated with the support of Matlab.
Table I presents the feature of the system, on the full load it consumes 62.7 mW, in idle 29.7 mW and the received signal strength indication with a range value compliant with the BLE specifics:

	Min	Max
Voltage supply	3.3 V	7 V
Idle Current	9 mA	13 mA
On Current	19 mA	24 mA
Idle Power	29.7 mW	42.9 mW
On Power	62.7 mW	79.2 mW
BLE RSSI	-95 dBm @ 18 m	-51 dBm @ 0 m

Table 1: Electric and communication features of the device

To validate the system, electrochemical experiments are finally performed. Four different samples catheterized by different concentration of paracetamol as benchmark drug have been prepared. The board is connected to a commercial SPE electrode through support and the zero concentration, the 100 μM, the 200 μM and the 300 μM are analyzed. For each concentration, a clean electrode is exploited and the data collected. The post-processing results are summarized in Figure 6.11:

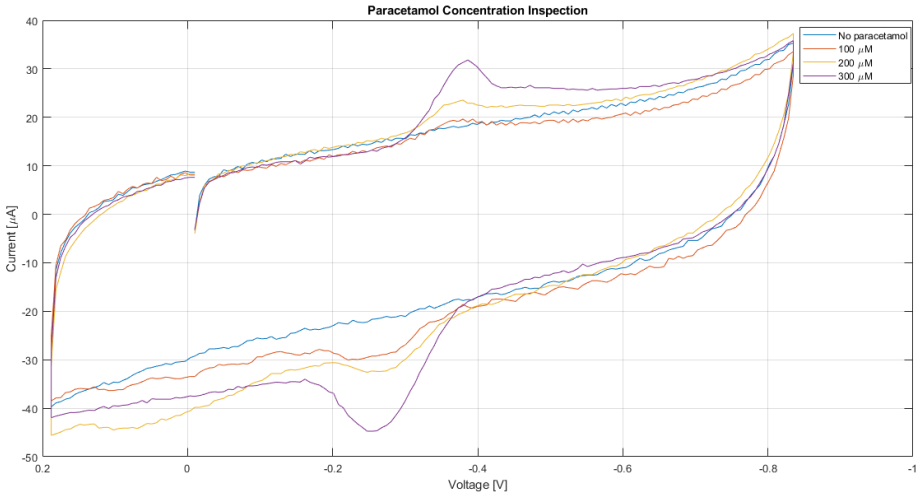

Figure 1: Test results: Voltammogram with the current response vs. driven electrode voltage.

In conclusion in Figure 6.11, it is possible to observe the expected results consistent with the state of art For each different paracetamol concentration, it is possible to observe two corresponding peaks. The peaks stand for the Reduction and Oxidation reactions. The increasing (with respect to the concentration) of the peak height, furthermore, is the indicator of the goodness of the device operation. Possible further development could consist in the evaluation and test with all the anaesthetic drugs element, and the design of the final device with the electrode integrated into a unique board and case for a final compact solution.

Contents

List of Figures

Chapter 1

Introduction

Anaesthesia or anaesthesia is a Greek term ("without sensation"), exploited to indicate a temporary, reversible and controlled loss of sensation or consciousness that is induced for medical purpose.[1] The anaesthesia is fundamental in medical practices to avoid pains to the patient exposed to medical surgery. During the operation, the patient is kept partially or totally insensitive to external stimulus. According to the different kinds of medical operation, the anaesthesia procedure can be:

- General Anesthesia (GA): suppression of the central nervous system activity and total lack of consciousness, through an intravenous liquid or inhalation gas.
- Sedation: less suppression of the central nervous system with respect to the GA, inducing inhibition of anxiety and creation of long-term memories (without unconsciousness).
- Regional and local anaesthesia: interdiction of nerve impulse transmission from a specific part of the body. The drug can be targeted at peripheral nerves to isolate locally the body.

1.1 Anaesthesia Drugs Composition and Interaction

In the anesthesiology, the drug composition and the interaction with the body, represent a cornerstone to guarantee the success of this procedure.
In order to define an anaesthetic process, not only based on the knowledge and the experience of the doctors and professors, it has been introduced the "magic formula" to administer the cocktail of a drug correctly in the largest part of the surgical operation. The model adopted for the description and balance of the main components of the anaesthesia takes the name of the triangle (AT introduced by Gray) [2] and it represents the key of anaesthesia. In the last year, the evolution of

the model and the technology has resulted in the focus on some aspects: concentration, targets, probability of not response and synergism, summarized with the term "interactions", the main idea at the base of safe anaesthesia.[2]
In sum, there are three basic components, opioid, hypnotic and relaxant, which presence in the body may cause different interaction: pharmaceutical, pharmacokinetics, pharmacodynamics and thermodynamics. The study of the aforementioned interactions allows the administration of anaesthesia with a multimodal approach that is safer and reproducible.

1.1.1 Anesthetic compound

The drugs component takes the name of "Triad of anaesthesia" and it is fundamental to define the correct balance of them in order to reduce the risk and side effects. Each component plays a different role at the body level, here a brief description is reported:

- Propofol: induction and/or maintenance of anaesthesia as the sedative and hypnotic component of balanced anaesthesia (depolarizing and nondepolarizing skeletal muscle relaxants, benzodiazepines, anticholinergic agents, opiate analgesics, inhalation and/or a regional anaesthetic) or total in patients undergoing inpatient or outpatient surgery.[3]

- Midazolam is a medication used for anaesthesia, trouble sleeping, severe agitation and procedural sedation .[4] It works by inducing sleepiness, causing a loss of ability to create new memories and decreasing anxiety.[4]

- Acetaminophen also called N-acetyl para-aminophenol or paracetamol is one of the most widely used over-the-counter analgesic and antipyretic agents. Studies have shown that acetaminophen lacks peripheral anti-inflammatory properties. It may be that acetaminophen inhibits the COX pathway in the central nervous system but not peripheral tissues. The reduction of the COX pathway activity by acetaminophen is thought to inhibit the synthesis of prostaglandins in the central nervous system, leading to its analgesic and antipyretic effects. Other studies have suggested that acetaminophen or one of its metabolites also can activate the cannabinoid system, contributing to its analgesic action [5].

- Opioid, common anaesthetic specific uses for opioids that have been FDA approved include use during almost every phase of surgery, including use during pre-induction for chronic pain conditions, induction of anaesthesia, maintenance, as well as to reduce immediate postoperative pain and decrease agitation. Opioids characteristically exert their effects by interacting with the few types of opioid receptors in the body. These interactions may result in a range

of receptor responses from inducing greatest receptor activity to no activity at all. Those medications that induce the most profound receptor response are referred to as agonists, while those inducing a partial response are known as partial agonists, and those which induce no activity are described as antagonists. These receptors are known as the mu-opioid receptor, delta-opioid receptor, and gamma-opioid receptor Opioid receptor-like (ORL1) receptor is also considered to be an opioid receptor system [6].

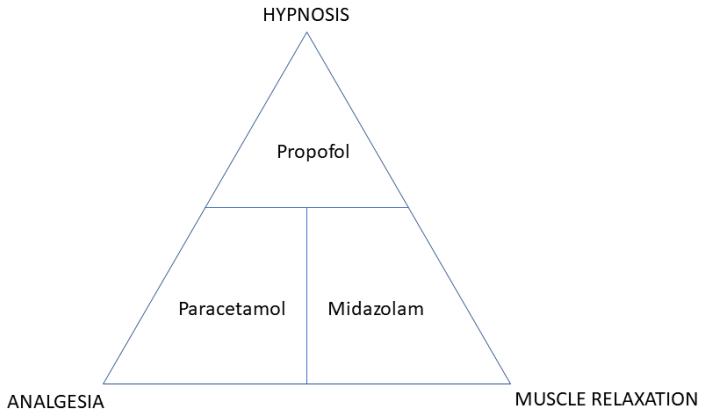

Figure 1.1: Triad of anesthesia

1.1.2 Interaction classification

- Pharmaceutical interactions "(PIs) are changes in the physical-chemical structure of a drug due to the action of a second drug when combined in the same solution, whether in a bag, a syringe, or in a Y-infusion system"[2]. This kind of interactions provides information about the compatibility between two or more drugs mixed for anaesthesia purposes for instance opioids and hypnotics components.

- Pharmacokinetic interaction takes into account the possible atypical behaviour of certain medicines, it is usually found in infusing remifentanil and propofol[7].

- Pharmacodynamic interaction is about the simultaneous administration of anaesthetic agents acts on various receptors resulting in different PDIs[7]. In

the additive case when the actions of components are similar on the patient; in the synergistic interaction the actual effect of drugs interaction is higher with respect to the expected; the last case, the opposite of synergistic, occurs when the real effect results lower than the expected one.

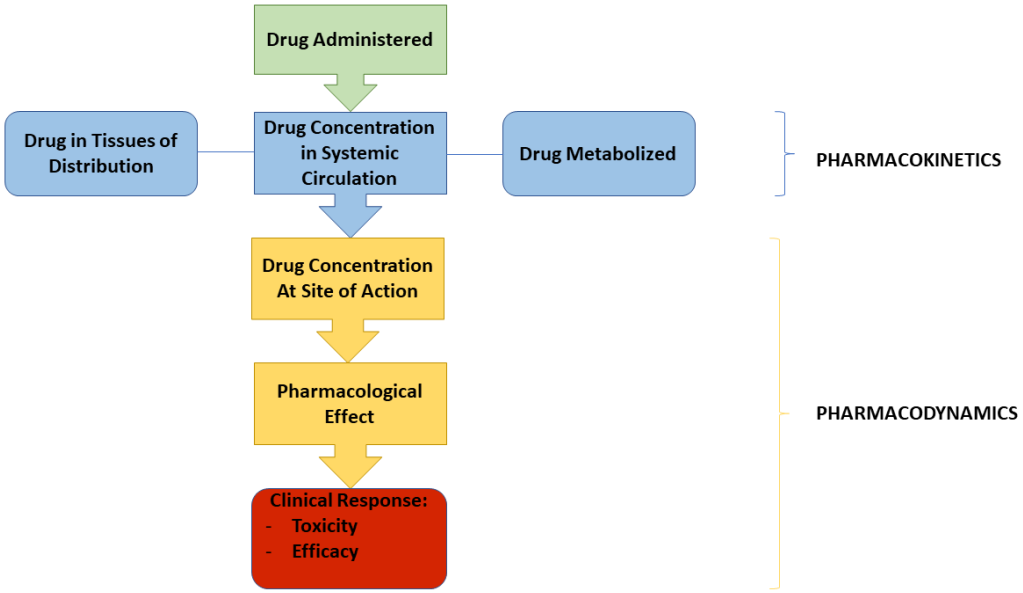

Figure 1.2: Pharmacodynamics vs. Pharamcokinectics

1.1.3 Risk and side effect

The anesthesia, as said, is exploited in many medical operation and plays an important role for the success of that, without unnecessary suffering. For this reason, the correctness of the process is fundamental for the patient safe. However, the percentage of error and side effect or risk is not zero, the main ones could be summarized as temporary confusion and memory loss, dizziness, nausea and vomiting, shivering and feeling cold, difficulty passing urine. Even if, general anaesthesia is safe in the large common case, sometimes there are complications linked to different causes.

According to [8] the most common causes of mistake in anaesthesia are the "Wrong drug", "Wrong route", "Wrong order" and "Wrong dose" (see 1.3).

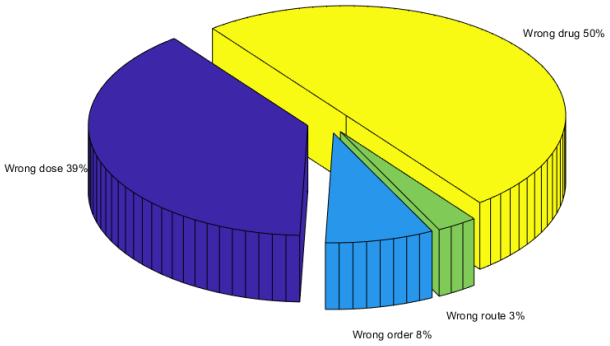

Figure 1.3: Classification of drug administration errors in anesthesia malpractice cases

The first three causes, above listed, are mainly connected to the attention and professionalism of the anesthesiologist. The order, the route and the drug type correctness are patient independent and must be carefully executed at the begin of operation. On the other hand, the "wrong dose" aspect can vary with respect to the patient and during the operation causing risk of overdose (and all the linked consequence), low-dose (possible pains) and wrong mix-dose. At the aim to help the doctors to minimize all the risk linked to the Wrong dose, a real-time support system can be placed side by side during the whole surgery duration.
The thesis work is focused on the design for a possible solution to this common problem. The project, described in the next chapters, pose the attention on the development of an electronic system able to monitoring in a continuous way, the response of an electrochemical sensor associated to the measure of drugs concentration in the patient's blood.

1.2 Drug Monitoring - State of the art

Nowadays, there are no commercial tools able to offer real-time monitoring of anaesthetics, indeed, there is still a lack in sensing technologies able to maintain high performances in long term monitoring within a portable miniaturized hardware system [9]. The monitoring of the patient during the administration of drugs per anaesthesia reasons (not only) is a crucial task required to avoid and to reduce potential health dangerous risk.
Personalized therapy offers the opportunity to increase the therapeutic efficacy of drugs by targeting the right dosage for each patient and, at the same time, by decreasing toxicity due to overdosing [10]. The main goal of this medical approach

is to maintain, ensuring patient-safe condition, the concentration of drug dosage in blood within a pre-defined range, in order to optimize the efficacy in patients treatments and reduce the toxicity for the body. At this scope, during anaesthesia, accurate balanced delivery of several compounds, including anaesthetics, analgesics and muscle relaxants is fundamental to achieve to avoid intoxication or awareness. The traditional methods evaluate the Depth of Anesthesia (DOA), very difficult-evaluation parameter, basing the approach on the observation of human parameter as physiological symptoms such as patient's heart rate and blood pressure. However, these parameters vary a lot depending on the type of surgery and the patient, then they are inadequate and with very poor predictive value[10]. Algorithms based on Bispectral Index (BIS) are actually used in surgery to monitor the DOA of the patient. The BIS is a weighted sum of several electroencephalographic (EEG) parameters and its value decreases linearly with the increasing of the DOA[10]. Unfortunately, also these techniques return wrong estimations in clinical situations due to abnormal EEG patterns or to different anaesthetics or interference by other drugs. [10] Currently, other technologies such as "mass spectrometry", liquid or gas "chromatography" and "immunoassay" [10], are exploited as well as EEG they result not suitable for Continuous Anesthetic Monitoring. In this thesis, in order to guarantee the commercial tools able to offer real-time monitoring of anaesthetics, the electrochemical biosensor approach is adopted.

1.2.1 Electrochemical sensor

Electrochemical sensors are considered as better candidates for the realization of such portable monitoring systems with respect to standard bulky techniques nowadays adopted [11]. The sensor cell is composed of a sensing part based on three-electrodes. Typically, the name of these three electrodes is WE (Working), RE (Reference) and CE (Counter) electrodes. The voltage potential between the RE and WE is kept constant thanks to the potentiostat circuit; the current is collected by the CE contact whereas the reduction and oxidation reactions take place on the WE electrode. The cell potential between RE and WE is distinct for each analyte, allowing specific compound recognition while the current through CE permits the quantification of such analyte.[11] In the system developed in order to drive the cell the Cyclic Voltammetry technique is applied. The method consists in the bias the cell with a linear-sweep potential ramp within a defined voltage range, and simultaneously measure the current provided by the cell at each voltage step, corresponding to a certain drugs concentration.
Among many electrochemical techniques are feasible, but Cyclic Voltammetry (CV) is the most suitable for detecting multiple compounds at the same time [11].

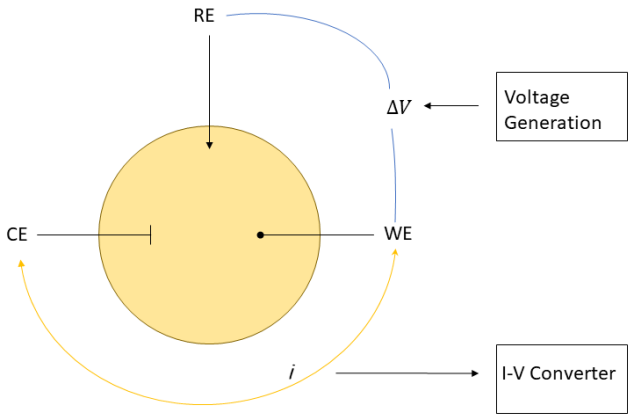

Figure 1.4: Electrochemical cell scheme

The chemical reaction in the cell is based on two half-reactions at the two electrodes where the potential difference is present between electrode and electrolyte. Generally, only one of the two half-reactions is considered and the electrode where it occurs. Driving the electrode towards more negative potentials causes an increase of electrons energy up to the value at which the electron is transferred to the electrolyte. This aforementioned case is called *Reduction* and the current flows from the electrode to the solution. In the opposite potential configuration, *Oxidation* the electron energy decreases and the current follows the opposite direction (w.r.t. reduction case).
In conclusion, the measured current flows WE and CE (the CE electrode should have a larger surface with respect to WE to avoid RedOx limitation kinetics); the potential difference is evaluated between RE and WE

RedOx chemical reaction in electrochemical cell

This section is dedicated to better explain, in the previous chapter mentioned chemical reaction exploited to measure the substance density. The electrochemical cell is composed of two half cell, one in which the oxidation of metal electrode occurs and the other where the reduction of metal ions in solution is considered. The reaction is (as previously said) called RedOx, it is a chemical reaction characterized by the transfer of electrons between chemical species: the reducing agent and the oxidizing one. The reducing agent loses electrons, gained by the oxidizing one.

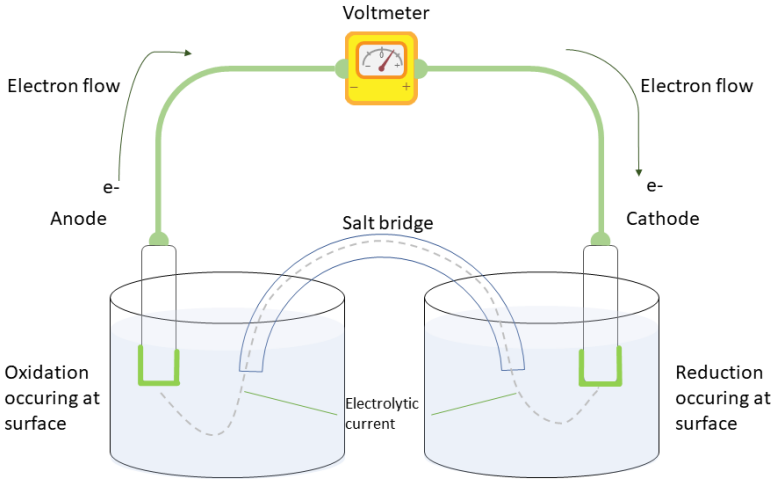

Figure 1.5: Redox and electron flow in Electrochemical cell

Each half-reaction has a standard electrode potential, it is equal to the potential difference at equilibrium (under standard conditions) of an electrochemical cell in which at the cathode occurs the considered half-reaction, at the anode the reference standard *Hydrogen electrode*, where hydrogen is oxided (losing electron). The standard cell potential is described by the following relation 1.1

$$E^{eq}_{cell} = E^{eq}_{cathode} - E^{eq}_{anode} \tag{1.1}$$

The electrons flowing is generated by the potential in electrochemical cells between the anode to become oxidized and the potential for the cathode to become reduces. The electrons fall from the anode (higher potential) to the cathode (lower potential). In conclusion, the electron flow generates the current which is the electrical parameter measured to evaluate the substance concentration in a fluid.

Potentiosat

The precise control of the electrochemical cell, during chemical analysis, represents one of the most important points in electrochemistry. Several techniques are available but the best in term of stability and control is given by potentiostat one. The potentiostat circuital solution has mainly two tasks: the control of the applied voltage to the cell and the measure of voltage response or current one (as in the thesis case). Other kinds of techniques result to be unstable and impedance dependent[12]. The potential control circuit is designed to control the potential at the working electrode (WE), and obtain a measure of current at a stable and constant voltage. The goal is obtained in common solution with a single op-amp connected

to the three-electrode cell, where the voltage is applied to the counter electrode (CE), which current compensate the RedOx reactions at WE. The RE electrode is connected in the negative feedback loop and the final configuration is 1.6:

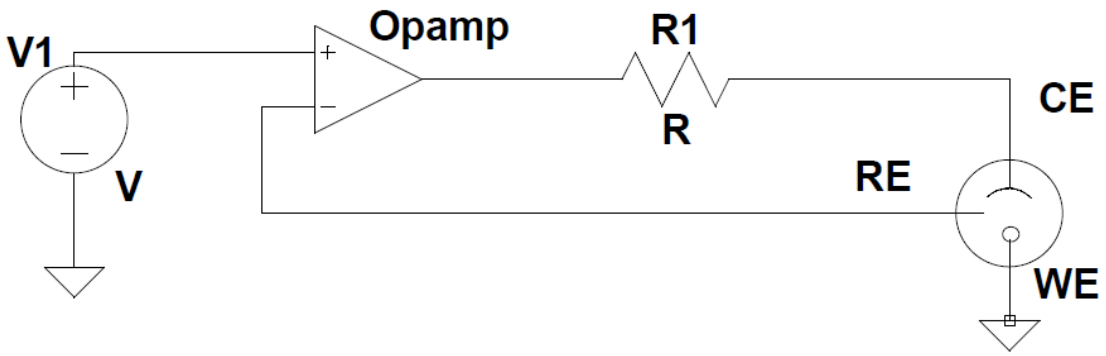

Figure 1.6: Potentiostat circuit configuration

The aforementioned circuit configuration plays an important role in the so call voltammetry-based-electro- analytical tools to obtain information about the RedOx process. Usually, the current measured in the CE and WE loop is reported versus the applied potential realizing the voltammogram which peculiar shape depends on the solution with current peaks relating to the phenomena[13]. The applied approach in the thesis analysis consists of the Scan Cyclic Voltammetry (SCV), where the driving voltage ramp between the WE and RE is generated through a staircase of voltage step[13].

Chapter 2

Circuit design

The potentiostat is an electronic circuit configuration able to correctly control a three electrodes cell in order to run electroanalytical experiments. The system functions by maintaining the potential of the working electrode at a constant level with respect to the reference electrode by adjusting the current at an auxiliary electrode. Besides the voltage control of the cell, the potentiostat plays the central role in the current flow measure among the Working electrode and Counter one.

2.1 Analog driver

In this section is reported in details the description of the circuit configuration aims to properly drive the electrochemical cell and acquire the sensor data.

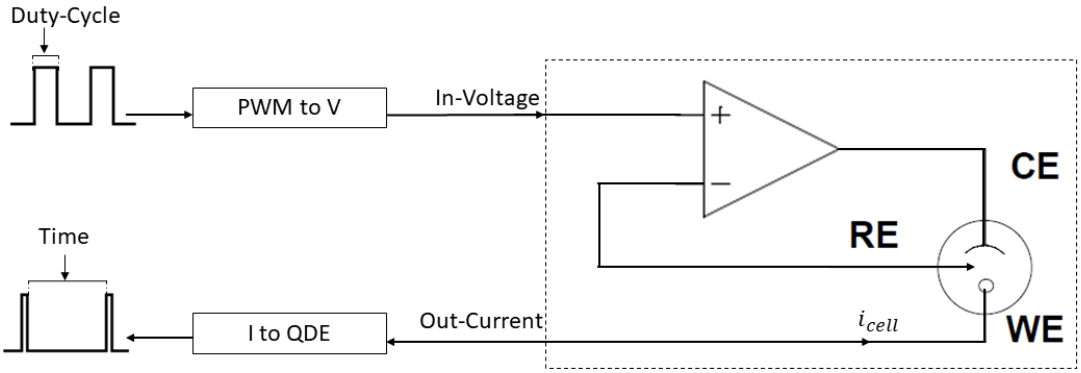

Figure 2.1: Schematic of Potentiostat architecture for the voltage control and current conversion

As shown in figure 2.1 the double tasks of the potentiostat results to drive the cell, by means a Pulse Width modulation that allows step-by-step voltage reference between WE and RE value, changing the duty-cycle of the square wave;on the other

hand the current flow among WE and CE electrode is properly converted to a quasi-digital signal, to exploit the event-based approach avoiding power-consuming ADC [9].
From the circuital point of view, the potentiostat consists of an electric circuit which is usually described in terms of simple op-amps.[14]

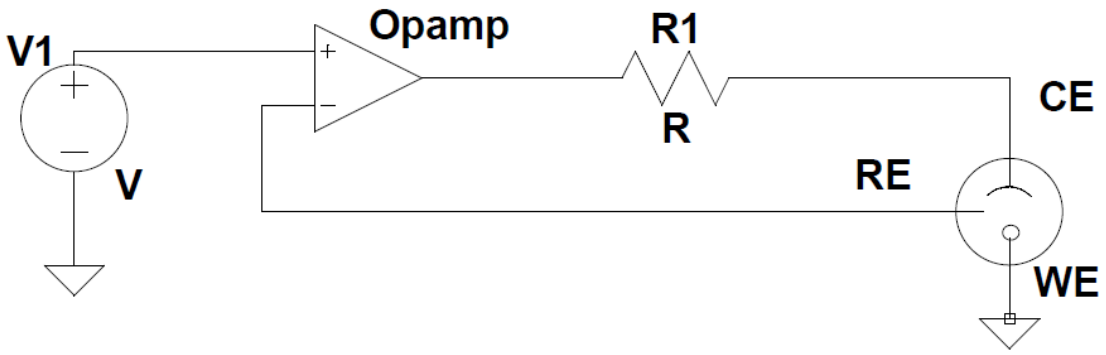

Figure 2.2: Potentiostat circuit configuration

2.1.1 Pulse width modulation and Voltage converter

Pulse width modulation (PWM) is a method of digital modulation which allows the obtaining of a variable average voltage based on the ratio among the time of the positive pulse with respect to the period of the square wave (duty cycle).
This technique has been proposed in the circuit implementation in order to provide a linear sweep voltage value to feed the electrochemical cell.
In figure 2.3 is reported the electronic circuit that receives the PWM signal with variable sequence of Duty-Cycle (waveform generated by the $\mu Controller$),the input enters in the filter circuit and the output(V_{out}) results in a voltage-ramp shape (as function of the input duty cycle), act to drive the chemical cell.

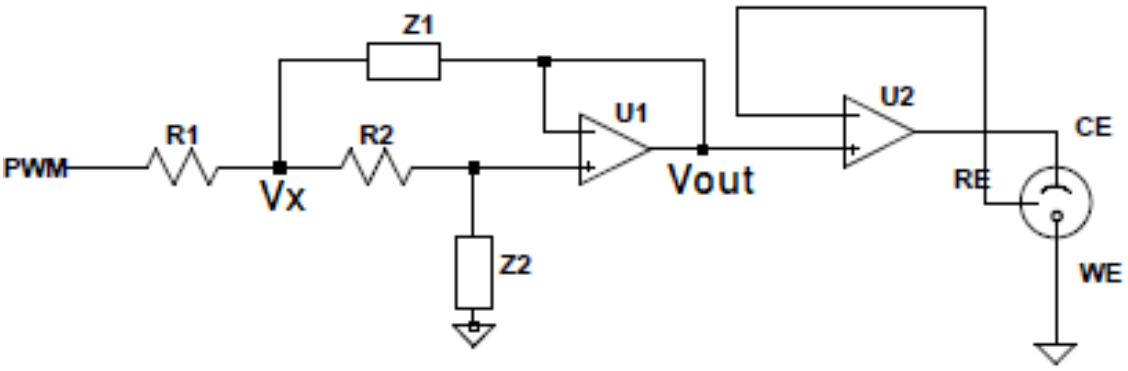

Figure 2.3: Circuit configuration to drive the electrochemical cell: PWM input signal and integral filter circuit

The input PWM signal, is filtered through the 2^{nd} order integrator circuit, maintaining only the DC harmonic component and rejecting the other contributes. In particular in figure 2.3 is described the *Sallen-Key* topology. The formula to describe the relation between the input and the output is reported in the following equations 2.1, 2.2, 2.3, 2.4:
By inspection the circuit with the K.C.L.

$$\frac{PWM - V_x}{R_1} = \frac{V_x - Vout}{Z_1} + \frac{V_x - V-}{R_2} \tag{2.1}$$

$$\frac{PWM - V_x}{R_1} = \frac{V_x - Vout}{Z_1} + \frac{V_x - V-}{R_2} \tag{2.2}$$

and

$$\frac{V_x - V_{out}}{R_2} = \frac{Vout}{Z_2} \tag{2.3}$$

Combining and rearranging the equation:

$$\frac{V_{out}}{PWM} = \frac{Z_1 Z_2}{R_1 R_2 + Z_1(R_1 + R_2) + Z_1 Z_2} \tag{2.4}$$

The equation 2.4 shows the transfer function, by replacing to Z_1 and Z_2 the value of the capacitor impedance, in the Laplace domain $s = j\omega t$, the final 2^{nd} order transfer function is available.

$$\frac{V_{out}}{PWM} = \frac{1}{1 + C_2(R_1 + R_2)s + C_1 C_2 R_1 R_2 s^2} \tag{2.5}$$

The design of the filter depends on two parameter Q and f_0, respectively the shape of the frequency response and the resonant frequency, to be chosen appropriately according the application. The definition of the two factors are:

$$f_0 = \frac{1}{2\pi\sqrt{R_1 R_2 C_1 C_2}} \tag{2.6}$$

$$Q = \frac{2\pi f_0}{2\alpha} = \frac{\sqrt{R_1 R_2 C_1 C_2}}{C_2(R_1 + R_2)} \tag{2.7}$$

For example the Q factor in order to obtain the maximum flat band response must be selected equal to $\frac{1}{\sqrt{2}}$.
At the end of the design, the unknowns (4) and the two parameters definition requires the imposition of a relation between R_1 and R_2 with R and C_1, C_2 with C by means of **m** and **n** .
In conclusion the relation to respect are:

$$R_1 = \mathbf{m}R; R_2 = \frac{R}{\mathbf{m}}; C_1 = \mathbf{n}C; C_2 = \frac{C}{\mathbf{n}}. \tag{2.8}$$

In order to "transform" the PWM signal in voltage ramp, the pole position must be chosen much lower with respect to the modulation frequency, the output integrated signal will be

$$V_{out} = V_{dd} \cdot DC \tag{2.9}$$

with DC equal to the modulation duty cycle.

2.1.2 Design proposal and LTspice Simulation

Exploiting all the considerations done in the previous section, the electrochemical cell driver has been designed and then simulated in LTspice. The resistance R_1 and R_2 have been chosen equal to 47 $k\Omega$ and 10 $k\Omega$ and the two capacitance $C_1 = 47$ nF and $C_2 = 10$ nF. The input signal has been modelled with a variable pulse in the time at frequency $f = 20$ kHz, simulating a signal with a variable duty cycle.

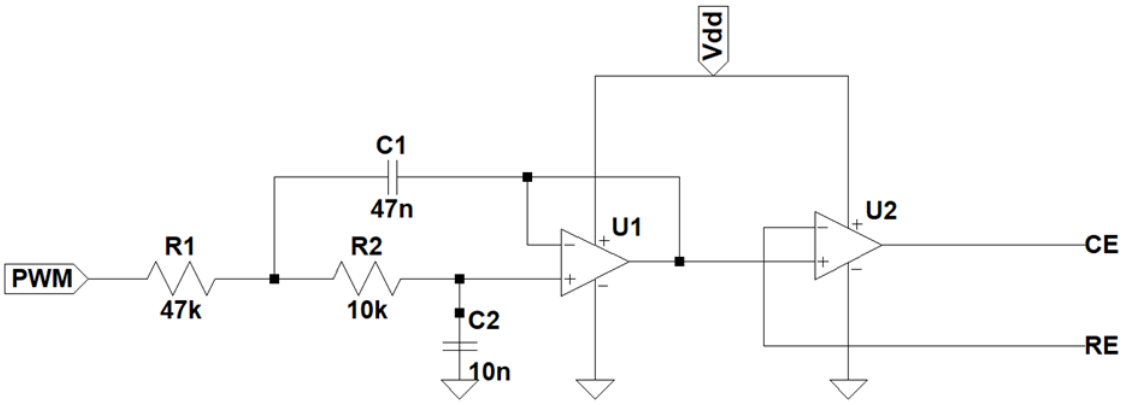

Figure 2.4: Simulation layout configuration

To obtain a more clear picture of the circuit behaviour, a single Duty Cycle per time has been simulated and here reported.

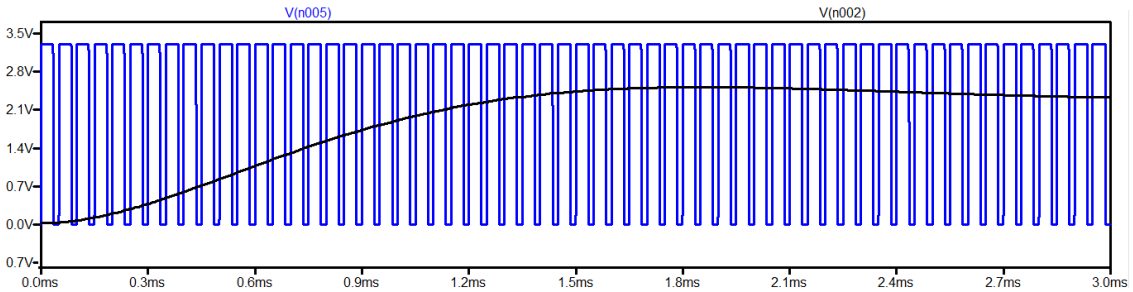

Figure 2.5: Input voltage signal (blue square wave) and Output voltage signal (black wave) with 70% of duty cycle

In figure 2.5 is possible to note the relation between the output voltage value and the input one, linked by the Duty cylce, indeed, $V_{out} = DC \cdot Vin = 0.7 \cdot 3.3 \simeq 2.31V$.

In order to observe the correctness of the behaviour of the circuit, it is useful to compare figure 2.5 with the one reported below 2.6. In the second case the duty cycle is equal to 30 %, indeed, the output voltage results to be lower than the first case following the expected behaviour.

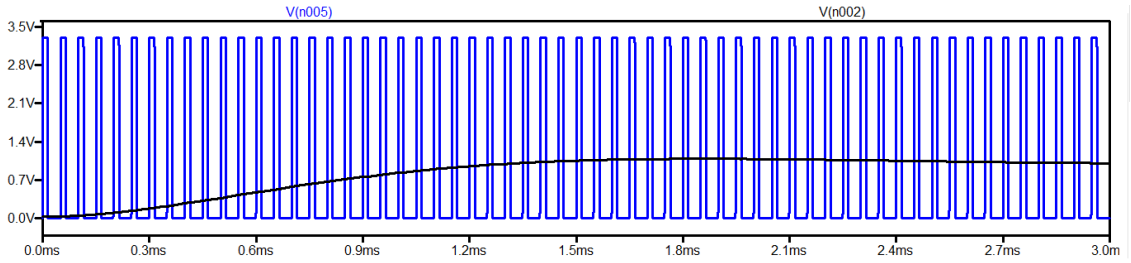

Figure 2.6: Input voltage signal (blue square wave) and Output voltage signal (black wave) with 30% of duty cycle

In conclusion, combining at different duty cycle the input signal (the PWM purpose), it is possible to obtain a linear-sweep voltage ramp (in order to drive the electrochemical cell) from $[DC_{min} \cdot V_{dd} \div DC_{max} \cdot V_{dd}]$ Volt.

2.2 Analog Read-out

In this section is described the circuit exploited in order to convert the signal comes from the output of the electrochemical cell in suitable signal for the microcontroller. The electronic circuit can be described by splitting the whole layout into the three stages that compose the configuration : the trans-impedance amplifier, the integrator stage and the edge trigger.

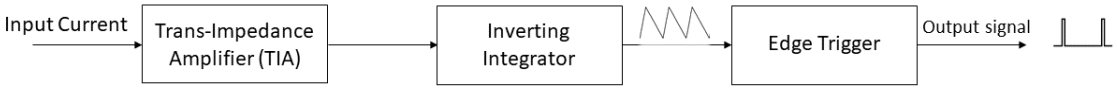

Figure 2.7: Read-out topology block diagram

2.2.1 Trans-Impedance Amplifier (TIA)

The first stage, the so call TIA configuration, is the fundamental electronic circuit aims to convert an input current signal into a voltage one. In the figure 2.8 the classical circuit configuration is reported:

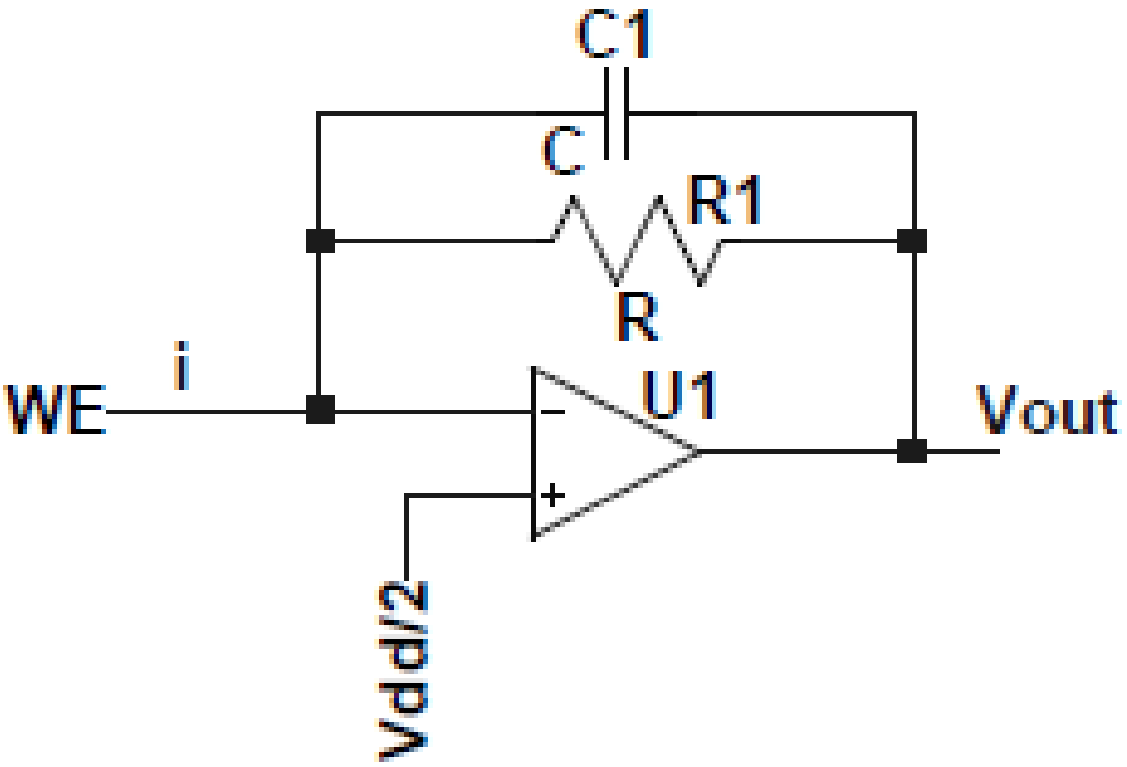

Figure 2.8: Trans-impedance amplifier configuration

The relation among the input current signal, generated by the sensor response, and the output voltage depends on the feedback resistance (in parallel to the capacitor to force the pole position).

$$V_{out} = -I_{WE} \cdot R_1 + \frac{V_{dd}}{2} \tag{2.10}$$

The current is measured from the Working electrode at each drive voltage step of the ramp voltage signal above described. This current signal derives from the oxidation or reduction of the analyte at the WE surface and it is strictly related to the presence and the concentration of the analyte itself[11]. The WE is connected to an half-supply node (Vdd/2) to achieve both a dual-voltage driving and a dual current measurement without the introduction in the system of a dual-voltage supply[9].

2.2.2 Integrator and Edge Trigger stage

Since the input signal is an analog signal, one possible circuital solution could be the employ of the Analog-to-Digital converter. This last solution has been replaced with two-stages, integrator and comparator, in order to transform the input analog signal in the so call Quasi-Digital signal [15], the event-based approach guarantees

a reduction in the term of power consumption with respect to the classical ADC solution. The circuit configuration with the two-stages is reported in 2.9

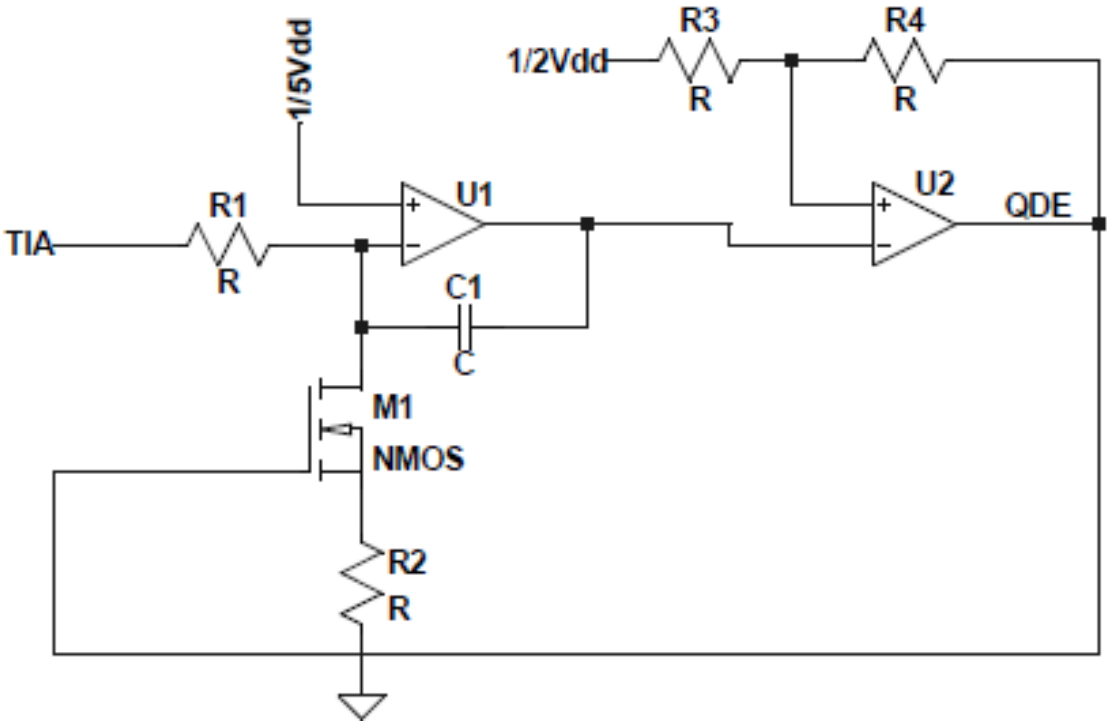

Figure 2.9: Two stages configuration: the integrator U1 and the edge trigger stage U2

The voltage output signal of the TIA becomes the input for the second and third stage. Starting from the second (integrator) stage:

$$I_c = \frac{V_{TIA} - \frac{1}{5}V_{dd}}{R_1} = -C \cdot \frac{dV}{dt} \tag{2.11}$$

Solving the differential equation, the output voltage proportional to the input one (and thus to the input current) is:

$$V_o = -\frac{1}{RC}\int (V_{TIA} + \frac{1}{5}V_{dd})dt \tag{2.12}$$

The capacitor, thus, is discharged by the current I_c and when it overcomes the threshold of the comparator (third stage) the QDE signal is set, the current start to flow through the MOSFET and the QDE rapidly is reset. This process represents the conversion loop between the input current and the output quasi-digital signal. When the integrator input is composed by a time-constant voltage

signal, the output is just proportional to the input. The integral relation 2.12 becomes:

$$V_o = -\frac{1}{RC}(V_{TIA} + \frac{1}{5}V_{dd}) \cdot T \tag{2.13}$$

Observing the equation 2.13, it is possible to describe the dependency time-current of the circuit in figure 2.10 just obtaining the input current from 2.10 and inverting the previous equation, leading to equation 2.14.

$$i_{cell} = -\frac{R_2 \cdot C_2 \cdot V_{th}}{R_1 \cdot T} + i_0 \tag{2.14}$$

Where T is the time interval, V_{th} is the last stage comparator threshold and i_0 the offset current computed when the cell current is equal to 0.

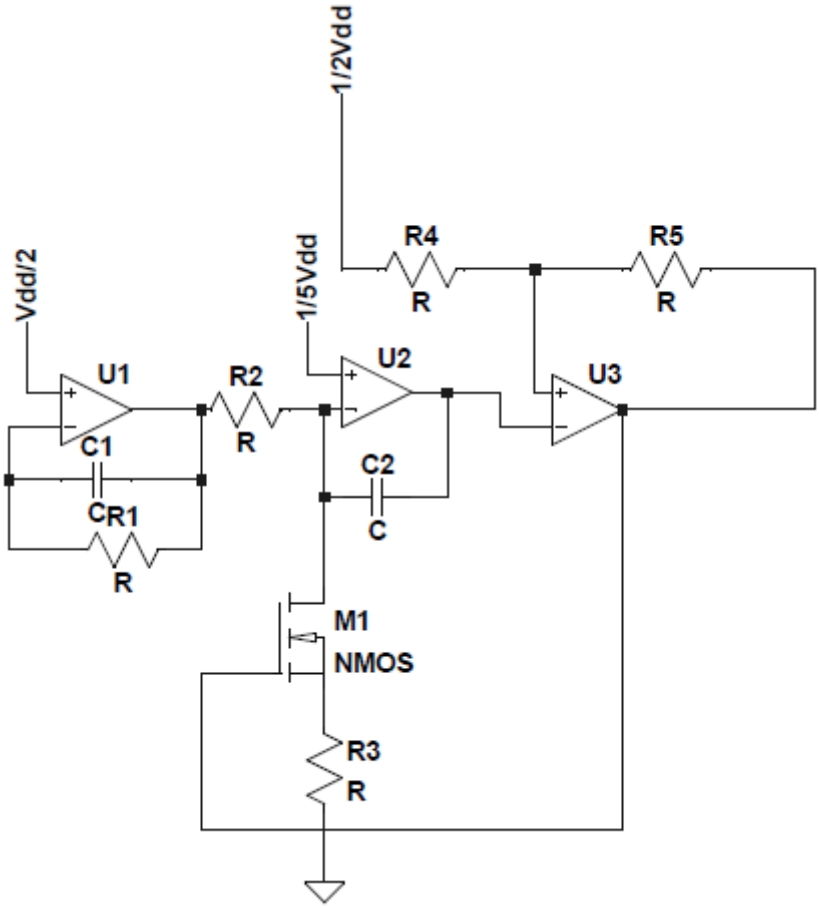

Figure 2.10: Three stage circuit configuration analog read-out

The value of T can be obtained by the following equation 2.15:

$$T = \frac{R_2 \cdot C_2 \cdot 1/2V_{dd}}{V_{TIA} - V_{low}}; V_{low} \simeq 0.6V \tag{2.15}$$

2.2.3 Design proposal and LTSPisce Simulation

Considering all the previous studies and relations, the simulation in LTspice for further validation of the configuration has been developed. In figure 2.11 is reported the layout for the simulation. The input RedOx current is modelled by a simple current generator(I_1).

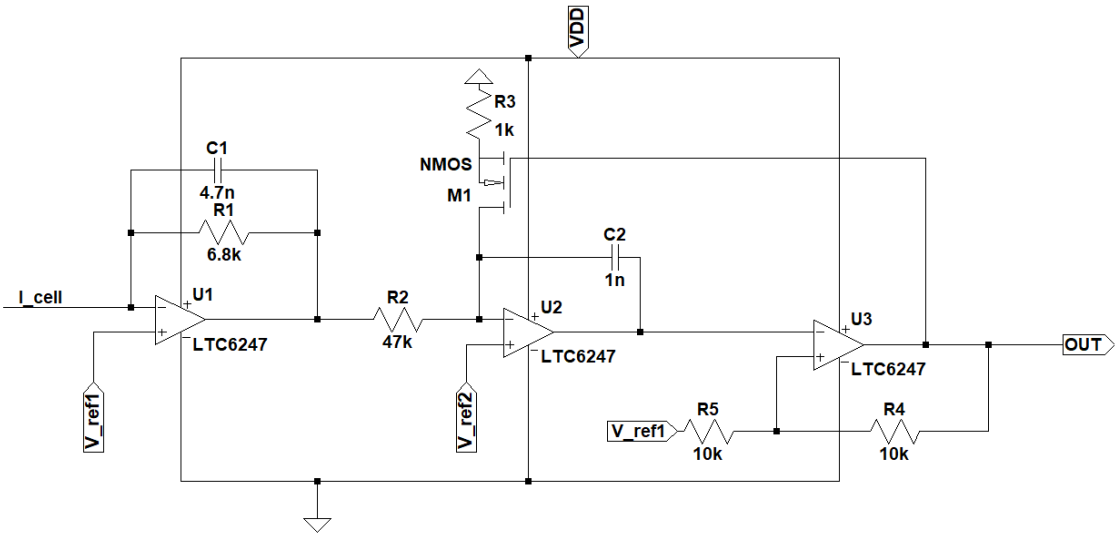

Figure 2.11: Circuit configuration analog read-out simulation in LTSpice

The results of the simulation are listed below : In figure 2.12 is reported the charge and discharge of the capacitor C_2 according to the mos current.

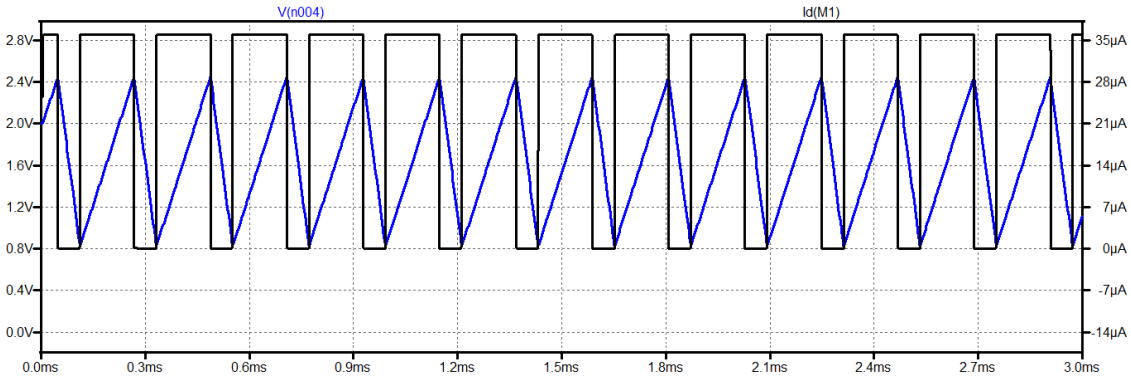

Figure 2.12: Charge and Discharge of the capacitor: current (black curve) and C_2 voltage (blue curve)

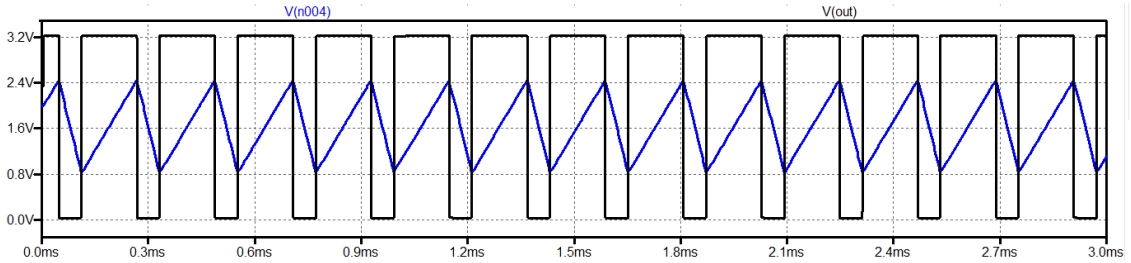

Figure 2.13: Output square quasi digital signal(black curve) and C_2 voltage signal (blue curve)

The capacitor voltage at the negative input of the third stage is compared with the threshold. The final square (quasi-digital) output in the example with input current $80\mu A$ is generated as shown in figure 2.14

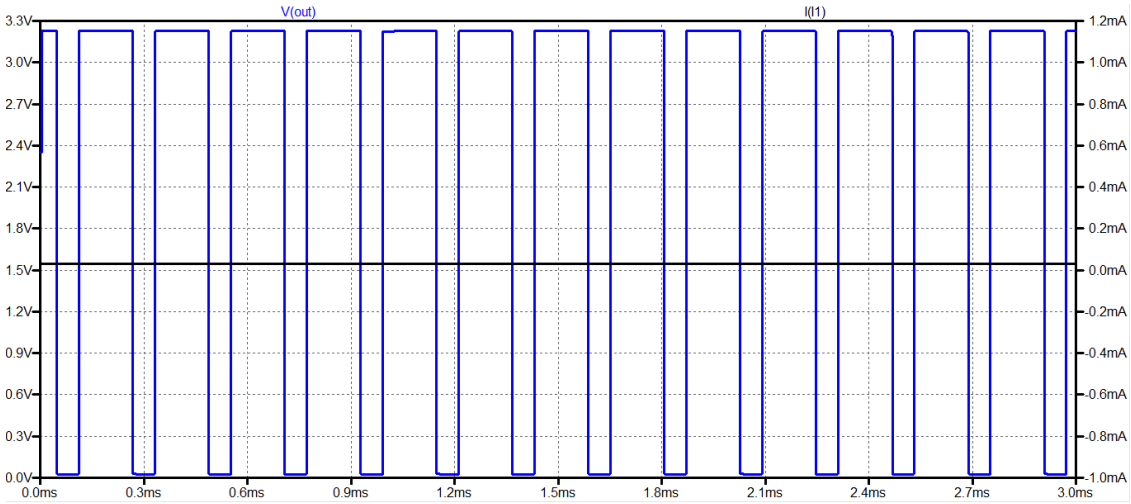

Figure 2.14: Output square quasi digital signal(blue curve) and input cell current $= 30\mu A$ (black line)

Finally, a variable current ramp signal has been exploited in order to observe the variation of event rate (evaluated as the time distance between two consecutive rise edge). Some case of input current and output event rate of the quasi-digital signal obtained from the simulation are reported here in the following table 2.1

Input cell current $[\mu A]$	Output event rate $[\frac{kevent}{s}]$
-100	4.24
-90	4.97
-70	6.73
-50	8.54
-10	12.10
20	14.75
30	15.65
40	16.54
50	17.41
60	18.32
70	19.22
80	20.14

Table 2.1: Results of the simulation in LTSpice

The general trend of the event rate vs. the current evaluation is reported in the Matlab graph 2.15 obtained exploiting the LTspice simulation data.

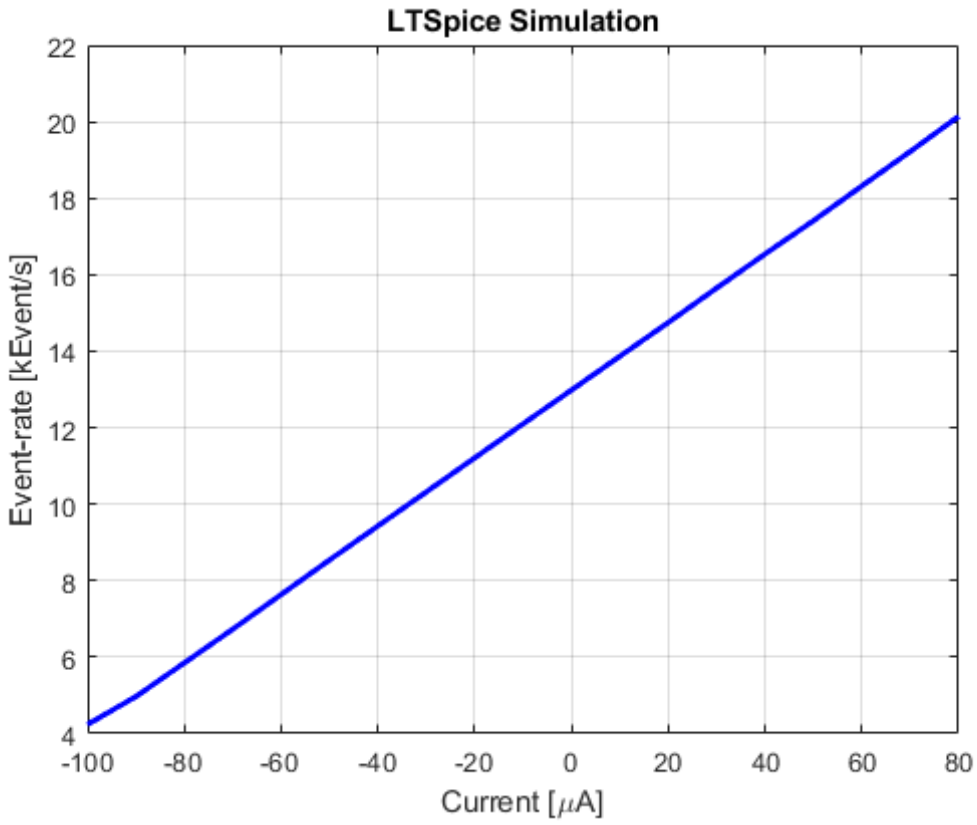

Figure 2.15: Trend of the event-rate vs input cell current in Matlab plot (simulation data)

Chapter 3

Printed circuit design: PCB

This chapter is dedicated to the description and the analysis of the design of the printed circuit board.
The printed circuit board (PCB) is mechanical support that electrically connects electronic components using conductive paths. In the thesis project, it has been chosen this kind of technology realization, because it is the best one to implement a customizable and portable device. For the final medical application, the main features required well marry with the PCB implementation. In order to obtain a possible commercial and usable medical equipment, the device has to respect some indications:

- it has to be small with a proper shape suitable for the measure of the blood substance of the patient. At this purpose, the main idea has seen the realization of a board "long and narrow" as a medical syringe, in order to facilitate the application;

- it has to be portable and at this purpose, the circuit, the components and the firmware have been selected in order to reduce the power consumption;

In this chapter will be discussing the steps for the implementation of the different parts of the project. Will be reported the details for the circuit schematics for the analogue part seen in chapter 2, the layout for the micro-controller, the realization of the antenna for the communication and all the components symbol and footprint, parameters (etc...) for the actual realization of the device.
The project for the realization of the printed circuit board, the main part of the thesis project and of the final device, has been developed employing the software "Altium designer", developed by Australian software company Altium Limited. Altium is a world-leading provider of software for the PCB and electronic design automation, for the management of the components and data. The four main functional areas are schematic capture, 3D PCB design, Field-programmable gate array development and release/data management [16]. Moreover, it has got the further

feature as integration with several components distributors, interactive 3D editing (etc.).

3.1 Analog driver & Read-out layout

In this section, it is reported in detail the Altium project relative to the analogue part described in the previous chapter. The analogue driver and the read-out circuit cover the great part of the PCB in term of area. In the following figure 3.1, it is reported the realized layout.

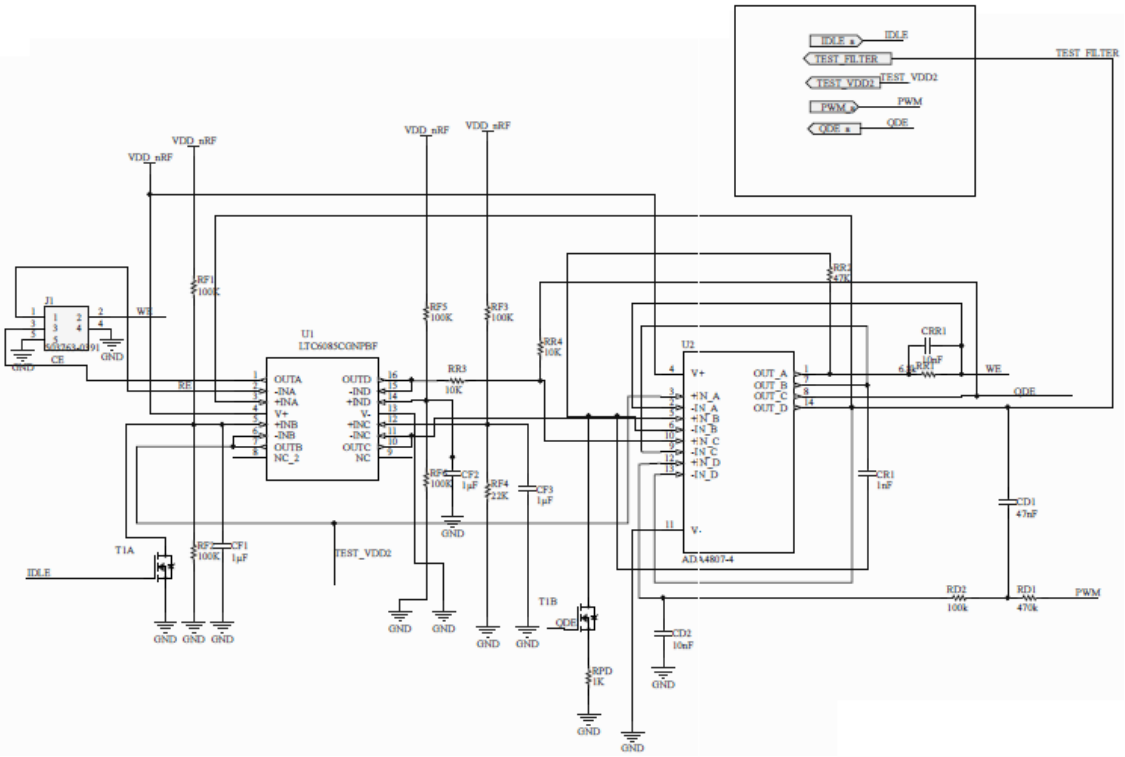

Figure 3.1: Altium configuration for the Analog driver & Read-out layout

Starting from the left of the layout, it is possible, as first, to note the connector (J1 in Altium), this component represents the connector for the input electrochemical cell from WE, RE and CE electrodes. From this part of the project and for all the PCB design, an important role has covered by the components selection, in term of availability and dimension. The connector J1 has been chosen for its small dimension (see later section 3.6). Continuing toward the right, the first integrated component is present(U1), this component as the second one (U2) is composed by four op-amps, thus compose by eight input signal, 2 input for the supply and four signal output. Always for space reason, the eight op-amps, needed to implement

the driver and read out part of the circuit, have been integrated into just two components. Moreover, it is also important to underline the main features of U1 and U2. The U1 is the amplifier series LTC6085 whereas the second one ADA4807, the difference between the two lies in the performance and power consumption. The ADA4807 has been chosen for its speed performance, indeed, that part of the circuit is addressed to the PWM conversion in linear-sweep voltage for the sensor, and the comparison task to obtain the Quasi-digital signal as output. On the other hand, to preserve power, all the stage that not needs high performance were realized with LTC6085. The following table 3.1 summarizes the two op-amp's parameters in order to compare them in terms of power and performance.

Parameter	LTC6085	ADA4807
Input bias current	1 pA	1.6 μA
Gain Bandwidth Product	1.5 MHz	165 MHz
Slew rate	0.5 $V/\mu s$	118-237 $V/\mu s$

Table 3.1: Comparison between LTC6085 and ADA4807 performance from datasheet [17] [18]

Focusing the attention on the U1 component, the LTC series, it is possible to describe the different circuital configurations implemented in the PCB. Input *-INA* and *+INA* are connected respectively to RE electrode signal and the PWM filtered signal as the circuit 1.6, and obviously the output *OUTA* linked to CE electrode signal. This first configuration follows the potentiostat circuit commented in section 2.1.1.
The *-INB* and *+INB* are attached to a voltage divider composed by two resistor with equal value (this choice turns out in an output signal equal to $\frac{V_{nRF}}{2}$); the output of the voltage follower is exploited as voltage reference for the electrochemical cell driver stage 2.8, implemented with ADA4807 (U2) due to the speed required by the operation.
As well as for the op-amp U1-B, also the U1-D input and output of the LTC6085 have been used as voltage follower once, by means of the two resistors (voltage divider), the input has been set to $\frac{V_{nRF}}{2}$. The output is connected to the *+INC* of the U2 component.
About the input *+INC* of the U1 component, the voltage divider is not act to realize $\frac{V_{nRF}}{2}$, but the *OUTC* of the voltage follower will be mirror the input voltage set to $V_{nRF} \cdot \frac{RF_4}{RF_4+RF_3}$, obtaining $\frac{V_{nRF}}{5}$ exploited as input in U2 *+INB* as described in figure 2.9.
Before to enter in details with the U2 four op-amp, it is possible to observe the N-MOS in common source configuration, with the drain linked to *+INB*. The aim of the MOSFET is to "turn off" the analogue circuit when needed. It is controlled on the gate by the IDLE signal, the flag is asserted by the μController when the

whole device is in the stand-by condition, in this way the first stage has the input connected to ground and the current flowing is minimized. This system has been adopted in order to reduce the power consumption and make the device low-power in standby condition.
Centring the attention, on the second U2 integrated component it is possible to analyze the circuital configuration. The input *+INA* is connected to the voltage reference, the output is negative feedback (linked to *-INA*) defines the Trans-impedance amplifier (figure 2.8), where the output voltage depends on the current from WE electrode and of course by the feedback resistance RR1.
The key part for the Read-out circuit has been implemented with the B amplifier, indeed the input *-INB* results connected to the drain of the N-MOS, driven by QDE signal in the negative feedback chain that includes *OUTC*. *+INB*, on the other hand, is connected to the voltage reference $\frac{V_{nRF}}{5}$ (as previously defined). The inner feedback circuit, with CR1, works in order to obtain the triangular voltage signal on the capacitance it-self that will be compared in the following stage (see figure 2.12.). Completing the description, the *-INC* and *+INC* define the last comparator where the positive input is connected to the voltage divider composed by the resistor RR3 and RR4 that sets the reference voltage to $\frac{V_{nRF}}{4}$ (the divider input is connected to the U1-*OUTD* equal to $\frac{V_{nRF}}{2}$); the negative node linked to the feedback capacitor, the voltage drop on CR1 is compared and finally the quasi-digital, QDE signal is obtained. In conclusion the above circuital configuration implements the circuit in figure 2.10.
The last amplifier the U2-D is dedicated to the realization of the integrator Sallen-Key filter as in figure 2.3. The RD1 and RD2 resistor with CD1 and CD2 capacitor corresponds to the resistor and impedance described in 2.3 and allow for the voltage-sweep ramp output signal (*OUTD*) modulated by the PWM wave with a variable duty cycle (set by the microcontroller).
In conclusion, the configurations analyzed in chapter two have been accurately implemented in Altium designer in order to obtain the final front-end circuit composed by the Readout and the PWM driver for the electrochemical cell.
The last mention is for the square on the right corner of figure 3.1, in which the output and input signal connection are reported. As said, the project has been done in a modular way and the final connections have been defined in the 3.5 section in this chapter. In the case of the front-end circuit the main signal were *QDE* and *PWM*, respectively output and input, furthermore there were the input *idle* signal (as above introduced) and finally two test signals, to check the correct device function, called *TEST_FILTER* and *TEST_VDD*.
In the following figure 3.2, the main aforementioned circuits, the above discussed are highlighted.

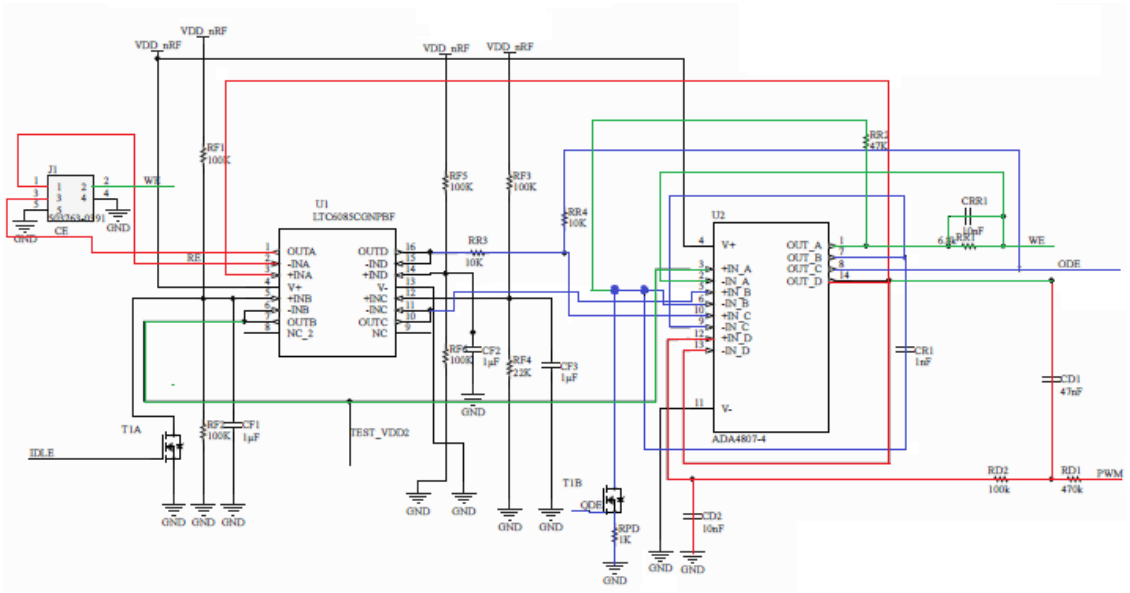

Figure 3.2: Zoom on the three main front-end parts: The red path follows the PWM driver circuit interconnection, the blue follows the Read-Out layout, the green lines follows the ìtrans impedance amplifier circuit.

The red path follows the PWM driver circuit interconnection, the blue one defines the Read-Out layout, crossed with the green lines, that point out the trans-impedance amplifier-circuit. As observable, the key tasks of the front-end in term of speed demand, are addressed to the ADA4807 op-amps.

3.2 Voltage regulator layout

The "last" part of the analog design is composed by the circuit aims to regular the voltage supply, to provide to the entire circuit, and to guarantee enough current. For the final medical application of the project, in order to realize a portable device, a battery supply will be adopted with a voltage value around 3.3V. In the developing stage also an external voltage supply has been exploited. The type of circuit configuration used for the voltage regulator is the so call *buck-boost* DC-DC converter. This kind of converter provides an output voltage with the same polarity of the input one with lower o higher value and it exploits a single inductor for both the buck and boost mode. The operation of the buck-boost is understood in terms of the inductor's "reluctance" to allow the rapid change in current. From the initial state in which nothing is charged and the switch is open, the current through the inductor is zero. When the switch is first closed, the blocking diode prevents current from flowing into the right-hand side of the circuit, so it must all flow through the inductor. However, since the inductor doesn't allow rapid current change, it will initially keep the current low by dropping most of the voltage provided by the source. Over time, the inductor will allow the current to slowly increase by

decreasing its voltage drop. Also during this time, the inductor will store energy in the form of a magnetic field.[19]

- On-state: the input is directly connected to the inductor (L). This results in accumulating energy in L. In this stage, the capacitor supplies energy to the output load.[19]
- Off-state: the inductor is connected to the output load and capacitor, so energy is transferred from L to C and R.[19]

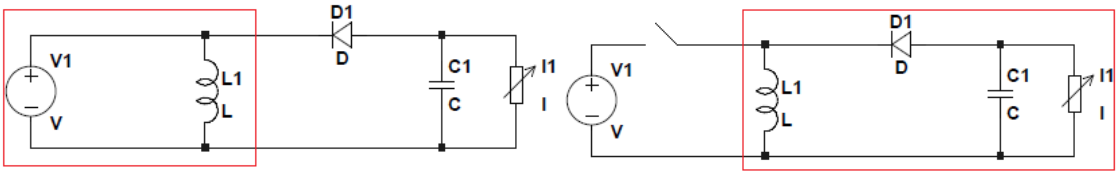

Figure 3.3: Left: buck-boost converter in on state; Right: buck-boost converter in on state

The circuit layout exploited in the PCB design is reported in figure 3.4

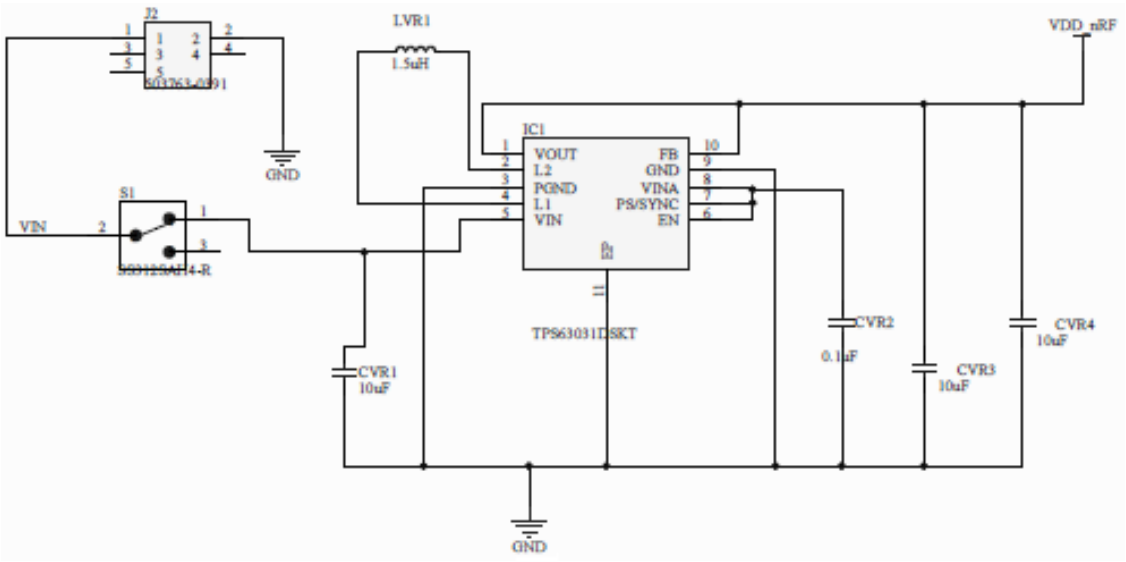

Figure 3.4: Altium configuration for the Voltage regulator circuit

First of all, it is possible to define the second connector (J2) dedicated, this time, to the voltage supply interconnection, V_{nRF} and ground reference. The connector bridges the out battery or voltage supplier with the mechanical switch (S1) through it is possible to completely switch off the device or turn on.
At this point, the circuit enters in the heart of its functionality, that is the control

of the voltage to provide to all components on the board. For this purpose, the component (IC1) has been introduced in the design, it is a voltage regulator, with fixed output voltage series TPS 63031. The whole external connections have been realized paying attention to the data-sheet indications. To better explain the circuit behaviour the inner links of the component are here reported from the data-sheet:

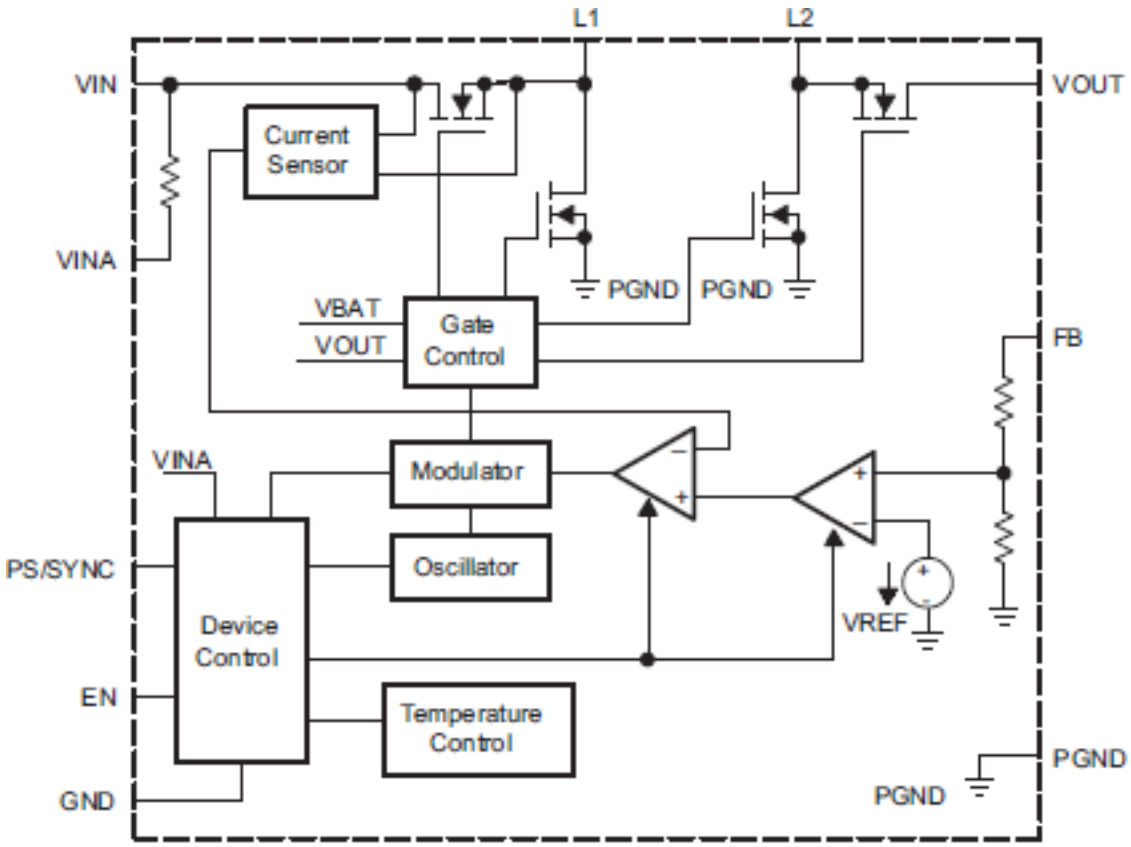

Figure 3.5: Voltage regulator circuit from datasheet [20]

The device has eight pins. Once the **EN** signal is asserted the device starts operating. The device is based on an average current mode topology, the average inductor (**L1** for both pins L1 and L2) current is regulated by a current regulator loop, controlled by means of voltage loop [20]. The average current limit ramps up from an initial 400 mA following the output voltage increases. At an output voltage of about 1.2 V, the current limit is at its nominal value, thus the output voltage overshoot at start-up, as well as the current is kept at a minimum.
To regulate the output voltage properly, the device automatically switches from step-down operation to boost operation and back as required by the configuration [20]. It operates according to the value of the input voltage with respect to the output one: if the **VIN** results higher than **VOUT**, it works as a step-down converter (buck state), and as boost converter when the input voltage drops below the output

one. Therefore **VIN** and **VOUT**, represent respectively the input and the output voltage pins. The **FB** pin works as voltage feedback and in case of constant output voltage must be connected to Vout to properly set the fixed output configuration. Finally, **PS/SYNC** pin sets different operation modes. Power safe is selected to improve efficiency at light load, in this situation the converter stops to operate if the average inductor current results are lower than 100 mA and voltage below its nominal value.
The capacitors CVR1, CVR2, CVR3 and CVR4 and the inductor LVR1 components have been selected as possible equal as the indication suggested by the data-sheet. Indeed the component, with the proposal configuration, has been tested before to be mounted, and the correct work validated.

3.3 Micro-controller layout

Once defined the front-end of the device with all the fundamental parts for the measure, for the driving and the for the voltage supply, it is possible to introduce the "mind" of the device. In this section, in particular, is described as the Micro-controller layout and functionality. The nRF52840 is the micro-controller chosen for the design (the description is postponed to chapter 4). The component (U3 in the Altium annotation) mounted in this specific case is the QIAA and it presents fifty-five pins on the 4 sides.
The layout and the connections have been executed following different reasons: the choice of the pin has been done considering first of all their functions and tasks and then considering the better way to place and route the component on the PCB. For the description of the layout, it is possible to take as reference figure 3.6 On the left top corner of the picture, in the square, the LEDs with their circuit are reported. The two LEDs, green and red, are controlled by the output pins p0.10 and p0.09. The LEDs result to be useful for mainly two reasons: during the debug in the developing phase and during the execution of the firmware in order to indicate to the operator the state in which the operation lies.
Always on the left, two buttons SW1 and SW1 are placed. The former is used to give some commands to the microcontroller, it is a mechanical user interface, the latter (SW2) is dedicated to the reset of the device, essential for the programming, moving the device in DFU mode, indeed it is connected to the RESET port of the micro-controller p0.18.
Closer to the device it is possible to observe the component X2, it is the first of two-oscillator, with the two capacitor C10 and C12. Broadly speaking, some parts, (as for the oscillator) of the circuit configuration, have been strongly realized considering the data-sheets indications and suggestions. Continuing describing the oscillator, the second one is placed on the top right corner with the name X1. The two oscillators work at a frequency equal to 32.707 kHz and 32 MHz.
Focusing the attention on the right bottom corner, it is possible to note a second

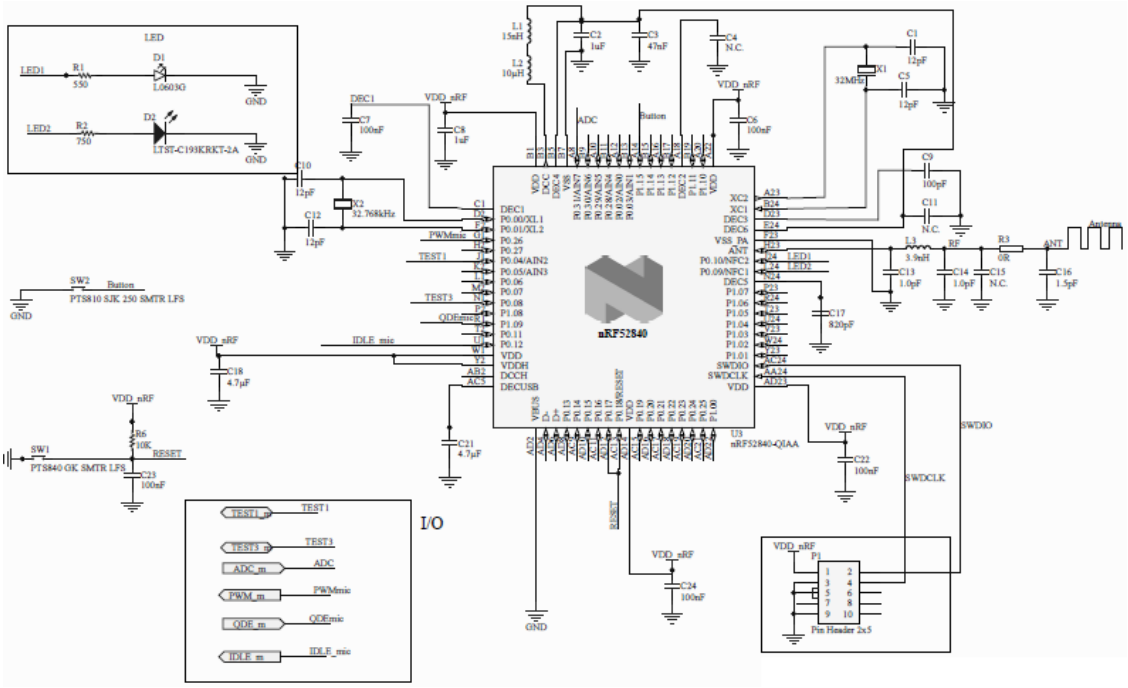

Figure 3.6: Altium configuration for the μController circuit

square, which is reported the second connector of the whole PCB (P1). The presence of this connector is fundamental to program the device with the firmware. The connector, indeed, is linked to *SWDIO* and *SWCLK* pin (furthermore linked to Voltage supplier and ground), dedicated to the programming of the microcontroller.
Finally, on the right is reported the antenna circuit: it starts from the pin called *ANT*, and it is composed by a common antenna matching π-network with L3, C13 and C14, to end with a bridge resistor (0Ω) and the antenna. The physical structure of the antenna is reported in section 3.6 of the same chapter.
The last point of the description concerns the input-output signals, reported in the square called I/O. As largely discussed, the are three main signals: QDE, PWM and IDLE, with the first analyzed in input by the nrf52840, the second and the latter transmitted by the control to the front-end circuit. The other ports listed in the square represent the test-signal described in the following section

3.4 Connection and Test layout

Finally, before to describe and show the PCB, it is possible to report the whole connection between the different parts. In order to obtain a more regular and modular design, indeed, the circuit layout has been divided in more mini-project and then connected by this circuit file as shown in figure 3.7.

The figure shows the whole circuit project as a schematic block. Each part from

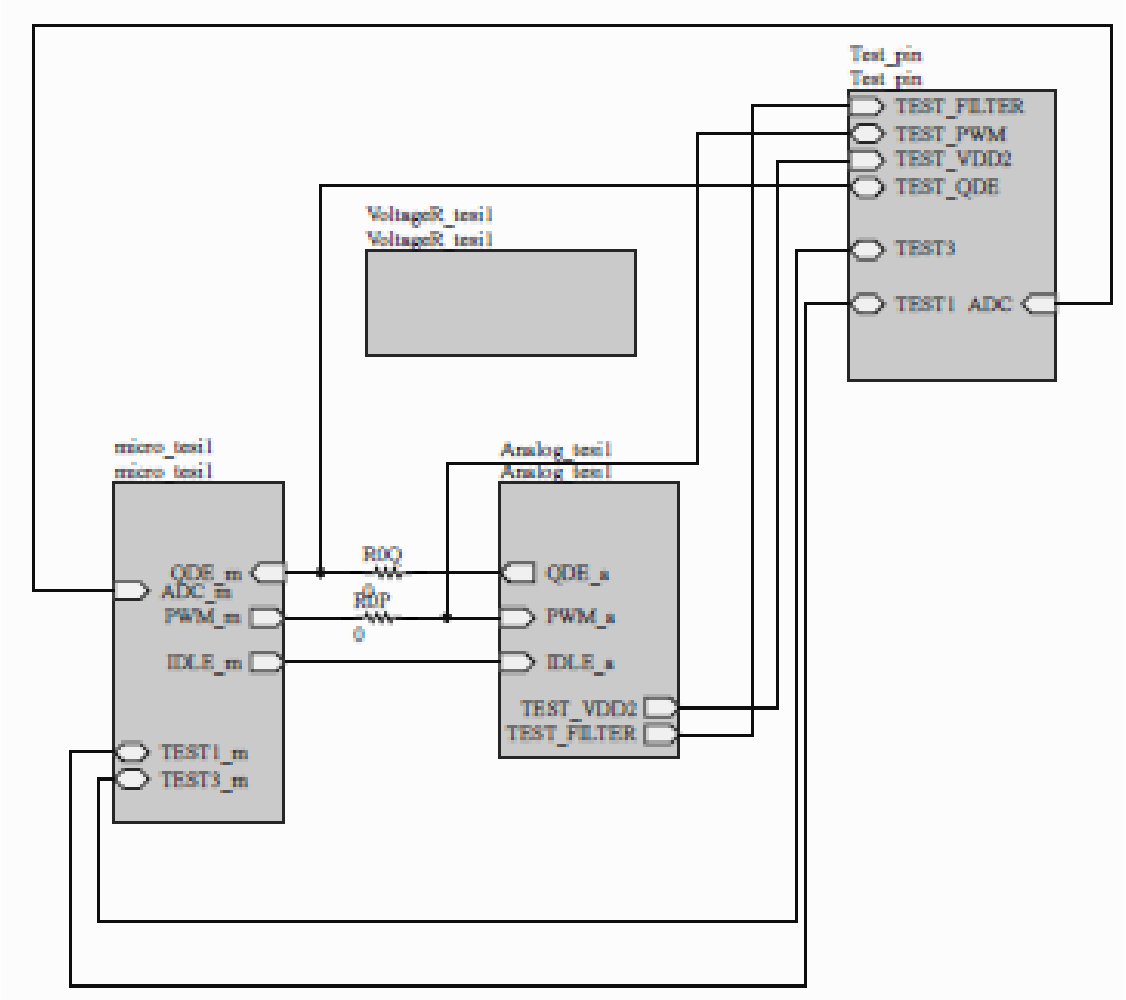

Figure 3.7: Altium configuration for the circuit interconnection

the front-end to the micro-controller circuit are represented by one block which presents a certain number of ports marked as input or output. As observed in every section, a part of the project was reserved for the definition of these ports.
In the first analysis, it is possible to start by the heart of the device, the connection between the microcontroller and the front-end circuit. The idle signal is a simple flag asserted or not according to the operation state by the microcontroller and it is sent to the analogue circuit in order to minimize the power consumption. The Quasi digital signal, generated by the analogue circuit, is transmitted to the controller and thus read and elaborated by the latter. Opposite path is dedicated to the PWM signal, indeed the variable duty-cycle modulation, set up by the micro, is sent to the analogue circuit in order to drive properly the electrochemical cell. It is possible to note on both the connections two bridge resistors with a nominal value equal to 0 Ω, their purpose is to connect or disconnect according to the need,

during the test and development step the three main blocks the micro-controller circuit, the analogue one and the unmentioned test-pin block.
The test pin is a simple connector exploited during the test and developing step. In order to favourite the circuit check, some nodes of the analogue circuit and some signals have been connected toward the external to be tested a to investigate potential problems, bad function and to fix it. At this purpose, the two main signal, QDE and PWM, the reference voltage and the filtered signal have been connected to the test-pin connector. Moreover, some micro-controller pin has been linked for possible future new functions. In figure 3.8 the aforementioned test connector is reported, it is composed of 10 pins, and only 8/10 have been exploited for the test.

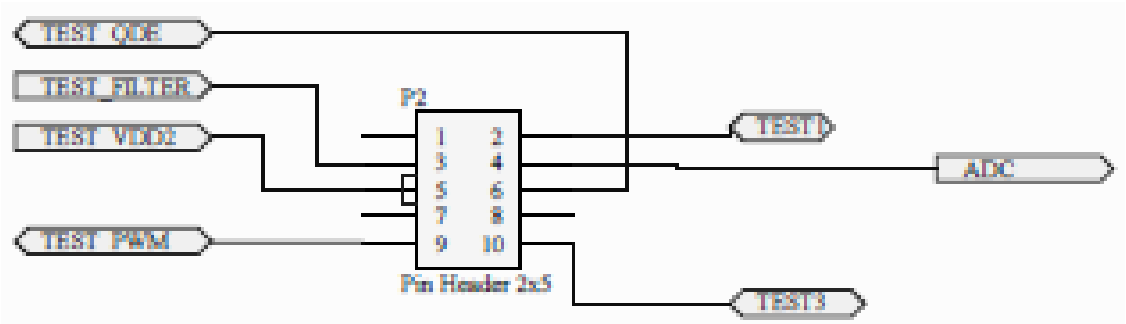

Figure 3.8: Altium configuration for the test circuit

3.5 PCB Description

In section 3.6 the steps to realize the PCB are reported. This step is the final one, once the functionality and the topology of the front-end circuit and the μController layout have been defined. The PCB has been designed on two layers the top and the bottom one

3.5.1 Top Layer

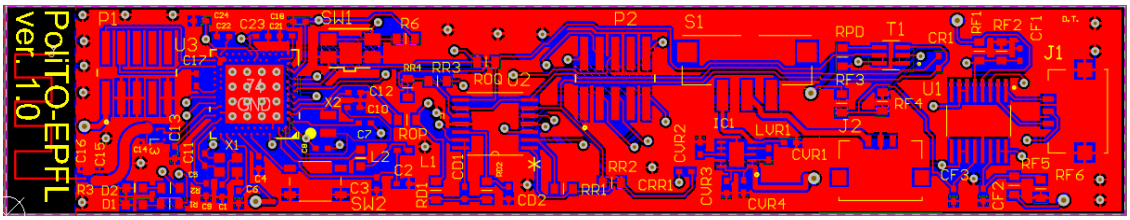

Figure 3.9: 2D PCB layout from the top (Image from *Altium designer*)

The top layer is the upper-side of the device, in this project all the components have been mounted on this layer. The milestone of the PCB realization is represented

by the **place** and **route** of the components which represent the principal challenge for the device project area. Due to the final application as "syringe", it is very important to guarantee the minimum area extended along the horizontal direction, to obtain a long and narrow board.
Therefore the placement and routing of the components must take into account the area constrains and not only: this limitation has influenced also the selection of components in term of dimension.

Footprint

The first step, before to start to import all the components and place them on the empty board, is the definition of the symbol and the footprint of each component itself. The symbol of the component is nothing else that its representation in the circuit schematic, the important point in its definition is the number of the pin and the correct (as indicated in the data-sheet) enumeration and connection with the footprint in order to reflect the reality. On the other hand, define the footprint means to realize the 2D (fundamental) or 3-D model of the component, that accurately mirrors the real component and to associate it to a defined symbol (pin correspondence). It is important at this aim to take as reference the data-sheet indication of dimension shape and area. In order to obtain the final results, it is possible to exploit the wizard tool available in Altium designer and to design the footprint from the begin, otherwise it is possible to import file ".lia" containing the accurate footprint already realized, follow an alternative Altium wizard, and finally check the correctness and validate them. Both the solution have been exploited for all the components. In the following (3.10 just an example is reported, in this case, it is possible also to note the star indicating the correct direction position of the device on the board (usually the star is placed to indicate the pin number one):

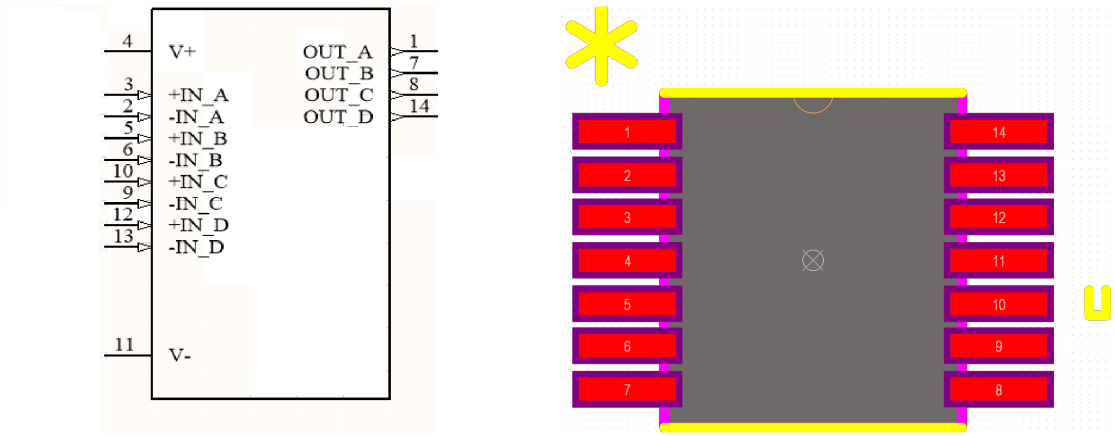

Figure 3.10: Example of symbol component and footprint (ADA4807)

When all the components footprint have been defined, the begin of the project of PCB starts with a provisional shape and dimension definition of the board, at

this point all the components imported will be available at the side of the empty board region for the placement.

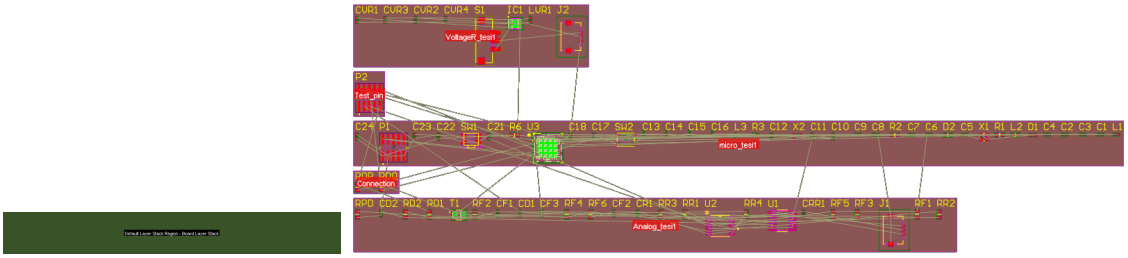

Figure 3.11: Empty provisional board and component to place

Antenna layout

The PCB trace antenna is a simple path of copper on the surface of the board.

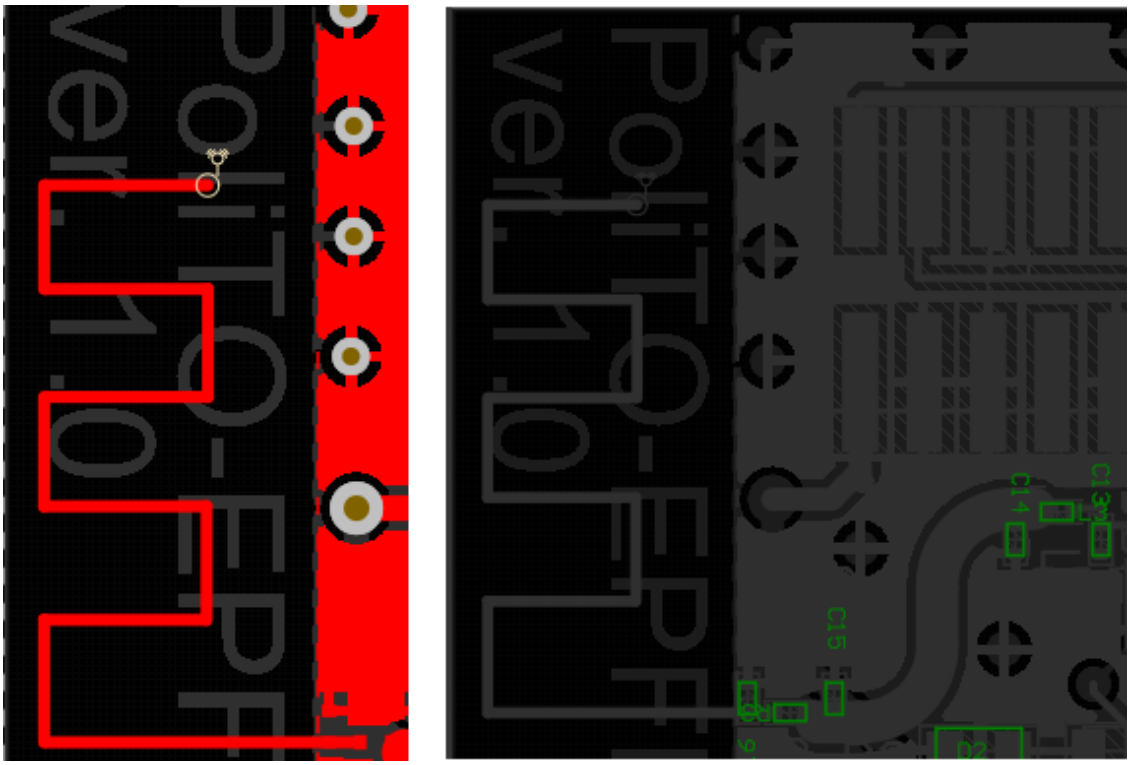

Figure 3.12: Antenna layout on PCB. Left: antenna geometry trace; Right: Antenna with matching network

Figure 3.12 (left) is the zoom of the antenna layout on the top surface of the board. This kind of antenna has been designed in order to work at a frequency of 2.4 GHz, for the BLE (Bluetooth Low Energy) protocol transmission. The total

height (vertical extension) is equal to 10.092 mm and the width equal to 2.964 mm. The geometrical design is a replica of the one proposes by the Nordic and developed on the dongle device Nrf52840, it has been chosen to reproduce the same geometrical layout after the test for the transmission and reception with the dongle provided. Together with the antenna copper trace, it is fundamental to a matching circuit. In the project, the π network matching has been realized composed by an inductor connected on two capacitors.

Components placement & Routing

Once defined the antenna region, it is possible to divide ideally the device into the other two parts. On the left, close to the antenna, the QIAA series of the Nrf52840 is present together with all the circuit connection, defining the region that is possible to call control part of the device. The rest area of the board is addressed to mount the front-end part of the project. This choice has been done in order to try to minimize the length of interconnection and to keep closed the components that "communicate" between them. The aforementioned division is here 3.13

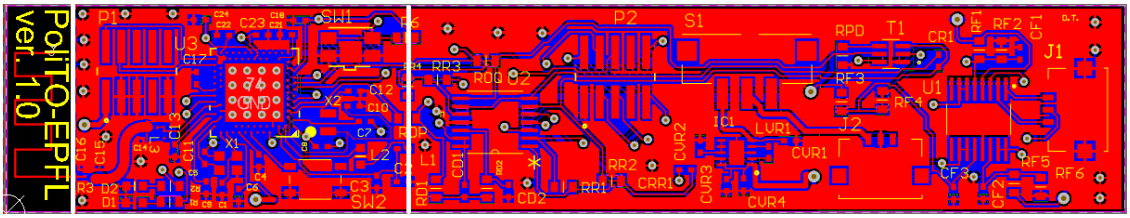

Figure 3.13: PCB placement region, the white line delimits the different region: from the left antenna, μController and front-end

At this point in the analysis, it is possible to introduce all the position of the main components. Starting from the middle region, it is possible to find the programmer connector P1 on the top and the two LEDs on the bottom (D1 and D2 annotation), on the left of the QIAA nrf52840 component. In between the connector and the LEDs, the matching network for the antenna lies, this configuration is an example of the decision to place the components one close to the other reducing the connection distance. Just below of the micro-controller, the X1 oscillator is reported, whereas the X2 one is on the right in the middle between the SW1 and the SW2 respectively the reset switch and the user button. All the aforementioned components represent the main parts of the micro-controller layout.
Moving the attention on the right zone, it is possible to note the first U2 integrated amplifiers with all the needed capacitors and resistors. On the top in the middle of this zone, the second P2 connector is present, it is, as said, dedicated to the test of the signal during the developing step (it becomes effectively useless once verified the correct device work). Next to the test-point, the on-off switch is placed just

above (for obvious reasons of closeness) the voltage regulator circuit. The signal front-end circuit ends with the second amplifier U1 and the two-channel N-mos. Final mention for the two external connectors, one, before the U1 component, for the voltage supplier (close to the switch and voltage regulator); and the latter at the lateral side of the board in order to simplify the connection with the sensor wires. As said all the components have been mounted on the top of the board, apart for the battery, indeed for it has been preferred as solution, an external location for the battery in the same case of the PCB.
About the connection routes, they have been designed on the top side, but mainly for space reasons, not all the routes have been linked on the top. To solve this problem, it has been very widespread the employment of the VIAs, conductive holes in the board, to connect the top layer with the bottom one. The aims of the VIAs are double, the former the link among top layer and bottom layer connections, to guarantee short-circuit and the continuity for the routes. The latter (of course based on the same principle) is to establish a connection between the two ground planes on the top and the bottom layers. Indeed, the red background of the board represents the ground plane on the top, and in order to obtain a unique ground reference with the bottom, it is connected, through Vias, to the ground on the bottom (blue background see figure 3.15. Thus to minimize the possible potential difference between the two ground plane, many vias are placed in free and available positions.
In conclusion, the 3D picture of the board is reported, this figure results to be interesting because it reproduces in a loyal manner the final PCB, considering also the vertical space occupation of the components (where the model is 3D).

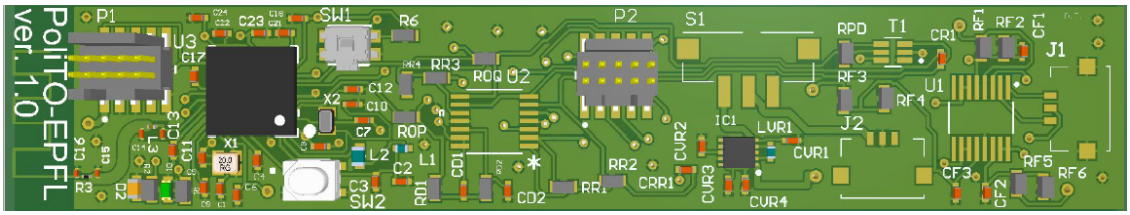

Figure 3.14: 3D PCB layout top view (Image from *Altium designer*)

3.5.2 Bottom Layer

The second layer description is very straightforward seen that only the remaining routes have been designed on this side. As is possible observe in figures 3.15 and 3.16, the bottom area is occupied by only conductive paths and VIAs in all the case in which the connection starts on one side, continues (through Via) on the bottom to end again on the top. Noteworthy is the externally visible route, this path has been made wider to the other and it follows the confine of the board. This connection is the voltage reference and for that, it is important that all the part of

the device were reached by the power signal in a similar way. Trying to minimize in this way the distance between voltage reference and components it is possible to reduce potential voltage drops, moreover the choice to make the path wider rows in the same direction: the reduction of voltage drop.

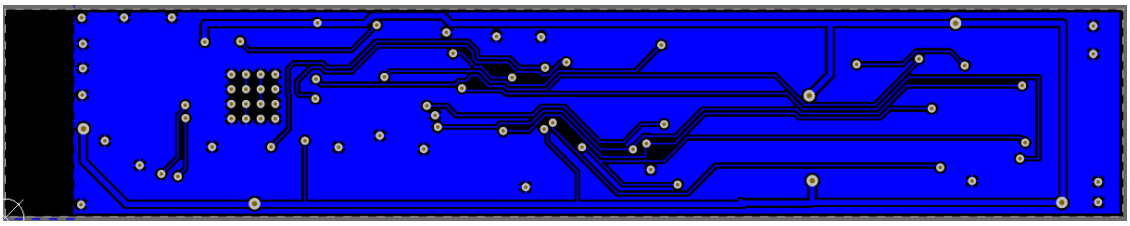

Figure 3.15: 2D PCB layout bottom view (Image from *Altium designer*)

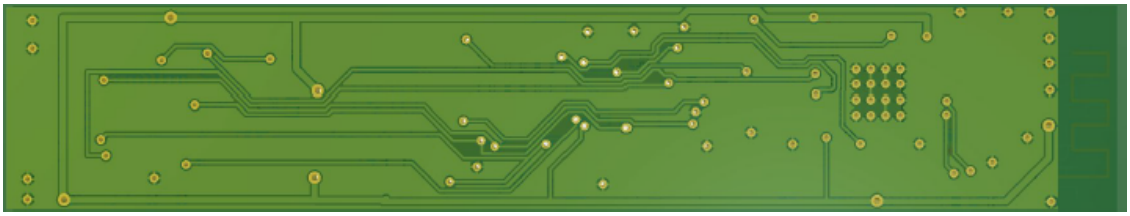

Figure 3.16: 3D PCB layout bottom view(Image from *Altium designer*)

3.6 PCB Production

3.6.1 Components

In parallel with the design of the schematic circuit and thus of the PCB project, the important role has been covered by the selection of the components. Short mentions about the components have been done during the previous description, in this section more information are given.

- The Integrated amplifier, U1 and U2, as above said, have been chosen according to their main features in terms of power and speed (see table 3.1. Moreover, the LTC608 and ADA4807 series of amplifier present 1,2 or 4 amplifier versions. The selection of the 4 op-amp configuration has been done in order to minimize the area occupation.
  Series: LTC6085 (annotation U1) and ADA4807-4 (annotation U2)

- The two-channel n-mos (annotation T1), it is composed of only (the needed) two mos for the front-end circuit.
  Series: 2N7002DW-TP Micro Commercial

- The on-off mechanical switch (S1), which task is to turn on and off the device.
  Series: SS312SAH4-R

- The voltage regulator component (annotation IC1), chosen for its characteristics, that is the output constant voltage equal to 3.3 V and the current control.
  Series: Texas instrument tps63031

- The connectors, J1 and J2, exploited to link respectively the sensor and the battery, have been chosen between a large totality of connectors for their number of pins equal to three and the small dimension always to preserve area occupation.
  Series: Molex 5037630391

- The ten pins connectors P1 and P2 have been chosen more for their task than for the area occupation. Indeed, they play an important role for the programming (the component series was the one suggested by the Nordic) and the test.
  Series: ftsh_smt-1316912

- The 32.768 kHz (X2 annotation) and 32 MHz (X1 annotation) oscillators have been chosen as indicated in the data-sheet on nRf52840, to guarantee the correct operation condition of the latter.
  Series: LFXTAL062558 (X1 annotation) XRCGB32M000F1H01R0 (X2 annotation)

- The reset switch SW1, it is a mechanical switch, with lateral button, to avoid unwanted pressure.
  Series: pts840-1280343

- The button switch SW2, it is a mechanical switch, with top button, exploited as user button to start and stop the measure.
  Series: pts810-1382876

- the nRf52840 controller component, it has been chosen for the GPIO (general purpose input and output) availability, the counter hardware, the PWM driver and the possibility to BLE (Bluetooth low energy) transmission.
  Series: nRF52840-QIAAC0

- The LEDs, green and red with dimension 0603 (mils) to indicate the operation state of the board
  Series SML-LX0402GC-TR (D1 annotation) and SML-P12U2TT86R (D2 annotation)

- The capacitors exploited in the circuit have been selected with different dimension in particular 0201 and 0402 (mils). The small dimension ones have been preserved in the layout of a matching network for the antenna, in order to avoid any possible failure, whereas in the rest of the board the 0402 have

been mounted, a good compromise between area and practicality. A further consideration in the decision about capacitors has concerned their class. All the capacitors selected have been chosen of class 1 and 2 (where the first class was not available), to guarantee the highest stability and lowest losses for resonant circuit applications. No particular considerations have been done about the work temperature range since it does not represent a hot spot.

- The inductors and resistors follow part of the consideration done for the capacitor.

3.6.2 Production

The production of the PCB represents the more "commercial" part of the project, as such the realization cost, the production time and the availability have been played the central role in this step.

Components order

The order of components has considered two important aspects of the cost and availability. Once defined all the aspect of the project, the proper components have been selected, in order to piratically make the order, different important and reliable distributor of electronic components have been compared in term of cost and delivery time. The final option has been Mouser Electronics, that with the minimum cost has guaranteed all the component with the minimum time.

PCB order

The physical realization of the PCB has been addressed to Eurocircuit the European reference for PCB prototypes & small series. In order to allow the specialist manufacturers and assemblers of prototype company, the realization of the PCB, some project files have been provided.
The first point to investigate, when the PCB is produced, is the technology exploited for the realization. The technology used has consequence in the technology adopted in the board, indeed the two have to share the same parameter to make possible the project. This fundamental consideration, at the design level, is translated in a set of rule to respect during the design. Since the technology is provided by Eurocircuit, the same rules have been imported by the same company in the Altium project. The rules concern a different aspect of the project and to make clear what they affect, the Altium set are here listed:

- Manufacturing:
  - Minimum annular ring, this rule specifies the minimum annular ring required for a pad or via. The annular ring is measured radially, from the edge of the pad/via hole to the edge of the pad/via (also referred to as the land perimeter).[21]

- Minimum solder mask sliver, this rule helps identify narrow sections of solder mask that may cause manufacturing problems at a later stage. Ensuring that there is a minimum width of solder mask across the board, this rule checks the distance between any two solder mask openings that are equal to or greater than a user-specified value. This includes the pads, vias, and any primitives that reside on solder mask layers. It also checks Top and Bottom sides independently[21]
- Silk to solder mask clearance, this rule checks the clearance between any silkscreen primitive and any solder mask primitive, or exposed copper-layer primitive (exposed through openings in the solder mask). The check ensures that the distance is equal to, or greater than, the value specified in the constraint. [21]
- Silk to silk clearance, this rule defines the minimum clearance allowed between any two objects on a silkscreen layer. [21]
- Net Antennae, this rule operates at a net level in the design to flag any track or arc end that is not connected to any other primitive and thus forms an antenna. [21]
- Board Outline Clearance, this rule defines the minimum clearance allowed from design objects that are fabricated, to edges of the board. Either a single clearance value can be specified for all object-to-edge possibilities, or different clearances for different pairings can be defined, through the use of a dedicated Minimum Clearance Matrix. [21]

At the conclusion of the project on Altium, it is fundamental to run the "PCB rules and violations" tool of the software itself to check that all the rules have been respected. The final control has highlighted just one violation about net antennae rule since an antenna layout is actually present on the device.
At this point, all the step of the project have been followed, and to produce the board some files have to be provided to Eurocircuit. The first file is the "Picks and Place" file, that contains all the assembly information about the position of each component on the board. In particular in this file are reported the position according to a reference point, the rotation degree and the layer in which the component is mounted.
The other documentation is focused on the fabrication output and the file considered are the "Gerber" and the "Drill drawings". The Gerber is the de-facto standard used by the PCB industry (called the backbone of the electronics) software to describe the printed circuit board image. It is based on an open 7-bit ASCII and it is exploited for the PCB data transfer. Gerber files contain copper layers, solder mask, legend and drill and route information. The Altium designer'tool allows automatically for the description of the board in term of Gerber file. The software generates a series of output document with all the information stored furthermore

a CAM file is produced that reads and shows the Gerber file: To complete the

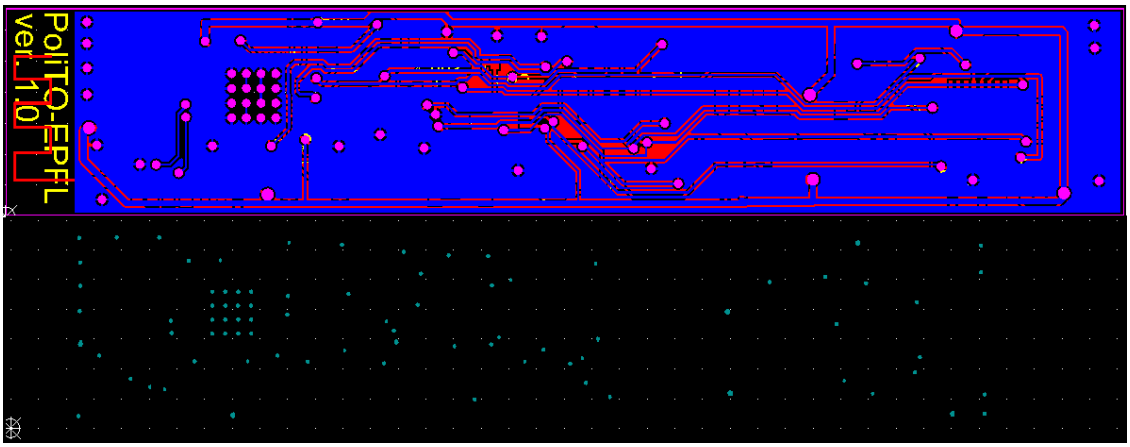

Figure 3.17: Top:Altium Cam file, picture of the Gerber data
Bottom: Drill drawing figure, the position of the vias hole.

board fabrication description, the "Drill Drawing" was added to the output file. The data stored outlines the position of the holes (VIA but not only) on the board 3.17 bottom figure.
Finally, all the files in a zip folder have been updated on the Eurocircuit portal. In this conclusive step of the PCB project, the key role is given by the correct trade-off of information between the designer and the producer. In addition to the aforementioned files, the data on the components (series name, value, type, etc...) have been provided. Indeed, the soldering of the critical part of the board (antenna matching network and micro-controller configuration) has been addressed to the producer automatic machine, to avoid every kind of defects. The rest of the components have been manually welded on the board in the laboratory with the electronic microscope and soldering iron.

Final PCB results and features

At the end of the exchange of data with the producer, the final device presents the following properties:

- Board definition:
  - Number of layers: 2;
  - PCB width (X): 91.42 mm;
  - PCB height (Y): 16.82 mm;
  - Top soldermask: green;
  - Top legend: white;
  - Bottom soldermask: green;

  - Panel outline: Routed;

- Board technology:
  - Pattern class: 7 (the manufacturability of the PCB);
  - Drill class: Drill B;
  - Outer layer trackwidth (OL-TW): 0.125 mm;
  - Outer layer isolation distance (OL-TT-TP-PP): 0.150 mm;
  - Outer layer anular ring (OAR): 0.125 mm;
  - Hole density: $<1000/dm^2$;
  - holes: 0.45 mm;
- Material definition:
  - Board thickness: 1.55 mm;
  - Base material: FR4IIMP;
  - Outer layer copper foil: 18 μm (End-Cu $+/-35\mu m$;
  - Board buildup: standard;
  - Material Tg: 145-155 °C;

All the parameters are the results of compromise among production cost and PCB quality, always considering the product as a laboratory prototype to test and develop in view of the final device.
Once ultimate the completion of all the needed operations as the soldering of the remaining components in the laboratory, the final device shows up:

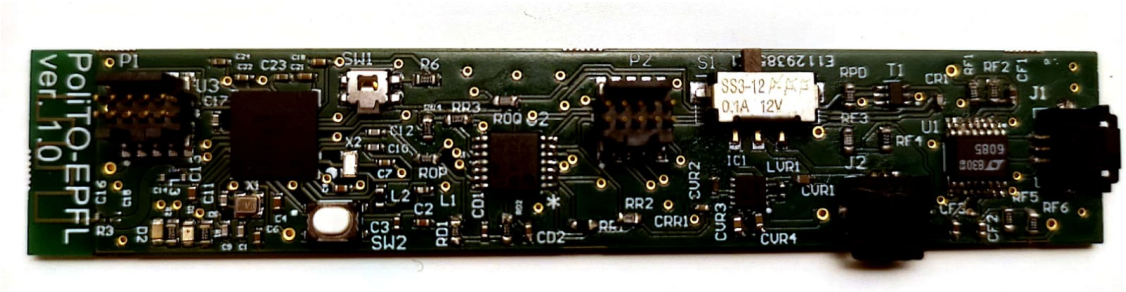

Figure 3.18: PCB real device photo

Chapter 4

Micro-controller & firmware

The present chapter is completely dedicated to the discussion and description of the micro-controller tasks and features and of the implemented firmware. This phase of the project is, together with the hardware designed, fundamental for the device working, it represents the "mind" of the whole system. The firmware reported is totally developed in SEGGER Embedded Studio for ARM, a streamlined integrated development environment, compilation tools, and libraries for building, testing, and deploying applications on ARM and Cortex microcontrollers[22].

4.1 Device Routine

The section is dedicated to the first glance to the routine idea for the device firmware.

The flow chart represents the sequence of the state covered by the processor during the operation. At the power-on of the board controlled by a mechanical switch, some configurations and set-up operations are executed, in order to make the system ready to the use. The configuration concerns the initialization of PWM and timer drivers, the definitions of interrupts and the BLE set-up. This operation is done just one time at the begin and every time the system is reset. Once all the preliminary steps have terminated the micro-controller waits for the start command in the idle state, where only the interrupt connected to the GPIO linked to the mechanical button, can wake up the system. In this condition, the power is preserved. The Idle state is called every time the user decides, always by means the pressure of the button, to pause the operation and to lie in stand-by. When the start is asserted, the firmware proceeds with the main loop. The operation is repeated in an infinite loop: the electrode cell is driven cyclically by the PWM signal, and the current response in term of event quasi-digital signal is continuous monitoring and measurement. During the main state, the device is discoverable and connectable by the central BLE receiver. Once the set of measure is ready, they are transmitted through the BLE technology and protocol, this step concludes the loop cycle. The only way to pause or stop the operations is to push again the same button (in this

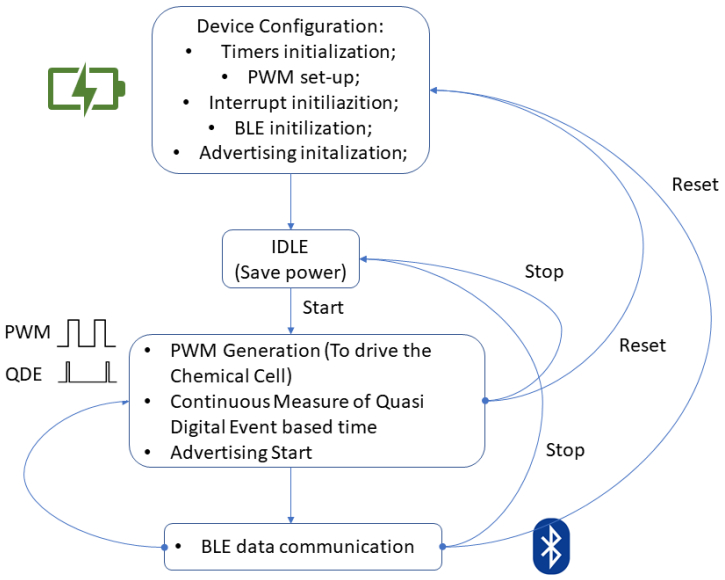

Figure 4.1: Flow Chart of the device firmware routine

case it stands for stop command) or alternatively reset the device.

4.2 Micro-controller

In the chosen of the μController, important aspects have been taken into account. The main features considered have been the presence of GPIO pin, the PWM (Pulse Width Modulation) driver, the available counter and overall the possibility for the BLE protocol communication. All the needed characteristics lie in the nRF52840 SoC.
The nRF52840 built around the 32-bit ARM® Cortex™-M4 CPU is the Nordic solution, perfectly matched with the specifics required by the project. The chip features allow for:

- ARM® Cortex®-M4 32-bit processor with FPU, 64 MHz;
- Bluetooth® 5, IEEE 802.15.4-2006, 2.4 GHz transceiver;
- 4x 4-channel pulse width modulator (PWM) unit with EasyDMA;
- 3x real-time counter (RTC);
- Up to 32 GPIO pins with configurable output drive strength; [23]

Where the listed points represent only a small cut of the possibilities, but they are fundamental for the project.
Once defined all the peculiarity, it is possible to enter in the details of some aspect.

4.2.1 Pulse Width Modulation for electrochemical cell driven

As first, the channel pulse width modulator (PWM), it enables the generation of pulse width modulated signals on GPIO. The module implements an up-down counter up to 4-channels. As said for the device application, just one channel is required, and it is necessary to drive the electrochemical cell to the established value of voltage where the response is evaluated. The PWM driver allows a programmable frequency, a definition of the sequence of PWM array values. To better explain the functionality, the schematic of the PWM driver is reported

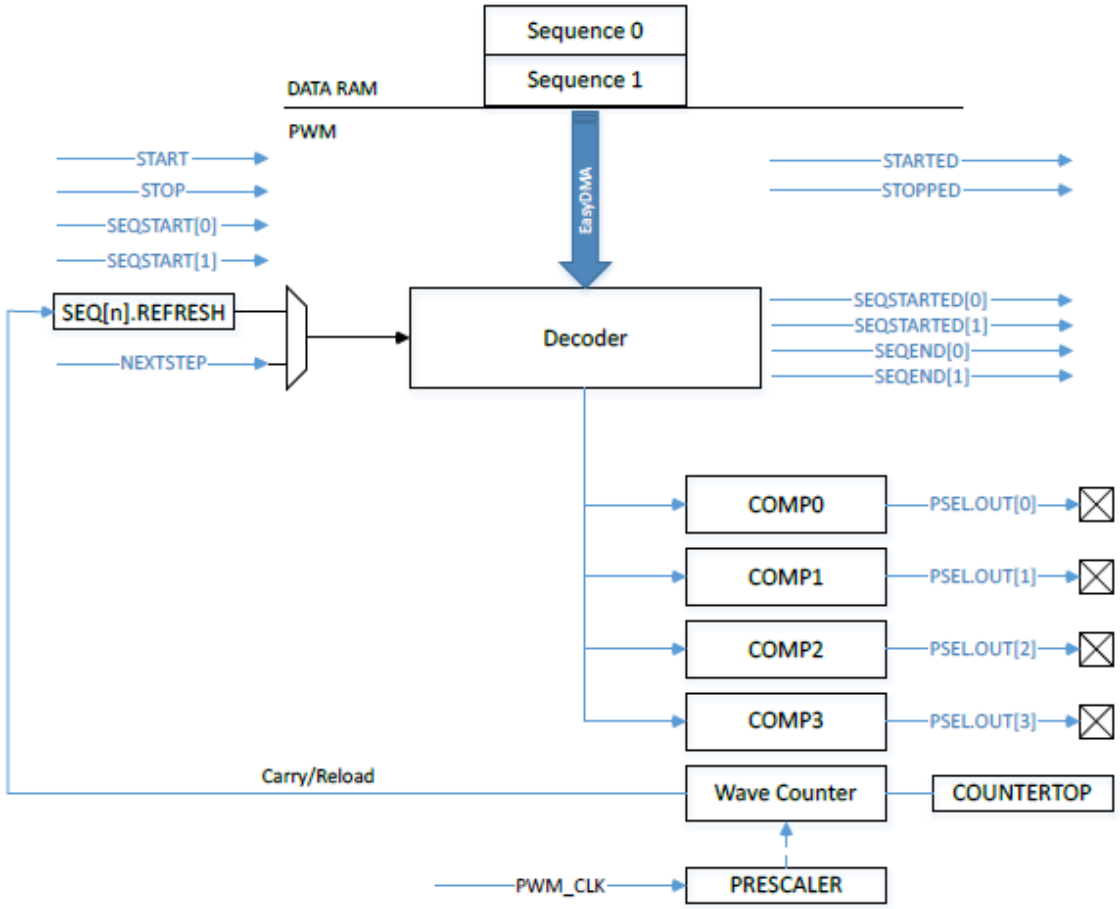

Figure 4.2: PWM module schematic nRF52840 datasheet image [23]

Dedicated Firmware

The function is implemented to configure the main aspects of the PWM driver.

```
1 void PWM_gen(void) {
2   NRF_PWM0->PSEL.OUT[1] = 26;
3
```

```
4     NRF_PWM0->MODE = 0x00000000;
5     NRF_PWM0->PRESCALER = 0x00000003;
6
7     NRF_PWM0->COUNTERTOP = 100;
8     NRF_PWM0->LOOP = 0x00000000;
9
10    NRF_PWM0->SEQ[0].PTR = (uint32_t)(pwm_seq);
11    NRF_PWM0->SEQ[0].CNT = 1;
12
13    NRF_PWM0->SEQ[0].REFRESH = 0;
14    NRF_PWM0->SEQ[0].ENDDELAY = 0;
15    NRF_PWM0->TASKS_SEQSTART[0] = 1;
16
17    NRF_PWM0->ENABLE = 0x00000001;
18  }
```

As first the output pin is defined according to the PCB layout. Then the PWM mode, the prescaler, and the countertop are set. In the example the number three stands for 2^3 that turns out in a frequency $f_p = \frac{16MHz}{2^3} = 2MHz$, and the value 100 sets the maximum value for the counter that turns out in the final signal frequency $f_{signal} = \frac{f_p}{100} = 20kHZ$. After that the PWM is configured to not have any loop, to start with the only sequence(CNT=1) "pwm_seq". At the end the driver is enabled.
The PWM sequence is then defined in the measure routine where two for-cycle are implemented to obtain two driven sequences (see appendix B.1 for the complete code).

4.2.2 Timer counter for event-based signal

The timer counter covers the important role in the measure of the time gap between two consecutive events when the quasi-digital signal is received according to the current generated in the RedOx chemical reaction.

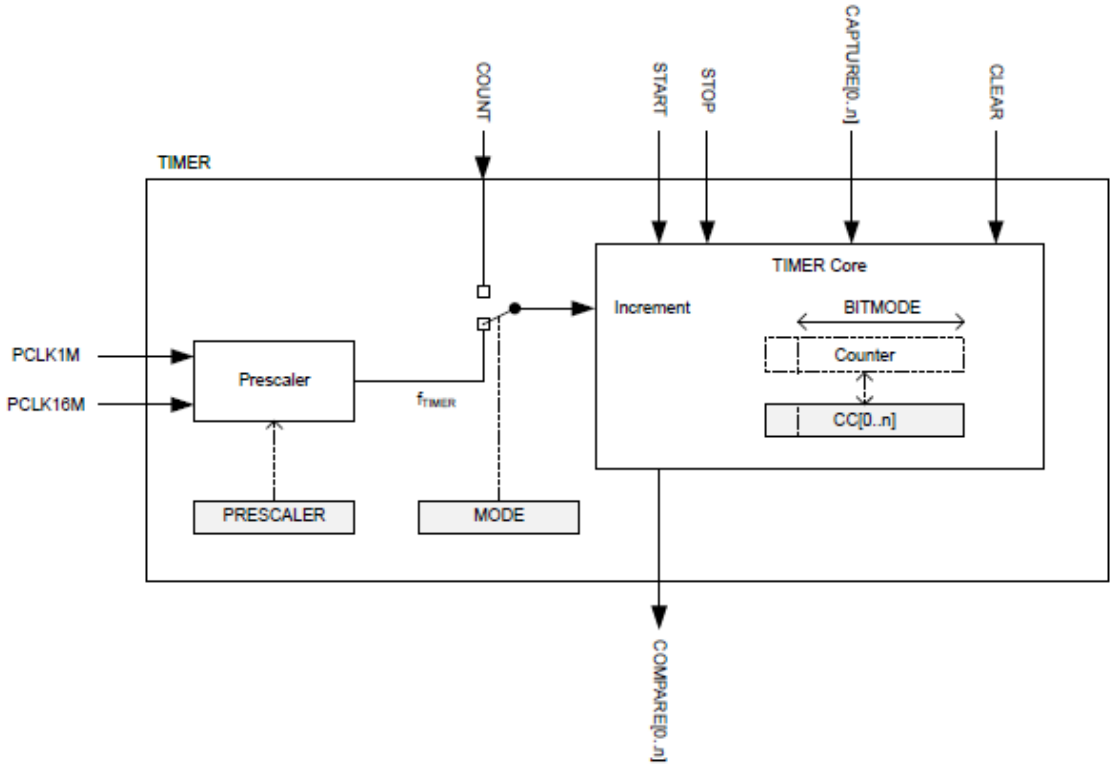

Figure 4.3: Block schematic for timer/counter nRF52840 data sheet image[23]

The counter runs on the high-frequency clock 16MHz mounted on the custom PCB. The register connection presents the possibility to define the number of the counter bit, the counter mode, and finally to capture a value of the counter in the "CC" register. Obviously, the counter can be started and stop and clear according to the requests.

Dedicated Firmware

```
1  void timer_setup() {
2    NRF_TIMER1->MODE = TIMER_MODE_MODE_Timer;
3    NRF_TIMER1->TASKS_CLEAR = 1;
4    NRF_TIMER1->BITMODE = TIMER_BITMODE_BITMODE_24Bit;
5    NRF_TIMER1->PRESCALER=0x00000000;
6    NRF_TIMER2->MODE = TIMER_MODE_MODE_Timer;
7    NRF_TIMER2->TASKS_CLEAR = 1;
8    NRF_TIMER2->BITMODE = TIMER_BITMODE_BITMODE_24Bit;
9    NRF_TIMER2->PRESCALER=0x00000000;
10 }
```

As reported in the code snippet above, the counter is configured on 24 bit, exploited

in timer mode in order to start counting and stop only when required. The counter exploited are three, the two sets here represent the first used for the measure of the event-based signal and the second for the button functionality. The third counter is set for BLE aims. The timer for the measurement starts and stops its task according to the interrupt routine called when the high to low transition occurs at the GPIO pin dedicated to the QD signal.

```
1 void qde_pin_handler(nrf_drv_gpiote_pin_t pin, ...
      nrf_gpiote_polarity_t action) {
2   NRF_TIMER1->TASKS_CAPTURE[0] = 1;
3   result[i] = NRF_TIMER1->CC[0];
4   NRF_TIMER1->TASKS_CLEAR = 1;
5   NRF_TIMER1->TASKS_START = 1;
6 }
```

As soon as the interrupt is called the timer value is stored, and the new counting operation started.

4.2.3 Bluetooth Low Energy BLE

The Bluetooth is standard wireless technology for short, low power and low-cost communication developed by "Bluetooth SIG" [24]. The main difference between a general Bluetooth and the low energy one (BLE) lies in the less power usage, indeed it is largest used in an application where the data transfer does not compromise the battery consumption. For this reason, the BLE results in the best communication protocol to apply in a portable device.
The definition of communication characters is the first step for the configuration of the correct connection. Generally speaking in the BLE protocol the roles are divided into Peripheral-Central, Master-Slave, Client-Server. At the begin, before the connection hold, one device starts the advertising, and the second scans, the former works as "Peripheral", the latter is the "Central", once the connection is stabilized, the Central takes the role of the Master and the Peripheral becomes the Slave. After the connection is established, the role of Master and Slave are not anymore restricted, indeed, it is better to introduce the Client and Server roles, respectively the first requires to write/read commands, whereas the second response with data required. In this phase both the devices could work as Master or Slave, usually the Server is the Slave and the Client is the master (not always).
Clarified the main protagonists, the chosen applied to the project defines the portable device designed to work as Peripheral, and Server-Slave during the connection, whereas the receiver as Central. In particular, since the task of the device is to send continuous data, the Server can transmit data without any request from the Client whose role is limited to the reception. In the described case the Server notifies the operation without any "Ack" signal from the client needed.

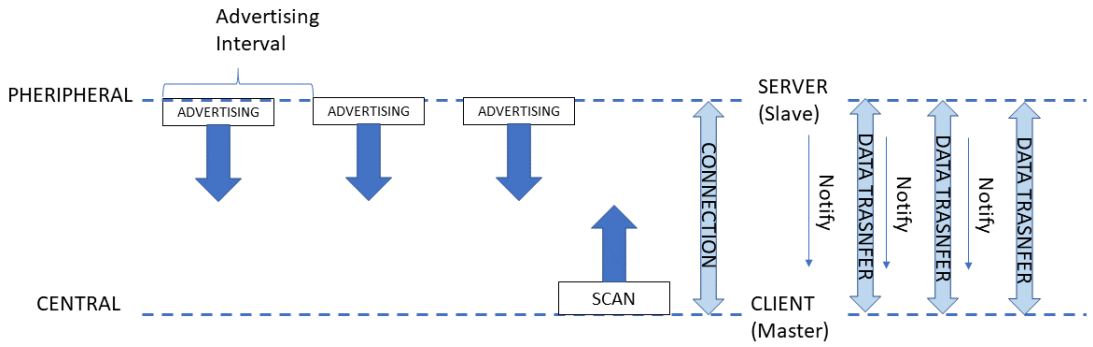

Figure 4.4: BLE protocol

Bluetooth Low Energy: Generic Access Profile (GAP)

The GAP is the "Generic Access Profile" and it is the guideline for the possible connection kinds: broadcasting or connecting. In details in case of broadcasting, the device advertises data packets, and the observer listens to the data sent, without any connections needed. On the other hand, the Connecting requires the handshake between peripheral and central points. The latter is the configuration chosen in the device project, in particular, the device can connect to any peer device.
Furthermore, in order to start the advertising, some parameters have to be provided: the advertising interval and timeout, that respectively describes the distance between each advertising and the maximum time to advertise; the slave latency, the minimum and maximum connection interval and the device name.

Bluetooth Low Energy: Generic Attribute Profile (GATT)

The GATT is the "Generic Attribute Profile", describes the way of the two Bluetooth devices transfer data based on Services and Characteristics concepts. Before to describe the GATT, the Attribute protocol (ATT) are introduced. The ATT defines the relation hold between the Server and the Client. The attribute structure includes:

- Handle: it is a 16-bit unique identifier with value in the range of 1 to ffff hexadecimal base[25];
- UUID: it is the universally Unique Identifier, 16-BIT in Bluetooth SIG convention or 128-BIT custom[25];
- Value: it holds the data that the server wants to share, it could have variable length and format based on type[25];
- Permission: it determines what attribute can be read or written or notified and indicated and the security[25].

The GATT exploits the ATT distributed in Services sections which group parts of user data called Characteristics. The GATT can be divided according to the previously discussed role of Client and Server:

- Client GATT sends the request to the server and waits for its response before to start reading and writing server's attribute[26].
- Server GATT receives a request from the client and it responds back with the information[26].

Dedicated Firmware

The salient parts of the BLE configuration and operation are here examined in-depth, the full code is postponed to the appendix section B.1.
As first the BLE parameters are defined:

```
1  #define APP_BLE_CONN_CFG_TAG 1
2  #define DEVICE_NAME "Anesthetic_device"
3  #define NUS_SERVICE_UUID_TYPE BLE_UUID_TYPE_VENDOR_BEGIN
4  #define APP_BLE_OBSERVER_PRIO 3
5  #define APP_ADV_INTERVAL 64
6  #define APP_ADV_DURATION 18000
7  #define MIN_CONN_INTERVAL MSEC_TO_UNITS(20, UNIT_1_25_MS)
8  #define MAX_CONN_INTERVAL MSEC_TO_UNITS(75, UNIT_1_25_MS)
9  #define SLAVE_LATENCY 0
10 #define CONN_SUP_TIMEOUT MSEC_TO_UNITS(8000, UNIT_10_MS)
11 #define FIRST_CONN_PARAMS_UPDATE_DELAY APP_TIMER_TICKS(5000)
```

The snippet code reports some define the BLE protocol. The first line indicates the Softdevice configuration mode (the only available), then the visible name for the discovery of the device is set up and the UUID type. From line five up to the eighth are initialized the advertising interval and duration in unit of 0.625 ms; the advertising minimum and maximum connection interval in unit of 1.25 ms. The latency defined is zero whereas the last two lines indicate the connection timeout, the time about the start notification.
Several functions are exploited to initialize the BLE stack, the GAP and GATT parameters and the advertising and connection parameter. Going beyond these functions, however, commented in the appendix, the transmission routine is here discussed:

```
1 static void nus_data_handler() {
2   uint32_t err_code;
3   if (stop == 0)
4   {
5     char data[12] = {0};
6     for (ind = 0; ind ≤ Ndata; ind++) {
```

```
7         uint16_t length = snprintf(data, sizeof(data), "%d,", ...
              result[ind]);
8         do {
9           err_code = ble_nus_data_send(&m_nus, data, &length, ...
                m_conn_handle);
10          if ((err_code != NRF_ERROR_INVALID_STATE) &&
11              (err_code != NRF_ERROR_RESOURCES) &&
12              (err_code != NRF_ERROR_NOT_FOUND)) {
13            APP_ERROR_CHECK(err_code);
14          }
15        } while (err_code == NRF_ERROR_RESOURCES);
16      }
17      uint16_t length = snprintf(data, sizeof(data), "\n");
18      do {
19        err_code = ble_nus_data_send(&m_nus, data, &length, ...
              m_conn_handle);
20        if ((err_code != NRF_ERROR_INVALID_STATE) &&
21            (err_code != NRF_ERROR_RESOURCES) &&
22            (err_code != NRF_ERROR_NOT_FOUND)) {
23          APP_ERROR_CHECK(err_code);
24        }
25      } while (err_code == NRF_ERROR_RESOURCES);
26    }
27  }
```

The function called "nus_data_hander" is the one designed for the transmission of data. After the measuring step, this code is executed, all the stored results in the namesake variable are converted in the sting format through the C snprintf function. The integer number is digit by digit inserted in the output data char type and the results divided by the comma divisor (for Matlab elaboration reasons). The dimension of data is chosen equal to 12 to guarantee the cover of all the possible case with an integer 32bit value. At this point, the timer measure is ready to send thanks to the "ble_nus_data_send", during the transmission some conditions are verified err_code in order to avoid the stop of the firmware in case of unwanted cases (line 10th to 15th). Finally, each dispatch is followed by the new line character to flush the buffer. According to the BLE configuration the maximum length of attribute data is "BLE_GATT_ATT_MTU", 20 bytes, the master physical layer data rate is set to the auto-configuration in order to impose always the maximum supported rate.

4.2.4 Receiver dongle Firmware description

To complete the project, separately to the device design, a receiver system has been implemented. In order to receive the sent data from the device, the Nordic nRF52840 Development Kit has been exploited. The system idea is based to establish a connection with the device through the BLE protocol and then the transmission to the final terminal (computer) of the data by means of USB serial link.

At this aim, the DK micro-controller has been programmed with the implemented firmware. The latter includes all the initialization of the BLE as well as done for the transmission firmware, with the difference that this time the data have to be received and transmit by serial communication. The dongle receiver simply continuously scan the BLE device and hold the connection as soon as the device is discoverable and connectable. In the following the piece of code about the reception and serial communication is highlighted:

```
1  static void nus_data_handler(ble_nus_evt_t * p_evt)
2  {
3
4      if (p_evt->type == BLE_NUS_EVT_RX_DATA)
5      {
6          uint32_t err_code;
7
8          printf("Received data from BLE NUS. Writing data on ...
               UART.");
9          NRF_LOG_HEXDUMP_DEBUG(p_evt->params.rx_data.p_data, ...
               p_evt->params.rx_data.length);
10
11         for (uint32_t i = 0; i < p_evt->params.rx_data.length; i++)
12         {
13             do
14             {
15                 err_code = ...
                       app_uart_put(p_evt->params.rx_data.p_data[i]);
16                 if ((err_code != NRF_SUCCESS) && (err_code != ...
                       NRF_ERROR_BUSY))
17                 {
18                     printf("Failed receiving NUS message. Error ...
                           0x%x. ", err_code);
19                     APP_ERROR_CHECK(err_code);
20                 }
21             } while (err_code == NRF_ERROR_BUSY);
22         }
23         if (p_evt->params.rx_data.p_data[p_evt->params.rx_data.
24 length - 1] == '\r')
25         {
26             while (app_uart_put('\n') == NRF_ERROR_BUSY);
27         }
28     }
29
30 }
```

As observable by the code, this function is called every time a BLE event occurs and in particular when the receiving of data (if condition line 4). Once the data arrived it is placed inside the UART buffer (app_uart_out) and then sends to the terminal computer.

Chapter 5

User Interface

In this chapter, the design of the user interface has been described. The aim of the development of the graphical user interface lies in the tentative to provide to the final user, not necessarily with specific technological knowledge, (as health worker nurses and similar) a tool able to manage the main tasks of the final device.
The IDE exploited for the realization has been Microsoft Visual Studio and the code in which it has been written C#. The IDE and the code have been chosen in order to realize a classical executable program in Windows, the most common operative system adopted.

5.1 User interface description

To better introduce the graphic interface, the images in the different situation of the operation is reported. The complete description is postponed in the appendix C where the whole code and some comments are added.

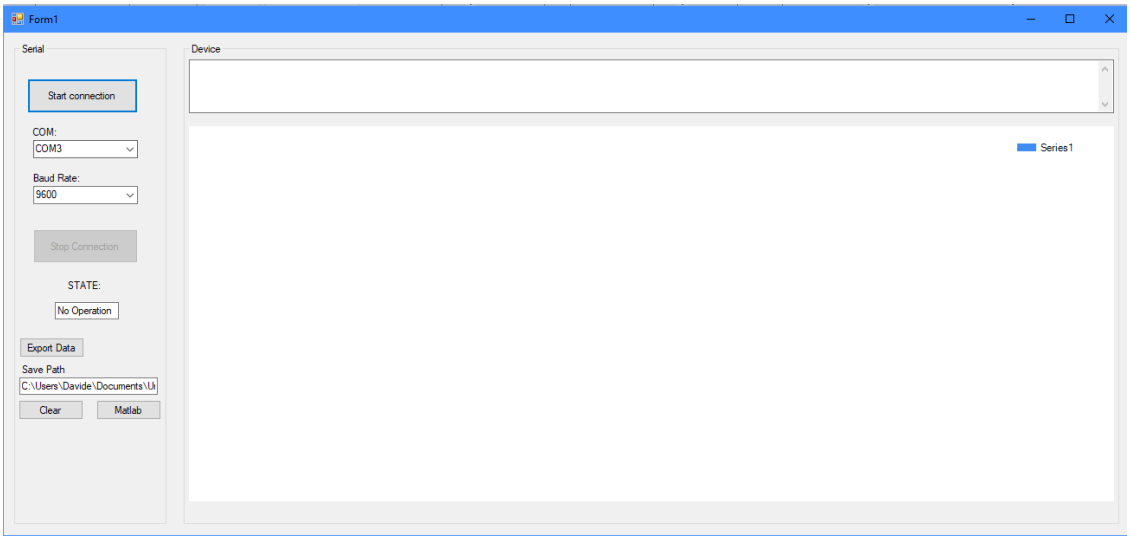

Figure 5.1: User interface at the start

The application, reported in picture, presents on the left in the "Serial box" some tools to manage the connection with the device, whereas the rest of the interface is dedicated to the device response: in the "Device box" the data collected were available in text format and primarily in graphical one (white space).
As first, on the right top corner, the "Start connection" button lies. Obviously, at the click action, it opens the communication with the receiver. The communication in the specific is serial, as explained in the firmware section, the data send via Bluetooth are received by a further device connected by USB to the terminal. To define the correct data transfer, the Baud rate and the COM to open have to be properly selected. The two combo-box, respectively, allow for the selection between different COM port of the computer from one to seven and the definition of the correct baud-rate, obviously according to the set-up of the serial transmitter. In the case in which is not possible to start the communication, a window message appears on the screen, to help the user, reporting the possible mistake as the incorrect port choice.
The last button before to introduce the "state" indicator of the device, is the "Stop connection", in figure 5.1, it is disabled because at the begin no-connection is still hold. The button will remain to deactivate up to the establishment of a working transmission see figure 5.2.
Finally, in the "Serial box" the state text-box is reported. According to the state of operation, the possible indication is: no operation (as in figure 5.1), connection, transmission and stop transmission; each of them highlighted with the proper colour.
The bottom left corner is dedicated to the management of the data: all the received values could be exported as text file in a custom folder thanks to the "Export data" button, moreover it is possible to run through the button "Matlab" the external Matlab software in order to elaborate and analyze more in deep the data. Obviously, by code modification, whatever external program can be run, according to the user request. Next to the Matlab button, the clear button allows for the removal of all the data transmitted by the device.

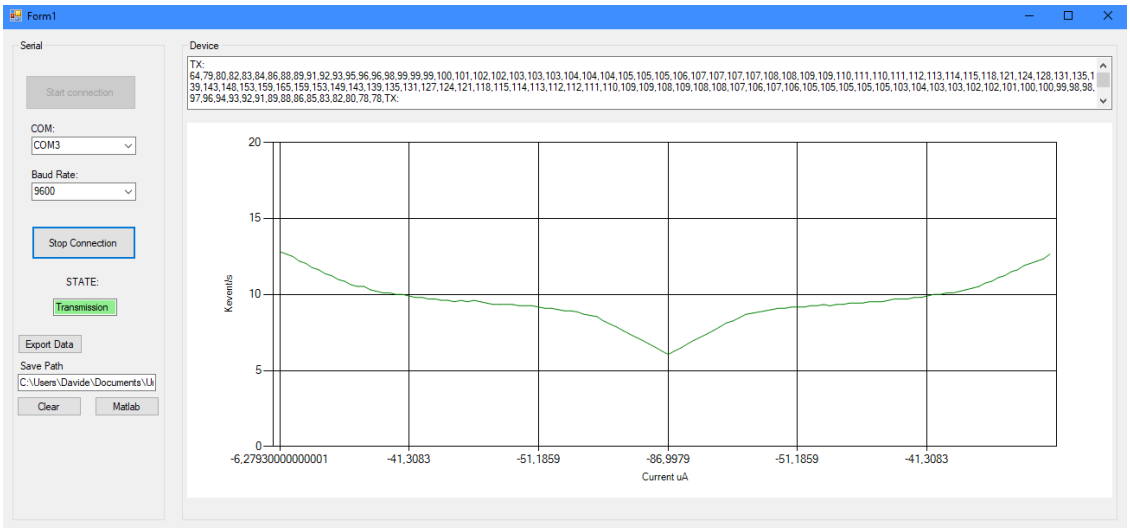

Figure 5.2: User interface working condition

In the second figure 5.2, it is the possible focus the attention on the main difference in the graphic. First of all the device state is changed, from "No Connection" to "Transmission" (the other case as said are "Stop" and "Connection"). As announced before, the white space, during the transmission, reports the data trend: on the vertical axis the "kEvent per second" and on the horizontal one, the current expresses in μA. The plot, automatically, updates at each data reception, and finally on the top, it is possible to note the numerical received value that represents the time between two events measured (in μs). For the sake of completeness, other two picture are reported where are shown the other two possible state 5.3, the "stop" case occurs when the "Stop Connection" button is pushed, whereas the "Connection" case is a transition between the "No connection" and the first data received.

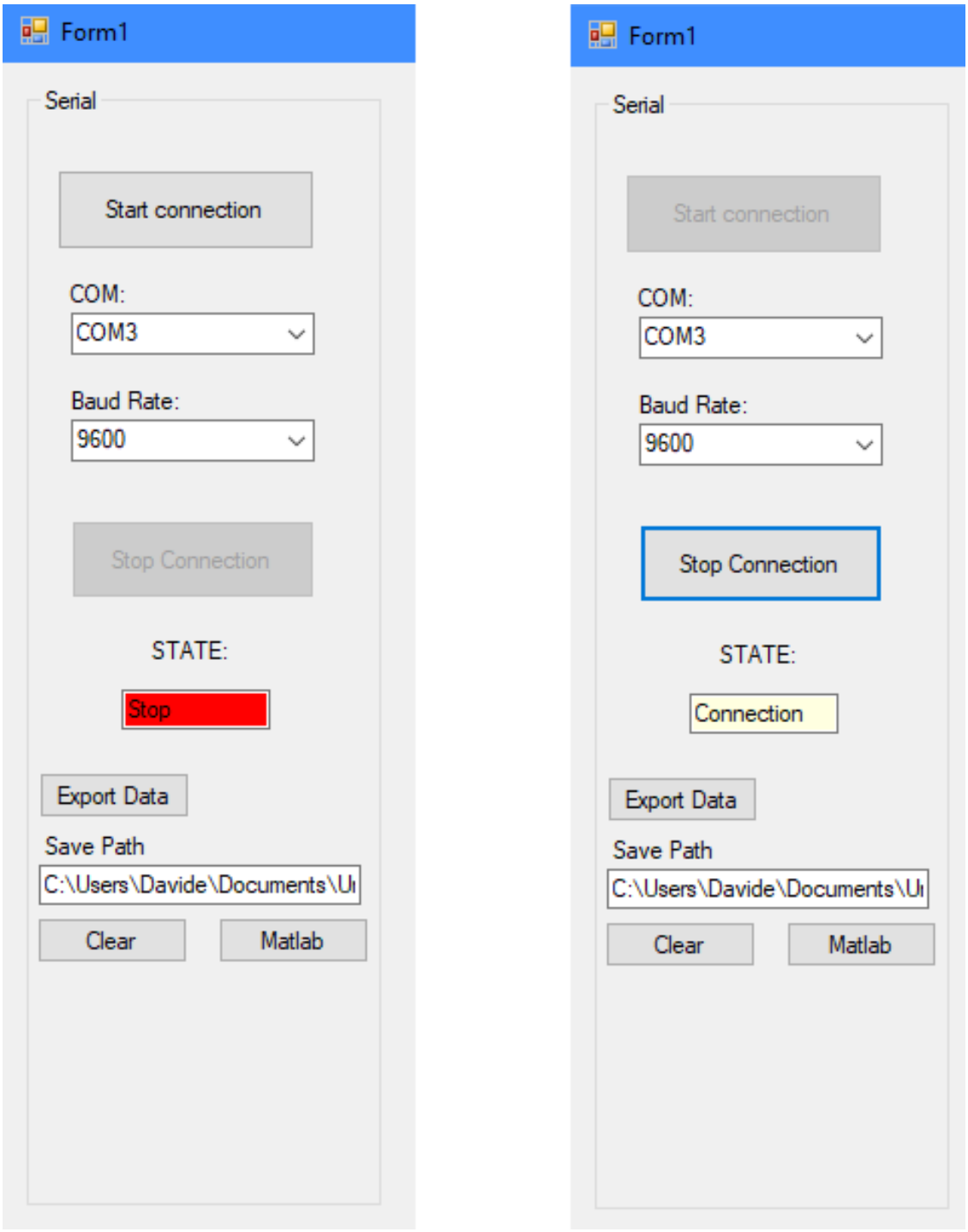

Figure 5.3: User interface working condition

Chapter 6

Test and Validation

The last step of the thesis work and thus of the project has seen the testing of the board. The testing has involved all the main aspects of the device, from the firmware stability, passing through the electric and power control to end with the functional features. During the testing and evaluation, the whole system has been taken in mind.

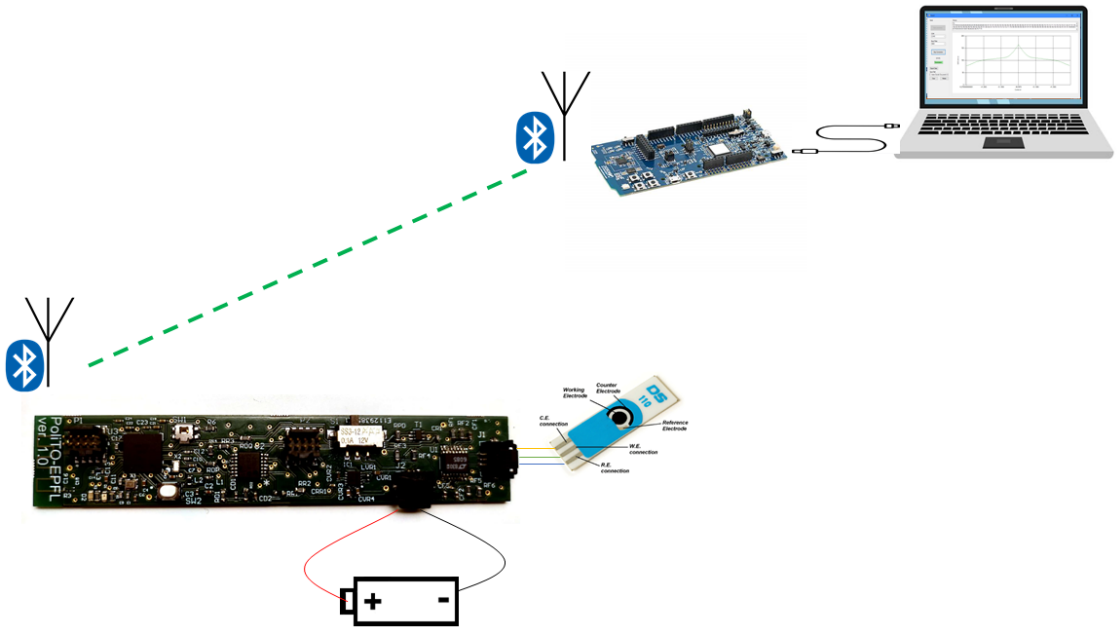

Figure 6.1: Final design system: In the bottom part there is the custom PCB connected to the electrode. In the top part, there is the receiver and terminal PC

6.1 Firmware Test

The firmware developed in Segger Embedded studio has been tested as first. Indeed during the production of the board, the Nordic DK and the Nordic dongle devices have been exploited. Of course, during the test of the firmware on the development kit, the slight differences with the custom board are all taken into account, the firmware has been evaluated thanks to the light indicator LED on the board and the oscilloscope, by what all the main signal generated have been investigated. Moreover, in order to verify the behaviour in the measuring task, the signal generator has been connected and a square wave, with different duty cycle and frequency, has been measured, observing the correct time gap between two consecutive high shapes of the wave acts to simulate the quasi digital signal. In this preliminary test, the BLE operation has been verified exploiting commercial available Android app that simulates the terminal and where the correctness of the connection and data sent has been checked. Once everything has resulted exact, the code firmware for the receiver has been flashed on the dongle device. And finally, the receiving of the data has been checked by means the personal computer terminal, verifying that also this last part of the system works well as expected.
The final step to verify the goodness of the firmware, the code has been adapted to the board. Through the J-link connector, it has been possible to program the custom board exploiting the debug connector.

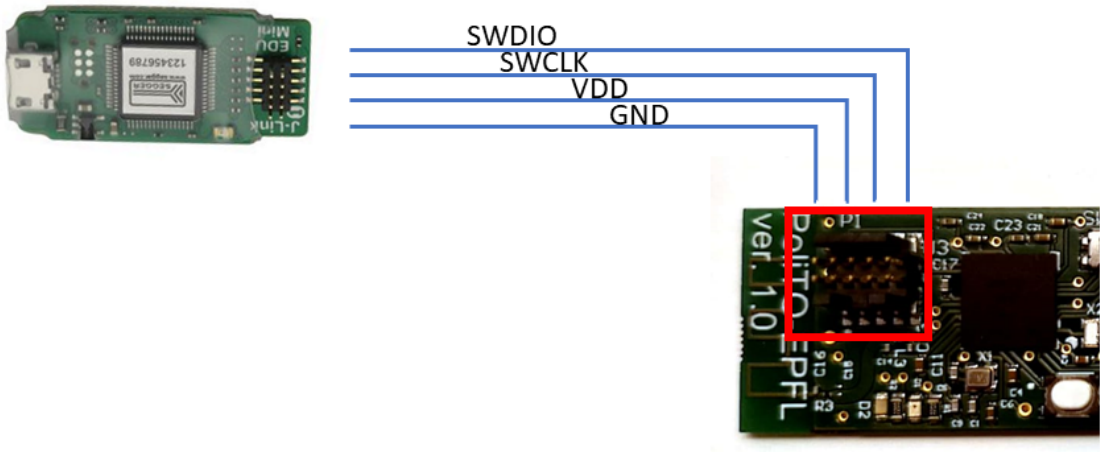

Figure 6.2: J-Link connector to flash and debug the custom PCB

As observable by 6.2, apart for the power reference pin (VDD and GND), the two main connections are the SWDIO and SWCLK. The serial wire debug (SWD) is a two pin SWDIO/SWCLK (it has the same JTAG protocol exploited in the DK programming) bi-directional wire protocol, defined in the ARM Debug interface.

In details, SWCLK is the clock signal to target CPU and the SWDIO is the input-output bi-directional data pin[27].

6.2 Electrical Test & Functional Test

Once executed all the previous operation to set-up the board from the hardware components point of view and the firmware configuration, the electrical test has been performed. The testing acts to control that in predefined points and pin of the circuit, the voltage expected reference values are correct. At this purpose, the testing header as described in the previous chapter plays an important role 6.3.

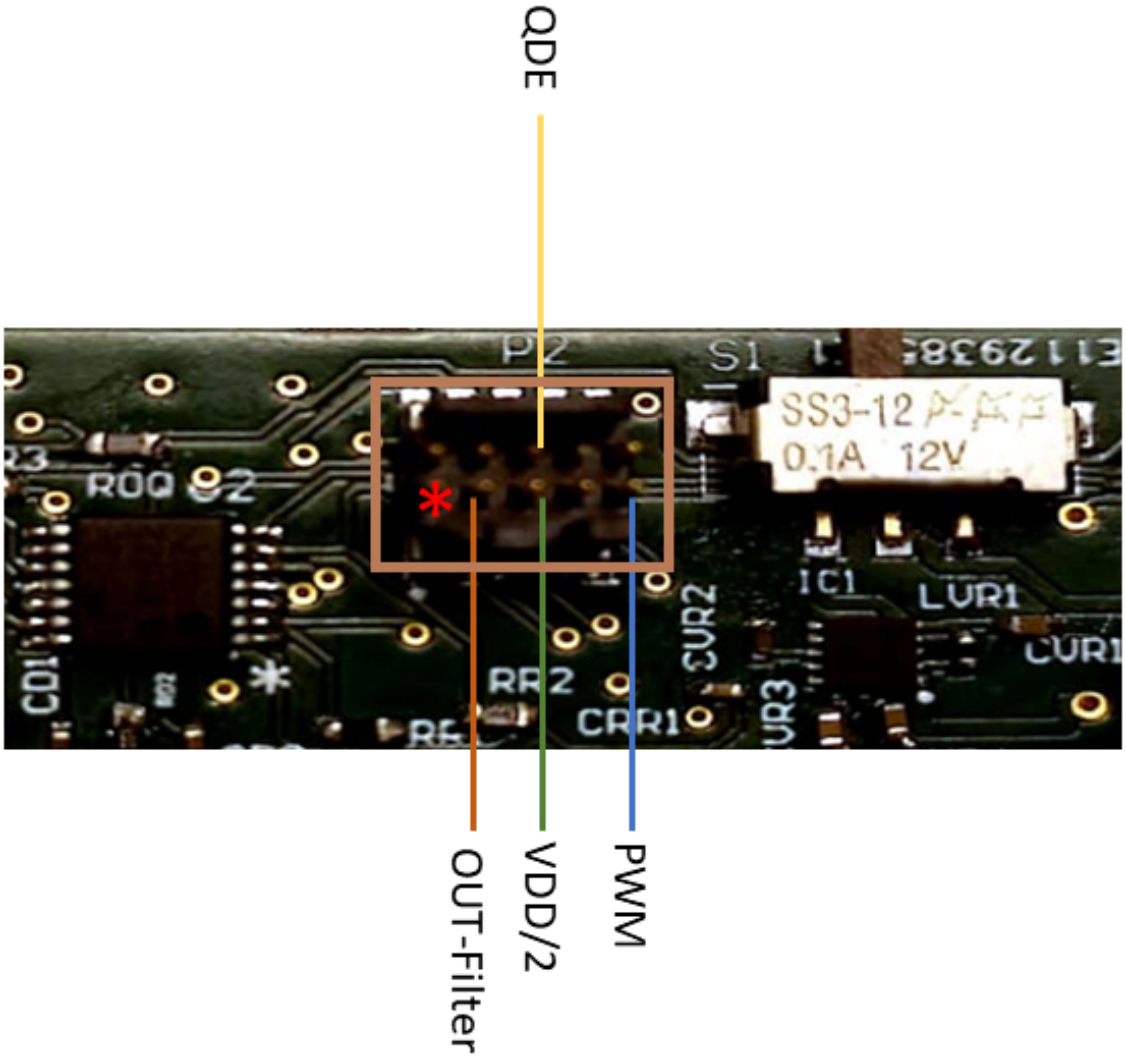

Figure 6.3: PCB header for the testing

As first, the mechanical switch, reported in figure 6.3 on the right, is controlled

to verify the ON-OFF operation. Successively the output-pin of the voltage regulator is tested with various input voltage level, in order to guarantee the correct voltage reference to the whole afterwards circuit for the safeness of the components. Then the header pin highlighted in the figure is exploited to control as first the known voltage value $VDD/2$ in a known circuit point. Furthermore, the voltage value provided by the voltage regulator is investigated in other points of the board as vias, components pin and debugger header where the ground and 3.3 V have to be stable. Once obtained positive feedback from the electrical point of view, the attention is moved to the verification of the two main signals. As notable from the 6.3 two pins are dedicated to the Pulse Width Modulation, the output of the micro-controller and Quasi-Digital events input of this latter. To check the PWM, different situations are tested just changing the firmware sequence of PWM cycle. As an example some cases measured with the oscilloscope are reported:

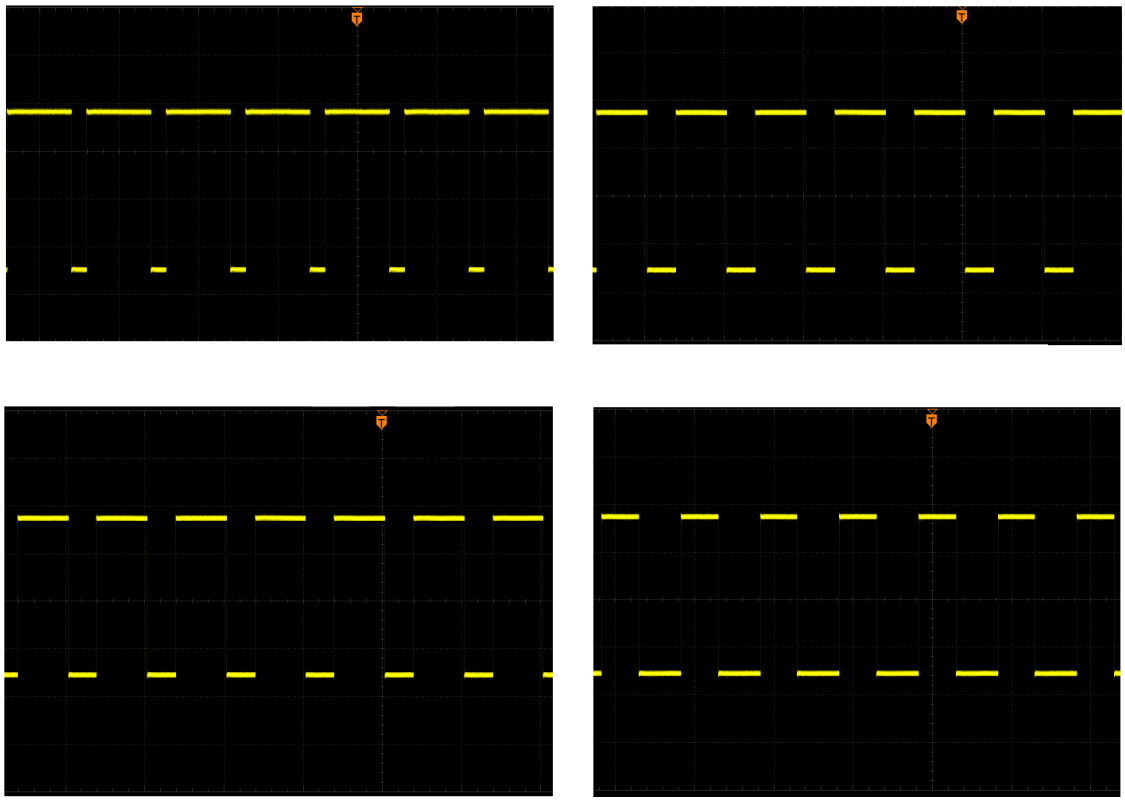

Figure 6.4: Oscilloscope image for various PWM duty-cycle

In all the cases the frequency generated and the expected duty-cycle are measured on the oscilloscope to obtain the double comparison and check. Strictly linked to the duty-cycle is the further pin "OUT-filter". This point of the circuit represents the output of the filter and the voltage ramp that drives the electrode cell. The

staircase output is verified by the oscilloscope and a part of the signal zoomed to observe the stair behaviour:

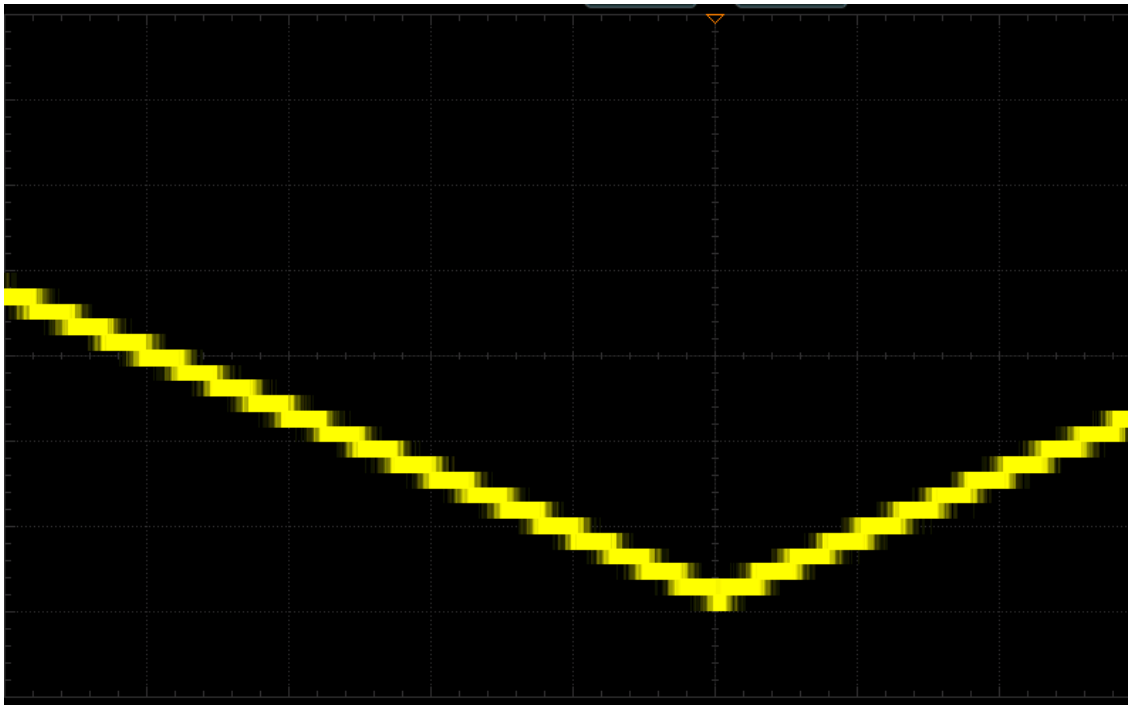

Figure 6.5: Output filter signal: at each duty-cycle value corresponds to a voltage reference level holds for few milliseconds. The staircase is obtained observing the signal with large time scale

At this point, the final verification about the driven electrode circuit is obtained exploiting two-channel of the oscilloscope, one connected to the WE output pin of the board for the reference value at $VDD/2$ and the other linked to the CE and RE shorted pins. The results as expected show a triangular curve that crosses the reference WE voltage value. The point where the two value overlap stands for the zero for the Cyclic Voltammetry graphic. In the verification an unexpected constant offset has been observed, however this not represents a problem because the important point is the difference between the voltage triangular value and the reference one and not the former absolute value, the offset implicates just a different selection in the PWM sequence of duty-cycle, easily configurable in the firmware, the aforementioned consideration are verifiable in the figure below 6.6 The last signal to verify consist in the micro-controller input quasi-digital event-based wave. The final shape, as described, is the front-end circuit result of the electrode current response. To check the correct behaviour and time-evaluation by the micro-controller a comparison among the time measured on the oscilloscope and the one reported in the final BLE data of the processor are done. In all the cases

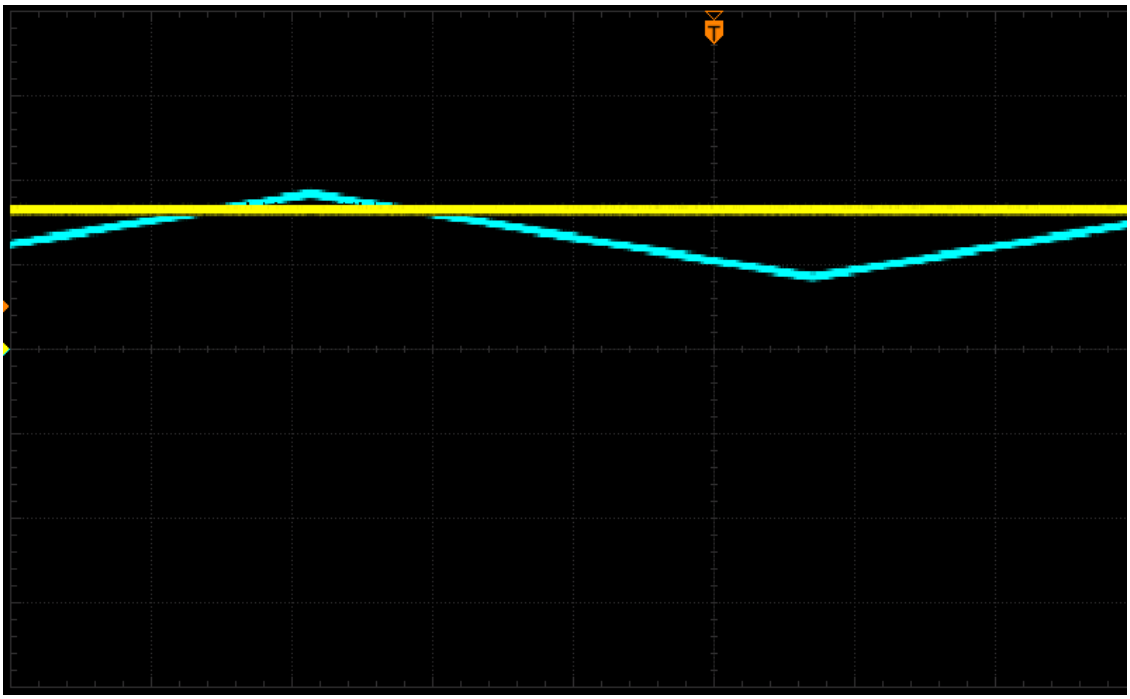

Figure 6.6: Driven electrode voltage signals. The yellow line represents the reference voltage value at WE. The blue line changes with respect to the duty-cycle sequence in the PWM generated. The oscilloscope picture is obtained with a large time scale

positive feedback has demonstrated the goodness of the system measuring. In the following (figure 6.7) the shape of the quasi-digital wave is reported in some example cases: To conclude the testing section, some test measurements are performed before to move on the drug's evaluation. The method exploited in this verification has consisted in the replacing of the electrode with a simple resistor model. In other words, in place of the sensor, it is connected to a resistor and the current response evaluated at each step of the staircase voltage. With this approach, knowing the value of the applied voltage and the resistance it has been possible to obtain the time-current relation.

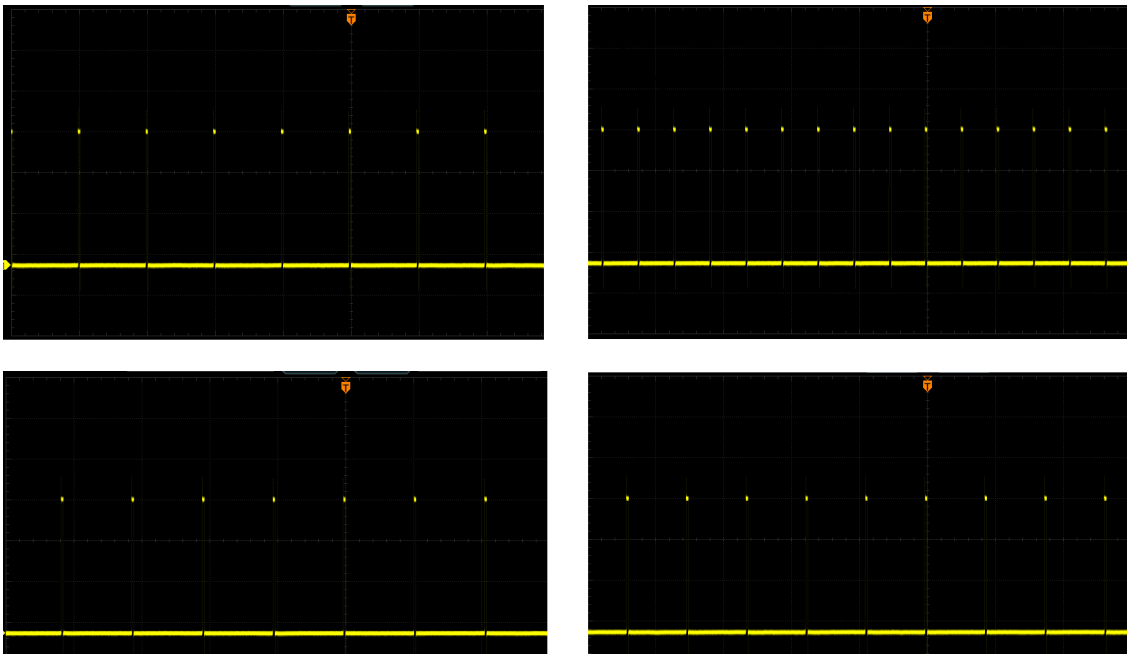

Figure 6.7: Four different cases of the quasi digital signal. The time gap between two events is proportional to the input current

6.3 Final Result

To conclude the thesis project the data obtained from the measure are summarized in this chapter. First of all, a brief table is reported to focus the attention on the electric and features of the device.

	Min	Max
Voltage supply	3.3 V	7 V
Idle Current	9 mA	13 mA
On Current	19 mA	24 mA
Idle Power	29.7 mW	42.9 mW
On Power	62.7 mW	79.2 mW
BLE RSSI	-95 dBm @ 18 m	-51 dBm @ 0 m

Table 6.1: Electric and communication features of the device

As disclosed, the first set of validation measures are collected exploiting a resistor. In this phase by means of Matlab support, the fitting of the data and the circuital model are computed. The measurements are done exploiting the different value of resistance and the final preliminary results reported in the following Matlab plot (figure 6.8). From figure 6.8 is observable as for the different value of resistance the relation between time-event and current measured remains more or

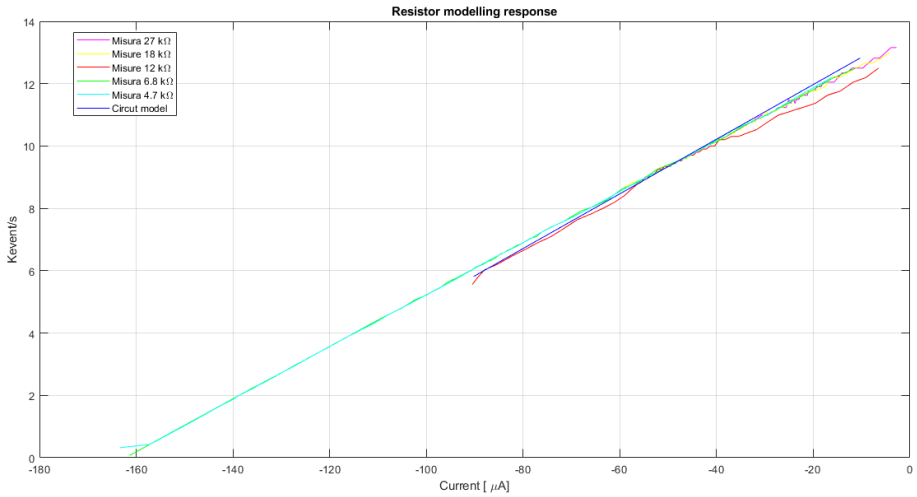

Figure 6.8: Fitting data measured with the resistor model at different values and circuital model curve

less constant. The data in this data are elaborate in order to obtain a fitting model to link in any case the time and the current.
The final experimental data and measuring are obtained exploiting one of the most common anaesthetic substance: the paracetamol. This last validation step of the device is dedicated to the direct measure of drug concentration in a solution. For the experiment, four different solutions are prepared with respectively no presence of paracetamol, 100 µM, 200 µM and 300 µM of paracetamol. These tests are designed to cover the therapeutic range of paracetamol and prove the capability of the system [28]. For all the cases several measurements are done. As a sensor, it is connected to the Screen-printed electrodes based on carbon, gold, platinum, silver or carbon nanotubes inks[29]. This kind of electrode, largest exploited for electrochemical analysis, presents 3.4 x 1.0 x 0.05 cm dimensions and it is mounted on the board employing adapter connector. The set-up of the firmware has consisted in the generation of a PWM sequence that guarantees the voltage drop between the CE (in short circuit with RE) and WE from, more or less, 0 to - 1 V and return up to 0.2 V. The whole sequence duration is configured in ten seconds at frequency of 16 kHz with 313 equidistant points in order to increase the duty-cycle of the 0.2% per time, to obtain an accurate curve. The described curve takes in literature the name of voltammogram and is the final result of the cycle voltammetry. The data collected and elaborated in Matlab have shown the expected behaviour. As first the immediate time quantity returned by the processor measurement is plotted versus the PWM duty-cycle. The exploiting the previous fitting the time between two successive events is converted in current physical quantity.

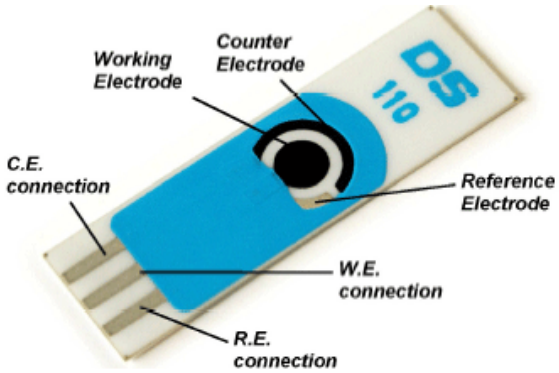

Figure 6.9: ELectrode DropSens DRP-110, image from [29]. In the picture are indicated the position of the three electrodes.

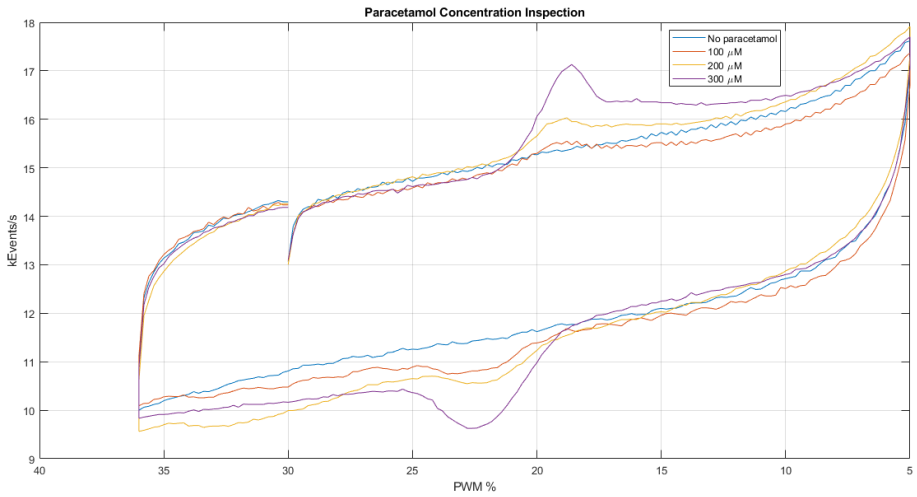

Figure 6.10: Matlab data elaboration: events rate vs. PWM duty-cycle

In the picture is possible to observe for each curve two peaks. One is the Reduction response and the other of the oxidation, the RedOx chemical reaction indeed is at the base of electron flow in the solution turns out in the current signal. The peaks show different maximum value according to the concentration, and this is the proof of the goodness of the customs system: investigating the peak response it is possible in conclusion to extrapolate all the information needed for the continuous monitoring of the anaesthetic drugs.

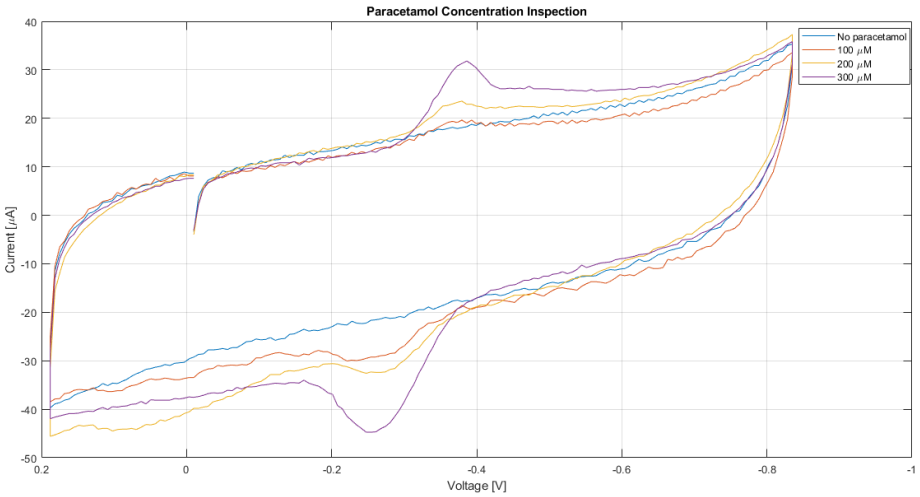

Figure 6.11: Matlab data elaboration: Voltammogram with current response vs. driven electrode voltage.

Chapter 7

Conclusion & Future works

The obtained results concern the evaluation of the paracetamol concentration and the linked voltammogram testify to the device's correct operation and function. The project mirrored the indications analyzed and studied in the state of art about the continuous monitoring of quantity, like drugs, for biomedical application. Moreover, the design of custom board provides the possibility to obtain a compact and low-cost dedicated device. The added of the on-board microcontroller permits to integrate into a unique board solution the front-end circuit acquisition and the driven electrochemical signal with the "mind controller". The decision in the not employing of AD/DA converters have allowed the reduction of power consumption fostering quasi digital solution, as the event-based signal and the pulse width modulation to directly interact with the processor with a square wave. The exploiting of the Bluetooth Low Energy 5 protocol to transmit the collected data constitutes an innovative solution being a communication low power solution and wireless allowing the application of the device with certain range respect to the terminal computer. Therefore, the latter solution is not only good in terms of power but also in terms of effective employment of the device making it portable and independent from wire constraints. The flexible firmware and design make the device easily usable for future development but also and overall ready for the effective application thanks also to the dedicated graphic user interface comfortable for the development phase but overall for the final actual user.
Possible future developments of the device consist of the integration of the board with the electrode sensor to obtain a more compact solution including the design of a comfortable case to protect also the electronic circuit. Furthermore, to completely characterized and validate the board project, the laboratory test with other kinds of opioids and drugs should be performed in other to guarantee an across-the-board application of the device.

Appendix A

Bill of material of PCB

Components	Producer Code	Components	Producer Code
C1	GRM1555C1H120GA01D	C2	GRM155C81C105KE11J
C3	GRM155R71E473KA88D	L1	HK100515NJ-T
C5	GRM1555C1H120GA01D	L2	LBMF1608T100K
C6	GCM155R71A104KA55D	L3	LQP03TN3N9B02D
C7	GCM155R71A104KA55D	LVR1	LK10051R5K-TV
C8	GRM155C81C105KE11J	P1	FTSH-105-01-L-DV-K-TR
C9	GRM1555C1H101JA01D	P2	FTSH-105-01-L-DV-K-TR
C10	GRM1555C1H120GA01D	ROP, ROQ	CRCW06030000Z0EAC
C12	GRM1555C1H120GA01D	R1	CRCW0603100RFKEAC
C13	GJM0335C1H1R0WB01D	R2	CRCW0603100RFKEAC
C14	GJM0335C1H1R0WB01D	R3	RC0201FR-070RL
C16	GJM0335C1E1R5CB01D	R6	CRCW060310K0FKEAC
C17	GRM1555C1H821JA01D	RD1	CRCW0603470KFKEAC
C18	GRM155R61A475MEAAD	RF1, RF2, RF3, RF5, RF6, RD2	CRCW0603100KFKEAC
C21	GRM155R61A475MEAAD	RF4	CRCW060322K0JNEAC
C22	GCM155R71A104KA55D	RF5	CRCW040222K0FKEDC
C23	GCM155R71A104KA55D	RPD	CRCW06031K00FKEAC
C24	GCM155R71A104KA55D	RR1	RCS06036K80FKEA
CD1	GRM155R71E473KA88D	RR2	CRCW060347K0FKEAC
CF1	GRM155C81C105KE11J	RR3	CRCW040210K0FKEDC
CF2	GRM155C81C105KE11J	RR4	CRCW060310K0FKEAC
CF3	GRM155C81C105KE11J	S1	SS312SAH4-R
CR1	GCM1555C1H102JA16J	SW1	PTS840 GK SMTR LFS
CRR1, CD2	GCM155R71H103KA55J	SW2	PTS810 SJK 250 SMTR LFS
CVR1, CVR3, CVR4	ZRB15XC80J106ME05D	T1	2N7002DW-TP
CVR2	GCG155R71C104KA01D	U1	LTC6085CGN#PBF
D1, D2	SML-P12U2TT86R	U2	ADA4807-4ARUZ-R7
IC1	TPS63031DSKT	U3	NRF52840-QIAA-R
J1, J2	503763-0391	X1	XRCGB32M000F1H01R0
J1, J2	503764-0301	X2	LFXTAL062558Reel

Table A.1: Bill of materiale

Appendix B

Firmware code

B.1 Device firmware

```
#include "app_timer.h"
#include "app_uart.h"
#include "app_util_platform.h"
#include "ble_advdata.h"
#include "ble_advertising.h"
#include "ble_conn_params.h"
#include "ble_hci.h"
#include "ble_nus.h"
#include "bsp_btn_ble.h"
#include "nordic_common.h"
#include "nrf.h"
#include "nrf_ble_gatt.h"
#include "nrf_ble_qwr.h"
#include "nrf_pwr_mgmt.h"
#include "nrf_sdh.h"
#include "nrf_sdh_ble.h"
#include "nrf_sdh_soc.h"
#include <stdint.h>
#include <stdio.h>
#include <string.h>
#if defined(UART_PRESENT)
#include "nrf_uart.h"
#endif
#if defined(UARTE_PRESENT)
#include "nrf_uarte.h"
#endif
#include "nrf_log.h"
#include "nrf_log_ctrl.h"
#include "nrf_log_default_backends.h"
#define APP_BLE_CONN_CFG_TAG 1                                  // ...
    A tag identifying the SoftDevice BLE configuration
```

```
31 #define DEVICE_NAME "Anhestetic_device"                          // ...
      Name of device. Will be included in the advertising data.
32 #define NUS_SERVICE_UUID_TYPE BLE_UUID_TYPE_VENDOR_BEGIN     // ...
      UUID type for the Nordic UART Service (vendor specific).
33 #define APP_BLE_OBSERVER_PRIO 3                                  // ...
      Application's BLE observer priority. You shouldn't need to ...
      modify this value.
34 #define APP_ADV_INTERVAL 64                                      // ...
      The advertising interval (in units of 0.625 ms. This value ...
      corresponds to 40 ms).
35 #define APP_ADV_DURATION 18000                                   // ...
      The advertising duration (180 seconds) in units of 10 ...
      milliseconds.
36 #define MIN_CONN_INTERVAL MSEC_TO_UNITS(20, UNIT_1_25_MS)    // ...
      Minimum acceptable connection interval (20 ms), Connection ...
      interval uses 1.25 ms units.
37 #define MAX_CONN_INTERVAL MSEC_TO_UNITS(75, UNIT_1_25_MS)    // ...
      Maximum acceptable connection interval (75 ms), Connection ...
      interval uses 1.25 ms units.
38 #define SLAVE_LATENCY 0                                          // ...
      Slave latency. */
39 #define CONN_SUP_TIMEOUT MSEC_TO_UNITS(8000, UNIT_10_MS)     // ...
      Connection supervisory timeout (4 seconds), Supervision ...
      Timeout uses 10 ms units.
40 #define FIRST_CONN_PARAMS_UPDATE_DELAY APP_TIMER_TICKS(5000) // ...
      Time from initiating event (connect or start of notification).
41 #define NEXT_CONN_PARAMS_UPDATE_DELAY APP_TIMER_TICKS(30000) // ...
      Time between each call to sd_ble_gap_conn_param_update after ...
      the first call (30 seconds).
42 #define MAX_CONN_PARAMS_UPDATE_COUNT 3                           // ...
      Number of attempts before giving up the connection parameter ...
      negotiation.
43 #define DEAD_BEEF 0xDEADBEEF                                     // ...
      Value used as error code on stack dump, can be used to ...
      identify stack location on stack unwind.
44 BLE_NUS_DEF(m_nus, NRF_SDH_BLE_TOTAL_LINK_COUNT); ...
                            // BLE NUS service instance.
45 NRF_BLE_GATT_DEF(m_gatt); ...
                                                  //GATT module ...
      instance.
46 NRF_BLE_QWR_DEF(m_qwr); ...
                                                    // Context ...
      for the Queued Write module.
47 BLE_ADVERTISING_DEF(m_advertising); ...
                                          // Advertising module ...
      instance.
48 static uint16_t m_conn_handle = BLE_CONN_HANDLE_INVALID; ...
                     // Handle of the current connection.
```

```
49 static uint16_t m_ble_nus_max_data_len = ...
       BLE_GATT_ATT_MTU_DEFAULT - 3; // Maximum length of data (in ...
       bytes) that can be transmitted to the peer by the Nordic ...
       UART service module.
50 static ble_uuid_t m_adv_uuids[] = ...
                                          //Universally unique ...
       service identifier.
51     {
52         {BLE_UUID_NUS_SERVICE, NUS_SERVICE_UUID_TYPE}};
53
54 #include "app_error.h"
55 #include "app_util_platform.h"
56 #include "boards.h"
57 #include "bsp.h"
58 #include "nrf.h"
59 #include "nrf_delay.h"
60 #include "nrf_drv_clock.h"
61 #include "nrf_drv_gpiote.h"
62 #include "nrf_drv_pwm.h"
63 #include <stdio.h>
64 #include <string.h>
65 #define QDE_IN NRF_GPIO_PIN_MAP(1, 9)          // QD signal input
66 #define button_IN NRF_GPIO_PIN_MAP(1, 15)      // Button_in
67 #define led_1 10                               // LED1
68 #define led_2 9                                // LED2
69 #define idle_front_end 12                      // Move in sleep ...
       the analog circuit
70 uint16_t pwm_seq[1] = {71};                    // Duty Cycle ...
       start point
71 int i;
72 int ind;
73 int result[163] = {0};                         // Output sensor ...
       measurement
74 volatile bool stop = 1;
75 volatile bool flag = 0;
76 volatile bool wait;
77 volatile bool flag_connection = 0;
78
79 // Idle state
80 void idle_state() {
81   nrf_gpio_pin_clear(led_1);
82   nrf_gpio_pin_clear(led_2);
83
84   flag = 0;
85   stop = 0;
86   wait = 0;
87   if (flag_connection == 1) {
88     sd_ble_gap_disconnect(m_conn_handle, ...
          BLE_HCI_REMOTE_USER_TERMINATED_CONNECTION);
89     sd_ble_gap_adv_stop(m_conn_handle);
```

```
90      nrf_drv_gpiote_in_event_disable(QDE_IN);
91      nrf_drv_gpiote_in_event_enable(button_IN, true);
92      nrf_gpio_pin_set(idle_front_end);
93
94      int contatore = 0;
95      do {
96
97        __WFI();
98      } while (wait == 0);
99    } else {
100     __disable_irq();
101     uint32_t icer0;
102     uint32_t icer1;
103     icer0 = NVIC->ICER[0];
104     icer1 = NVIC->ICER[1];
105     NVIC->ICER[0] = 0xffffffff;
106     NVIC->ICER[1] = 0xffffffff;
107     NVIC->ICPR[0] = 0xffffffff;
108     NVIC->ICPR[1] = 0xffffffff;
109     NVIC->ISER[0] = 0x00000040;
110     nrf_drv_gpiote_in_event_disable(QDE_IN);
111     nrf_drv_gpiote_in_event_enable(button_IN, true);
112     nrf_gpio_pin_set(idle_front_end);
113     __enable_irq();
114     __WFI();
115     __disable_irq();
116     NVIC->ISER[0] = icer0;
117     NVIC->ISER[1] = icer1;
118     NVIC->ICPR[0] = 0xffffffff;
119     NVIC->ICPR[1] = 0xffffffff;
120     __enable_irq();
121   }
122 }
123 // Interrupt routine
124 void qde_pin_handler(nrf_drv_gpiote_pin_t pin, ...
        nrf_gpiote_polarity_t action) {
125   NRF_TIMER1->TASKS_CAPTURE[0] = 1;
126   result[i] = NRF_TIMER1->CC[0];
127   NRF_TIMER1->TASKS_CLEAR = 1;
128   NRF_TIMER1->TASKS_START = 1;
129 }
130 static void advertising_stop(void) {
131   uint32_t err_code = ble_advertising_start(&m_advertising, ...
          BLE_ADV_MODE_IDLE);
132   APP_ERROR_CHECK(err_code);
133 }
134 void reset() {
135   NRF_TIMER2->TASKS_CLEAR = 1;
136   NRF_TIMER2->TASKS_START = 1;
137   while (nrf_gpio_pin_read(button_IN) ≤ 0.5)
```

```
138     ;
139   NRF_TIMER2->TASKS_STOP = 1;
140   NRF_TIMER2->TASKS_CAPTURE[0] = 1;
141   int reset_cond = NRF_TIMER2->CC[0];
142   if (reset_cond ≥ 3000000) {
143     flag_connection = 0;
144     stop = 1;
145     uint32_t err_code = sd_power_system_off();
146     APP_ERROR_CHECK(err_code);
147   }
148 }
149 void button_in_pin_handler(nrf_drv_gpiote_pin_t pin, ...
      nrf_gpiote_polarity_t action) {
150   wait = 1;
151   if (flag == 1) {
152     stop = 1;
153   }
154   reset();
155 }
156
157 void timer_setup() {
158   NRF_TIMER1->MODE = TIMER_MODE_MODE_Timer;
159   NRF_TIMER1->TASKS_CLEAR = 1;
160   NRF_TIMER1->BITMODE = TIMER_BITMODE_BITMODE_24Bit;
161   NRF_TIMER2->MODE = TIMER_MODE_MODE_Timer;
162   NRF_TIMER2->TASKS_CLEAR = 1;
163   NRF_TIMER2->BITMODE = TIMER_BITMODE_BITMODE_24Bit;
164 }
165
166 // interrupt initialization
167 static void interrupt_init(void) {
168   ret_code_t err_code;
169   err_code = nrf_drv_gpiote_init();
170   APP_ERROR_CHECK(err_code);
171   nrf_drv_gpiote_in_config_t in_config = ...
        GPIOTE_CONFIG_IN_SENSE_HITOLO(true);
172   in_config.pull = NRF_GPIO_PIN_PULLUP;
173   err_code = nrf_drv_gpiote_in_init(QDE_IN, &in_config, ...
        qde_pin_handler);
174   APP_ERROR_CHECK(err_code);
175   nrf_drv_gpiote_in_event_enable(QDE_IN, true);
176 }
177
178 // wake-up init
179 static void wake_up_init(void) {
180   ret_code_t err_code;
181
182   //Configure sense input pin to enable wakeup and interrupt on ...
        button press.
```

```
183   nrf_drv_gpiote_in_config_t in_config2 = ...
          GPIOTE_CONFIG_IN_SENSE_HITOLO(false);     //Configure to ...
          generate interrupt and wakeup on pin signal low.
184   in_config2.pull = NRF_GPIO_PIN_PULLUP; ...
                                                    //Configure ...
          pullup for input pin to prevent it from floting.
185   err_code = nrf_drv_gpiote_in_init(button_IN, &in_config2, ...
          button_in_pin_handler); //Initialize the pin with ...
          interrupt handler in_pin_handler
186   APP_ERROR_CHECK(err_code); ...
                                                                ...
          //Check potential error
187   nrf_drv_gpiote_in_event_enable(button_IN, true); ...
                                        //Enable event and ...
          interrupt for the wakeup pin
188 }
189
190 // PWM initilization
191 void PWM_gen(void) {
192   NRF_PWM0->PSEL.OUT[1] = 26;        // PWM out pin
193
194   NRF_PWM0->ENABLE = 0x00000001;
195   NRF_PWM0->MODE = 0x00000000;
196   NRF_PWM0->PRESCALER = 0x00000003; // 16MHz/2^3=2Mhz
197
198   NRF_PWM0->COUNTERTOP = 100;        // PWM_Freq = 20kHz
199   NRF_PWM0->LOOP = 0x00000000;
200
201   NRF_PWM0->SEQ[0].PTR = (uint32_t)(pwm_seq);
202   NRF_PWM0->SEQ[0].CNT = 1;
203
204   NRF_PWM0->SEQ[0].REFRESH = 0;
205   NRF_PWM0->SEQ[0].ENDDELAY = 0;
206   NRF_PWM0->TASKS_SEQSTART[0] = 1;
207
208   NRF_PWM0->ENABLE = 0x00000001;
209 }
210
211 void meas_state() {
212
213   nrf_drv_gpiote_in_event_enable(QDE_IN, true); // enable QDE ...
          interrupt
214                                                 // Start DC ...
                                                       variation ...
                                                       for PWM from ...
                                                       10% to 80% ...
                                                       of period
215     pwm_seq[0]=71;
216   for (i = 0; i ≤ 60; i++)                  // up Duty cycle
217   {
```

```
218       pwm_seq[0] -=1;                                  // 1% at time
219       NRF_PWM0->TASKS_SEQSTART[0] = 1;
220       nrf_delay_ms(50);                                // 1000 period ...
              before switch Duty cycle
221
222     }
223     for (i = 61; i ≤ 120; i++)                         // down Duty cycle
224     {
225       pwm_seq[0] +=1;                                  // 1% at time
226       NRF_PWM0->TASKS_SEQSTART[0] = 1;
227       nrf_delay_ms(50);                                // 1000 period ...
              before switch Duty cycle
228       nrf_gpio_pin_toggle(led_1);
229     }
230     nrf_drv_gpiote_in_event_disable(QDE_IN);           // disable QDE ...
          interrupt
231 }
232
233
234 // This function will be called in case of an assert in the ...
       SoftDevice.
235
236
237 void assert_nrf_callback(uint16_t line_num, const uint8_t ...
       *p_file_name) {
238   app_error_handler(DEAD_BEEF, line_num, p_file_name);
239 }
240
241 //Initializing the timer module.
242
243 static void timers_init(void) {
244   ret_code_t err_code = app_timer_init();
245   APP_ERROR_CHECK(err_code);
246 }
247
248 //Function for the GAP initialization.
249 //This function will set up all the necessary GAP (Generic ...
       Access Profile) parameters of
250 // the device. It also sets the permissions and appearance.
251
252 static void gap_params_init(void) {
253   uint32_t err_code;
254   ble_gap_conn_params_t gap_conn_params;
255   ble_gap_conn_sec_mode_t sec_mode; // security mode  0
256
257   BLE_GAP_CONN_SEC_MODE_SET_OPEN(&sec_mode);
258
259   err_code = sd_ble_gap_device_name_set(&sec_mode, // set del nome
260       (const uint8_t *)DEVICE_NAME,
261       strlen(DEVICE_NAME));
```

```
262   APP_ERROR_CHECK(err_code);
263
264   memset(&gap_conn_params, 0, sizeof(gap_conn_params));
265
266   gap_conn_params.min_conn_interval = MIN_CONN_INTERVAL; // ...
          intervalli di connessione
267   gap_conn_params.max_conn_interval = MAX_CONN_INTERVAL;
268   gap_conn_params.slave_latency = SLAVE_LATENCY;
269   gap_conn_params.conn_sup_timeout = CONN_SUP_TIMEOUT;
270
271   err_code = sd_ble_gap_ppcp_set(&gap_conn_params); //set the ...
          packet 8bytes
272   APP_ERROR_CHECK(err_code);
273 }
274
275 //Function for handling Queued Write Module errors.
276 //  A pointer to this function will be passed to each service ...
      which may need to inform the
277 //  application about an error.
278
279
280 static void nrf_qwr_error_handler(uint32_t nrf_error) {
281   APP_ERROR_HANDLER(nrf_error);
282 }
283
284 //Function for handling the data from the Nordic UART Service.
285 //This function will process the data received from the Nordic ...
      UART BLE Service and send
286 //  the sensor result
287
288 static void tx_ready_handler(ble_nus_evt_t *p_ble_evt) {
289   if (p_ble_evt->type == BLE_NUS_EVT_TX_RDY) {
290   }
291 }
292
293 static void nus_data_handler() {
294   uint32_t err_code;
295   if (stop == 0)
296
297   {
298
299     char data[120] = {0};
300     for (ind = 0; ind ≤ 120; ind++) {
301       uint16_t length = snprintf(data, sizeof(data), "%d,", ...
              result[ind]);//
302       do {
303         err_code = ble_nus_data_send(&m_nus, data, &length, ...
              m_conn_handle);
304         if ((err_code != NRF_ERROR_INVALID_STATE) &&
305             (err_code != NRF_ERROR_RESOURCES) &&
```

```
306               (err_code != NRF_ERROR_NOT_FOUND)) {
307             APP_ERROR_CHECK(err_code);
308           }
309         } while (err_code == NRF_ERROR_RESOURCES);
310       }
311       uint16_t length = snprintf(data, sizeof(data), "\n");
312       do {
313         err_code = ble_nus_data_send(&m_nus, data, &length, ...
              m_conn_handle);
314         if ((err_code != NRF_ERROR_INVALID_STATE) &&
315             (err_code != NRF_ERROR_RESOURCES) &&
316             (err_code != NRF_ERROR_NOT_FOUND)) {
317           APP_ERROR_CHECK(err_code);
318         }
319       } while (err_code == NRF_ERROR_RESOURCES);
320     }
321   }
322
323   //Function for initializing services that will be used by the ...
        application.
324
325   static void services_init(void) {
326     uint32_t err_code;
327     ble_nus_init_t nus_init;
328     nrf_ble_qwr_init_t qwr_init = {0};
329
330     // Initialize Queued Write Module.
331     qwr_init.error_handler = nrf_qwr_error_handler;
332
333     err_code = nrf_ble_qwr_init(&m_qwr, &qwr_init);
334     APP_ERROR_CHECK(err_code);
335
336     // Initialize NUS.
337     memset(&nus_init, 0, sizeof(nus_init));
338
339     //nus_init.data_handler = nus_data_handler;
340     nus_init.data_handler = tx_ready_handler;
341     err_code = ble_nus_init(&m_nus, &nus_init);
342     APP_ERROR_CHECK(err_code);
343   }
344
345   //Function for handling an event from the Connection Parameters ...
        Module.
346
347   static void on_conn_params_evt(ble_conn_params_evt_t *p_evt) {
348     uint32_t err_code;
349
350     if (p_evt->evt_type == BLE_CONN_PARAMS_EVT_FAILED) {
351       err_code = sd_ble_gap_disconnect(m_conn_handle, ...
            BLE_HCI_CONN_INTERVAL_UNACCEPTABLE);
```

```
352       APP_ERROR_CHECK(err_code);
353     }
354   }
355
356   //Function for handling errors from the Connection Parameters ...
        module.
357
358
359   static void conn_params_error_handler(uint32_t nrf_error) {
360     APP_ERROR_HANDLER(nrf_error);
361   }
362
363   //Function for initializing the Connection Parameters module.
364   static void conn_params_init(void) {
365     uint32_t err_code;
366     ble_conn_params_init_t cp_init;
367
368     memset(&cp_init, 0, sizeof(cp_init));
369
370     cp_init.p_conn_params = NULL;
371     cp_init.first_conn_params_update_delay = ...
          FIRST_CONN_PARAMS_UPDATE_DELAY; // Time from initiating ...
          event (connect or start of notification).
372     cp_init.next_conn_params_update_delay = ...
          NEXT_CONN_PARAMS_UPDATE_DELAY;
373     cp_init.max_conn_params_update_count = ...
          MAX_CONN_PARAMS_UPDATE_COUNT;
374     cp_init.start_on_notify_cccd_handle = BLE_GATT_HANDLE_INVALID;
375     cp_init.disconnect_on_fail = false;
376     cp_init.evt_handler = on_conn_params_evt;
377     cp_init.error_handler = conn_params_error_handler;
378     err_code = ble_conn_params_init(&cp_init);
379     APP_ERROR_CHECK(err_code);
380   }
381
382   //Function for putting the chip into sleep mode.
383   //This function will not return.
384
385   static void sleep_mode_enter(void) {
386     uint32_t err_code = bsp_indication_set(BSP_INDICATE_IDLE);
387     APP_ERROR_CHECK(err_code);
388
389     // Prepare wakeup buttons.
390     err_code = bsp_btn_ble_sleep_mode_prepare();
391     APP_ERROR_CHECK(err_code);
392
393     //Go to system-off mode (this function will not return; ...
          wakeup will cause a reset).
394     //nrf_gpio_cfg_sense_input(button_IN, NRF_GPIO_PIN_PULLUP, ...
          NRF_GPIO_PIN_SENSE_HIGH);
```

```
395   err_code = sd_power_system_off();
396   APP_ERROR_CHECK(err_code);
397 }
398
399 //Function for handling advertising events.
400
401 static void on_adv_evt(ble_adv_evt_t ble_adv_evt) {
402   uint32_t err_code;
403
404   switch (ble_adv_evt) {
405   case BLE_ADV_EVT_FAST:
406     err_code = bsp_indication_set(BSP_INDICATE_ADVERTISING);
407     APP_ERROR_CHECK(err_code);
408     break;
409   case BLE_ADV_EVT_IDLE:
410     sleep_mode_enter();
411     break;
412   default:
413     break;
414   }
415 }
416
417 // Function for handling BLE events.
418
419 static void ble_evt_handler(ble_evt_t const *p_ble_evt, void ...
      *p_context) {
420   uint32_t err_code;
421
422   switch (p_ble_evt->header.evt_id) {
423   case BLE_GAP_EVT_CONNECTED:
424     NRF_LOG_INFO("Connected");
425     err_code = bsp_indication_set(BSP_INDICATE_CONNECTED);
426     APP_ERROR_CHECK(err_code);
427     m_conn_handle = p_ble_evt->evt.gap_evt.conn_handle;
428     err_code = nrf_ble_qwr_conn_handle_assign(&m_qwr, ...
          m_conn_handle);
429     APP_ERROR_CHECK(err_code);
430     break;
431
432   case BLE_GAP_EVT_DISCONNECTED:
433     NRF_LOG_INFO("Disconnected");
434     // LED indication will be changed when advertising starts.
435     m_conn_handle = BLE_CONN_HANDLE_INVALID;
436     break;
437     // PHY = Physical layer
438   case BLE_GAP_EVT_PHY_UPDATE_REQUEST: {
439     NRF_LOG_DEBUG("PHY update request.");
440     ble_gap_phys_t const phys =
441         {
442             .rx_phys = BLE_GAP_PHY_AUTO,
```

```
443             .tx_phys = BLE_GAP_PHY_AUTO,
444         };
445     err_code = ...
            sd_ble_gap_phy_update(p_ble_evt->evt.gap_evt.conn_handle, ...
            &phys);
446     APP_ERROR_CHECK(err_code);
447   } break;
448
449   case BLE_GAP_EVT_SEC_PARAMS_REQUEST:
450     err_code = sd_ble_gap_sec_params_reply(m_conn_handle, ...
            BLE_GAP_SEC_STATUS_PAIRING_NOT_SUPP, NULL, NULL);
451     APP_ERROR_CHECK(err_code);
452     break;
453
454   case BLE_GATTS_EVT_SYS_ATTR_MISSING:
455     // No system attributes have been stored.
456     err_code = sd_ble_gatts_sys_attr_set(m_conn_handle, NULL, ...
            0, 0);
457     APP_ERROR_CHECK(err_code);
458     break;
459
460   case BLE_GATTC_EVT_TIMEOUT:
461     // Disconnect on GATT Client timeout event.
462     err_code = ...
            sd_ble_gap_disconnect(p_ble_evt->evt.gattc_evt.conn_handle,
463          BLE_HCI_REMOTE_USER_TERMINATED_CONNECTION);
464     APP_ERROR_CHECK(err_code);
465     break;
466
467   case BLE_GATTS_EVT_TIMEOUT:
468     // Disconnect on GATT Server timeout event.
469     err_code = ...
            sd_ble_gap_disconnect(p_ble_evt->evt.gatts_evt.conn_handle,
470          BLE_HCI_REMOTE_USER_TERMINATED_CONNECTION);
471     APP_ERROR_CHECK(err_code);
472     break;
473
474   default:
475     // No implementation needed.
476     break;
477   }
478 }
479
480 //Function for the SoftDevice initialization.
481 //This function initializes the SoftDevice and the BLE event ...
       interrupt.
482
483 static void ble_stack_init(void) {
484   ret_code_t err_code;
485
```

```
486   err_code = nrf_sdh_enable_request();
487   APP_ERROR_CHECK(err_code);
488
489   // Configure the BLE stack using the default settings.
490   // Fetch the start address of the application RAM.
491   uint32_t ram_start = 0;
492   err_code = nrf_sdh_ble_default_cfg_set(APP_BLE_CONN_CFG_TAG, ...
          &ram_start);
493   APP_ERROR_CHECK(err_code);
494
495   // Enable BLE stack.
496   err_code = nrf_sdh_ble_enable(&ram_start);
497   APP_ERROR_CHECK(err_code);
498
499   // Register a handler for BLE events.
500   NRF_SDH_BLE_OBSERVER(m_ble_observer, APP_BLE_OBSERVER_PRIO, ...
          ble_evt_handler, NULL);
501 }
502
503 //Module for negotiating and keeping track of GATT connection ...
       parameters and updating the data length
504 //Function for handling events from the GATT library.
505 void gatt_evt_handler(nrf_ble_gatt_t *p_gatt, ...
       nrf_ble_gatt_evt_t const *p_evt) {
506   if ((m_conn_handle == p_evt->conn_handle) && (p_evt->evt_id ...
          == NRF_BLE_GATT_EVT_ATT_MTU_UPDATED)) {
507     m_ble_nus_max_data_len = p_evt->params.att_mtu_effective - ...
          OPCODE_LENGTH - HANDLE_LENGTH; // MAX transmission unit
508     NRF_LOG_INFO("Data len is set to 0x%X(%d)", ...
          m_ble_nus_max_data_len, m_ble_nus_max_data_len);
509   }
510   NRF_LOG_DEBUG("ATT MTU exchange completed. central 0x%x ...
          peripheral 0x%x",
511       p_gatt->att_mtu_desired_central,
512       p_gatt->att_mtu_desired_periph);
513 }
514
515 //Function for initializing the GATT library.
516 void gatt_init(void) {
517   ret_code_t err_code;
518
519   err_code = nrf_ble_gatt_init(&m_gatt, gatt_evt_handler);
520   APP_ERROR_CHECK(err_code);
521
522   err_code = nrf_ble_gatt_att_mtu_periph_set(&m_gatt, ...
          NRF_SDH_BLE_GATT_MAX_MTU_SIZE);
523   APP_ERROR_CHECK(err_code);
524 }
525
526 //Function for initializing the Advertising functionality.
```

```
527
528 static void advertising_init(void) {
529   uint32_t err_code;
530   ble_advertising_init_t init;
531
532   memset(&init, 0, sizeof(init));
533
534   init.advdata.name_type = BLE_ADVDATA_FULL_NAME;
535   init.advdata.include_appearance = false;
536   init.advdata.flags = ...
        BLE_GAP_ADV_FLAGS_LE_ONLY_LIMITED_DISC_MODE; // limited ...
        discoverable mode for certain time
537   //Unique Identifier
538   init.srdata.uuids_complete.uuid_cnt = sizeof(m_adv_uuids) / ...
        sizeof(m_adv_uuids[0]);
539   init.srdata.uuids_complete.p_uuids = m_adv_uuids;
540
541   init.config.ble_adv_fast_enabled = true;
542   init.config.ble_adv_fast_interval = APP_ADV_INTERVAL;
543   init.config.ble_adv_fast_timeout = APP_ADV_DURATION;
544   init.evt_handler = on_adv_evt;
545
546   err_code = ble_advertising_init(&m_advertising, &init);
547   APP_ERROR_CHECK(err_code);
548
549   ble_advertising_conn_cfg_tag_set(&m_advertising, ...
        APP_BLE_CONN_CFG_TAG);
550 }
551
552 //Function for initializing the nrf log module.
553
554 static void log_init(void) {
555   ret_code_t err_code = NRF_LOG_INIT(NULL);
556   APP_ERROR_CHECK(err_code);
557
558   NRF_LOG_DEFAULT_BACKENDS_INIT();
559 }
560
561 //brief Function for starting advertising.
562
563 static void advertising_start(void) {
564   uint32_t err_code = ble_advertising_start(&m_advertising, ...
        BLE_ADV_MODE_FAST);
565   APP_ERROR_CHECK(err_code);
566 }
567
568 int main(void) {
569   nrf_gpio_cfg_output(led_1);
570   nrf_gpio_cfg_output(led_2);
571
```

```
572   nrf_gpio_pin_set(led_1);
573   nrf_gpio_pin_set(led_2);
574   NRF_CLOCK->TASKS_HFCLKSTART = 1;
575   // Wait for clock to start
576   while (NRF_CLOCK->EVENTS_HFCLKSTARTED == 0)
577     ;
578       NRF_CLOCK->TASKS_LFCLKSTART = 1;
579   // Wait for clock to start
580   while (NRF_CLOCK->EVENTS_LFCLKSTARTED == 0)
581     ;
582   nrf_gpio_cfg_output(idle_front_end);
583   // Initialize BLE connection
584
585   log_init();
586   timers_init();
587   ble_stack_init();
588   gap_params_init();
589   gatt_init();
590   services_init();
591   advertising_init();
592   conn_params_init();
593   // Call timer set
594
595
596   timer_setup();
597   // Call interrupt set
598   interrupt_init();
599   wake_up_init();
600   // Call PWM set
601   PWM_gen();
602   stop = 1; // idle condition
603   for (;;) {
604     if (stop == 1) {
605       idle_state();
606       nrf_gpio_pin_clear(idle_front_end);
607       stop = 0;
608       if (flag_connection == 0) {
609         advertising_start();
610         flag_connection = 1;
611       }
612     }
613     nrf_gpio_pin_clear(led_1);
614     nrf_gpio_pin_set(led_2);
615
616     meas_state();
617     nus_data_handler();
618     flag = 1;
619   }
620   return (0);
621 }
```

B.2 Receiver firmware

```
#include <stdio.h>
#include <stdint.h>
#include <stdbool.h>
#include "nordic_common.h"
#include "app_error.h"
#include "app_uart.h"
#include "ble_db_discovery.h"
#include "app_timer.h"
#include "app_util.h"
#include "bsp_btn_ble.h"
#include "ble.h"
#include "ble_gap.h"
#include "ble_hci.h"
#include "nrf_sdh.h"
#include "nrf_sdh_ble.h"
#include "nrf_sdh_soc.h"
#include "ble_nus_c.h"
#include "nrf_ble_gatt.h"
#include "nrf_pwr_mgmt.h"
#include "nrf_ble_scan.h"
#include "nrf_log_ctrl.h"
#include "nrf_log_default_backends.h"
#include "nordic_common.h"
#include "nrf.h"
#include "ble_hci.h"
#include "ble_advdata.h"
#include "ble_advertising.h"
#include "ble_conn_params.h"
#include "nrf_sdh.h"
#include "nrf_sdh_soc.h"
#include "nrf_sdh_ble.h"
#include "nrf_ble_gatt.h"
#include "nrf_ble_qwr.h"
#include "app_timer.h"
#include "ble_nus.h"
#include "app_uart.h"
#include "app_util_platform.h"
#include "bsp_btn_ble.h"
#include "nrf_pwr_mgmt.h"

#if defined (UART_PRESENT)
#include "nrf_uart.h"
#endif
#if defined (UARTE_PRESENT)
#include "nrf_uarte.h"
#endif
#include "nrf_log.h"
```

```
48
49 #define APP_BLE_CONN_CFG_TAG    1 ...
                                        /**< Tag that refers ...
      to the BLE stack configuration set with @ref sd_ble_cfg_set. ...
      The default tag is @ref BLE_CONN_CFG_TAG_DEFAULT. */
50 #define APP_BLE_OBSERVER_PRIO   3 ...
                                        /**< BLE observer ...
      priority of the application. */
51 #define COMMON_CONFIG_LOG_LEVEL 4
52 #define NRF_LOG_BACKEND_SERIAL_USES_RTT 3
53
54 #define UART_TX_BUF_SIZE        256 ...
                                       /**< UART TX buffer ...
      size. */
55 #define UART_RX_BUF_SIZE        256 ...
                                       /**< UART RX buffer ...
      size. */
56
57 #define NUS_SERVICE_UUID_TYPE   BLE_UUID_TYPE_VENDOR_BEGIN ...
                  /**< UUID type for the Nordic UART Service ...
      (vendor specific). */
58
59 #define ECHOBACK_BLE_UART_DATA  1 ...
                                        /**< Echo the UART ...
      data that is received over the Nordic UART Service (NUS) ...
      back to the sender. */
60
61
62 BLE_NUS_C_DEF(m_ble_nus_c); ...
                                             /**< BLE Nordic ...
      UART Service (NUS) client instance. */
63 NRF_BLE_GATT_DEF(m_gatt); ...
                                               /**< GATT ...
      module instance. */
64 BLE_DB_DISCOVERY_DEF(m_db_disc); ...
                                         /**< Database ...
      discovery module instance. */
65 NRF_BLE_SCAN_DEF(m_scan); ...
                                               /**< Scanning ...
      Module instance. */
66
67 static uint16_t m_ble_nus_max_data_len = ...
      BLE_GATT_ATT_MTU_DEFAULT - OPCODE_LENGTH - HANDLE_LENGTH; ...
      /**< Maximum length of data (in bytes) that can be ...
      transmitted to the peer by the Nordic UART service module. */
68
69 // NUS UUID.
70 static ble_uuid_t const m_nus_uuid =
71 {
72     .uuid = BLE_UUID_NUS_SERVICE,
```

```
73      .type = NUS_SERVICE_UUID_TYPE
74  };
75
76
77  //Function for handling asserts in the SoftDevice.
78
79  void assert_nrf_callback(uint16_t line_num, const uint8_t * ...
        p_file_name)
80  {
81      app_error_handler(0xDEADBEEF, line_num, p_file_name);
82  }
83
84
85  //Function for starting scanning.
86  static void scan_start(void)
87  {
88      ret_code_t ret;
89
90      ret = nrf_ble_scan_start(&m_scan);
91      APP_ERROR_CHECK(ret);
92
93      ret = bsp_indication_set(BSP_INDICATE_SCANNING);
94      APP_ERROR_CHECK(ret);
95  }
96
97
98  //Function for handling Scanning Module events.
99
100 static void scan_evt_handler(scan_evt_t const * p_scan_evt)
101 {
102     ret_code_t err_code;
103
104     switch(p_scan_evt->scan_evt_id)
105     {
106          case NRF_BLE_SCAN_EVT_CONNECTING_ERROR:
107          {
108               err_code = ...
                     p_scan_evt->params.connecting_err.err_code;
109               APP_ERROR_CHECK(err_code);
110          } break;
111
112          case NRF_BLE_SCAN_EVT_CONNECTED:
113          {
114               ble_gap_evt_connected_t const * p_connected =
115                  p_scan_evt->params.connected.p_connected;
116              // Scan is automatically stopped by the connection.
117              printf("Connecting to target ...
                     %02x%02x%02x%02x%02x%02x",
118                        p_connected->peer_addr.addr[0],
119                        p_connected->peer_addr.addr[1],
```

```
120                         p_connected->peer_addr.addr[2],
121                         p_connected->peer_addr.addr[3],
122                         p_connected->peer_addr.addr[4],
123                         p_connected->peer_addr.addr[5]
124                         );
125             } break;
126 
127             case NRF_BLE_SCAN_EVT_SCAN_TIMEOUT:
128             {
129                 printf("Scan timed out.");
130                 scan_start();
131             } break;
132 
133             default:
134                 break;
135         }
136 }
137 
138 
139 
140 //This function will process the data received from the Nordic ...
       UART BLE Service and send it to the UART .
141 
142 static void nus_data_handler(ble_nus_evt_t * p_evt)
143 {
144 
145     if (p_evt->type == BLE_NUS_EVT_RX_DATA)
146     {
147     uint32_t err_code;
148 
149     printf("Received data from BLE NUS. Writing data on UART.");
150     NRF_LOG_HEXDUMP_DEBUG(p_evt->params.rx_data.p_data, ...
           p_evt->params.rx_data.length);
151 
152     for (uint32_t i = 0; i < p_evt->params.rx_data.length; i++)
153     {
154         do
155         {
156             err_code = ...
                   app_uart_put(p_evt->params.rx_data.p_data[i]);
157             if ((err_code != NRF_SUCCESS) && (err_code != ...
                   NRF_ERROR_BUSY))
158             {
159                 printf("Failed receiving NUS message. Error ...
                       0x%x. ", err_code);
160                 APP_ERROR_CHECK(err_code);
161             }
162         } while (err_code == NRF_ERROR_BUSY);
163     }
```

```
164     if ...
            (p_evt->params.rx_data.p_data[p_evt->params.rx_data.length ...
            - 1] == '\r')
165      {
166          while (app_uart_put('\n') == NRF_ERROR_BUSY);
167      }
168      }
169
170  }
171
172  //initializing the scanning and setting the filters.
173
174  static void scan_init(void)
175  {
176      ret_code_t          err_code;
177      nrf_ble_scan_init_t init_scan;
178
179      memset(&init_scan, 0, sizeof(init_scan));
180
181      init_scan.connect_if_match = true;
182      init_scan.conn_cfg_tag     = APP_BLE_CONN_CFG_TAG;
183
184      err_code = nrf_ble_scan_init(&m_scan, &init_scan, ...
             scan_evt_handler);
185      APP_ERROR_CHECK(err_code);
186
187      err_code = nrf_ble_scan_filter_set(&m_scan, ...
             SCAN_UUID_FILTER, &m_nus_uuid);
188      APP_ERROR_CHECK(err_code);
189
190      err_code = nrf_ble_scan_filters_enable(&m_scan, ...
             NRF_BLE_SCAN_UUID_FILTER, false);
191      APP_ERROR_CHECK(err_code);
192  }
193
194
195  // Function for handling database discovery events. Depending ...
       on the UUIDs that are discovered
196  static void db_disc_handler(ble_db_discovery_evt_t * p_evt)
197  {
198      ble_nus_c_on_db_disc_evt(&m_ble_nus_c, p_evt);
199  }
200
201
202  //Function for handling characters received by the Nordic UART ...
       Service (NUS).
203  //This function takes a list of characters of length data_len ...
       and prints the characters out on UART.
204
205
```

```
206 static void ble_nus_chars_received_uart_print(uint8_t * p_data, ...
        uint16_t data_len)
207 {
208     ret_code_t ret_val;
209 
210     NRF_LOG_HEXDUMP_DEBUG(p_data, data_len);
211 
212     for (uint32_t i = 0; i < data_len; i++)
213     {
214         do
215         {
216             ret_val = app_uart_put(p_data[i]);
217             if ((ret_val != NRF_SUCCESS) && (ret_val != ...
                    NRF_ERROR_BUSY))
218             {
219                 printf("app_uart_put failed for index 0x%04x.", i);
220                 APP_ERROR_CHECK(ret_val);
221             }
222         } while (ret_val == NRF_ERROR_BUSY);
223     }
224     if (data_len && p_data[data_len-1] == '\r')
225     {
226         while (app_uart_put('\n') == NRF_ERROR_BUSY);
227     }
228 #if 0
229     if (ECHOBACK_BLE_UART_DATA)
230     {
231         // Send data back to the peripheral.
232         do
233         {
234             ret_val = ble_nus_c_string_send(&m_ble_nus_c, ...
                    p_data, data_len);
235             if ((ret_val != NRF_SUCCESS) && (ret_val != ...
                    NRF_ERROR_BUSY))
236             {
237                 printf("Failed sending NUS message. Error 0x%x. ...
                        ", ret_val);
238                 APP_ERROR_CHECK(ret_val);
239             }
240         } while (ret_val == NRF_ERROR_BUSY);
241     }
242 #endif
243 }
244 
245 //Function for handling app_uart events.
246 This function receives a single character from the app_uart ...
        module and appends it to
247 // a string. The string is sent over BLE when the last ...
        character received is a
```

```
248 //         'new line' '\n' (hex 0x0A) or if the string reaches ...
       the maximum data length.
249
250 void uart_event_handle(app_uart_evt_t * p_event)
251 {
252     static uint8_t data_array[BLE_NUS_MAX_DATA_LEN];
253     static uint16_t index = 0;
254     uint32_t ret_val;
255
256     switch (p_event->evt_type)
257     {
258         //Handling data from UART
259         case APP_UART_DATA_READY:
260             UNUSED_VARIABLE(app_uart_get(&data_array[index]));
261             index++;
262
263             if ((data_array[index - 1] == '\n') || (index ≥ ...
                    (m_ble_nus_max_data_len)))
264             {
265                 printf("Ready to send data over BLE NUS");
266                 NRF_LOG_HEXDUMP_DEBUG(data_array, index);
267
268                 do
269                 {
270                     ret_val = ...
                            ble_nus_c_string_send(&m_ble_nus_c, ...
                            data_array, index);
271                     if ( (ret_val != NRF_ERROR_INVALID_STATE) ...
                            && (ret_val != NRF_ERROR_RESOURCES) )
272                     {
273                         APP_ERROR_CHECK(ret_val);
274                     }
275                 } while (ret_val == NRF_ERROR_RESOURCES);
276
277                 index = 0;
278             }
279             break;
280
281         case APP_UART_COMMUNICATION_ERROR:
282             printf("Communication error occurred while handling ...
                    UART.");
283             APP_ERROR_HANDLER(p_event->data.error_communication);
284             break;
285
286         case APP_UART_FIFO_ERROR:
287             printf("Error occurred in FIFO module used by UART.");
288             APP_ERROR_HANDLER(p_event->data.error_code);
289             break;
290
291         default:
```

```
292             break;
293     }
294 }
295
296
297
298 //This function is called to notify the application of NUS ...
       client events.
299
300
301 static void ble_nus_c_evt_handler(ble_nus_c_t * p_ble_nus_c, ...
       ble_nus_c_evt_t const * p_ble_nus_evt)
302 {
303     ret_code_t err_code;
304
305 switch (p_ble_nus_evt->evt_type)
306 {
307     case BLE_NUS_C_EVT_DISCOVERY_COMPLETE:
308         printf("Discovery complete.");
309         err_code = ble_nus_c_handles_assign(p_ble_nus_c, ...
               p_ble_nus_evt->conn_handle, &p_ble_nus_evt->handles);
310         APP_ERROR_CHECK(err_code);
311
312         err_code = ble_nus_c_tx_notif_enable(p_ble_nus_c);
313         APP_ERROR_CHECK(err_code);
314         printf("Connected to device with Nordic UART Service.");
315         break;
316
317     case BLE_NUS_C_EVT_NUS_TX_EVT:
318         ble_nus_chars_received_uart_print(p_ble_nus_evt->p_data, ...
               p_ble_nus_evt->data_len);
319         break;
320
321     case BLE_NUS_C_EVT_DISCONNECTED:
322         printf("Disconnected.");
323         scan_start();
324         break;
325     }
326 }
327
328
329 // Function for handling shutdown events.
330
331 static bool shutdown_handler(nrf_pwr_mgmt_evt_t event)
332 {
333     ret_code_t err_code;
334
335     err_code = bsp_indication_set(BSP_INDICATE_IDLE);
336     APP_ERROR_CHECK(err_code);
337
```

```
338     switch (event)
339     {
340         case NRF_PWR_MGMT_EVT_PREPARE_WAKEUP:
341             // Prepare wakeup buttons.
342             err_code = bsp_btn_ble_sleep_mode_prepare();
343             APP_ERROR_CHECK(err_code);
344             break;
345 
346         default:
347             break;
348     }
349 
350     return true;
351 }
352 
353 NRF_PWR_MGMT_HANDLER_REGISTER(shutdown_handler, ...
        APP_SHUTDOWN_HANDLER_PRIORITY);
354 
355 
356 //Function for handling BLE events.
357 
358 static void ble_evt_handler(ble_evt_t const * p_ble_evt, void * ...
        p_context)
359 {
360 ret_code_t            err_code;
361 ble_gap_evt_t const * p_gap_evt = &p_ble_evt->evt.gap_evt;
362 
363 switch (p_ble_evt->header.evt_id)
364 {
365     case BLE_GAP_EVT_CONNECTED:
366         err_code = ble_nus_c_handles_assign(&m_ble_nus_c, ...
                p_ble_evt->evt.gap_evt.conn_handle, NULL);
367         APP_ERROR_CHECK(err_code);
368 
369         err_code = bsp_indication_set(BSP_INDICATE_CONNECTED);
370         APP_ERROR_CHECK(err_code);
371 
372         // start discovery of services. The NUS Client waits ...
                for a discovery result
373         err_code = ble_db_discovery_start(&m_db_disc, ...
                p_ble_evt->evt.gap_evt.conn_handle);
374         APP_ERROR_CHECK(err_code);
375         break;
376 
377     case BLE_GAP_EVT_DISCONNECTED:
378 
379         printf("Disconnected. conn_handle: 0x%x, reason: 0x%x",
380                     p_gap_evt->conn_handle,
381                     p_gap_evt->params.disconnected.reason);
382         break;
```

```
383
384     case BLE_GAP_EVT_TIMEOUT:
385         if (p_gap_evt->params.timeout.src == ...
                BLE_GAP_TIMEOUT_SRC_CONN)
386         {
387             printf("Connection Request timed out.");
388         }
389         break;
390
391     case BLE_GAP_EVT_SEC_PARAMS_REQUEST:
392         // Pairing not supported.
393         err_code = ...
                sd_ble_gap_sec_params_reply(p_ble_evt->evt.gap_evt.
394         conn_handle,
395         BLE_GAP_SEC_STATUS_PAIRING_NOT_SUPP, NULL, NULL);
396         APP_ERROR_CHECK(err_code);
397         break;
398
399     case BLE_GAP_EVT_CONN_PARAM_UPDATE_REQUEST:
400         // Accepting parameters requested by peer.
401         err_code = ...
                sd_ble_gap_conn_param_update(p_gap_evt->conn_handle, ...
                &p_gap_evt->
402         params.conn_param_update_request.conn_params);
403         APP_ERROR_CHECK(err_code);
404         break;
405
406     case BLE_GAP_EVT_PHY_UPDATE_REQUEST:
407     {
408         printf("PHY update request.");
409         ble_gap_phys_t const phys =
410         {
411             .rx_phys = BLE_GAP_PHY_AUTO,
412             .tx_phys = BLE_GAP_PHY_AUTO,
413         };
414         err_code = sd_ble_gap_phy_update(p_ble_evt->evt.gap_evt.
415         conn_handle, &phys);
416         APP_ERROR_CHECK(err_code);
417     } break;
418
419     case BLE_GATTC_EVT_TIMEOUT:
420         // Disconnect on GATT Client timeout event.
421         printf("GATT Client Timeout.");
422         err_code = sd_ble_gap_disconnect(p_ble_evt->evt.gattc_evt.
423         conn_handle,
424         BLE_HCI_REMOTE_USER_TERMINATED_CONNECTION);
425         APP_ERROR_CHECK(err_code);
426         break;
427
428     case BLE_GATTS_EVT_TIMEOUT:
```

```
429             // Disconnect on GATT Server timeout event.
430             printf("GATT Server Timeout.");
431             err_code = sd_ble_gap_disconnect(p_ble_evt->evt.gatts_evt.
432             conn_handle,
433                         BLE_HCI_REMOTE_USER_TERMINATED_CONNECTION);
434             APP_ERROR_CHECK(err_code);
435             break;
436 
437         default:
438             break;
439         }
440 }
441 
442 
443 //Function for initializing the BLE stack.
444 //Initializes the SoftDevice and the BLE event interrupt.
445 
446 static void ble_stack_init(void)
447 {
448     ret_code_t err_code;
449 
450     err_code = nrf_sdh_enable_request();
451     APP_ERROR_CHECK(err_code);
452 
453     // Configure the BLE stack using the default settings.
454     // Fetch the start address of the application RAM.
455     uint32_t ram_start = 0;
456     err_code = ...
            nrf_sdh_ble_default_cfg_set(APP_BLE_CONN_CFG_TAG, ...
            &ram_start);
457     APP_ERROR_CHECK(err_code);
458 
459     // Enable BLE stack.
460     err_code = nrf_sdh_ble_enable(&ram_start);
461     APP_ERROR_CHECK(err_code);
462 
463     // Register a handler for BLE events.
464     NRF_SDH_BLE_OBSERVER(m_ble_observer, APP_BLE_OBSERVER_PRIO, ...
            ble_evt_handler, NULL);
465 }
466 
467 
468 //Function for handling events from the GATT library.
469 void gatt_evt_handler(nrf_ble_gatt_t * p_gatt, ...
        nrf_ble_gatt_evt_t const * p_evt)
470 {
471     if (p_evt->evt_id == NRF_BLE_GATT_EVT_ATT_MTU_UPDATED)
472     {
473         printf("ATT MTU exchange completed.");
474 
```

```
475             m_ble_nus_max_data_len = ...
                    p_evt->params.att_mtu_effective - OPCODE_LENGTH - ...
                    HANDLE_LENGTH;
476             printf("Ble NUS max data length set to 0x%X(%d)", ...
                    m_ble_nus_max_data_len, m_ble_nus_max_data_len);
477         }
478     }
479
480
481     //Function for initializing the GATT library.
482     void gatt_init(void)
483     {
484         ret_code_t err_code;
485
486         err_code = nrf_ble_gatt_init(&m_gatt, gatt_evt_handler);
487         APP_ERROR_CHECK(err_code);
488
489         err_code = nrf_ble_gatt_att_mtu_central_set(&m_gatt, ...
                NRF_SDH_BLE_GATT_MAX_MTU_SIZE);
490         APP_ERROR_CHECK(err_code);
491     }
492
493
494
495     void bsp_event_handler(bsp_event_t event)
496     {
497         ret_code_t err_code;
498
499         switch (event)
500         {
501             case BSP_EVENT_SLEEP:
502               nrf_pwr_mgmt_shutdown(NRF_PWR_MGMT_SHUTDOWN_GOTO_SYSOFF);
503                 break;
504
505             case BSP_EVENT_DISCONNECT:
506                 err_code = ...
                        sd_ble_gap_disconnect(m_ble_nus_c.conn_handle,
507                         BLE_HCI_REMOTE_USER_TERMINATED_CONNECTION);
508                 if (err_code != NRF_ERROR_INVALID_STATE)
509                 {
510                     APP_ERROR_CHECK(err_code);
511                 }
512                 break;
513
514             default:
515                 break;
516         }
517     }
518
519     // Initializing the UART.
```

```
520 static void uart_init(void)
521 {
522     ret_code_t err_code;
523 
524     app_uart_comm_params_t const comm_params =
525     {
526         .rx_pin_no    = RX_PIN_NUMBER,
527         .tx_pin_no    = TX_PIN_NUMBER,
528         .rts_pin_no   = RTS_PIN_NUMBER,
529         .cts_pin_no   = CTS_PIN_NUMBER,
530         .flow_control = APP_UART_FLOW_CONTROL_DISABLED,
531         .use_parity   = false,
532         .baud_rate    = UART_BAUDRATE_BAUDRATE_Baud9600
533     };
534 
535     APP_UART_FIFO_INIT(&comm_params,
536                        UART_RX_BUF_SIZE,
537                        UART_TX_BUF_SIZE,
538                        uart_event_handle,
539                        APP_IRQ_PRIORITY_LOWEST,
540                        err_code);
541 
542     APP_ERROR_CHECK(err_code);
543 }
544 
545 // Initializing the Nordic UART Service (NUS) client.
546 static void nus_c_init(void)
547 {
548     ret_code_t       err_code;
549     ble_nus_c_init_t init;
550 
551     init.evt_handler = ble_nus_c_evt_handler;
552 
553     err_code = ble_nus_c_init(&m_ble_nus_c, &init);
554     APP_ERROR_CHECK(err_code);
555 }
556 
557 
558 
559 static void buttons_leds_init(void)
560 {
561     ret_code_t err_code;
562     bsp_event_t startup_event;
563 
564     err_code = bsp_init(BSP_INIT_LEDS, bsp_event_handler);
565     APP_ERROR_CHECK(err_code);
566 
567     err_code = bsp_btn_ble_init(NULL, &startup_event);
568     APP_ERROR_CHECK(err_code);
569 }
```

```
570
571
572
573 static void timer_init(void)
574 {
575     ret_code_t err_code = app_timer_init();
576     APP_ERROR_CHECK(err_code);
577 }
578
579
580 //Function for initializing the nrf log module.
581 static void log_init(void)
582 {
583     ret_code_t err_code = NRF_LOG_INIT(NULL);
584     APP_ERROR_CHECK(err_code);
585
586     NRF_LOG_DEFAULT_BACKENDS_INIT();
587 }
588
589
590 //Function for initializing power management.
591
592 static void power_management_init(void)
593 {
594     ret_code_t err_code;
595     err_code = nrf_pwr_mgmt_init();
596     APP_ERROR_CHECK(err_code);
597 }
598
599
600 // Function for initializing the database discovery module.
601 static void db_discovery_init(void)
602 {
603     ret_code_t err_code = ble_db_discovery_init(db_disc_handler);
604     APP_ERROR_CHECK(err_code);
605 }
606
607
608 //Function for handling the idle state (main loop). Then sleeps ...
       until the next event occurs.
609
610 static void idle_state_handle(void)
611 {
612     if (NRF_LOG_PROCESS() == false)
613     {
614         nrf_pwr_mgmt_run();
615     }
616 }
617
618
```

```
619 int main(void)
620 {
621     // Initialize.
622     log_init();
623     timer_init();
624     uart_init();
625     buttons_leds_init();
626     db_discovery_init();
627     power_management_init();
628     ble_stack_init();
629     gatt_init();
630     nus_c_init();
631     scan_init();
632 
633     // Start execution.
634 
635     printf("BLE UART central example started.\r\n");
636     printf("BLE UART central example started.");
637     scan_start();
638 
639     // Enter main loop.
640     for (;;)
641     {
642         idle_state_handle();
643     }
644 }
```

Appendix C

User interface code

```
1  using System;
2  using System.Collections.Generic;
3  using System.ComponentModel;
4  using System.Data;
5  using System.Drawing;
6  using System.Linq;
7  using System.Text;
8  using System.Threading.Tasks;
9  using System.Windows.Forms;
10 using System.IO.Ports;
11 using System.Windows.Forms.DataVisualization.Charting;
12 using System.Diagnostics;
13
14
15 namespace WindowsFormsApplication1
16 {
17     public partial class Form1 : Form
18     {
19         private SerialPort myport;
20         private string in_data;
21         private string Data;
22         private int[] Xinput ;
23         private int flag = 0;
24
25         public Form1()
26         {
27
28             InitializeComponent();
29             this.stop.Enabled = false;
30             this.Start.Enabled = true;
31         }
32
33         private void comboBox1_SelectedIndexChanged(object ...
               sender, EventArgs e)
```

```
34          {
35              COM.Text = COM.SelectedText;
36          }
37
38          private void Baud_rate_SelectedIndexChanged(object ...
                sender, EventArgs e)
39          {
40              Baud_rate.Text = Baud_rate.SelectedText;
41          }
42
43          void myport_DataReceived(object sender, ...
                SerialDataReceivedEventArgs e)
44          {
45
46
47              in_data = myport.ReadLine();
48              Data = in_data;
49                          if (in_data[0] == ',')
50              {
51                  Data = in_data.Remove(0, 1);
52              }
53              else if (in_data[in_data.Length-1] == ',')
54              {
55                  Data = in_data.Remove(in_data.Length-1, 1);
56              }
57              else
58              {
59                  Data = in_data;
60              }
61              try {
62              Xinput = Array.ConvertAll(Data.Split(','), int.Parse);
63              }
64              catch (FormatException)
65              {
66              }
67              flag = 2;
68              this.Invoke(new EventHandler(acquisition_event));
69              this.Invoke(new EventHandler(Form1_Load));
70              flag = 1;
71
72          }
73          private void acquisition_event(object sender, EventArgs e)
74          {
75              textBox2.Text += in_data + "TX:\n";
76
77              switch (flag) {
78
79                  case 1:
80                  textBox1.BackColor = Color.LightYellow;
81                  textBox1.Text = "Connection";
```

```
82                      break;
83                  case 2:
84                      textBox1.BackColor = Color.LightGreen;
85                      textBox1.Text = "Transmission";
86
87                      break;
88                  case 3:
89                      textBox1.BackColor = Color.Red;
90                      textBox1.Text = "Stop";
91                      break;
92                  default:
93                      textBox1.BackColor = Color.Empty;
94                      textBox1.Text = "No Operation";
95                      break;
96              }
97          }
98
99          private void Start_Click(object sender, EventArgs e)
100         {
101
102             myport = new SerialPort();
103             if (!myport.IsOpen)
104             {
105                 myport.PortName = COM.Text;
106                 myport.DataBits = 8;
107                 myport.StopBits = StopBits.One;
108                 try
109                 {
110                     myport.BaudRate = ...
                            Convert.ToInt32(Baud_rate.Text);
111                 } catch(Exception ex)
112                 {
113                     MessageBox.Show(ex.Message, "SELECT CORRECT ...
                            BAUD RATE");
114                 }
115                 myport.DataReceived += myport_DataReceived;
116                 try
117                 {
118                     myport.DtrEnable = true;
119                     myport.Handshake = Handshake.None;
120                     myport.ReceivedBytesThreshold = 1;
121                     myport.Open();
122                     flag = 1;
123                     this.Invoke(new ...
                            EventHandler(acquisition_event));
124                     this.stop.Enabled = true;
125                     this.Start.Enabled = false;
126                 }
127                 catch (Exception ex)
128                 {
```

```
129
130                     MessageBox.Show(ex.Message, "Port not ...
                            available");
131
132                 }
133
134             }
135
136         }
137         private double current(int i)
138         {
139             double R1 = 6800;
140             double R2 = 47000;
141             double C1 = 2;
142             double VDD = 3.3;
143             double Vref = 1;
144             double Vlow = 0.4;
145             double i0 = 0;
146             double f1;
147             try
148             { //0.0093   -3.1877  185.7801
149
150                 f1 = (double)0.0093 * Xinput[i] * Xinput[i] + ...
                        Xinput[i] * -3.1877 + 185.7801;// ...
                        (double)Xinput[i] * 0.909756526727153 - ...
                        51.122722625016756; //(1/R1 ...
                        *(-R2*C1*0.5*VDD/Xinput[i]*1000000 + ...
                        Vref-Vlow)+i0);
151             }
152             catch (NullReferenceException)
153             {
154                 f1 = 0;
155             }
156                 return f1;
157         }
158
159
160         private void Form1_Load(object sender, EventArgs e)
161         {
162             chart1.Series.Clear();
163             var series1 = new ...
                    System.Windows.Forms.DataVisualization.Charting.Series
164             {
165                 Name = "Data",
166                 Color = System.Drawing.Color.Green,
167                 IsVisibleInLegend = false,
168                 IsXValueIndexed = true,
169                 ChartType = SeriesChartType.Line
170
171             };
```

```
172             chart1.ChartAreas[0].AxisX.Title = "Current uA";
173             chart1.ChartAreas[0].AxisY.Title = "Kevent/s";
174             this.chart1.Series.Add(series1);
175 
176             for (int i = 0; i <120 ; i++)
177             {
178                 try {
179                 series1.Points.AddXY((double)current(i), ...
                        (double)1000/Xinput[i]);
180                 }
181                 catch(NullReferenceException) { }
182             }
183             chart1.Invalidate();
184         }
185 
186 
187 
188         private void stop_Click(object sender, EventArgs e)
189         {
190 
191             myport.Close();
192             flag = 3;
193             this.Invoke(new EventHandler(acquisition_event));
194             this.stop.Enabled = false;
195             this.Start.Enabled = true;
196         }
197 
198         private void Clear_Click(object sender, EventArgs e)
199         {
200             textBox2.Clear();
201         }
202 
203 
204 
205 private void button2_Click(object sender, EventArgs e)
206         {
207             string pathfile = textBox3.Text;
208             string filename = "Anaesthetic_dev_data.txt";
209             string imagename = "Anaesthetic_dev_data.png";
210 
211             System.IO.File.WriteAllText(pathfile + filename, ...
                    textBox2.Text);
212             chart1.SaveImage(pathfile + imagename, ...
                    ChartImageFormat.Png);
213         }
214 
215         private void button1_Click(object sender, EventArgs e)
216         {
217 
218 
```

```
219                 System.Diagnostics.Process.Start(@"C:\Program ...
                        Files\MATLAB\R2018b\bin\matlab.exe");
220
221
222             }
223         }
224     }
```

Bibliography

[1] (). Definition of Anesthesia, [Online]. Available: https://en.wikipedia.org/wiki/Anesthesia.

[2] L. A. Tafur-Betancourt, «El mundo oculto de las interacciones farmacológicas en anestesia», *Revista Colombiana de Anestesiologìa*, vol. 45, no. 3, pp. 216–223, 2017.

[3] (). The American Society of Health-System Pharmacists. Archived from the original on 9 October 2016. Retrieved 21 January 2017., [Online]. Available: https://www.drugs.com/monograph/propofol.html.

[4] (). The American Society of Health-System Pharmacists.Archived from the original on 2015-09-05. Retrieved Aug 1, 2015., [Online]. Available: https://www.drugs.com/mtm/midazolam.html.

[5] (). Acetaminophen, [Online]. Available: https://www.ncbi.nlm.nih.gov/books/NBK482369/.

[6] (). Opioid Anesthesia, [Online]. Available: https://www.ncbi.nlm.nih.gov/books/NBK532956/.

[7] (). Newman, Tim. What to know about general anesthesia. Medical News Today. MediLexicon, Intl., 5 Jan. 2018. Web. 19 Sep. 2019., [Online]. Available: https://www.medicalnewstoday.com/articles/265592.php.

[8] J. Cranshaw, K. Gupta, and T. Cook, «Litigation related to drug errors in anaesthesia: an analysis of claims against the NHS in England 1995–2007», *Anaesthesia*, vol. 64, no. 12, pp. 1317–1323, 2009.

[9] S. Aiassa, F. Stradolini, A. Tuoheti, S. Carrara, and D. Demarchi, «Quasi-Digital Biosensor-Interface for a Portable Pen to Monitor Anaesthetics Delivery», 2019.

[10] F. Stradolini, T. Elboshra, A. Biscontini, G. De Micheli, and S. Carrara, «Simultaneous monitoring of anesthetics and therapeutic compounds with a portable multichannel potentiostat», in *2016 IEEE International Symposium on Circuits and Systems (ISCAS)*, IEEE, 2016, pp. 834–837.

[11] F. Stradolini, A. Tuoheti, P. M. Ros, D. Demarchi, and S. Carrara, «Raspberry pi based system for portable and simultaneous monitoring of anesthetics and therapeutic compounds», in *2017 New Generation of CAS (NGCAS)*, IEEE, 2017, pp. 101–104.

[12] A. J. Bard, L. R. Faulkner, J. Leddy, and C. G. Zoski, «Potential Sweep Method», in *Electrochemical methods: fundamentals and applications*, 2nd ed., New York: Wiley, 1980, pp. 226–260.

[13] S. Aiassa, S. Carrara, and D. Demarchi, «Optimized Sampling Rate for Voltammetry-Based Electrochemical Sensing in Wearable and IoT Applications», *IEEE Sensors Letters*, vol. 3, no. 6, pp. 1–4, Jun. 2019.

[14] (). Definition of Potentiostat, [Online]. Available: https://en.wikipedia.org/wiki/Potentiostat.

[15] S. Aiassa, P. Motto Ros, G. Masera, and M. Martina, «A low power architecture for AER event-processing microcontroller», in *2017 IEEE Biomedical Circuits and Systems Conference (BioCAS)*, Oct. 2017, pp. 1–4.

[16] (). Definition of Altium Designer, [Online]. Available: https://en.wikipedia.org/wiki/Altium_Designer.

[17] LinearTechnology, «LTC6085 Datasheet»,

[18] AnalogDevices, «ADA4807 Datasheet»,

[19] (). Definition of Buck–boost converter, [Online]. Available: https://en.wikipedia.org/wiki/Buck-boost_converter.

[20] TexasInstrument, «TPS6303x High Efficiency Single Inductor Buck-Boost Converter With 1-A Switches Datasheet»,

[21] (). Documentation for Altium Designer, [Online]. Available: https://www.altium.com/documentation/.

[22] (). SEGGER Embedded Studio for ARM, [Online]. Available: https://studio.segger.com/index.htm?https://studio.segger.com/home.htm.

[23] Nordic, «nRF52840 product specification datasheet»,

[24] P. McDermott-Wells, «What is Bluetooth?», *IEEE Potentials*, vol. 23, no. 5, pp. 33–35, Dec. 2005. DOI: 10.1109/MP.2005.1368913.

[25] (). Bluetooth Low Energy, [Online]. Available: https://www.bluetooth.com/.

[26] (). Bluetooth Low Energy2, [Online]. Available: https://www.oreilly.com/library/view/getting-started-with/9781491900550/ch04.html.

[27] (). Serial Wire Debug, [Online]. Available: https://wiki.segger.com/SWD.

[28] S. Aiassa, F. Grassi, R. Terracciano, S. Carrara, and D. Demarchi, «Live Demonstration: Quasi-Digital Portable Pen to Monitor Anaesthetics Delivery», in *2019 IEEE Biomedical Circuits and Systems Conference (BioCAS)*, 2019, pp. 1–4.

[29] (). Screen-printed electrodes, [Online]. Available: http://www.dropsens.com/en/screen_printed_electrodes_pag.html.