POLITECNICO DI TORINO

Collegio di Ingegneria Elettronica, delle Telecomunicazioni e Fisica (ETF)

Corso di Laurea Magistrale in Ingegneria Elettronica
(Electronic Engineering)

Tesi di Laurea Magistrale

Smart wearable wrist ECG with BLE interface

Relatore

firma del relatore
prof. Pasero Eros Gian Alessandro (DET)

Candidato

firma del candidato
Puntillo Corrado

Marzo 2019

Acknowledgments

ABSTRACT

Nowadays the wearable devices market is robustly booming, especially in a society where an ever-increasing importance is given to the healthcare. This thesis project is aimed to fill a gap in the current ECG state of art, by implementing a smart wrist device to detect the ECG signal and send it to any other mobile device via Bluetooth Low Energy (BLE) interface. The main challenge is to move almost all the signal processing to the digital domain, in order to reduce the hardware and energy consumption and make the system smaller and easier to manage. This is done by redesign the analog front-end of the circuit and selecting a small and compact microcontroller with an embedded Bluetooth module. The system is composed by a single board circuit, supplied by a USB rechargeable battery, two electrodes (to detect the heart voltage pair from the wrists) and a dedicated application, able to display the received data. A great emphasis is given to the quality of the signal, with the aim of denoising it, by removing all the artefacts, and using it to supervise the heart condition and detect some cardiac diseases.

Keywords:
electrocardiography, *healthcare*, *wearable*, *wristband*, *mobile application*, *Bluetooth low energy*

What is an ECG sensor?

An ECG sensor is a system targeted to measure the contractions of the heart muscle tissue during the heartbeat, by means of electrodes over the skin. ECG stands for "electrocardiogram", i.e. a record of the activity of the heart from an electrical point of view. The result of this measurement is a waveform which describes the cardiac cycle and is used by cardiologists to check the state of health and diagnose cardiac diseases.
This waveform is nothing more than the sum of the waves representing the single movements of the heart during the depolarizing/repolarizing process, intended to pump the blood around the body (Fig. 1).
Different ECG versions can be plotted, depending on where the electrical activity is measured. Each version (so called *lead*) can be thought as a vision of the same phenomenon from a different angle, each one capable of adding some new information about the heart condition.

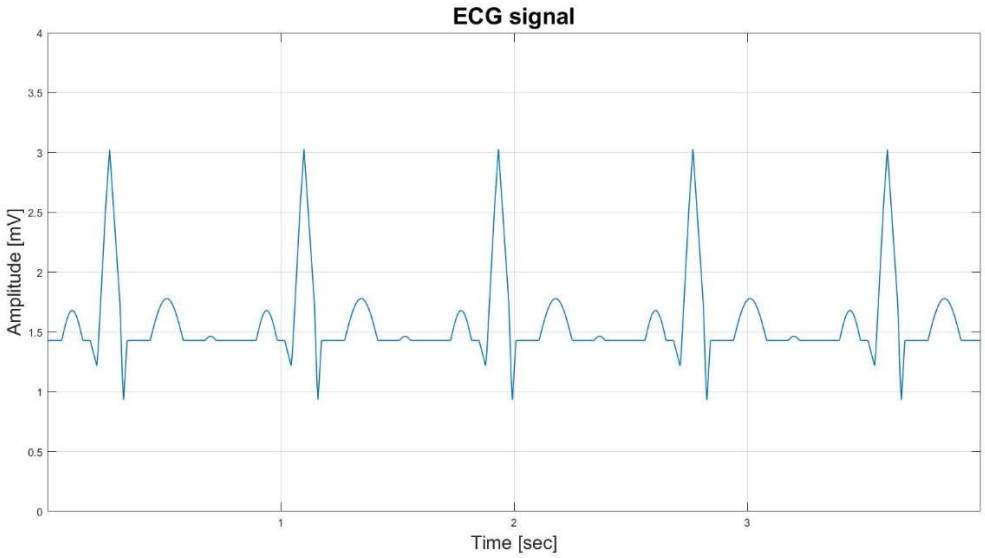

Figure 1: Normal ECG.

A professional ECG can give a maximum of 12 leads, by applying 4 electrodes on the patient's limbs and 6 electrodes on the chest. Anyway, most of the information necessary to monitor a patient and perform an initial check of his health can be obtained just reading the first lead, that is the one derived from the right and left arm electrodes.
Many devices have been developed for this purpose: desktop or wearable equipment, which require intrusive electrodes attached to the skin or just the two thumbs to lean on them.
That's where this thesis project comes in, with the hard ambition to combine the comfort of a wristband with a good quality of the ECG signal.

Current state of the art

Today the ECG sensors market supplies a wide range of products. Some of them can offer excellent results (often quite expensive as well), some other is unsatisfactory in terms of signal quality, many others extremely uncomfortable to wear and to use. The main goal of this project is to try to overcome those shortcomings by developing a comfortable and handy product (due to a reduced size) but still able to give a high-quality ECG, compared to a diagnostic one (Fig. 2).

Figure 2: Diagnostic electrocardiography equipment.

If one happened to be in a cardiologist's office for a heart workup, he would notice what a real 12-lead ECG looks like: a large and heavy machine, capable of elaborating with precision all the electrical stimuli coming from the body. Such devices are certainly the best for a careful analysis, but their use is highly limited within the hospital rooms and, therefore, unsuitable for a constant monitoring of the heart activity. Even though there are many other mid-range products intended to aim this purpose (Fig. 3), a good quality ECG signal usually must deal with the large size of the detecting device and the difficulties in making use of it. In other words, the more comfortable the better.

Figure 3: ECG non-wearable devices.

With the increasing technological development over recent years (e.g. the IoT, Internet of Things), more and more products are taking advantage of the possibility of being connected among them, exchanging data and being remotely controlled by a mobile device, through a proper application. As a result, stand-alone devices (which are objects that embed sensors, hardware, software and a dedicated display) are becoming increasingly obsolete, subsumed into non-specific devices that bring together the functionalities of thousands of products into one

single object (Fig. 4). Here too, though, the high prices and the poor versatility make these products affordable to a chosen few.

Figure 4: Apple Watch Series 4 with ECG.

In summary, there is still a strong demand for wearable and easy to use products (Fig. 5), capable to get a high-quality signal for the constant patient monitoring. The project here submitted attempts to meet this necessity. This wrist ECG exploits the great flexibility offered by the Bluetooth connection, such that it can be used by all kinds of mobile devices and provides a clean and easy to read ECG just placing a finger on top and waiting 10 seconds.

Figure 5: General purpose plastic wristband.

Previous versions

The NeuronicaLabs, in the Politecnico of Turin, for years has been working to develop an ECG watch able to meet the existing market demands. A first version of the system (Fig. 6) was implemented in 2014, by the student Federico Caffarelli, who succeeded to achieve a properly working device and an acceptable quality ECG [1].

Figure 6: ECG Watch, first version.

This thesis project seeks to reach a new goal, namely, to improve that previous version in several aspects. Hence, attempts have been made to considerably reduce the object size (previously very bulky and uncomfortable to wear), by choosing smaller and more efficient components, a microprocessor with an integrated Bluetooth 4.2 module and a minimized analog front-end. By doing so, it was possible to get the best results with the minimum resources. As a result, the ECG signal could be closely processed in the digital domain, thus making the system more flexible and open to further changes. In addition, the device was programmed to make the most of the hardware features it is equipped with, in order to reduce the power consumption and allow a long battery life. Last, a proper mobile application was developed to allow the device to connect with any tablet or mobile phone, to make it universally usable by anyone.

Improvements

The project mainly focused on three fundamental objectives:

- To improve the ECG quality w.r.t. the first-generation device, which contained errors and inaccuracies due to a massive analog filtering. In this respect, it was necessary a complete rearrangement of the front-end (i.e. all the circuitry before the analog-to-digital conversion), to provide the ADC (analog-to-digital converter) with a raw and non-contaminated signal and hence to process it with less aggressive digital filters. A detailed analysis on the ECG frequency behaviour is fundamental to identify the useful band of and ECG and develop more performing filters.

- To reduce to the minimum the number of components, by choosing small and high-performing integrated circuits and paying special attention to the PCB design phase to make best use of the reduced size. In this way, the board can be easily inserted into a small case and, then, into a common wristband.

- To reach a better energy performance, by avoiding a real-time data transmission via Bluetooth and, on the contrary, by storing them in the microcontroller flash memory. By doing so, the entire block can be transmitted at the end of the acquisition, thus in a shorter time. For this reason, it has also become necessary the selection of a new microcontroller which, despite its small size (4 x 4 mm), contains enough memory space for managing all the acquisition and signal-processing operations (thanks to its 20 Kbytes RAM and 128 Kbytes flash memory).

Methods and materials

After a preliminary analysis, a new version of the circuit was developed by maintaining almost unchanged the structure in terms of power management. Subsequently, the required simulations have been performed through the Allegro PSpice tool and the prototypes implementation, under the supervision of the NeuronicaLabs team. The 4-layer PCB was designed with OrCad PCB Editor with a focus on all the electrical constraints, in a way to avoid any possible disturbance during the signal acquisition and elaboration.
The firmware was built from scratch (with the important support of the Giacomo Zanichelli) and debugged with the TI CC2640R2 Launchpad Development Kit. The main objective was to implement a program capable of receive data from the analog-front end, store them in the flash memory, process and transmit them to a host device via Bluetooth.
On the other hand, to achieve a better ECG quality, a new digital filtering was designed. In addition to the typical IIR filters, a mathematical approach was pursued, by using functions such as moving mean and median, to remove the high-frequency noise and the baseline drift. This signal processing functions have been at first simulated with MATLAB and then implemented in firmware.
Finally, a proper app was developed with JavaScript to communicate via Bluetooth and display the received data.
In particular, the phases of the thesis project can be summarized as follows:

- Research on most common systems for the ECG detection;
- Design and simulation of a new front-end circuit;
- Creation of a prototype in through-hole technology and testing in the laboratory;
- Full circuit design with Capture CIS;
- 4-layer PCB design and optimization with OrCad PCB Editor
- Firmware implementation for the data acquisition and storage;
- Bluetooth protocol development, service and characteristics creation;
- Signal processing algorithms implementation and MATLAB simulation;
- Incorporation of the signal processing into the firmware and testing of the results;
- App development in JavaScript;
- Data collection and analysis.

Conclusion

The results collected during all the project phases have highlighted improvements in terms of performances and signal quality. In particular, the signal processing techniques have allowed to obtain a less contaminated signal (thus more similar to the professional one) and almost completely free from high-frequency noise and artefacts caused by the filtering (e.g. the unpleasant undershoots readily visible in the previous version). As a result, it will be easier to detect any abnormalities and to perform a constant monitoring of the actual heart activity. Moreover, it has been possible to significantly reduce the board size, thereby facilitating the creation of a handy and comfortable object. Last but not the least, the stress to which the board has been subjected during normal use has outlined a good energy efficiency (still to be officially validated), allowing the battery to preserve high performances even after hundreds of cycles of use. The result achieved has been shown up to the expectations, at the same time opening the way for many other possibilities and improvements.

Table of contents

1. Introduction

1.1 Einthoven triangle and leads description

As stated above, the ECG is the most widely used instrument for monitoring the heart activity and diagnose every type of anomalies and cardiac diseases. To perform a complete evaluation of the actual patient condition it is required 360-degree view of his heart, i.e. a 12-lead ECG. Each lead, indeed, provides a different way to look at the heart, such as a unique information about the electrical heart activity.
The simplest way to derive a lead is to compute the difference in electrical potential between two points of the body, according to the so-called Einthoven's triangle (Fig. 1.1). These are defined as primary leads.

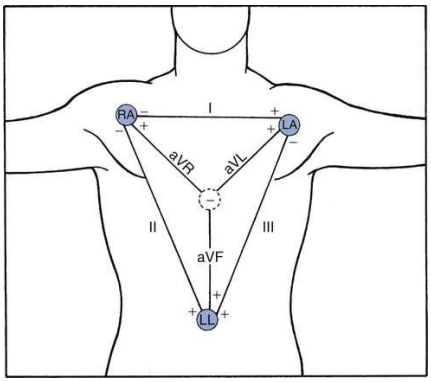

Figure 1.1: Einthoven's triangle.

Therefore, the first three leads can be easily detected through electrodes placed on the body's extremities, which are the two arms and the left leg (in addition to a fourth electrode, used to provide a voltage reference). The three differences of these voltages give the "primary leads":

$$\text{Lead I} = \text{LA} - \text{RA}$$

$$\text{Lead II} = \text{LL} - \text{RA}$$

$$\text{Lead III} = \text{LL} - \text{LA}$$

For instance, the lead I describes the electrical activity that flows from the upper right side to the left upper one. Consequently, le graphical view of this lead (voltage as a function of time) has a positive trend as long as the heart keeps moving in such a direction, while it is negative when the heart moves in the opposite direction.
Three other leads, so called "augmented leads" are mathematically computed since the first three.

$$\text{Lead aVR} = \text{RA} - \frac{LL+LA}{2}$$

$$\text{Lead aVL} = \text{LA} - \frac{LL+RA}{2}$$

$$\text{Lead aVF} = \text{LL} - \frac{RA+L}{2}$$

The last six (so called "chest leads") on the other hand, require six electrodes placed on the chest, capable to detect six more voltages. The leads are computed as a difference between these voltages and an artificial value, called WCT (Wilson Central Terminal) and defined as the arithmetic mean of the first three voltages:

$$\text{Lead V1} = \text{V1} - \text{WCT}$$

$$\text{Lead V2} = \text{V2} - \text{WCT}$$

$$\text{Lead V3} = \text{V3} - \text{WCT}$$

$$\text{Lead V4} = \text{V4} - \text{WCT}$$

$$\text{Lead V5} = \text{V5} - \text{WCT}$$

$$\text{Lead V6} = \text{V6} - \text{WCT}$$

Basically, each lead gives information about the atrial and ventricular contractions during a heartbeat, aimed to move blood around the body and transport the metabolic waste away.
Since this project is intended to develop a device able to extract the Lead I by detecting the voltages on the wrist and the opposite hand's finger, this lead will be the only one which the entire document will refer to.

1.2 Graphical waves representation

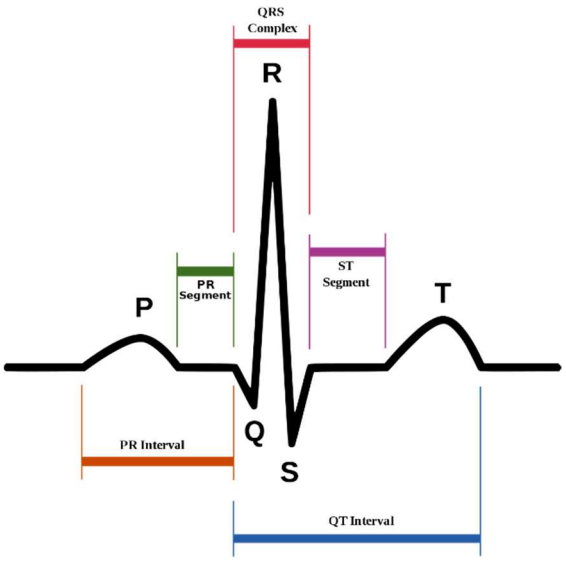

Figure 1.2: Waves, segments and intervals of a normal ECG.

In each heart cycle it is possible to observe all the single heart movement as parts of the ECG waveform (Fig. 1.2). In particular:

- the P wave is a pulse at beginning of the ECG and represents the atrial depolarization; it has a rounded shape, a low amplitude and a duration of about 80ms;

- the QRS complex, that corresponds to the ventricular depolarization, is a sequence of narrow and rapid spikes; that is because the ventricle muscles are bigger than the atrial ones; the first (Q wave) and the third (S wave) have a small negative peak, while the second (R wave) reaches very high positive values, compared with the P wave; overall, the complex lasts as long as 80ms to 100ms;

- the T wave corresponds to the beginning of the ventricular repolarization; it has a positive value and a rounded shape, often bigger than the P wave and so easier to detect;

- the U wave is often absent in the heart cycle representation; it represents the repolarization of the papillary muscles.

The detailed analysis of these waves and the time intervals which elapse among them allows an accurate diagnosis of a huge number of cardiovascular diseases. By measuring the distance between the same wave of two consecutive cycles, it is moreover possible to compute the heartrate, i.e. how many beats occur in a minute. The most widely used method for heartbeat detection is based on the Pan-Tompkins algorithm [2].

1.3 Circuitry for single-lead detection

Since the 12-lead ECG detection is out of the aim of this project, here in after only the circuit for the lead I acquisition will be proposed and analysed. The fundamental circuit (Fig. 1.3) foresees the use of an instrumentation amplifier (internally composed of three op-amps) which, as well as an impedance decoupling, is responsible for amplifying the differential mode and significantly reducing the common mode. Before passing through this instrumentation amplifier, the signal couple (coming from the electrodes) get through some protection circuitry, such as diodes, which isolate the patient from fault currents, an RC antialiasing filter and two common mode rejection capacitors. The resulting signal, after overcoming the right leg feedback (used to provide a reference voltage and to cancel common-mode interference [3]) and after a suitable conditioning, is sent to the ADC to be sampled and elaborated in the digital domain. The typical primary lead detection circuit is the following.

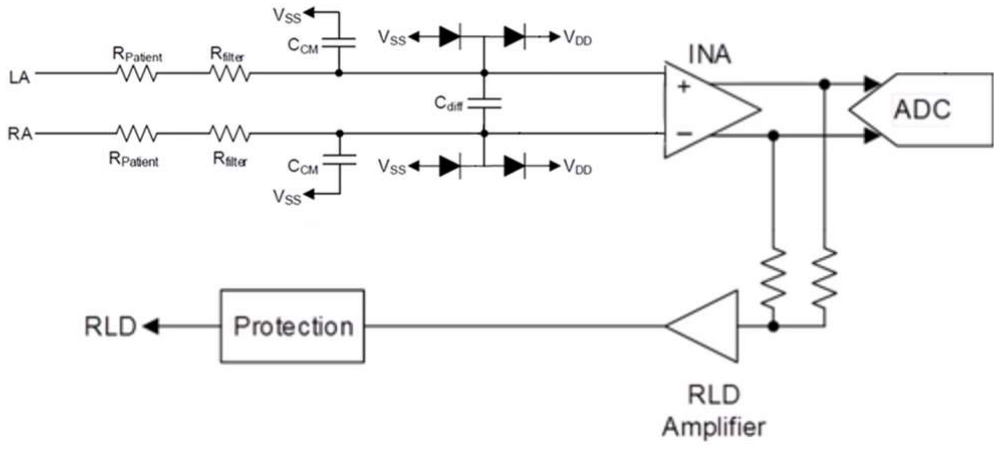

Figure 1.3: Lead I detection circuit.

Such a path is also important to clean up the signal as much as possible, before getting to the microcontroller. Furthermore, it is crucial to choose an instrumentation amplifier with a high CMRR (Common Mode Rejection Ratio), in order to prevent the signal from high frequency noise due to the network and, thus, to reduce any sort of leakage current (ensuring then a better energy performance). Nonetheless, the ECG signal preserves many artefacts along the analogic front-end (because of the human body's movements during detection, as well as the electrical disturbance inside the circuit), which must necessary be filtered in the digital domain.

1.4 Typical artefacts to be adjusted

The ECG signal, after being detected by the electrodes placed on the body, is affected by many artefacts that make it look very different from the final form (i.e. the one to be considered for evaluating the heart condition), since it is impossible to isolate the heart from the rest of the human body and the circuit from the rest of the electric fields. The most evident alterations can have two different natures, so that they can be grouped into external and endogenous noises. Here are listed the most serious artefacts to be adjusted in an ECG:

- 50 Hz noise due to the power line (Fig. 1.4); it may be easily removed through a notch filter (i.e. a band-stop filter) centred on that frequency;

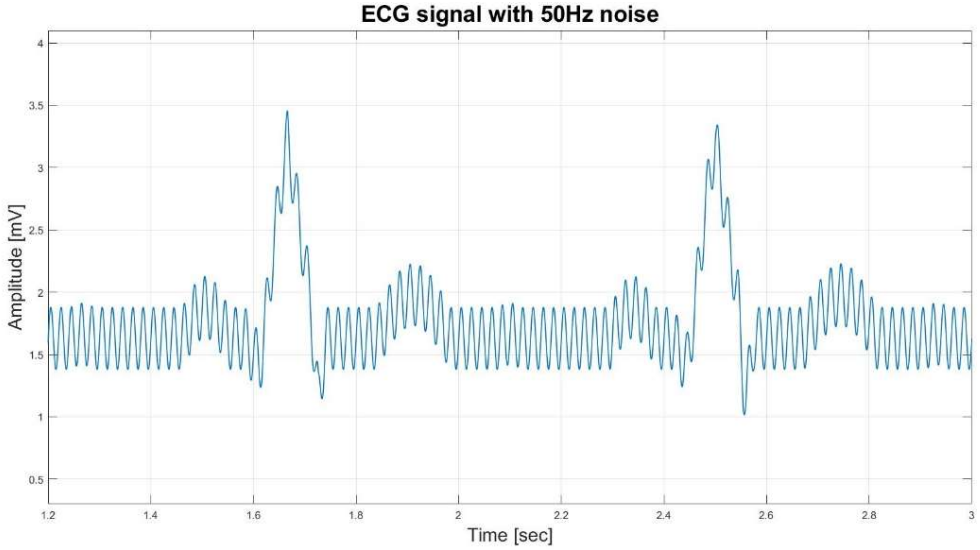

Figure 1.4: 50 Hz noise, due to the network.

- high frequency noise due to the electrical activity of the skeletal muscles, that is added to the heart activity making the signal fuzzy and distorted (Fig. 1.5); hence, it must be removed using a well-designed low-pass filter, to preserve only the most significant ECG frequencies;

- disturbance due either to an incorrect placement of the electrodes on the skin or to a rubbing with the metallic surface;

- environmental EM disorders caused by other electronic devices nearby which interfere with the circuit.

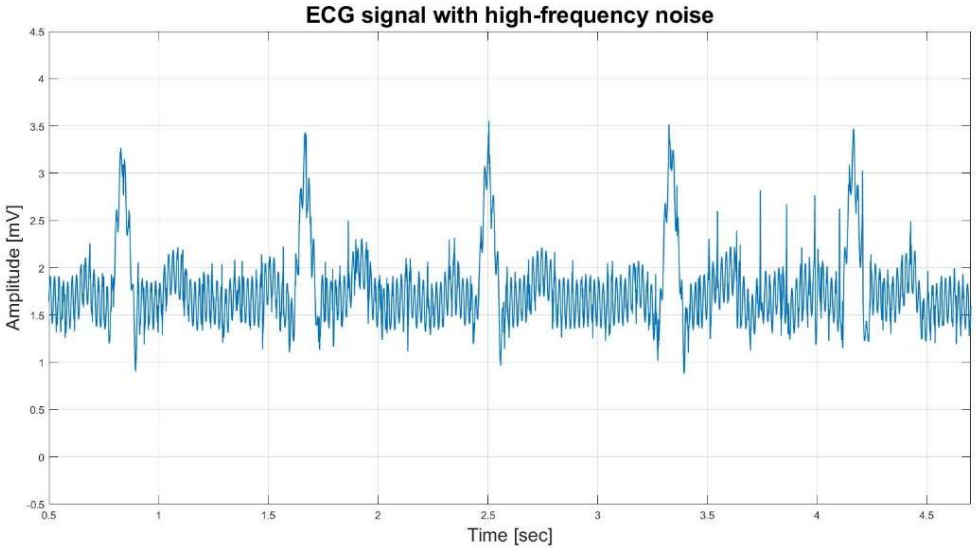

Figure 1.5: High-frequency noise.

- baseline drift, i.e. the displacement of the central line where the waveform is based on (Fig. 1.6); such a drift is far slower than the heart contractions (normally its periodicity is about 4-5 seconds) and is the result of the chest movement during respiration; it is possible to partially or completely remove this alteration by means of a high-pass filtering with a cutoff frequency of 0.2-0.3 Hz;

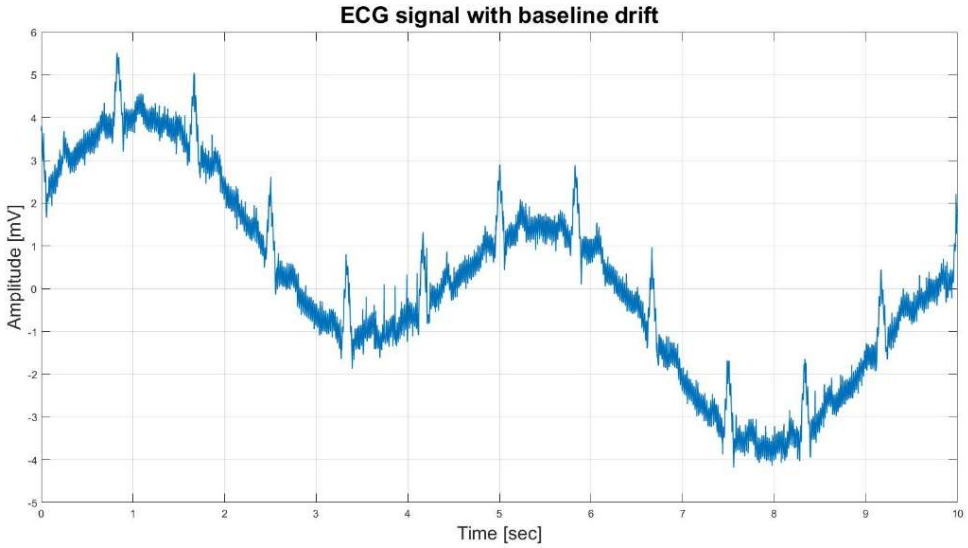

Figure 1.6: Baseline drift, due to respiration.

Considering that all these types of errors will affect the signal, it becomes crucial to figure out which is the most significant frequency range of an ECG signal, in a way to leave it unaltered while the signal processing phase takes place.
Generally, a diagnostic ECG must be able to ensure a frequency band between 0.05 Hz and 150 Hz (250 Hz for a paediatric ECG) [4]. In this way, it is possible to preserve the characteristics of every waveform (P, T and U) and not to reduce the QRS complex amplitude (which is the most sloping part in the cycle), without altering the signal at all. However, an ECG with monitoring purposes can settle for a reduced band, between 0.5 Hz and 50 Hz (where most of the power is bounded), by paying attention on the low-pass filtering, not to attenuating the spiky waves amplitude and to avoid inconvenient artefacts such as undershoots. Moreover, it is imperative to consider the sampling frequency, which shall be at least twice the maximum ECG significant frequency (this threshold is defined as Nyquist Frequency).

1.5 Aim of the project

According to this analysis, a system capable of elaborating the signal in two phases was developed. The aforementioned phases consisted into an analog front-end (composed of active filters and amplifiers) and a subsequent digital processing. In the analog session the signal is deprived of its common mode by means of a proper instrumentation amplifier (INA333, by Texas Instruments), in addition to a common mode rejection circuitry at the very path beginning. Furthermore, the signal is filtered in a slight and minimally invasive way, in order to preserve all the frequencies out of the 0.5-100 Hz band. The obtained signal, properly amplified and adapted to the ADC input dynamic range, is thus sampled and elaborated in the digital domain. Finally, the signal is refined through a non-aggressive signal processing, able to eliminate the high-frequency noise (without smoothing the highest spikes) and to straighten the baseline drift.

2. Hardware architecture

Two of the main challenges of this thesis project were to reduce the final object size and exploit more advanced components to move most of the signal processing in the digital domain. After analysing the possibilities offered by the current market and the typical architectures used to implement small wearable ECG devices, it became necessary a complete rearrangement of the analog part of the circuit, leaving instead almost unchanged the power management section, w.r.t the previous version. All the circuitry applied to achieve these results, the reasons of the component selection, their features and limits are explained in detail below, as well as the performed simulations in the process and the prototypes developed to verify the circuit correctness before the final version manufacturing.

2.1 Analog front-end

2.1.1 Proposed circuit

On the basis of the circuit for the ECG detection shown and commented above, and with the support of Jacopo Ferretti, from the NeuronicaLabs (whose initial idea was taken as a reference point throughout the design phase), an alternative version was developed, to ensure the signal to be processed by active filters and to guarantee the common mode rejection and the ESD (electrostatic discharge) protection, in addition to a proper adjustment in relation to the ADC input dynamic range. To do this, following design constrains have been considered:

- To ensure an initial analog active filtering for the signal, in order to reduce the disturbance out of the 0.5–100 Hz band;
- To select an instrumentation amplifier with a high CMRR (100 dB) and not to go above four operational amplifiers for the filtering and conditioning operations, thus being able to use a single integrated circuit which include them all;
- To shift the reference voltage from 0V to half the supply voltage, namely 1.65V;
- To guarantee a massive amplification to let the signal fully exploit the ADC dynamic range, dealing with a quite low resolution (i.e. 12 bit); since the initial ECG (the one coming from the instrumentation amplifier) has an amplitude of few millivolts, it was desirable to amplify such a signal enough to achieve a 1 volt peak-to-peak amplitude; a factor of 700 was selected for this purpose.

Bearing in mind all these constraints, the schematic in Fig. 2.1 was proposed.

A fundamental characteristic is the absence of the right leg feedback, with respect to the schematic shown above for the lead I detection. This choice became necessary to implement a wearable design. A consequence of such a choice is a higher noise in the signal from the surrounding circuit, in any case balanced by the selection of a high CMRR instrumentation amplifier.

The circuit is divided into five parts, according to the phases which the signal goes through from the time when the voltage couple is detected from the electrodes:

- ESD protection and AC decoupling;
- Instrumentation amplifier, common mode rejection and differential mode amplification;
- 0.5 Hz high-pass filter;
- 100 Hz low-pass filter;
- Conditioning and final amplification.

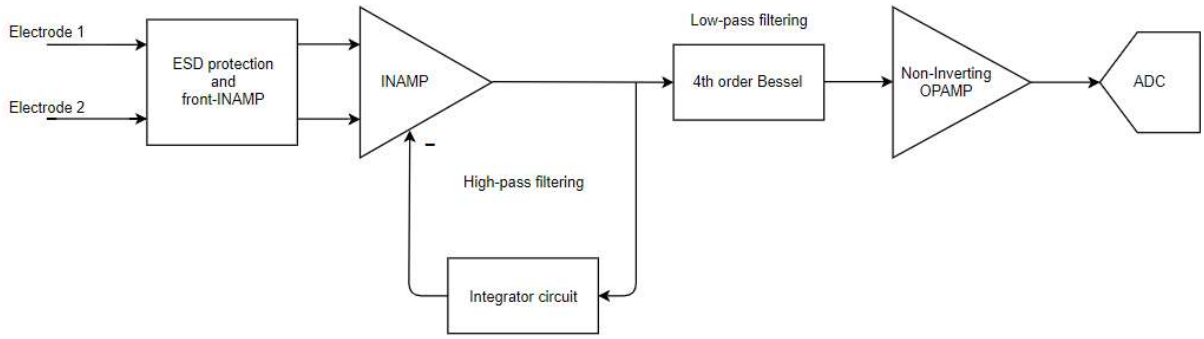

Figure 2.1: ESD protection and front-INAMP block.

The original choice of adding a band-stop filter (also said *notch* filter) centered at 50 Hz was immediately abandoned because such a filtering would have led to an extra components' addition, thus an increase in terms of area and power consumption, besides an unavoidable deterioration of the signal quality, which would have shown more conspicuous artefacts.

2.1.2 Circuit analysis

Each of the parts of the front-end is now described, reporting the computations which allowed the above-mentioned circuit decisions.

ESD protection and front-INAMP

For the electrostatic discharge protection an integrated circuit was selected, that is the DVIULC6-2M6 (provided by Texas Instruments [4]) to ensure a full protection on both the lines from high voltage peaks, paying a small price in terms of area and leakage currents (Fig. 2.2**)**. The main features of this component are listed below:

- 2-line ESD protection at 15 kV air and contact discharge;
- Fast response time
- Low leakage current, lower than 0.5 uA, for longer operation of battery powered devices;
- A very small package: 500 uM pitch fr uQFN 6 leads.

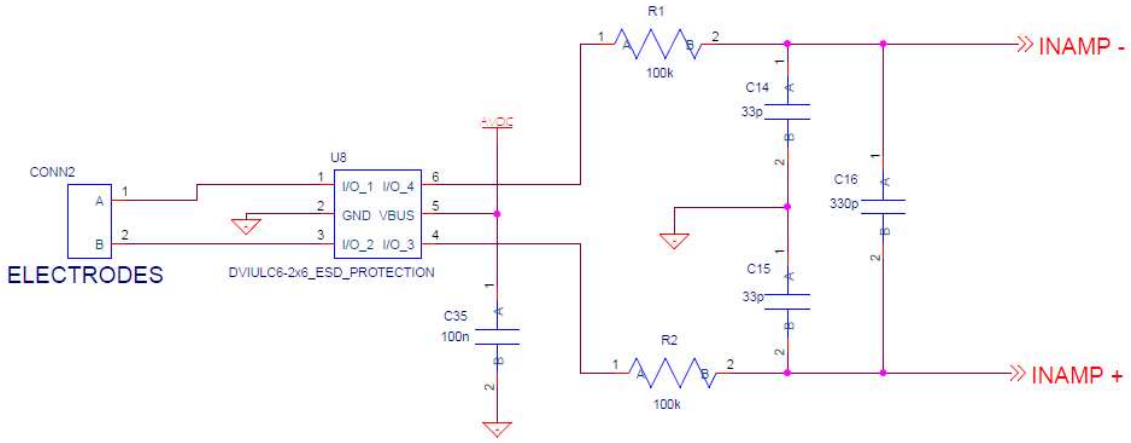

Figure 2.2: ESD protection and front-INAMP block.

After this component, two resistors were placed to minimise the current flow back to the electrodes. Such resistors, coupled to a differential capacitor placed between the two lines, are used as anti-aliasing filter, in order to eliminate the high frequencies out of the signal bandwidth.
Finally, two more capacitors were placed along the two lines to reduce the common mode propagation. Nevertheless, this is not the bulk of the common mode rejection, which otherwise would be degraded by the passive components' tolerances. A massive common mode rejection is performed afterwards, through the instrumentation amplifier.

Instrumentation amplifier

A very high CMRR instrumentation amplifier is usually required for ECG applications, i.e. able to almost completely remove the common mode component of the signal and to propagate only its differential mode, properly amplified. The integrated component *INA333* (provided by Texas instruments, datasheet [6]) was selected for this purpose: it boasts a CMRR higher than 100dB and a three operational amplifiers structure which allows to set the differential gain to any value between 1 and 1000 just setting a single resistor (Fig. 2.3).

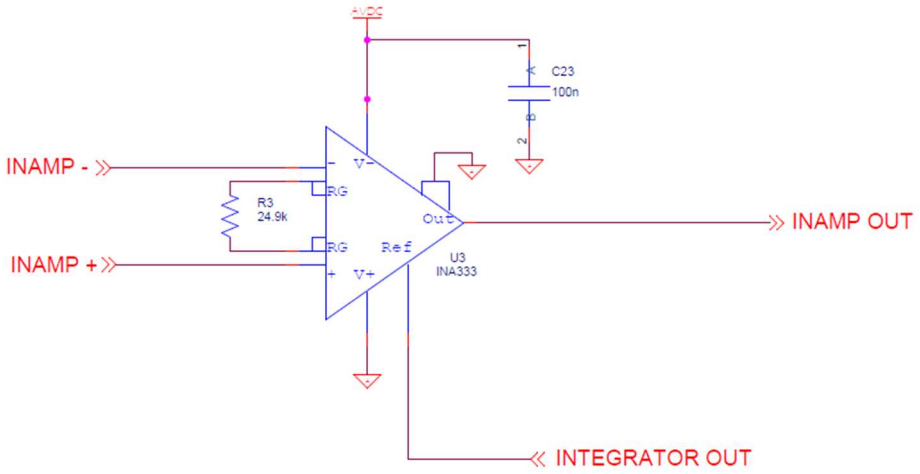

Figure 2.3: Instrumentation amplifier INA333.

Furthermore, the component can be supplied by voltages of 1.8V up to 5.5V, with a negligible leakage current and a sufficiently small package (3x3mm). Such features make it the ideal instrumentation amplifier for mobile applications which need small size batteries and aim to minimise the power consumption. The resistor R_G value was computed as shown in the Formula 2.1, by setting the gain equal to 5.

$$\boldsymbol{G = 1 + \left(\frac{100k\Omega}{R_G} \right)} \tag{2.1}$$

An initial simulation in PSpice environment confirmed the correctness of the performed calculation, showing the signal amplification w.r.t. the selected resistor value. The simulation was executed in the time domain, with a 1.65V constant voltage value and a 200mV$_{pp}$ sinewave as inputs. The simulation results are shown in the Fig. 2.4**.**

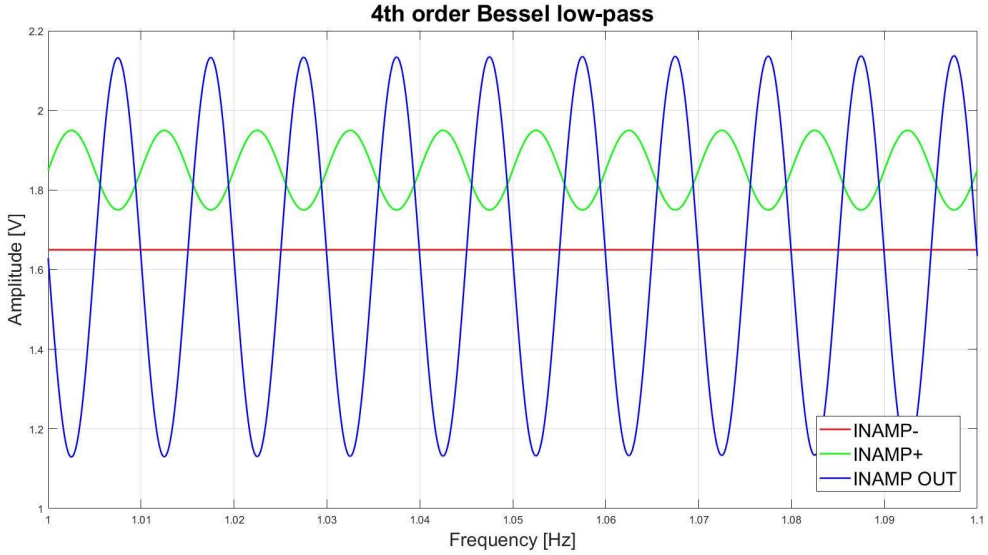

Figure 2.4: Instrumentation amplifier INA333 time response.

High-pass filtering

A first high-pass filtering was implemented by means of a first order active filter. In this way, the very low frequencies could be removed from the signal (i.e. the ones lower than 0.5Hz). Such a filtering, as hard as it is not effective enough, it helps to remove the baseline drift caused by the lung movement during respiration. A further high-pass filtering is subsequently performed in the digital domain.
In this case, a circuit implementing a 0.5Hz low-pass filter was designed (also said *integrator circuit*) and thus placed in the feedback loop, between the main line and the reference pin of the instrumentation amplifier. By doing so, the low frequencies are isolated from the signal and given back to the amplifier, in order to be subtracted from the main signal. The Formula 2.2 exposes how the filter was designed, i.e. by setting the cutoff frequency and resistor values and computing the capacity value. The filter schematic is also provided in Fig. 2.5.

$$f_c = \frac{1}{2\pi RC} \tag{2.2}$$

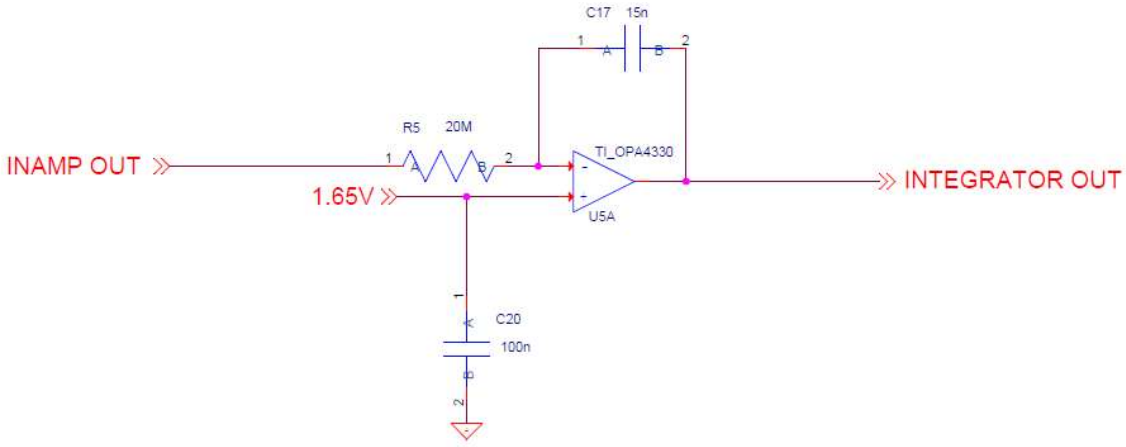

Figure 2.5: Integrator circuit, cutoff frequency at 0.5 Hz.

It can be noted that the operational amplifier, besides getting the signal to be filtered in the inverting input (to be hence subtracted from the original signal), receives an additional 1.65V voltage in the non-inventing input (to be now added to the signal). The voltage reference is thereby shifted from 0V to 1.65V, which is half of the supply voltage equal to 3.3V. This was necessary to allow the propagation of both the signal components, the positive and the negative ones, which otherwise would be completely deleted.
All the operational amplifiers of this analog front-end (i.e. the ones used for high-pass and low-pass filtering and for signal conditioning) are contained in the integrated component OPA4330, by Texas Instruments [7], selected because of its compactness and its optimal performances within battery-powered mobile devices. The component, indeed, can be supplied by voltages of 1.8V up to 5.5V and provides great versatility due to its small size (with a package of 3.5x3.5mm). In this specific case, one operational amplifier was used to implement the integrator circuit, two others for the two cascading Sallen-Key topologies and the fourth one for the final non-inverting amplifier.

Low-pass filtering

Many possible scenarios were considered for the low-pass filtering phase, with the purpose of deprive the signal, in the least possible aggressive way, of the frequencies higher than 100 Hz

(namely, the ones over the signal useful bandwidth). The obtained result is a light filtering, which doesn't add any artefacts like undershoots or ringing in the output response. This was aimed to improve the signal quality from the previous ECG watch version which, instead, filtered the signal almost completely in the analog domain, with a consequent extra defects' insertion.
Many analog filters behaviours were analysed, by exploring their pros and cons. Greater emphasis was placed on the following types of filters:

- *Butterworth*, with a maximally flat magnitude response in the passband and monotonic in the stopband;
- *Chebyshev I*, with an equiripple magnitude response in the passband and monatomic in the stopband;
- *Chebyshev II*, with a monotonic magnitude response in the passband and equiripple in the stopband;
- *Elliptic*, with an equiripple magnitude response both in the bandpass and in the stopband;
- *Bessel*, with a maximally flat magnitude response in the bandpass, a monotonic response in the stopband ad a maximally flat linear phase response.

The latter was selected because its linearity in the magnitude and phase response, without adding any artefact like ripples or undershoots. This configuration can be further modified, just changing the resistors and capacitors values of the two blocks. On the basis of those analyses and simulations, a fourth-order Bessel active filter was designed, by means of two cascading Sallen-Keys [8]. In the following all the calculations for this circuit design are exposed, besides the circuit schematic (Fig. 2.6) and the performed PSpice simulations (Fig. 2.7).

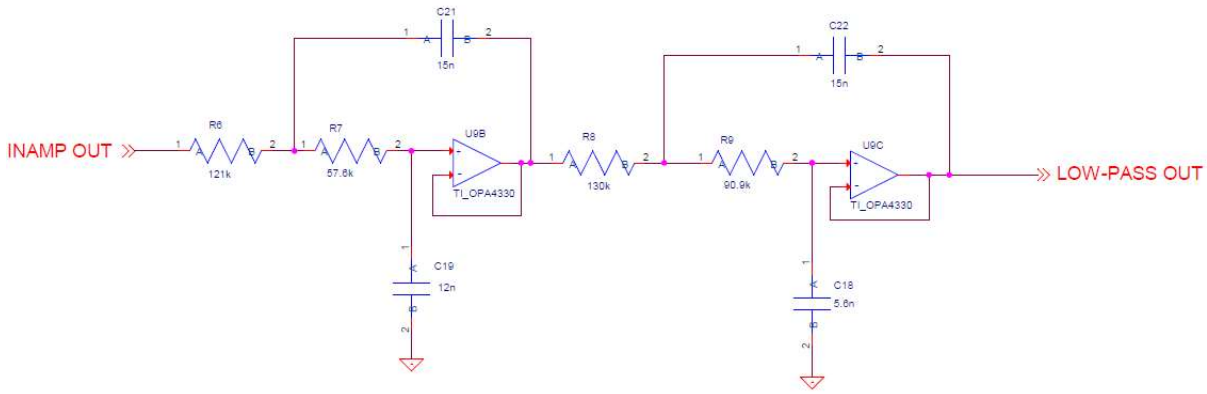

Figure 2.6: 4th order Bessel low-pass filter, cutoff frequency at 100 Hz.

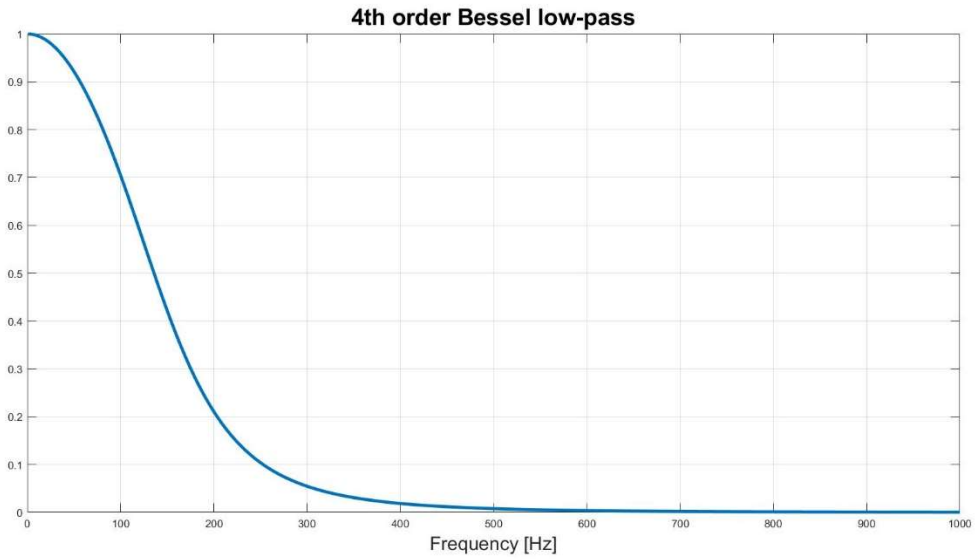

Figure 2.7: Frequency response of the 4th order Bessel low-pass filter, with a 100 Hz cutoff frequency.

Each of the two Sallen-Keys must be designed separately, by using the fourth-order Bessel coefficients (Table 2.1). The capacitor values are computed as shown in the Formula 2.3, by setting the first capacity to an accessible value, e.g. 15nF. The resistor values are instead computed by inserting the a and b coefficients in the Formula 2.4 **[9]**.

Table 2.1. 4th order Bessel coefficients.

i	a_i	b_i
1	1.3397	0.7743
2	0.4889	0.3890

$$C_2 \geq C_1 \cdot \frac{4b_i}{a_i^2} \tag{2.3}$$

$$R_{1,2} = \frac{a_i C_2 \mp \sqrt{a_i^2 C_2^2 - 4b_i C_1 C_2}}{4\pi f_c C_1 C_1} \tag{2.4}$$

Conditioning and final amplification

The last block provides the final signal amplification before it is sampled by the ADC (Fig 2.8). In order to exploit as well as possible the ADC input dynamic range (between 0V and 3.3V) a non-inverting amplifier was designed, with a gain equal to 140 (see the Formula 2.5) such as to lead the ECG signal to an amplitude of the order of one Volt. The final amplification is now $5 \cdot 141 = 705$.

$$G = 1 + \frac{R_{11}}{R_{10}} \tag{2.5}$$

The same 1.65V voltage seen above is send to the operational amplifier inverting input, to center the amplification around this voltage value and to avoid, therefore, to amplify the offset voltage together with the signal.

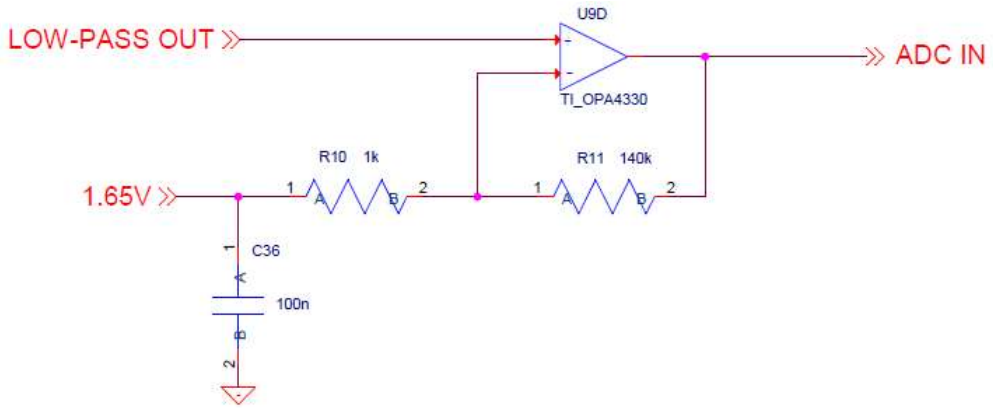

Figure 2.8: Non-inverting amplifier, with a gain of 140.

The whole circuit was subsequentially implemented in through hole technology and its proper functioning was verified in the laboratory by means of specific measurement instrumentation.

2.1.3 Prototyping and first results

A prototype of the prior analysed circuit was implemented and suitably simulated in all its parts, except for the ESD protection. The laboratory equipment, necessary for the simulation phase, included a waveform generator, an oscilloscope (Fig 2.9), a desktop multimeter and a power supply kit. Moreover, a patient simulator was used to replicate the typical ECG signal and many of the most common cardiac diseases.

Figure 2.9: Oscilloscope displaying an ECG signal affected by the 50 Hz disturbance.

Such simulations were taken as a basis to develop an effective prototype in SMT (Surface Mount Technology). A simple data acquisition data was further developed with the TI LaunchPad BLE kit, in the MATLAB environment. The obtained results, still raw and full of disturbance and all sorts of noise, allowed to trace a first ECG shape. The effective correctness of the proposed circuit was thereby verified, to use it as a starting point for an accurate digital sampling and a more powerful and detailed signal processing.

2.2 Microcontroller selection

A key issue of the whole project consisted in a proper sampling and storage of the signal from the analog front-end, the subsequent data processing and the transmission of data via Bluetooth. The selection of a suitable microcontroller was thus crucial. The limitations of the ECG watch in its previous version also lied in a bulky microcontroller, with not enough memory to perform complex operations nor an embedded Bluetooth module. Then the microcontroller CC2640R2F, provided by Texas Instruments (datasheet [ ]), was chosen as a compact and high-performing component. Among its main features, it can be seen:

- An ARM Cortex-M3 processor, with a 16-bit architecture and up to 48-MHz clock speed;
- 275kB of non-volatile memory, including a 128kB flash memory;
- 20kB of system SRAM and an extra 8kB SRAM for cache or system RAM use;
- A 12-bit ADC, with 2 channels which can be connected to each of the analog inputs;
- Three different packages, including a 3x3mm one with 10 GPIOs (General Purpose Input/Output);
- Wide supply voltage range, normally between 1.8V and 3.8V;
- UART, SPI and I2C peripherals;
- Bluetooth BLE 4.2 module which allows, together with the rest of the device, a great energy performance and a long battery lifetime.

In addition, this device gives the possibility of debugging in an integrated development environment that is Code Composer Studio, through the LaunchPad kit (Fig 2.10) provided by Texas Instruments.

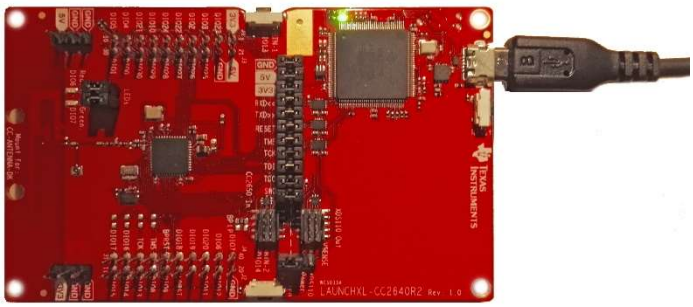

Figure 2.10: CC2640R2 LaunchPad Board.

This allowed to develop a system capable to sample a ten-second signal at a 500Hz sampling frequency (16 bits per sample), to store it in the flash memory, to process and transmit it via Bluetooth in payloads of 8 samples each. Moreover, the microcontroller also handles all the control signals necessary to let the device work correctly, which include the battery voltage gauge and the recharge status indicator, the input signal to get the push button status and the output signal to drive the two LEDs.

2.2.1 12-bit ADC

There are two different approaches to sample an ECG signal. The first is to use a low-resolution ADC (i.e. lower than 16 bits) and to significantly amplify the signal in the analog domain, still taking care of using low-noise operational amplifiers. The second, on the contrary, is to use a high-resolution ADC (e.g. 24 bits) and to amplify the signal downstream by a lower value, such as only 5 times [ ]. Both approaches lead to a similar result in terms of input dynamic range and signal quality (Fig. 2.11). Nevertheless, the second one needs less effort and shall ensure the best outcome, even requiring an enhanced microcontroller and a higher computational cost.

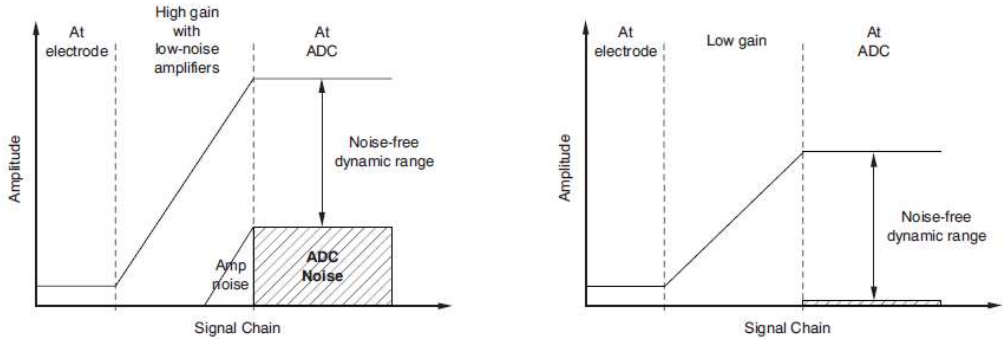

Figure 2.11: Comparison between high-resolution and low-resolution ADCs [ ].

Unfortunately, it was necessary to settle for a 12-bit ADC, with two channels connected to the eight microcontroller analog inputs. Before data acquisition, the ECG signal is amplified by a total value of 700 (5, by the instrumentation amplifier, multiplied by 140, by the non-inverting operational amplifier). After acquisition the signal quality appears still riddled with noise, although to be removed during the digital signal processing phase.

2.2.2 GPIOs and final configuration

Being the microcontroller is the core of the proposed system, it was necessary an accurate design to configure the whole input/output system. Besides the ECG signal sampling, the microcontroller (in the selected format, that is the 4x4mm package), allows the exploitation of nine additional input/output pins, some of which provide a high-drive capability (8mA versus the typical 4mA). Such GPIOs, properly connected to other components, allow information and controls to be exchanged within the device, enabling many actions like:

- To detect of the push button state (ON/OFF);
- To drive the two LEDs, through an 8-mA current;
- To transmit and receive data through JTAG connection;
- To transmit and receive data through I2C protocol, used to detect the battery voltage level;
- To enable the voltage references provider which allows to supply the analog front-end and give it a reference voltage;
- To read the *CHGn* signal, which gives information on the battery recharge status;

Moreover, the microcontroller presents other connections to the crystal oscillators and the patch antenna. The final microcontroller configuration is shown in Fig 2.12 and Fig 2.13.

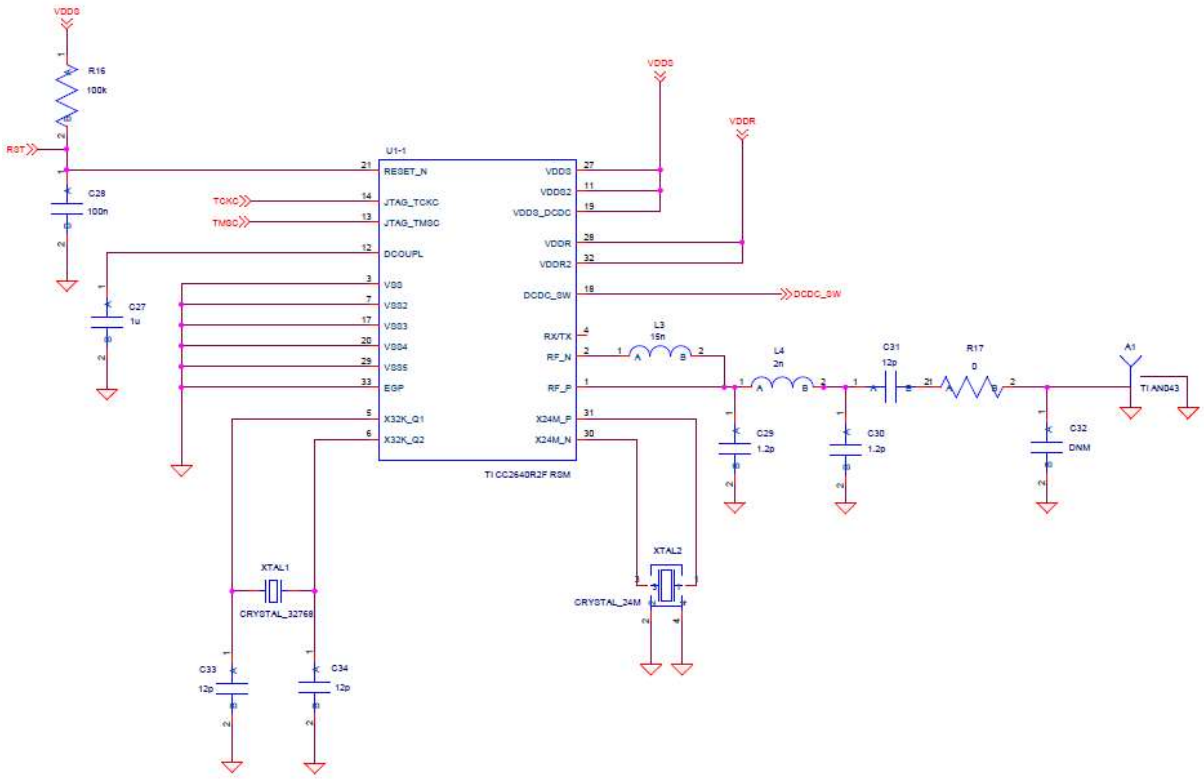

Figure 2.12: Microcontroller configuration, except for the GPIO pins.

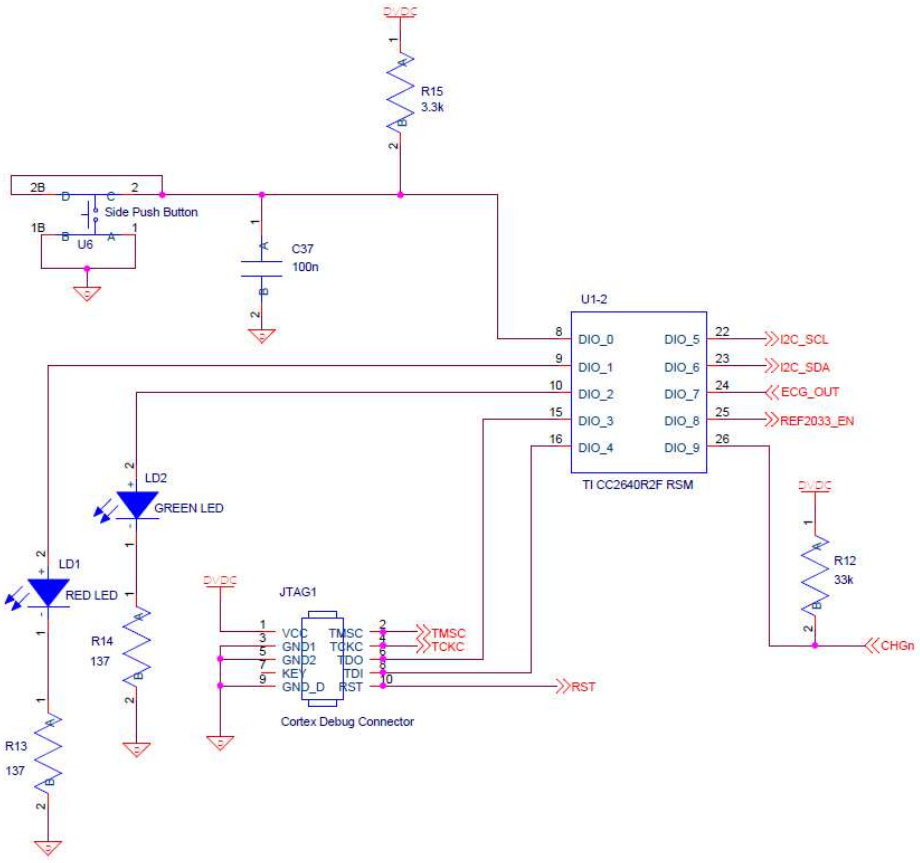

Figure 2.13: Microcontroller GPIOs configuration.

2.3 Power management

An accurate component selection, regarding the device power management, was necessary to let all the parts of the system properly work and to optimize the energy performance. A great emphasis was given to four components, i.e. those useful or even fundamental for a proper energy-efficient administration:

- a buck/boost regulating charge pump;
- a USB battery charger
- a battery gauge, to detect the battery voltage level;
- a voltage references provider

Their main characteristics and usage inside the circuit are described below.

2.3.1 Buck/boost regulating charge pump

The MAX1759 buck/boost regulator was selected (Fig. 2.14), since it has been already used in the previous version of the ECG watch. It is a 10-pin integrated circuit able to receive an input voltage from +1.65V to +5.5V and generate a 3.3V output voltage. Among its features there is a quiescent supply current lower than 50uA, besides the possibility to provide an open-drain power OK (POK) output pin, used to signal when the output voltage is in regulation [ ].

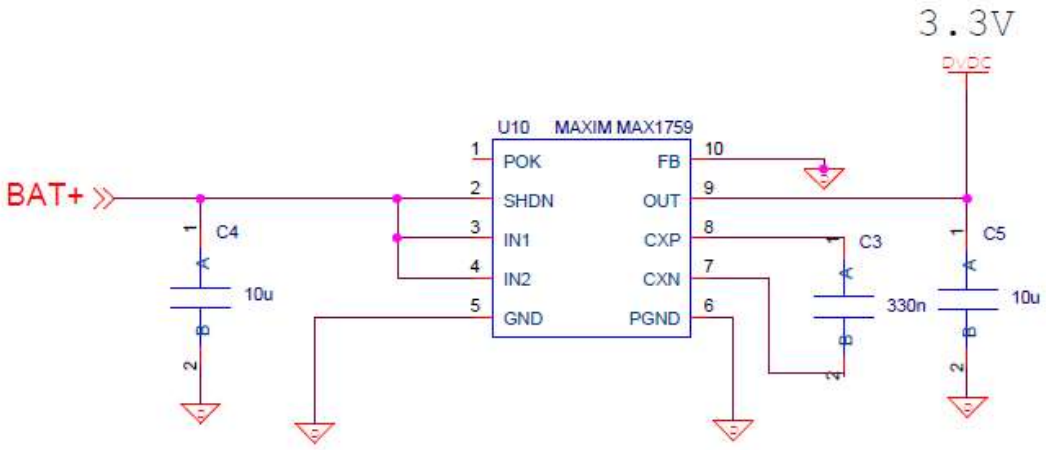

Figure 2.14: MAX1759 buck/boost regulator configuration.

2.3.2 Battery charger

The MAX1555 component (also included in the previous version) was selected to recharge the battery via USB connection (Fig.2.15). It is characterized by a 5-pin package, an input voltage up to 7V and an indicator (active-low) which signals when the input power is present [ ].

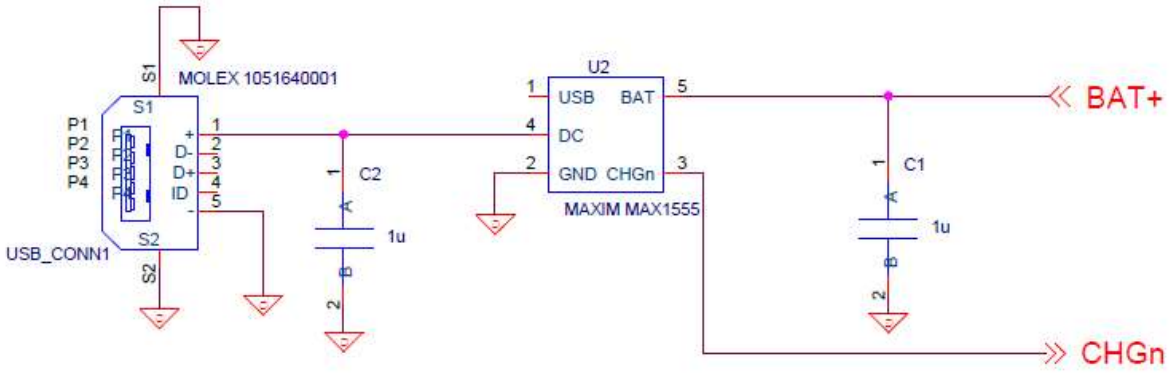

Figure 2.15: MAX1555 battery charge configuration.

2.3.3 Battery gauge

The Max17048 battery gauge was selected to measure the battery voltage level (with a precision of ±7.5mV) and to send this information to the microcontroller through I2C protocol [ ]. The component will be mounted on the board but not yet programmed via firmware. For the proper functioning of the I2C interface, it was configured as in Fig. 2.16.

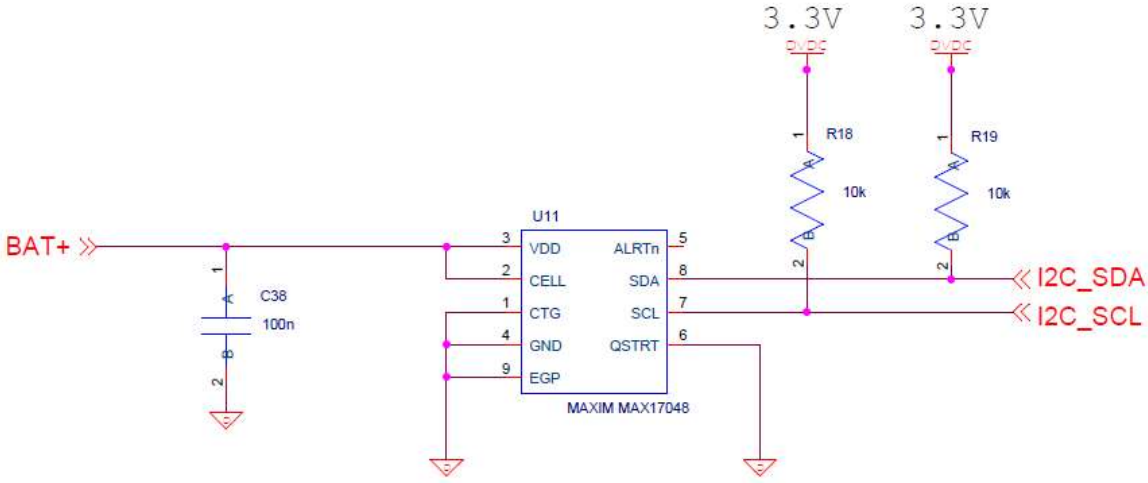

Figure 2.16: MAX17048 battery gauge configuration.

2.3.4 Dual-output voltage references provider

The REF2033 component was necessary to provide the analog front-end a proper reference voltage, halfway through the voltage supply, i.e. 1.65V [ ]. Such component is characterized by a 5-pin package, whose an input to receive the voltage supply directly from the battery, two constant outputs on the values of 3.3V and 1.65V, a ground reference and an input used to be enabled by the microcontroller (Fig. 2.17).

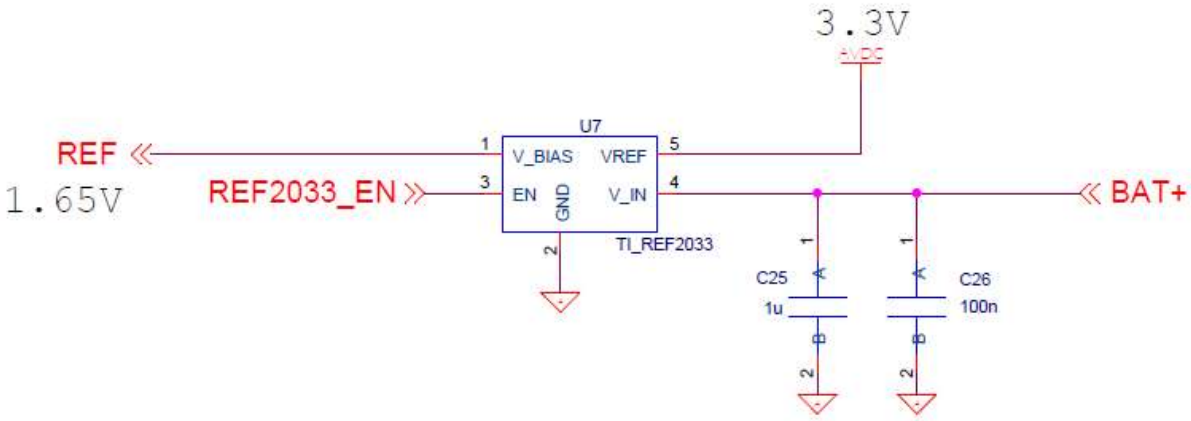

Figure 2.17: REF2033 dual-output voltage reference configuration.

3. PCB design

A fundamental part of this project consisted in designing a small-sized and versatile PCB (printed circuit board), without affecting the technical and functional characteristics of the circuit. The CAD implementation was developed from the already existing design from OrCad Capture CIS (Component Information System) environment, where all the circuit schematics were created, and the simulations were run.
Alongside the firmware developing, the hardware design was then transferred on Cadence Allegro PCB Editor (that is an EDA – electronic design automation – software tool) and using it the project files were finally generated (the so-called *Gerber* files, necessary for the manufacturing process).

3.1 Project constraints

The CAD implementation demanded a particular care and attention to detail, since the requirement of producing a small-sized board had to be combined to the need to maintain all the components necessary for its proper functioning. A special care was taken to limit as much as possible all kinds of electromagnetic interference and energy waste.
The following constraints have been used as a guide during the whole design phase:

- To use a board outline approximately sixed (3 x 2) cm;
- To divide the board into four layers, so as to use the external layers to place the components and the inner ones for the ground and voltage supply planes (Fig. 3.1);

Objects		Types >>		Thickness >>	Physical >>	
#	Name	Layer	Layer Function	Value mm	Layer ID	Material
*	*	*	*	*	*	*
		Surface				
1	TOP	Conductor	Conductor	0.03048	1	Copper
		Dielectric	Dielectric	0.2032		Fr-4
2	GND	Conductor	Conductor	0.03048	2	Copper
		Dielectric	Dielectric	0.2032		Fr-4
3	VDC	Conductor	Conductor	0.03048	3	Copper
		Dielectric	Dielectric	0.2032		Fr-4
4	BOTTOM	Conductor	Conductor	0.03048	4	Copper
		Surface				

Figure 3.1: 4-layer PCB OrCad Cross Section Editor.

- To divide crosswise the board in two well-isolated parts, in order to separate the analog section from the digital one;
- To observe the manufacturer rules (Figure) about the minimum manufacturable trace width (0.15mm), the spacing between two traces (0.15mm), the minimum drill size (0.2mm) and annular ring size for pads with vias in the middle (0.15mm);

- To guarantee a current return to all components, in a way not to generate additional disturbances inside the board; such a purpose is achieved by adding a ground plane (instead of routing a ground net) and by isolating the analog ground plane from the digital one;
- To properly place the decoupling and bypass capacitors along the traces, so that to fully exploit their high-frequency decoupling capability; their connection was always carried out with short traces and vias as closer as practicable to the pads (Fig. 3.2);

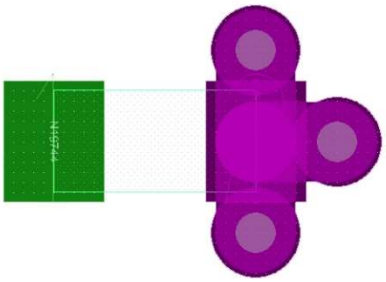

Figure 3.2 Bypass capacitor vias.

- To place the crystal oscillators as far away as possible from the traces which lead the analog signal (e.g. the ones in the analog front-end path);
- To let the antenna to be connected to the microcontroller through a short and straight path and to be free to transmit the signal in any direction.

An attempt was constantly made to keep the costs to a minimum during the component selection and the assembly phase. For this reason, the used approach was to implement the connections among the four PCB layers by means of through hole vias, instead of blind or buried vias, because of their expensive assembly costs. Such a decision, however, restricted the design flexibility and, together with other design constraints, led to a final board size of (3.2 x 2.1) cm. Nevertheless, the result is more than satisfactory, since it was possible to halve the board size, as compared to the previous version (Fig. 3.3). All the 4 layers of the final PCB are shown in the following.

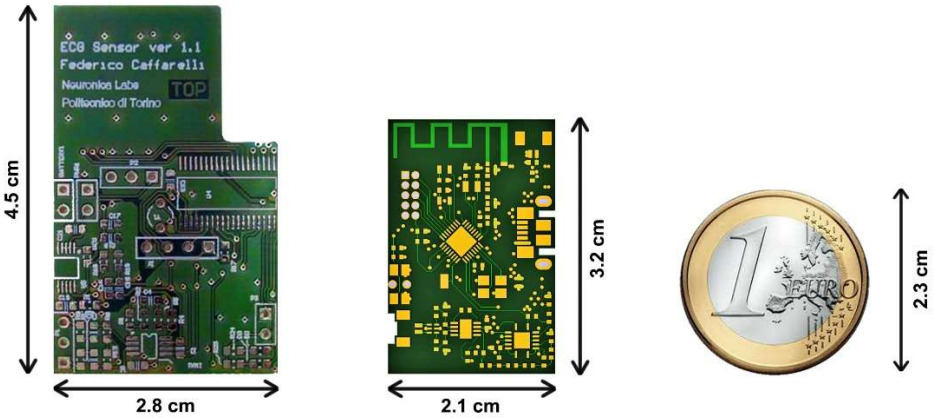

Figure 3.3: Previous PCB size, compared with the new one and a one-euro coin.

3.2 Layers overview

3.2.1 Top layer

This layer was used to place the entire analog front-end and the main components (except for the ones related to the power management, i.e. the buck-boost regulator, the battery recharge, the battery gauge and the voltage references generator). Such decision was made to ensure that all the high-frequency components can be placed as closer as possible to the ground plane, in a manner to minimize the noise propagation along the board due to the non-null ground plane impedance.
Special focus was placed on the placement of the components along the path along which the ECG signal is generated, elaborated and sent to the microcontroller. The greatest symmetry was aimed between the two lines before the instrumentation amplifier, in a way to avoid affecting the differential signal and increasing the common mode gain. This circuit section was completely separated from the rest, in order to exploit separated ground and supply voltage planes and thus to be in no way contaminated by the digital signals. Moreover, all the peripherals, i.e. the push button, the LEDs, the JTAG connector for the firmware upload and the crystal oscillators, were placed on this layer, aiming to locate the latter far away from any analog trace, to prevent them to be affected by high-frequency noises (Fig. 3.4).

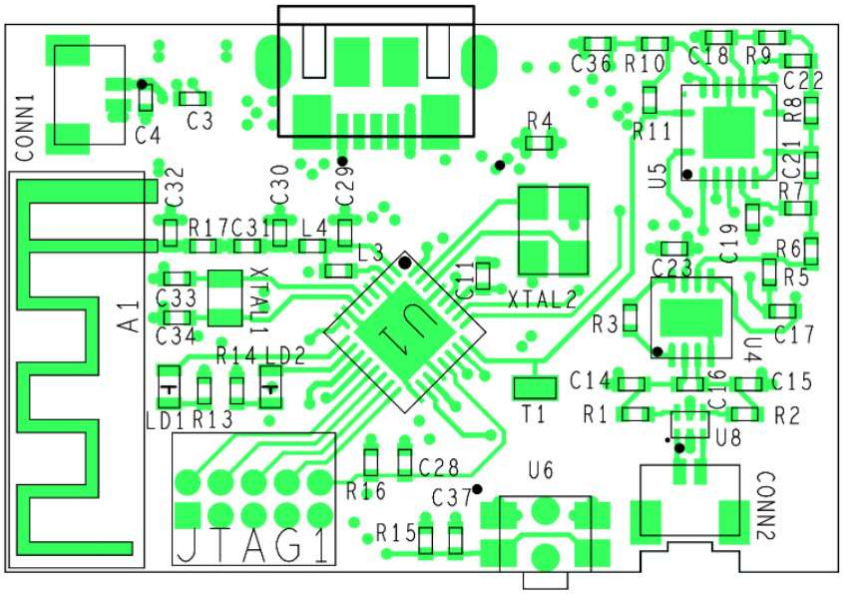

Figure 3.4: TOP layer.

3.2.2 Bottom layer

It was agreed to place on this layer all the low-frequency components, such as the ones aimed at the power management and the battery recharge (this being the farthest layer from the ground plane). Here again, the strict division between the analog and digital section was met, where possible. For reasons of space, the battery gauge was located near the antenna front-end. However, it was believed that such decision doesn't affect significantly the proper functioning thereof and, therefore, the data transmission (Fig. 3.5).

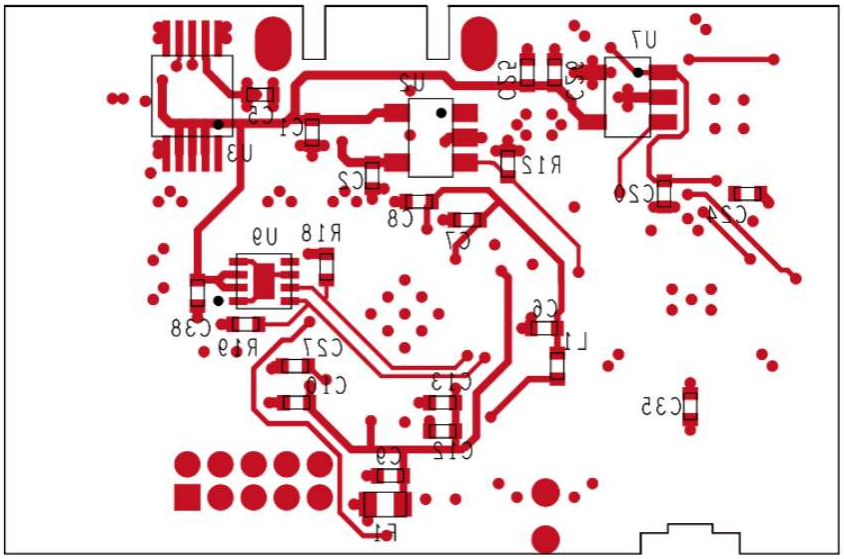

Figure 3.5: BOTTOM layer.

3.2.3 VCC and ground layers

As mentioned above, a 4-layer implementation was chosen to optimize the board design. Hence, VCC (Fig. 3.6) and ground (Fig. 3.7) planes were created inside the board with the aim of not routing any power network inside the two external layers. A ground plane plays an extremely important role, limiting the voltage drop due to the non-null impedance of the conductive material, which otherwise would affect the accuracy of the whole system. This impedance, indeed, can be reduced by creating a wider ground reference (the larger the conductor, the lower the impedance) and by interconnecting it through very short traces. Furthermore, an extended ground plane facilitates the flow of the return currents among the high-frequency components, reducing thus possible interferences.

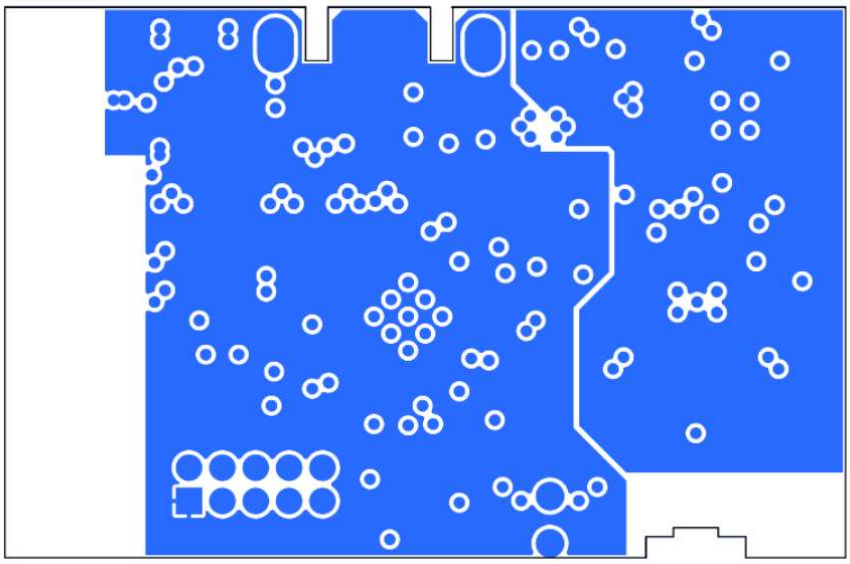

Figure 3.6: VDC layer.

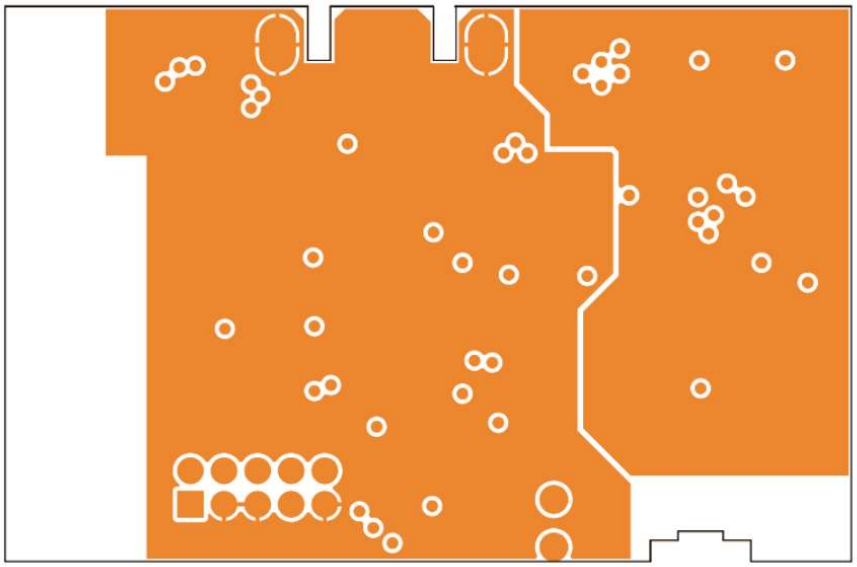

Figure 3.7: GND layer.

3.3 Results and possible improvements

The PCB design can be currently evaluated from the perspective of area and use of resources. A significant improvement can be readily observed in the total size and the space management. Moreover, each component was selected taking care to the costs and the market availability. The electrical characteristics of every component were considered as well, e.g. the tolerances and the maximum voltage and power ranges. In the Appendix the full BOM (bill of materials) is exposed in this respect.
As mentioned above, the design could be enhanced by using blind/buried vias which would optimize the space management and the connections among layers. Nonetheless, this expedient proves very expensive during assembly and, consequently, difficult to implement for a low-cost device.
After manufacturing, the circuit will need a proper case which can exploit the reduced size of the board. All the user interfaces peripherals and connectors were located in strategic points, so as to make the device easy to use. In particular, the on/off push button and the micro-USB connector were placed on the long sides of board, while a green and a red LEDs (to indicate the operating and recharging status) were located on the top. The forthcoming purpose is to supply the board with proper plastic light pipes, aimed to funnel the LED light to the top wall of the case. Two 0.8 mm pitch wire-to-board connectors (to connect the electrodes and the battery) were instead placed on the board opposite corners, so as not to be in the way during assembly.

4. Firmware

All the firmware developing phases, necessary to program the microcontroller and to allow it to interface with an external user, are described in the following, together with the developing environment and all the supports for simulations and debugging (some section of the C code is thus reported as well). An accurate analysis of the microcontroller features and functionalities was indeed the basic for using it correctly and exploiting its full potential. Finally, a detailed overview of the BLE protocol stack is given, in order to comprehend the used approach for data transmission to a mobile device, as well as the format in which the data is elaborated and transmitted. The discussion of the signal processing methods used to denoise the digital signal is postponed to the next chapter. The firmware described here is therefore the one intended for sampling, storing and raw data transmission, as coming from the analog front-end.

4.1 Development environment

The firmware development was conducted union of five fundamental resources, used to assembly, debug and simulate the whole system:

- The integrated development environment (IDE), which is Code Composer Studio, provided by Texas Instruments for embedded applications. Such IDE contains a C/C++ compiler, a source code editor, a low-level JTAG based debugger, a linker and many other functionalities. This was the key tool to implement the entire code, based on several drivers and examples provided by SimpleLink MDU Software Development Kit (SDK), made available by Texas Instruments to support the developers in microcontroller programming;

- The TI LaunchPad kit (LAUNCHXL-CC2640R2), also provided by SimpleLink SDK for a rapid prototyping of radio-frequency low-power applications. This board was used as Code Composer Studio's hardware support, to configure the microcontroller and debug the code;

- The aforementioned prototype of the analog front-end circuit, used to detect the electrical signal from the human body, elaborate it and transmit a raw version to the ADC (such as a not yet filtered version and thus full of any sort of noise);

- The MATLAB numeric computing environment (provided by MathWorks), used during many development phases as a serial terminal for the data receiving and the subsequent processing;

- Lastly, an Android Bluetooth terminal become necessary to interface with the device in a way to receive the transmitted data.

Along the various development phases, the drivers provided by SimpleLink were exploited and the SDK examples were taken as models to start with. At first a simple design was implemented to sample data at a default frequency (namely 500 Hz), to store it in the flash memory and then to transmit it in a raw form to the pc, through serial port, so as to display it in MATLAB and

draw initial conclusions. Sequentially, the Bluetooth Low Energy protocol stack was implemented for the wireless connection to a mobile device and the data packets transmission, after reading them from the flash memory. An additional intermediary step (which will be subsequently discussed) consists in processing the data before sending it via Bluetooth.

4.2 Program structure

The presented program executes a set of operations inserted into a never-ending loop, starting from a simple command given by the user. The algorithm was implemented on the basis of a state machine explained in Fig. 4.1. After Bluetooth connection and after receiving the boot command, the machine starts the 10-second acquisition of data from the analog channel of the microcontroller, with a sampling frequency of 500 Hz (thus acquiring 5000 16-bit samples). The acquisition takes place by blocks of 100 samples which are immediately stored into the flash memory (after having conveniently erased enough space to contain all the samples, i.e. a total of 10 Kbytes). At the end of the acquisition a proper interrupt signal is activated to start to read data in blocks and temporarily store it into a signal processing buffer. The processed data is then packed and transmitted.

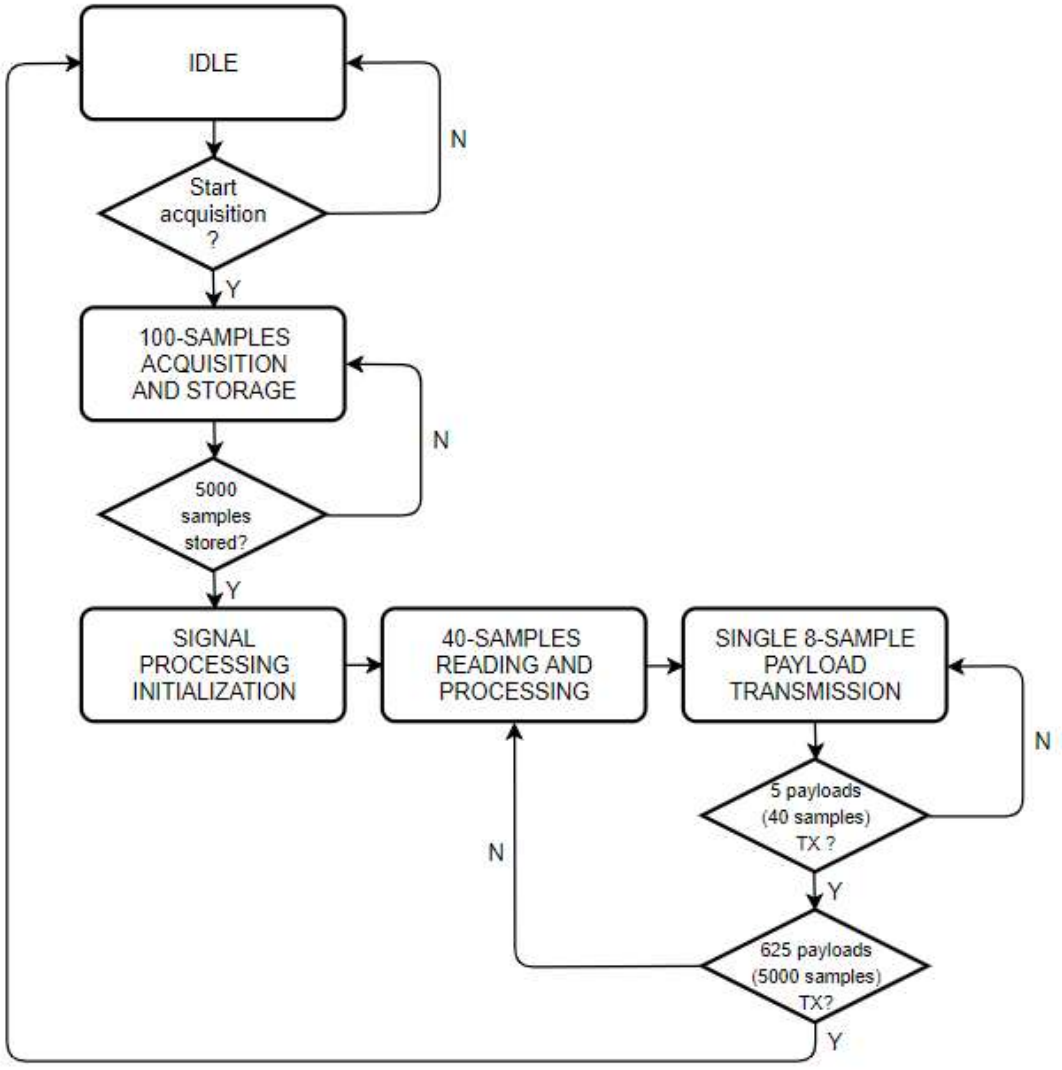

Figure 4.1: Algorithm flow chart.

It is necessary, at this point, an estimation of the internal memory capacity [ ]. The CC2640R2F microcontroller includes several types of both volatile and non-volatile memories:

- 128 Kbytes flash memory, where all the system configuring variables are stored (including hardware interrupts), the stack and application code, a maximum of 8 Kbytes dedicated to the GAP bond manager (the Bluetooth protocol handler) and a last 88 bytes section to store customer-specific chip configuration (GGFG) parameters. This memory is used to store the firmware as well as the global variables (namely, the variables non intended to be often updated during the program execution) and can be also used to save huge amounts of data coming from the ADC. Nevertheless, it is not possible to allocate data everywhere. On the contrary, only few memory sections can be designated for this purpose. In addition, erase operations are only permitted for entire block of 4 Kbytes;

- 20 Kbytes RAM, used to store variables necessary for the RTOS execution (real time operating system), as well as the runtime interrupt vector table and the statically allocated application and stack data. The remaining space is used as heap for the dynamic data allocation. As it is possible to notice, the available RAM memory is not that large and hence a correct space partitioning is crucial, to properly connect to other devices and execute the sampling and signal processing algorithms;

- 8 Kbytes cache memory, which can be potentially used as RAM in a way to obtain additional space and to extend the RAM up to 28 Kbytes;

- 115 Kbytes ROM, where the ROM image of the CC2640R2F board is stored;

- AUX (auxiliary) 2 Kbytes RAM, which can be used as additional SRAM but whose access is slower w.r.t. the standard SRAM. Its exploitation would slow down the program execution and increase the power consumption.

Because of the low RAM capacity, it is appropriate to make sure not to create long data arrays and to minimise the number of operations, especially during the signal processing phase.

4.3 Data acquisition

The first function executed after receiving the boot command is the 10-second data acquisition, at a sampling frequency of 500 Hz. However, before acquisition, it is necessary to configure one of the two microprocessor ADC channels and to connect it to one of the eight analog input of the board, as shown in the Listing 4.1.

Listing 4.1. ADC configuration before starting the acquisition.

```
1.      ADCBuf_Handle adcBuf;
2.      ADCBuf_Conversion continuousConversion;
3.
4.      /* Initialize ADCBuf driver */
5.      ADCBuf_init();
6.
7.      /* Set the ADC parameters */
```

```
8.      ADCBuf_Params adcBufParams;
9.      ADCBuf_Params_init(&adcBufParams);
10.       adcBufParams.callbackFxn = adcBufCallback;
11.       adcBufParams.recurrenceMode =ADCBuf_RECURRENCE_MODE_CONTINUOUS;
12.       adcBufParams.returnMode = ADCBuf_RETURN_MODE_CALLBACK;
13.       adcBufParams.samplingFrequency = ECG_SAMPLING_FREQUENCY;
14.       *adcBuf = ADCBuf_open(Board_ADCBUF0, &adcBufParams);
15.
16.       continuousConversion->arg = NULL;
17.       continuousConversion->adcChannel = Board_ADCBUF0CHANNEL0;
18.       continuousConversion->sampleBuffer = sampleBufferOne;
19.       continuousConversion->sampleBufferTwo = sampleBufferTwo;
20.       continuousConversion->samplesRequestedCount = ADCBUFFERSIZE;
21.
22.       /* Start acquisition */
23.       ADCBuf_convert(adcBuf, &continuousConversion, 1);
```

The so-called *RECURRENCE_MODE_CONTINUOUS* mode was so set, I such a way to sample data through the alternation between two 100-element buffers. The acquisition starts when the *ADCBuf_convert* function is called. Once the first buffer is filled, the code jumps to a callback function connected to the ADC channel (*adcBufCallback*), which elaborates those 100 samples while the second buffer is filled in parallel. After filling it, the *adcBufCallback* function is again called and the two buffers can alternate anew. This cycle continues until 5000 samples are acquired (the *adcBufCallback* is therefore called 50 times). The counting is performed by a global variable (so-called *buffersCompleteCounter*) which increments its value every time the *adcBufCallback* function is called (Fig. 4.2).

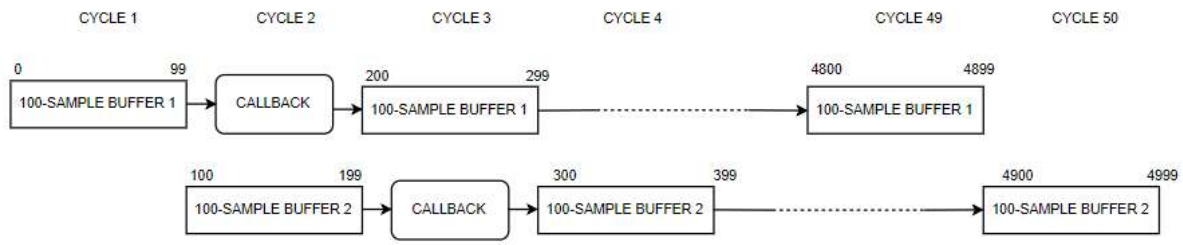

Figure 4.2: 2-beffer ADC acquisition.

The *adcBufCallback* function (Listing 4.2), after receiving the samples in a block of 100 elements, is responsible to adjust every single 16-bit sample into a microvolt value. The full block is then stored in the flash memory and the counter is incremented by one unit. After acquiring all the 5000 samples, the conversion is cancelled by calling the *ADCBuf_convertCancel* function, waiting to start the first reading cycle.

Listing 4.2. adcBufCallback() function.

```
1. /*
2.  * This function is called whenever an ADC buffer is full.
3.  * The content of the buffer is then converted into human-readable
4.  * format and ECG_taskFxn is notified.
5.  */
6.
7. static void adcBufCallback(ADCBuf_Handle handle,
```

```
                                    ADCBuf_Conversion *conversion,
                                    void *completedADCBuffer,
                                    uint32_t completedChannel)
{
    uint16_t nvsWriteBuffer[ADCBUFFERSIZE];
    unsigned int i;

    /* Adjust raw ADC values and convert them to microvolts */

    ADCBuf_adjustRawValues(handle, completedADCBuffer,
        ADCBUFFERSIZE, completedChannel);

    ADCBuf_convertAdjustedToMicroVolts(handle, completedChannel,
       completedADCBuffer, microVoltBuffer, ADCBUFFERSIZE);

    /* Convert from 32 bits to 16 bits */
    for(i=0; i<ADCBUFFERSIZE; i++)
        {
            microVoltBuffer16[i] = microVoltBuffer[i] >> 10;
            nvsWriteBuffer[i] = microVoltBuffer16[i];
        }

    /* Write 100 samples into the flash memory*/

    NVS_write(nvsHandle, buffersCompletedCounter*ADCBUFFERSIZE*2,
        (void *)nvsWriteBuffer, sizeof(nvsWriteBuffer), NULL);

    buffersCompletedCounter++;

    /* Once the 50th buffer has been filled, send an interrupt

    if(buffersCompletedCounter == WRITECNT)
    {
        Event_post(ecgSyncEvent, ECG_MEM_READY_EVT);
    }
}
```

4.4 Data storage

In order to write all the data into the flash memory, it is first needed to erase a sufficient portion to contain everything. The microcontroller CC260R2F allows to dedicate entire blocks of 4 Kbytes to this purpose, inside a maximum writable space which depends on the fullness of the rest of the memory. As already mentioned, a 4 Kbytes sector is the smallest unit of flash memory that can be erased at one time. Consequently, the erasable region size must be a multiple of the sector size. In any case, it is strictly forbitten to write data over the address 0x1FFA8, used to store customer-specific chip configuration (CCFG).
Since the goal is to store 10 Kbytes of data, the writable range was set within the interval of 0x1A000 – 0xCFFF, which is the size of three sectors of 4 Kbytes each (Listing 4.3).

Listing 4.3. Flash memory allocation.

```
#define NVS_REGIONS_BASE        0x1A000
#define SECTORSIZE              0x1000
#define REGIONSIZE              SECTORSIZE * 3
```

Just like the writing operation, reading from the flash memory is allowed for a single 8-bit data as well as for an entire data block, as long as the block size and the starting address are declared. The two functions NVS_write and NVS_read were used respectively to write and read blocks of 100 samples of 16 bits, to an ever-increasing address (Listing 4.4).

Listing 4.4. Non-volatile write and read functions.

```
1. int_fast16_t NVS_write(NVS_Handle handle, size_t offset, void
2. *buffer, size_t bufferSize, uint_fast16_t flags);
3.
4. int_fast16_t NVS_read(NVS_Handle handle, size_t offset, void *buffer,
5. size_t bufferSize);
```

To monitor the writing operation and the following reading of all the 5000 samples, the CCS *Memory Browser* tool was used and an initial system able to send data to the MATLAB environment through serial port was built. In this first phase, as noisy as the signal can be, it appears in line with expectations.

4.5 Bluetooth interface

The device Bluetooth interface, that is the protocol which allows the interconnection and the information exchange between a server and a client, was developed from the Bluetooth low energy (BLE) protocol stack provided by the CC2640R2F SDK platform. Based on the *Simple_Peripheral* project, which implements a basic BLE peripheral system, a custom application was created, by generating a proper profile able to manage the data and control signal exchange. In particular, a new service with two different characteristics was added to the GATT service (Generic ATTribute profile, i.e. the top layer which controls all the others). Such characteristics allow the device to receive a command code from a mobile device and to transmit data packets. An overview on the Bluetooth stack basic concepts is provided below, along with a detailed description of the developed system.

4.5.1 Basic concepts

Thanks to the BLE protocol stack [ ], the microcontroller can connect to another host device. It is the most important task of the system (the one with the highest priority) and it thus controls all the other tasks. Hence, it can be seen as the structure which supports the whole system and without whom many other functions cannot work properly. The protocol top layer is covered by the GAP (Generic Access Profile) [ ] and the GATT [ ]. The first one is responsible to manage the connection status of the device, passing through the state of *Standby*, *Advertising*, *Scanning*, *Initiating* and *Connected* (Fig. 4.4). On the other hand, the second one is a profile (i.e. a set of services and characteristics, which in turn are packets of info, so-called attributes) that controls the other profiles of the systems, including the one created ad hoc for this project. Each profile, thereby, must pass through the GAP to manage its connection status and through the GATT to interface with the protocol stack.

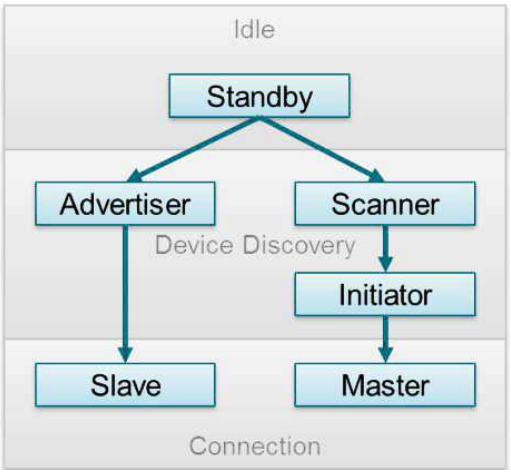

Figure 4.4: Connection states diagram [ ].

In addition, every application must go through a framework, called ICall (Indirect Call Framework) to use the BLE protocol stack (Fig. 4.5). Such a module allows the application and protocol stack to work efficiently, communicate and share resources []. The ICall functionalities, together with the GAP, the GATT profile and all the applications which interface with the protocol stack, need to be activated by a main function, contained in the file so-called *main.c*. This function is on the bottom of every project and is responsible for the system configuration. Moreover, the main function also activates the TI RTOS (Real-Time Operating System), whose image is stored in the flash memory.

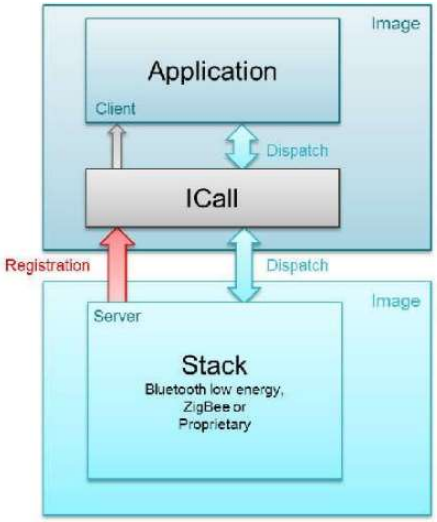

Figure 4.5: ICall as a framework between the application and the protocol stack [ ].

A new task (e.g. the ECG task) is created by defining a new stack and setting the task parameters, such as the stack size and the task priority (Listing 4.5). When the new task is added and it is supposed to communicate with the protocol stack, it is moreover important to register this task with ICall. This is done by creating a new entity and a new synchronization event. By doing so, the task can be registered to ICall through the function *ICall_registerApp* and be synchronized with the main function by letting it pend on this new sync event. Finally, it necessary to modify the maximum number of ICall enabled tasks and entities, by increasing the corresponding preprocessor symbols.

Listing 4.5. Task registration to ICall and synchronization to the main function.

```
1. // ECG Task configuration
2. #define ECG_TASK_PRIORITY                1
3. #define ECG_TASK_STACK_SIZE              1200
4. static ICall_EntityID ecgSelfEntity;
5. static ICall_SyncHandle ecgSyncEvent;
6.
7. // ECG Task registration to ICall
8. ICall_registerApp(&ecgSelfEntity, &ecgSyncEvent);
9.
10.  // Synchronization to the main function
11.  events=Event_pend(ecgSyncEvent,Event_Id_NONE,ECG_ALL_EVENTS,
12.                     ICALL_TIMEOUT_FOREVER);
```

4.5.2 Developed system

As already explained, the *Simple_Peripheral* [ ] project (provided by SimpleLink) was used as the basis for implementing the BLE protocol stack. A new profile was added, which was expressly created to handle the information exchange aimed to transfer the ECG signal to a mobile device. Such profile contains a service (*ecg_service*) that in turn consists of two characteristics, called *Payload* and *Command*:

- *Payload* is the characteristic which allows to receive data packets after they are read from the flash memory, processed and adjusted to a specific format. This characteristic has thus two main properties (i.e. functionalities):
    - *Read*, which lets the host device read the newly arrived data;
    - *Notify*, to let the host activate the characteristic functionality and be aware of when a new payload has arrived.

- *Command* is the characteristic which allows to send a command to the microcontroller. Its properties are the following:
    - *Write*, to send a command to the host;
    - *Notify*, to notify the host every time a new command is transmitted.

Table 4.1. Possible commands to be transmitted to the device.

Command	Description	Value
CMD_START_ACQ	It makes the acquisition start	0x03
CMD_REQ_SEND	It sends a generic interrupt	0x0C
CMD_ABORT_ACK	It sends an abort request	0xAA
CMD_DONE	It makes the algorithm stop	0xC0

The above-described functions of acquisition and storage in the flash memory and the signal processing methods (to be described in the next chapter) are inserted in this structure. Once the system has been initiated, it connects to the mobile device making the request (by means of a *pairing* operation) and waits to get a command (Table 4.1). After receiving it through the just outlined service, the system begins to sample the signal from the ADC, stores it into the flash memory and elaborates it through dedicated signal processing algorithms. When the data are ready for the transmission, they are packed and sent to the mobile device by interfacing with the protocol stack.
The data are transmitted in 19-byte payloads, where the first 16 bytes (from 0 to 15) contain the ECG samples, 2 bytes per sample (Listing 4.6).
The bytes 16 and 17 contain the counting of the transmitted payloads, from 0 to 625 (0x271), i.e. 5000 divided by 8.
The last byte, that is the byte 18, contains a value which gives information on the just received payload, e.g. it is equal to "0x03" when a payload is successfully transmitted, "0xC0" when the last payload arrives and "0x55" when an error occurred during the transmission (Table 4.2).

Table 4.2. Transmission codes.

Code name	Description	Value
TxCode_START_ACQ	It is sent when the acquisition starts	0x03
TxCode_ERR_ACQ	It is sent when an error occurred	0x55
TxCode_ACQ_DONE	It is sent when the acquisition is cancelled	0x0C
TxCode_DATA_VALID	It is sent every time a valid payload is transmitted	0x30
TxCode_DATA_LAST	It is sent when the last payload is transmitted	0xC0

Each byte of the chunk (part of the payload which contains data) stores an int8 value, i.e. an 8-bit signed integer number. By combining two successive bytes in a little-endian format, an ECG int16 sample is obtained. This means that the least significant byte of the sample is the first to arrive.

Listing 4.6. Payload definition prior to being sent.

```
1. static void ECG_sendPayload(char * Chunk, uint16_t Offset,  UInt8
   TxCode)
2. {
3.     uint8_t payload[ECG_CHUNK_SIZE + 3] = {0};
4.     if (Chunk)
5.     {
6.         memcpy(payload, Chunk, ECG_CHUNK_SIZE);
7.         memcpy(payload+ECG_CHUNK_SIZE, &Offset, 2);
8.     }
9.
10.      payload[ECG_CHUNK_SIZE + 2] = TxCode;
11.      Ecg_service_SetParameter(ECG_SERVICE_PAYLOAD,
12.                               ECG_SERVICE_PAYLOAD_LEN, payload);
13.
14.      payloadCounter++;
15.  }
```

4.6 Raised issues and results

The biggest challenge in the firmware development consisted in organizing the code in a way to avoid any waste of resources, i.e. memory. The most serious faced issued was the one concerning the device *pairing*. Such operation, indeed, requires a careful memory management, which has a very limited capacity itself. It was so necessary to fully understand how much memory this operation needed, in order to allocate a suitable space (within the limits). In this respect, two flash pages were reserved for the SNV (storage non-volatile), which is the area used for non-volatile memory storage by the GAP Bond Manager. Furthermore, for a better memory management, the stack size of the *simple_peripheral* task was limited to 300 bytes while the stack size of the *ecg* task was increased to 1200 bytes. Such a decision was taken after checking the two task stacks situation, by means of the CCS Memory Allocation tool.
The obtained results are satisfactory and show that a significant amount of data can be acquired and subsequently transmitted without any loss of information. The device flash memory was also properly used without slowing down the execution, thus opening the device to many other possibilities which require the processing of the data stored in the internal memory.

5. Signal processing

In this section all the analysed signal processing methods are described, in a way to provide an overview on some possibilities as regards the ECG signal denoising. All the outcomes of the examined algorithms are reported, to compare them and evaluate their effectiveness. Since the possibilities are countless, a final signal processing method to be incorporated in the final firmware has not yet been selected. Nevertheless, the hardware features of the device and the firmware structure allow to retain a certain amount of flexibility. It is still important to consider right away which obstacles have been encountered during the insertion of some algorithms inside the firmware, such as those connected to the limited memory capacity, often not able to handle complex and elaborate processes.

As described above, in the analog part of device the signal is detected by two electrodes and then, after passing through the instrumentation amplifier, is sent almost unchanged to the ADC, undergoing a soft noise filtering out of the (0.5-100) Hz bandwidth. As can be seen in Fig. 5.1, several remarkable artefacts are still visible in the signal, e.g. the high-frequency noise (due to electrical and physical defects inside the circuit, muscular movements and incorrect connections between the fingers and the electrodes), besides the 50 Hz network interference and the baseline drift caused by the respiration.

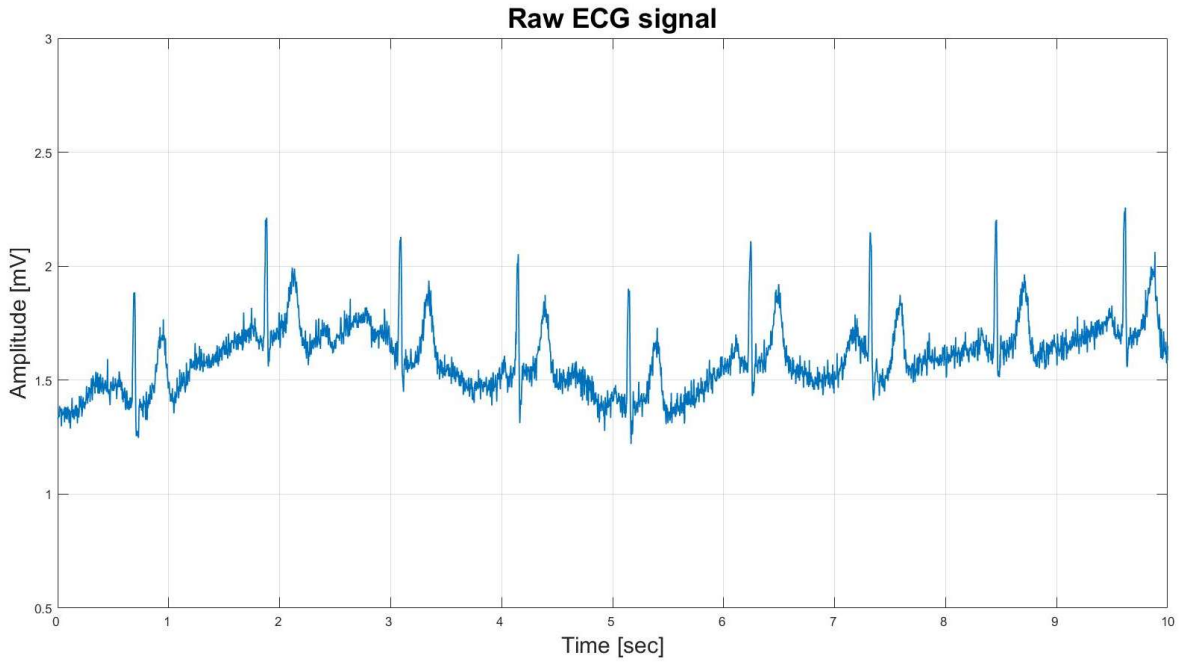

Figure 5.1: Raw ECG, before being processed.

The techniques used to remove each of these artefacts are presented, outlining the differences in terms of signal quality and computational cost. The analysed methods are listed below:

- 50Hz noise filtering
  - Notch filter (stop-band)
  - Comb filter

- High frequencies filtering
  - Several IIR low-pass filters, with a 35Hz cutoff frequency
  - Moving average, with a small window size
  - Moving median, with a small window size

- Low frequencies filtering
  - Several IIR high-pass filters, with a 0.5Hz cutoff frequency
  - Moving average, with a wide window size
  - Moving median, with a wide window size

During the development process, each algorithm was analysed along three abstraction levels. Firstly, their mere effectiveness in the signal denoising was evaluated in the MATLAB environment. Secondly, the development evolved into the algorithm implementation in C language, through a different compiler w.r.t. the one inside Code Composer Studio. Lastly, the developed algorithms were inserted in the state machine described in the preceding section. During this step, several issues were addressed, such as the memory constraints as regards the variables and array allocation. Another significant issue to be considered was the data conversion from the floating-point format (in which they were elaborated in the previous stage) to the fixed-point format, due to the absence of a floating-point unit inside the microprocessor.

5.1 MATLAB simulations

The starting raw signal was here processed through proper simulations in the MATLAB environment, and possible improvements were considered. During this phase, a fundamental instrument was the signal transposition of the signal in the frequency domain, by means of the Fourier transform. The first step toward a complete denoising was, indeed, to determine which frequencies belong to the ECG signal frequency band [5] and, by contrast, which ones should be removed, being the result of external disturbances. A proper function was consequently used to decompose the signal in the frequency domain and to plot the resulting Fourier transform (Listing 5.1).

Listing 5.1. fourier_plot function.

```
1. function fourier_plot(sig, f, nf)
2.
3.     if(nargin == 1)
4.         f = 500;
5.     elseif (nargin < 3)
6.         nf = 2^ceil(3+log2(length(sig)));
7.     end
8.
9.     sig_fshift = fftshift(abs(fft(sig, nf)));
10.      nl = length(sig_fshift);
11.      df = f / (nl);
12.      fk = df * (-nl/2:nl/2-1);
13.      figure()
14.      plot(fk(nl/2:end),20*log10(sig_fshift(nl/2:end)));
15.  end
```

The Figure 5.2 shows how, in an ECG signal, most of the power is concentrated on frequencies lower than 100 Hz. Furthermore, the frequencies close to 0 Hz are very pronounced, as well as the ones in the vicinity of the network frequency (i.e. 50 Hz) and its multiples.

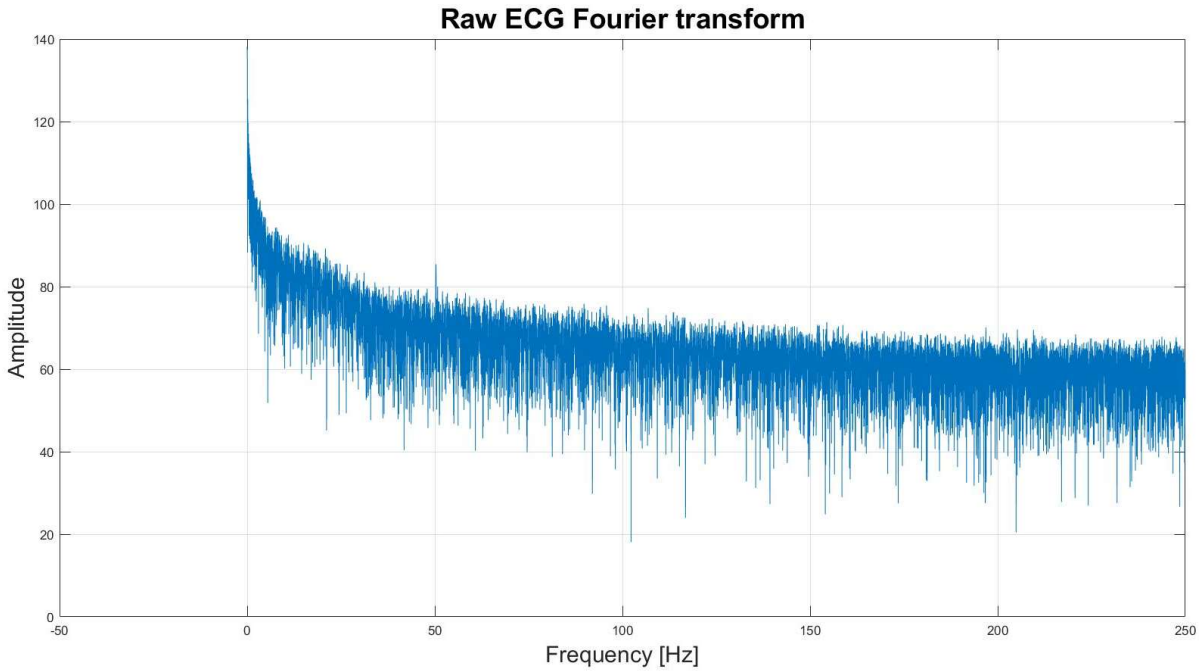

Figure 5.2: ECG Fourier transform.

5.1.1 50 Hz noise filtering

The elimination of the 50 Hz disturbance, caused by the network, and its multiples (i.e. 100 Hz, 150 Hz etc) should be the first step to deal with during the denoising process. During this initial stage, a proper function, provided by Mathworks, was used to implement a simple IIR (Infinite Impulse Response) notch filter, that is a stop-band filter, centered on 50 HZ and with a bandwidth at -3dB point set as in Listing 5.2.

Listing 5.2. Notch filter configuration and implementation.

```
1. wo = 50/(500/2);  bw = wo/1;
2. [b_notch,a_notch] = iirnotch(wo,bw);
3.
4. ecg_notch = filter(b_notch,a_notch,ecg);
```

In Fig. 5.3 it is possible to see how this filter can remove the network noise, without altering the original signal shape, e.g. without smoothing the R spikes (while considering that the QRS complex has some high-frequency components as well, which instead must not be absolutely removed). It was decided not to implement a more aggressive notch filtering, with the attendant risk of affecting the signal and adding artefacts.
The quality of this filter was, moreover, evaluated through the SNR (Signal to Noise Ratio) calculation, which is the ratio between the relevant signal power and the noise power, both measured in decibel. This dimensionless value gives information on the quantity of noise removed from the original signal, but also allows to determine how altered the signal comes out

from the filtering, due to any sort of manipulations (such as spike reductions or mismatches). The SNR has high positive values if the filtering is effective, while it has high negative values in case of a low-quality filtering or even when the signal results corrupted.
In this case, the SNR value for the notch filter is equal to 34.9654. This value was calculated for all the implemented filters.

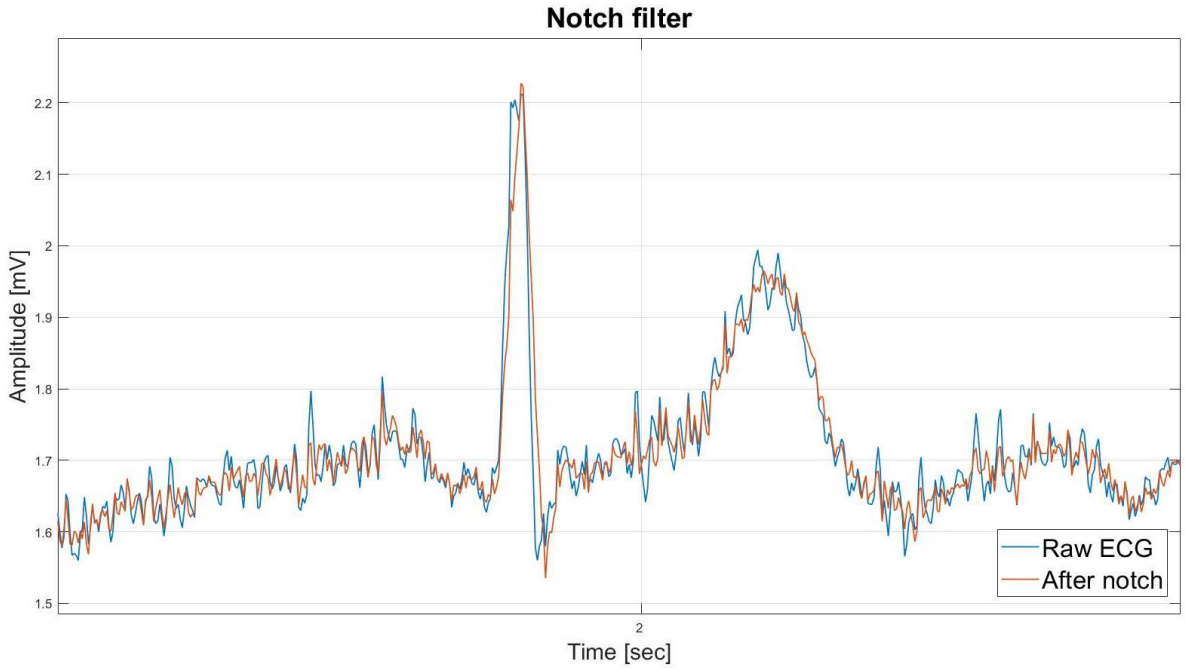

Figure 5.3: ECG Fourier transform.

Another solution to remove the 50 Hz noise consists in implementing a comb filter (Listing 5.3), which is a filter able to remove all the 50 Hz multiples and the 0 Hz component as well (Fig.5.4).

Listing 5.3. Comb filter configuration and implementation.

```
1. d=fdesign.comb('notch','N,BW',8,0.02);
2.
3. hd=design(d,'systemobject',true);
4.
5. co=tf2sos(hd.Numerator, hd.Denominator);
6.
7. rcgf=sosfilt(co, ecg);
```

This filter was, however, not considered because of the increase of the S spikes and the rounding of the R spikes. In addition, the signal presents undershoots after the T wave, because of an unproper high-pass filtering. The resulting ECG shape thus results altered w.r.t. the original (Fig. 5.5).

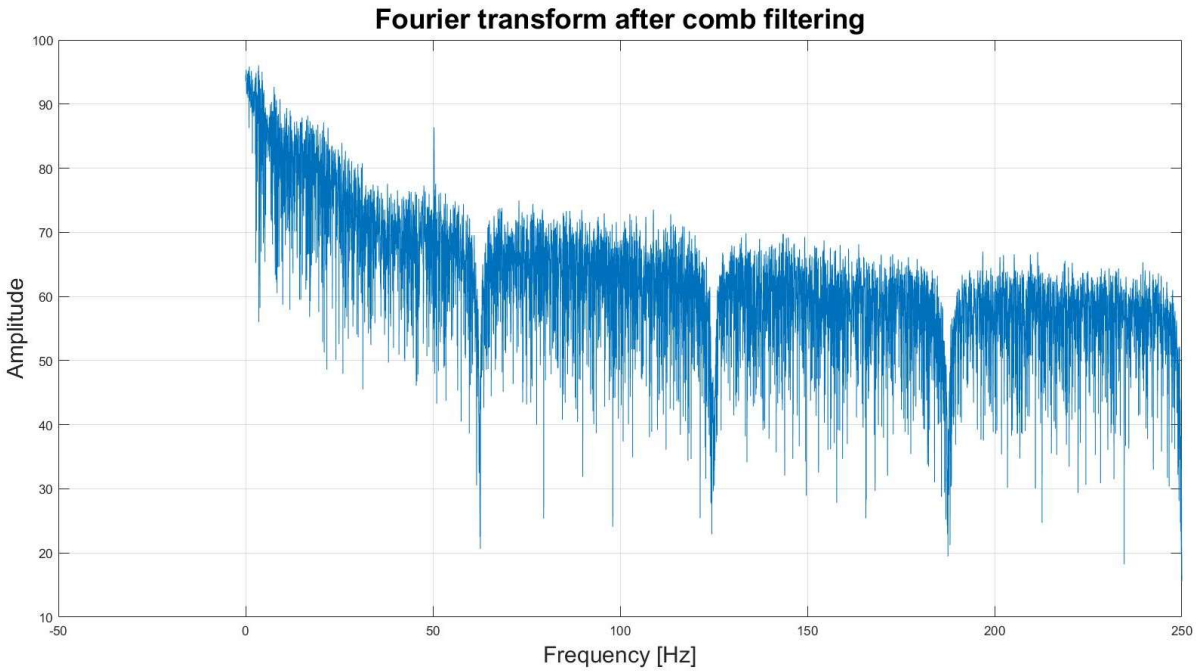

Figure 5.5: ECG Fourier transform.

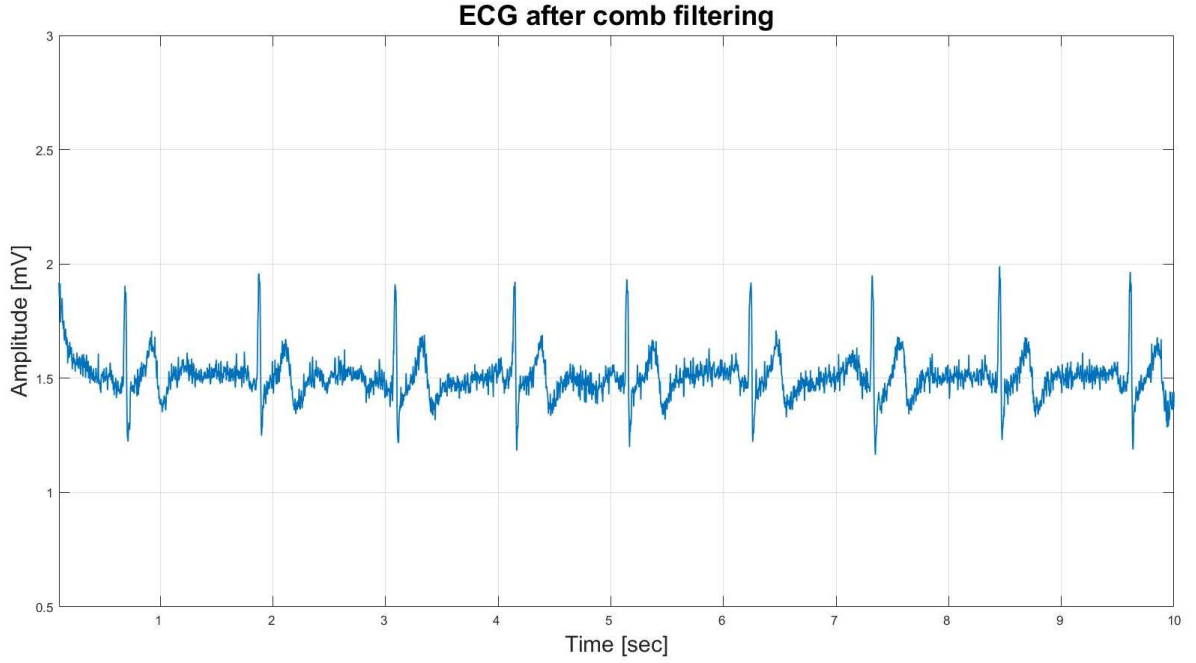

Figure 5.5: ECG signal after comb filtering.

5.1.2 Low-pass filtering

The next step is to analyse some methods to remove all the high-frequency noise from the ECG signal. Firstly, the most common types of low-pass IIR filters were evaluated, such as Butterworth, Chebyshev I, Chebyshev II and elliptic (each with a cutoff frequency of 35Hz), and the difference were examined in terms of effectiveness and artefacts which instead could

be added to the signal by the filters. The parameters used to design the filters are shown in (Listing 5.4).

Listing 5.4. Butterworth, Chebyshev I, Chebyshev II and elliptic low-pass filters configuration and implementation.

```
Wn_butter = (35/500)*2;
[b_butter,a_butter] = butter(2,Wn_butter);
ecg_lp_butter = filter(b_butter,a_butter,ecg);

Rp_cheby1 = 3;
Wp_cheby1 = (35/500)*2;
[b_cheby1,a_cheby1] = cheby1(15,Rp_cheby1,Wp_cheby1);
ecg_lp_cheby1 = filter(b_cheby1,a_cheby1,ecg);

Rs_cheby2 = 20;
Ws_cheby2 = (35/500)*2;
[b_cheby2,a_cheby2] = cheby2(6,Rs_cheby2,Ws_cheby2);
ecg_lp_cheby2 = filter(b_cheby2,a_cheby2,ecg);

Rp_ellip = 5;
Rs_ellip = 40;
Wp_ellip = (35/500)*2;
[b_ellip,a_ellip] = ellip(6,Rp_ellip,Rs_ellip,Wp_ellip);
ecg_lp_ellip = filter(b_ellip,a_ellip,ecg);
```

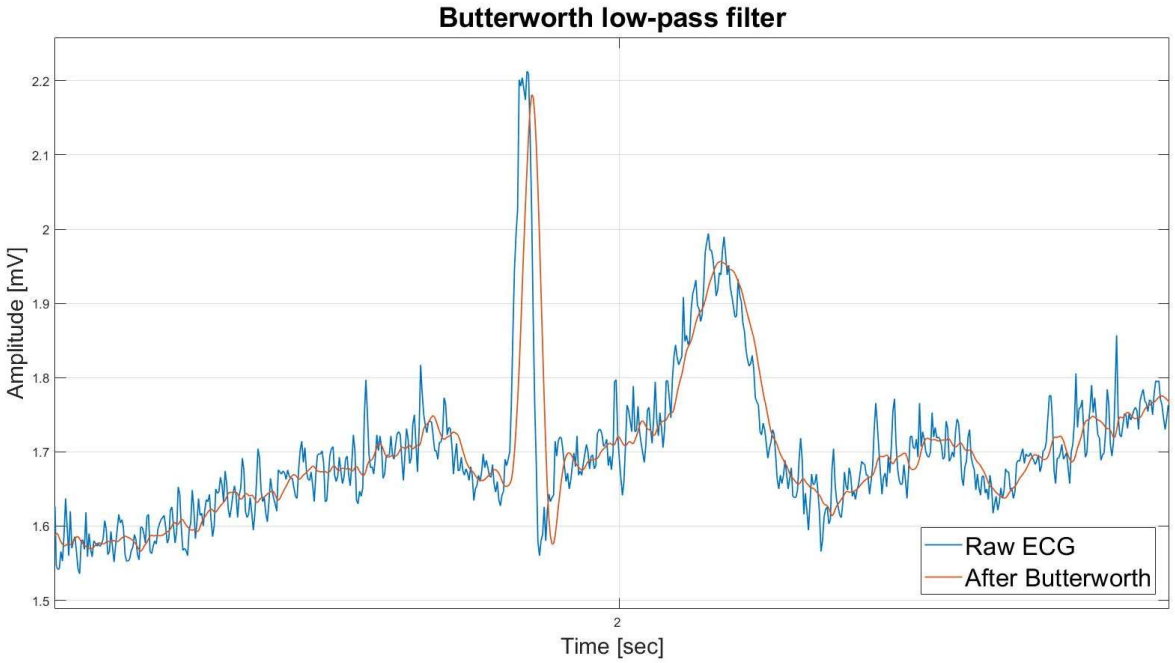

Figure 5.6: ECG signal after Butterworth 2nd order low-pass filter.

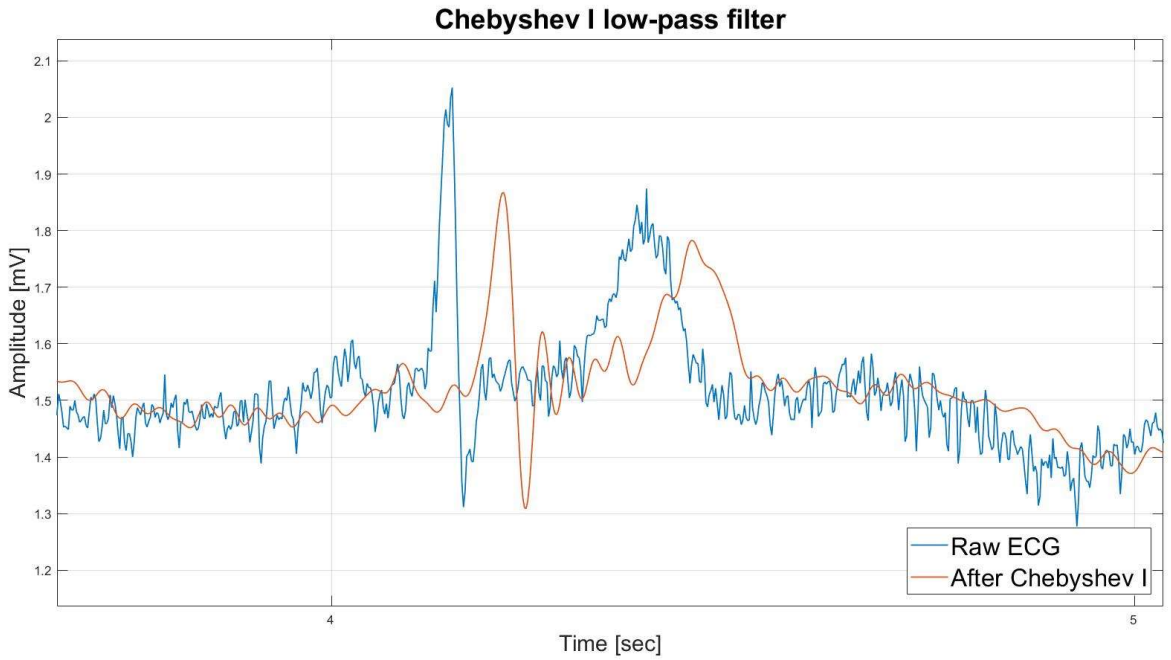

Figure 5.7: ECG signal after Chebyshev I 15th order low-pass filter, with a 3dB ripple in the passband.

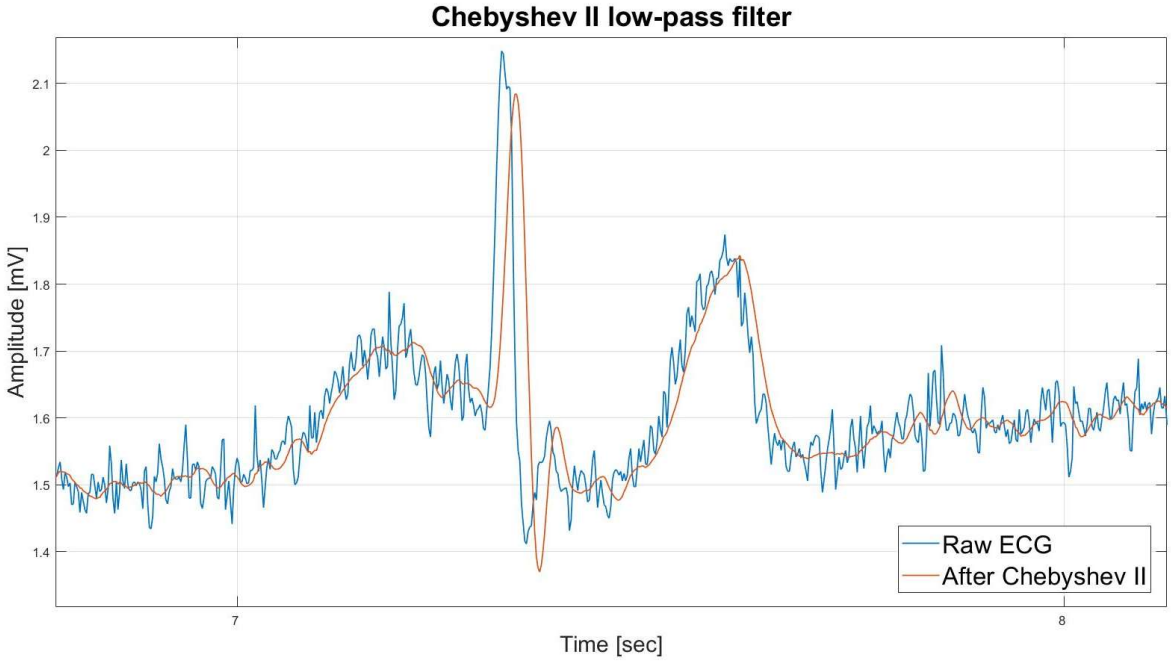

Figure 5.8: ECG signal after Chebyshev II 6th order low-pass filter, with a -20dB ripple in the stopband.

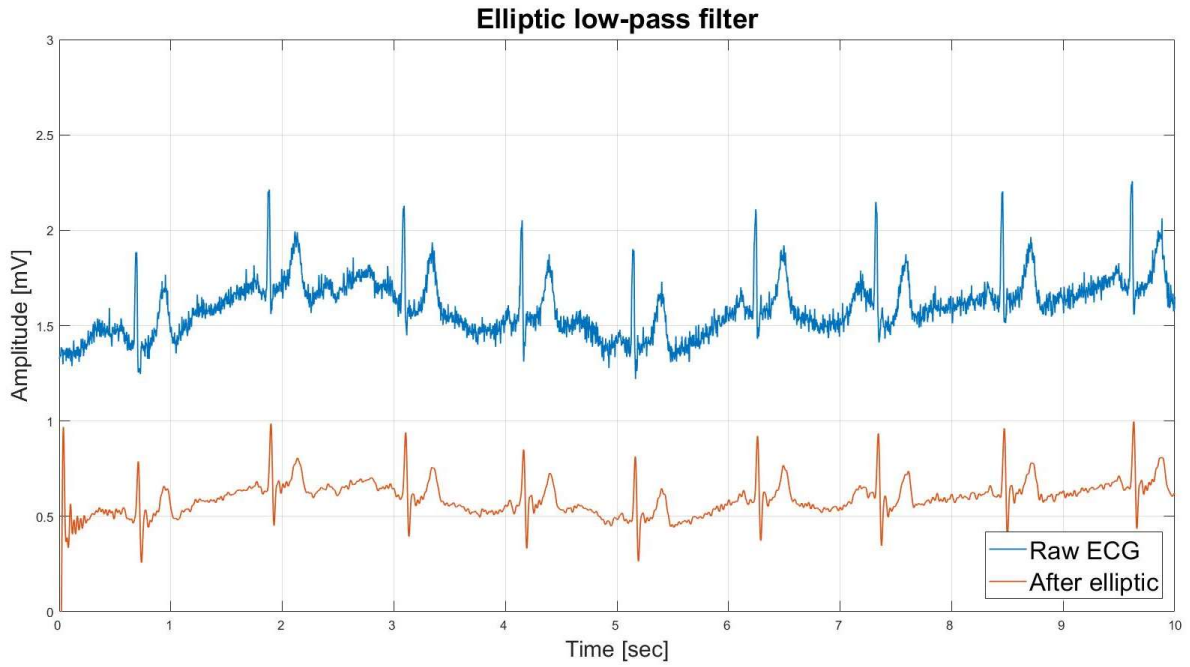

Figure 5.9: ECG signal after elliptic 6th order low-pass filter, with a 5dB ripple in the passband and a - 40dB ripple in the stopband.

It is possible to see how the Butterworth filter (Fig 5.6) removes the high frequencies in a non-aggressive fashion, thanks to its maximally flat response in the passband and its smoothed response around the knee frequency. The same result would be achieved with the Chebyshev II (Fig 5.8) filter, if its ripple in the stopband was set on a low value, but in this case, it would miss its main characteristic which is to have a sloping response after the cutoff frequency. It is also clear that the Chebyshev I (Fig 5.7) and elliptic (Fig 5.9) filters cannot be considered, since their responces implies a signal alteration around the narrowest spikes, adding artefacts such as undershoots and ringing effect, besides a reduction of the spikes in the QRS complex, in some cases coming to 50 %.

Two more techniques for the high frequencies filtering were considered. Such methods exploit the moving average and moving median functions (Listing 5.5) to smooth the signal, i.e. to cut off all its narrowest spikes. The risk is, however, that of rounding also the spikes in the QRS complex, which instead should be left completely untouched.
A moving average is a mathematical tool which allows to assess the average development of a curve by eliminating the values which mostly deviate. Hence, it can be seen as a low-pass filter capable to remove in an aggressive fashion the high frequencies from the signal. A moving average with a window width of 10 values was used for this case. This means that each sample of the signal is replaced by the mathematical average of the ten samples around it.
Similarly, the moving median is defined. Each sample is now replaced by the sample with the middle value inside the window. Here again, a 10 samples window was selected. In Fig 5.10 and Fig. 5.11, the results of these two techniques implementation is exposed.

Listing 5.5. Moving average and median implementation, with a window of 10 samples.

```
1. ecg_lp_mean = movmean(ecg_notch,10)
2. ecg_lp_med = movmedian(ecg_notch,10)
```

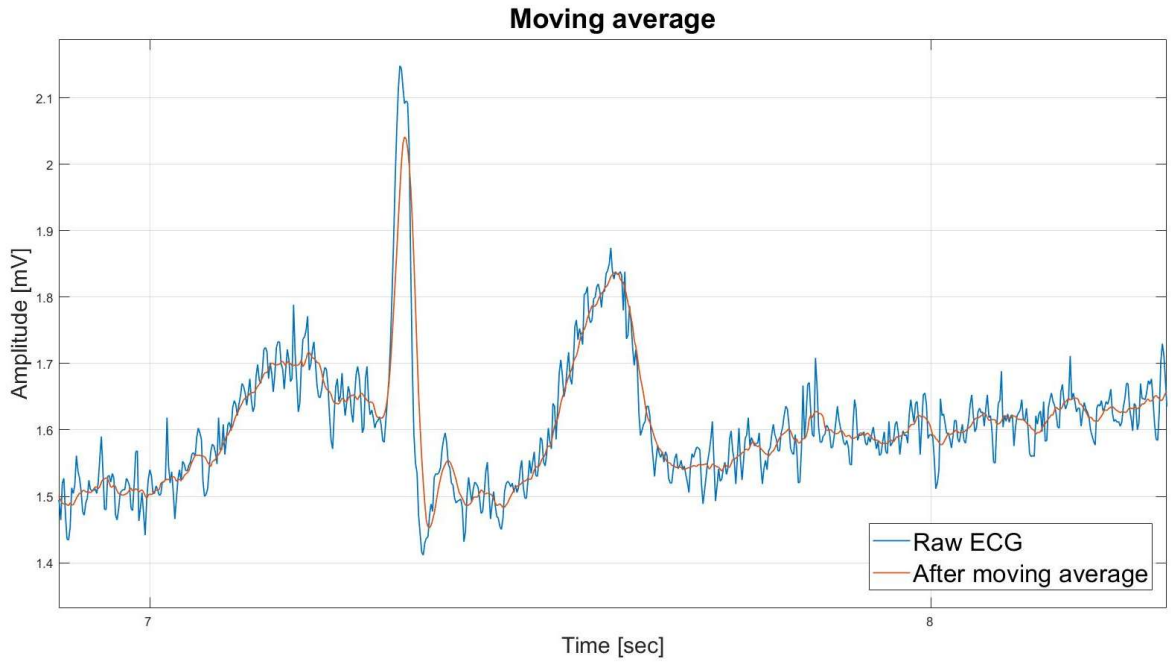

Figure 5.10: ECG signal after moving average low-pass filtering, with a window of 10 samples.

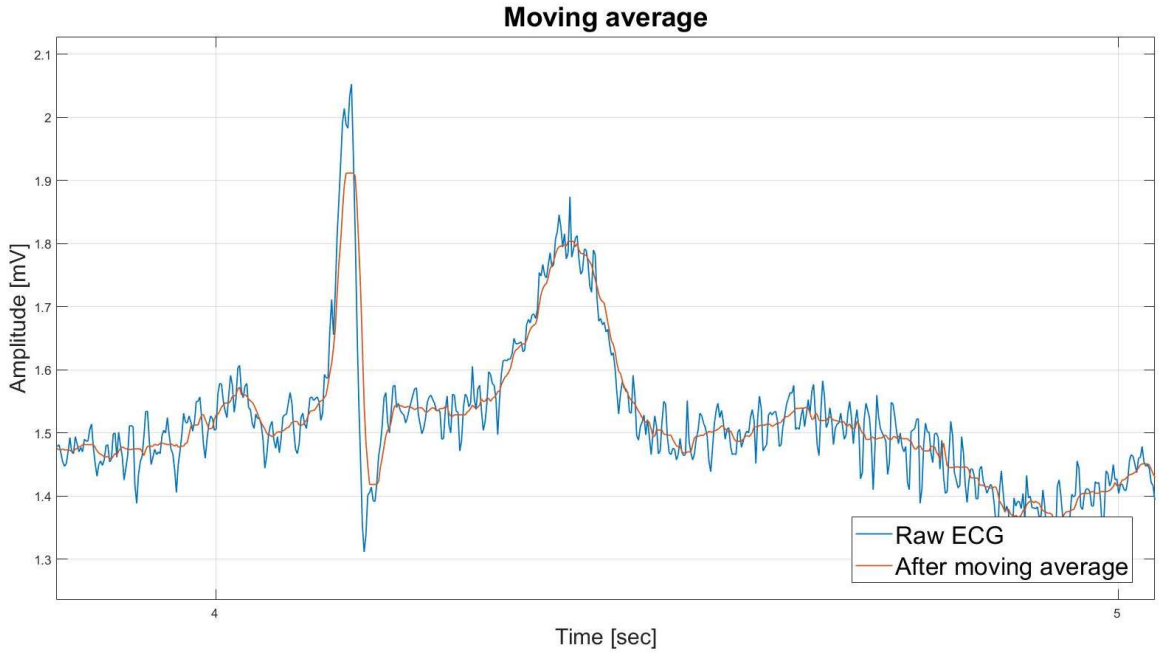

Figure 5.11: ECG signal after moving median low-pass filtering, with a window of 10 samples.

The performances of the just described low-pass filtering techniques can be summarised by calculating their SNR and estimating the reduction of the R spike amplitude, as compared to the raw signal (Table 5.1). It indicates that the Butterworth configuration is the one, among the IIR filters, which can mostly remove the noise. At the same time, the best outcome was achieved with the moving average and median, capable of providing a scrubbed and almost devoid of

alterations caused by the filtering. Nonetheless, the reduction of the signal amplitude caused by these techniques (due to the rounding of the QRS complex spikes) should be considered.

Table 4.1. Possible commands to be transmitted to the device.

Filter	*SNR*	*R spike attenuation*
Butterworth	32.4594	5%
Chebyshev I	21.0042	35%
Chebyshev II	29.7190	12%
Elliptic	2.1516	40%
Moving average	39.0413	18%
Moving median	39.8743	28%

5.1.3 High-pass filtering

The final step of this study consists in evaluating some methods to remove the 0 Hz component and the very low frequencies from the ECG signal, in order to straighten the baseline and to obtain a signal which develops horizontally around the 0V line. In this respect, several options were considered. Firstly, the IIR filters were analysed in the configurations seen above, but this time set as high-pass filters, with a cutoff frequency of 0.5Hz (Listing 5.6). Before the filtering was executed, a value equal to the reference voltage, i.e. 1650 (1.65V) was subtracted from all the signals. In doing so, the filtered signal can reach more quickly the 0V line.

Listing 5.6. Butterworth, Chebyshev I, Chebyshev II and elliptic high-pass filters configuration and implementation.

```
1.  Wn_butter_hp = (0.5/500)*2;
2.  [b_butter_hp,a_butter_hp] = butter(2,Wn_butter_hp, 'high');
3.  ecg_hp_butter = filter(b_butter_hp,a_butter_hp,ecg_lp_butter-1650);
4.
5.  Rp_cheby1_hp = 3;
6.  Wp_cheby1_hp = (0.5/500)*2;
7.  [b_cheby1_hp,a_cheby1_hp] =
8.                  cheby1(2,Rp_cheby1_hp,Wp_cheby1_hp, 'high');
9.  ecg_hp_cheby1 = filter(b_cheby1_hp,a_cheby1_hp,ecg_lp_butter-1650);
10.
11.  Rs_cheby2_hp = 20;
12.  Ws_cheby2_hp = (0.5/500)*2;
13.  [b_cheby2_hp,a_cheby2_hp] =
14.                 cheby2(2,Rs_cheby2_hp,Ws_cheby2_hp, 'high');
15.  ecg_hp_cheby2 = filter(b_cheby2_hp,a_cheby2_hp,ecg_lp_butter-1650);
16.
17.  Rp_ellip_hp = 5;
18.  Rs_ellip_hp = 40;
19.  Wp_ellip_hp = (0.5/500)*2;
20.  [b_ellip_hp,a_ellip_hp] =
21.                 ellip(6,Rp_ellip_hp,Rs_ellip_hp,Wp_ellip_hp, 'high');
22.  ecg_hp_ellip = filter(b_ellip_hp,a_ellip_hp,ecg_lp_butter-1650);
```

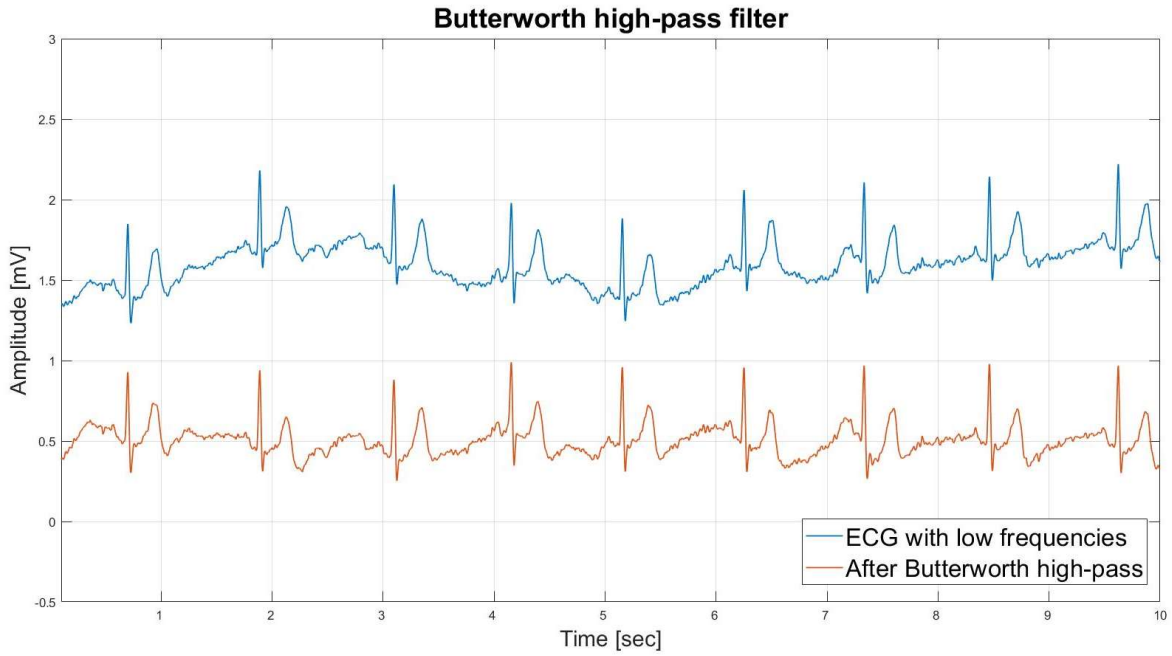

Figure 5.12: ECG signal after Butterworth 2nd order high-pass filter.

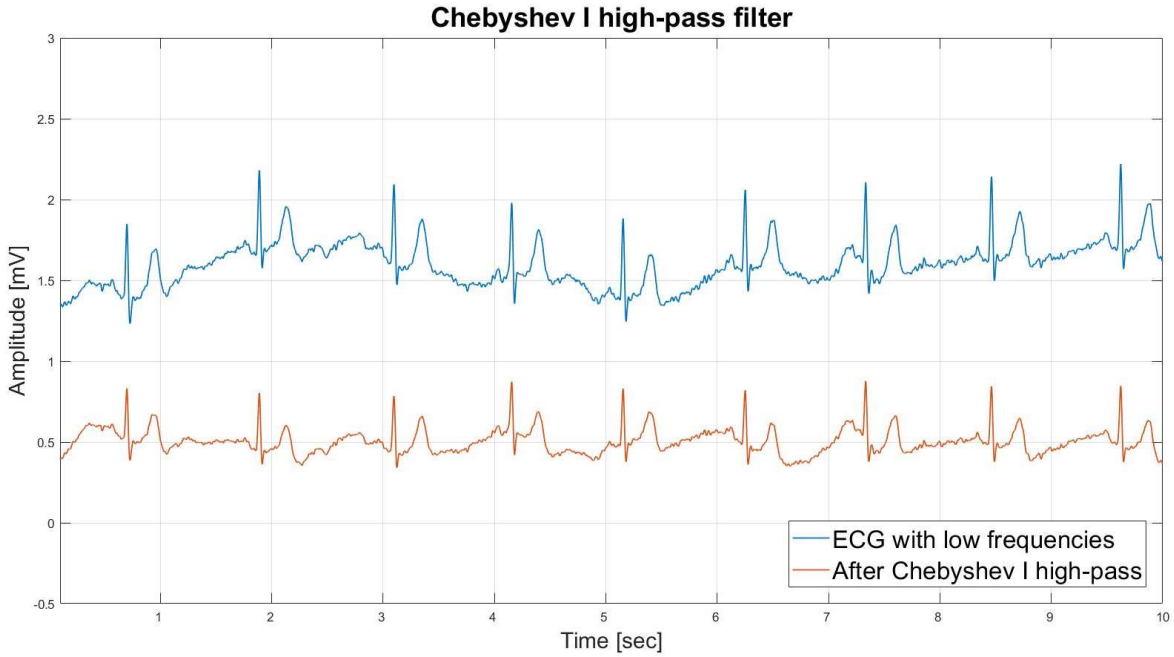

Figure 5.13: ECG signal after Chebyshev I 2nd order high-pass filter, with a 3dB ripple in the passband.

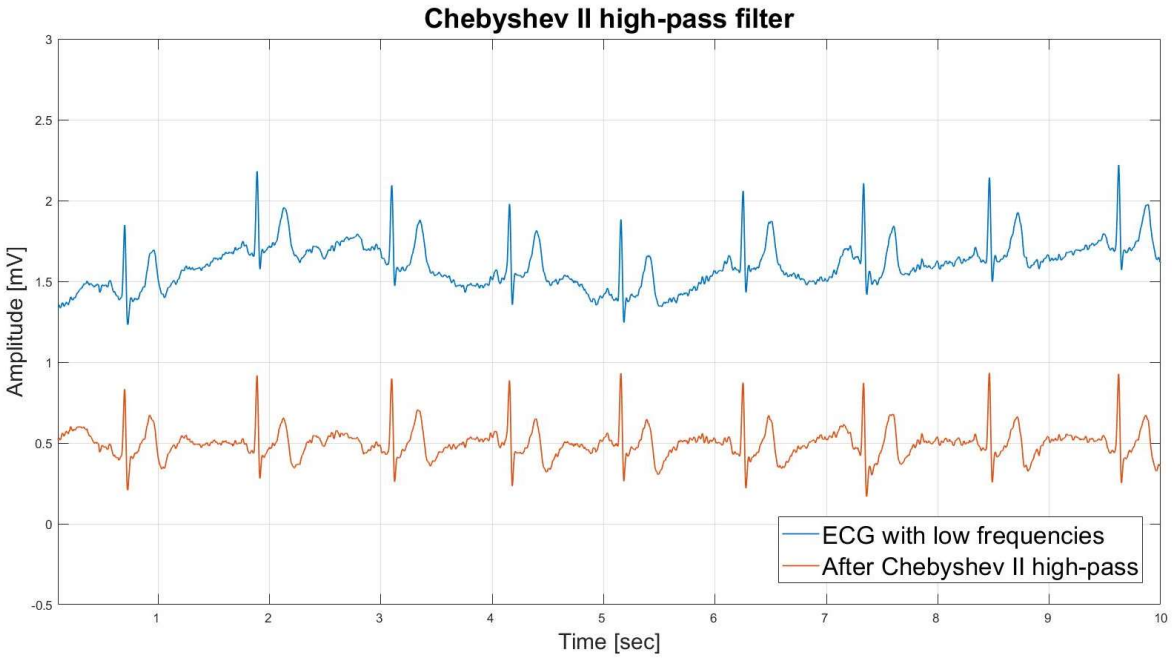

Figure 5.14: ECG signal after Chebyshev II 2nd order high-pass filter, with a -20dB ripple in the stopband.

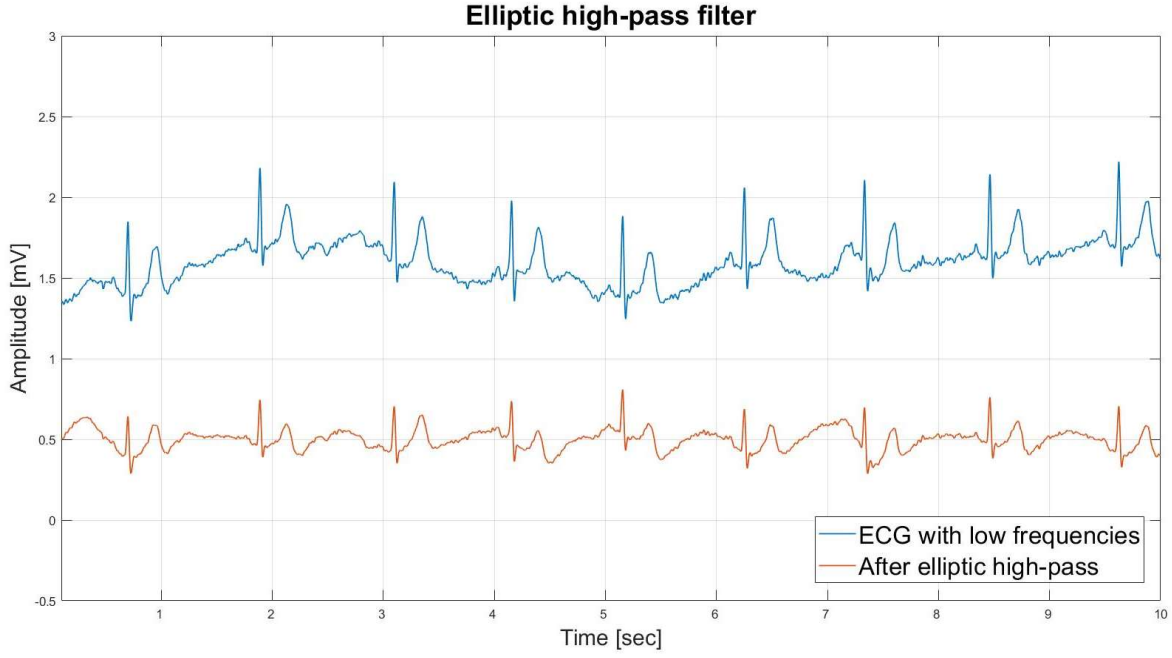

Figure 5.15: ECG signal after elliptic 6th order high-pass filter, with a 5dB ripple in the passband and a - 40dB ripple in the stopband.

Again, the best results are provided by the Butterworth filter, because of its flat response before the cutoff frequency (i.e. 0.5 Hz) and softly smoothed in the stopband. The main weakness of such filter is the inability to completely straighten the signal along the horizontal line. This could be achieved by increasing the filter order (therefore raising the slope of the response in the stopband and thus tightening the angle between passband and stopband), thereby affecting the quality of the resulting signal, as with the Chebyshev II filter. For this purpose, Butterworth filter clearly appears more efficient than any other IIR filter, whose response can at most get closer it by varying some parameters.
Finally, even for high-pass filtering two supplementary methods (Fig. 5.12 and Fig. 5.13) were evaluated to remove the eliminate the baseline drift, which consist in extracting the baseline itself through the moving average or median with a very wide window (e.g. 250 samples) and the subsequent elimination of this baseline from the original signal, by means of a simple subtraction (Listing 5.7).

Listing 5.7. Butterworth, Chebyshev I, Chebyshev II and elliptic high-pass filters configuration and implementation.

```
1. ecg_hp_mean_sub = movmean(ecg_lp_butter,250)
2. ecg_hp_med_sub = movmedian(ecg_lp_butter,250)
3.
4. ecg_hp_mean = ecg_lp_butter - ecg_hp_mean_sub
5. ecg_hp_med = ecg_lp_butter - ecg_hp_med_sub
```

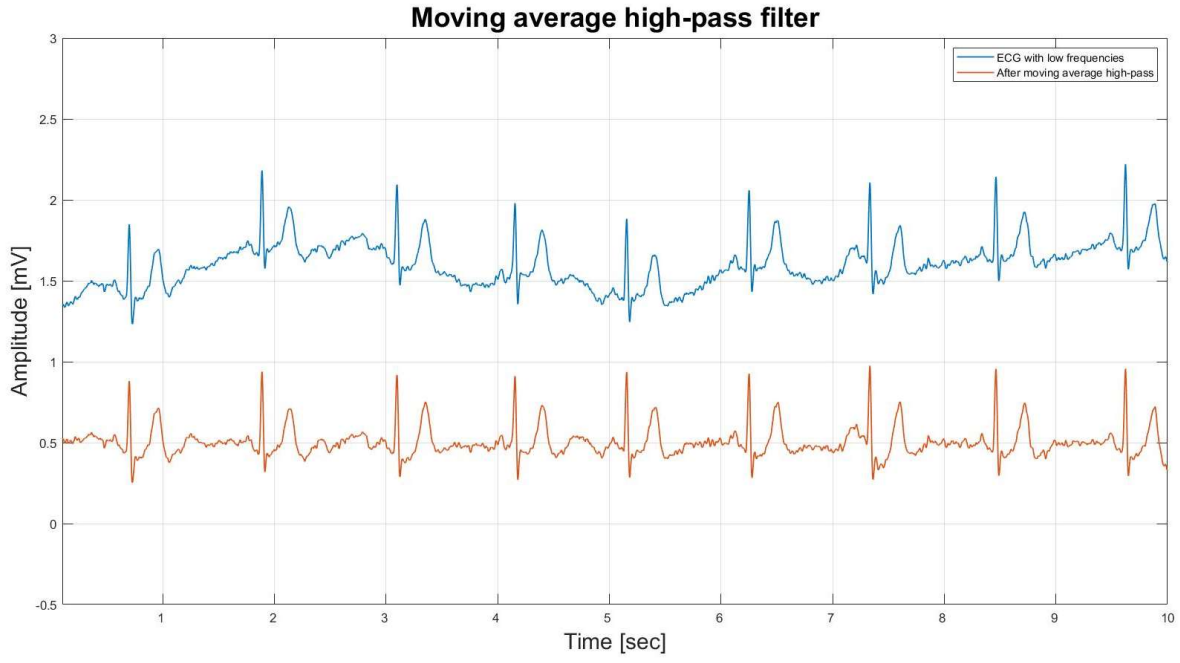

Figure 5.16: ECG signal after moving average high-pass filtering, with a window of 250 samples.

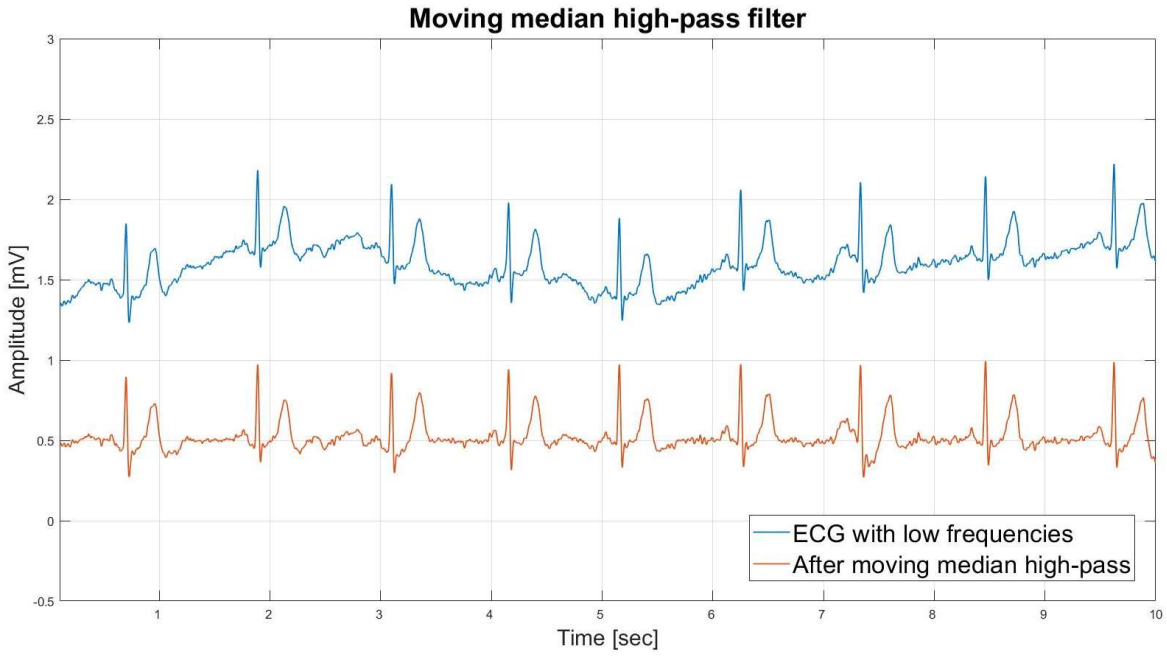

Figure 5.17: ECG signal after moving median high-pass filtering, with a window of 250 samples.

These two techniques are clearly very effective. Among all the high-pass filters, the moving median achieves the best outcome in perfectly straightening the signal without any alteration. At the same time, these techniques are difficult to implement out of the MATLAB environment because of their high computational cost and memory consumption.

5.2 Implementation in C language

Once some of the signal processing methods to be used for the ECG signal denoising were analysed, the study evolved into implementing those methods in C language. This phase can be split in two parts. Firstly, the mathematical functions were developed without caring of the data format nor the computational cost. Secondly, it was necessary to adapt such code to the CC2640R2F microprocessor features, which does not include a floating-point unit and whose memory is limited. Thus, a prior operation was to convert all data from the 16-bit *signed int* format to the fixed-point format, to properly elaborate the data almost with no loss of information.
After developing these algorithms, their correctness was tested using MALAB as serial terminal to acquire the signal and display it. Furthermore, a Bluetooth terminal was used, through the proper Android application. The results thus obtained were compared with the expected ones and the possibility of inserting them into the final device firmware was examined.

5.2.1 Fixed-point conversion

A decimal number can be transferred in binary in two different ways: fixed-point or floating-point. In the first case, the decimal value is derived from a significand (or mantissa) scaled using an exponent in base 2. Therefore, the MSB (most significant bit) is used to represent the sign, a fixed number of bits are used for the significand and the rest of the bits for the exponent. This is a more complex representation and so it is not supported by many microprocessors (such

as the CC2640R2F one). However, it allows to execute decimal operations in a faster and more optimized way, thus it is used by systems that require fast processing times. In the second case, by contrast, a decimal number is represented with a fixed number of digits after the radix point, which is the point which separates the decimal part (MSBs) from the fractional one (LSBs) [Wikipedia]. In this way, it is possible to perform operations between fractional values, by considering them as integers. The fixed-point representation is a good trade-off between performance and approximation.
Hence, before filtering the signal through the methods described above, it was required to convert all samples once they were read from the memory, in order to extend them to 32-bit (signed int type) and then to convert them to the fixed-point format, by shifting all bits by ten positions (thus obtaining a value with ten fractional digits). Additionally, the functions of fixed-point multiplication and division were defined, to be used in the proposed algorithms (Listing 5.8).

Listing 5.8. Butterworth, Chebyshev I, Chebyshev II and elliptic high-pass filters configuration and implementation.

```
1. #define FRACT_BITS              10
2. #define FIXED_POINT_ONE         (1 << FRACT_BITS)
3. #define MAKE_INT_FIXED(x)       ((x) << FRACT_BITS)
4. #define MAKE_FP_FIXED(x)        ((int_fixed_t)((x) * FIXED_POINT_ONE))
5. #define MAKE_FIXED_INT(x)       ((x) >> FRACT_BITS)
6. #define MAKE_FIXED_FP(x)        (((fp_t)(x)) / FIXED_POINT_ONE)
7.
8. #define FIXED_MULT(x, y)        ((int_fixed_t) ((((long_int_fixed_t)
   (x)) * ((long_int_fixed_t) (y))) >> FRACT_BITS))
9.
10.  #define FIXED_DIV(x, y)       ((int_fixed_t)((((long_int_fixed_t)
   (x)) << FRACT_BITS) / ((long_int_fixed_t) (y))))
```

5.2.2 IIR filters implementation

As previously evidenced, the IIR filter allows to perform many types of digital filtering, such as low-pass, high-pass, pass-band and stop-band. Based on the settings, it is possible to remove from a signal nearly all the unwanted frequency bands, in a soft or aggressive fashion, depending on the filter order. To implement such kind of filters it is necessary to build on its basic component, which is the *biquad filter* (Fig. 5.18). This component allows to derive every type of 1st and 2nd order IIR filter, depending on the values given to its matrix of coefficients, according to the equation (). The biquad filter was implemented in C language in its transposed direct form II, which foresees the use of only five multipliers, three adders and two registers (Listing 5.9).
Starting from this component, some of the abovementioned IIR filters were implemented. For instance, to build a 2nd order Butterworth low-pass IIR filter, it is sufficient to derive its coefficients (so called a and b) and use them to configure the filter. A similar operation can be performed for a high-pass or notch filter (Listing 5.10).

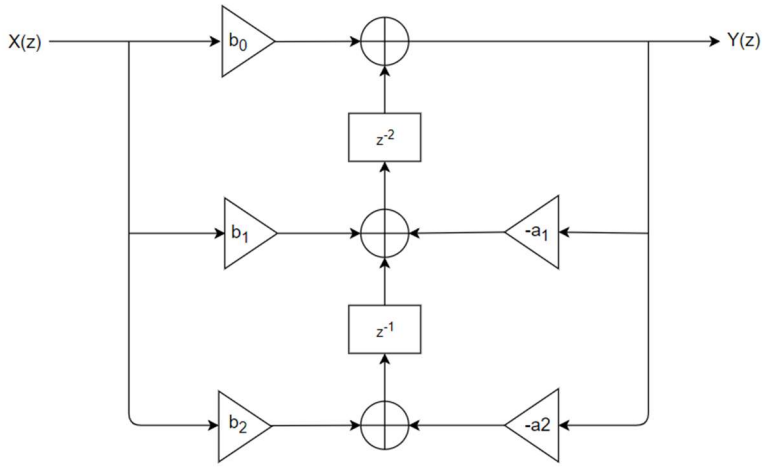

Figure 5.18: Biquad filter, transposed direct form 2.

Listing 5.9. Biquad filter implementation

```
1. void iir_biquad_fixed(iir_biquad_fixed_t * cell, int_fixed_t *
2.                       data_in, int_fixed_t * data_out, int size)
3. {
4.     unsigned int k;
5.     int_fixed_t input;
6.     long_int_fixed_t temp;
7.     for (k = 0; k < size; k++)
8.     {
9.         input = data_in[k];
10.
11.          temp = FIXED_MULT(cell->B[0], input) + cell->z[0];
12.          data_out[k] = (int_fixed_t) temp;
13.
14.          temp = FIXED_MULT(cell->B[1], input) + cell->z[1] -
15.                            FIXED_MULT(cell->A[0], data_out[k]);
16.          cell->z[0] = (int_fixed_t) temp;
17.
18.          temp = FIXED_MULT(cell->B[2], input)              -
19.                            FIXED_MULT(cell->A[1], data_out[k]);
20.          cell->z[1] = (int_fixed_t) temp;
21.      }
22.  }
```

Listing 5.10. IIR 0.5Hz high-pass, notch and 35Hz low-pass filters implementation.

```
1. /*---------biquadratic high-pass-----------------*/
2.
3. const float HPF_A[2] = {-0.9844, 0};
4. const float HPF_B[3] = {0.9922, -0.9922, 0};
5.
6. /*---------biquadratic notch 50Hz----------------*/
7.
8. const float NoF_A[2] = {-1.6399, 0.7252};
9. const float NoF_B[3] = {0.8626, -1.6399, 0.8626};
```

```
10.
11.  /*---------biquadratic low-pass Butterworth 35Hz-----------------*/
12.
13.  const float LPF_A[2] = {-1.3909, 0.5372};
14.  const float LPF_B[3] = {0.0366, 0.0731, 0.0366};
15.
16.  ECG_processing_t dsp;
17.
18.  dsp->biquads[0].A[0] = MAKE_FP_FIXED(HPF_A[0]);
19.  dsp->biquads[0].A[1] = MAKE_FP_FIXED(HPF_A[1]);
20.  dsp->biquads[0].B[0] = MAKE_FP_FIXED(HPF_B[0]);
21.  dsp->biquads[0].B[1] = MAKE_FP_FIXED(HPF_B[1]);
22.  dsp->biquads[0].B[2] = MAKE_FP_FIXED(HPF_B[2]);
23.  dsp->biquads[0].z[0] = 0;
24.  dsp->biquads[0].z[1] = 0;
25.
26.  dsp->biquads[1].A[0] = MAKE_FP_FIXED(NoF_A[0]);
27.  dsp->biquads[1].A[1] = MAKE_FP_FIXED(NoF_A[1]);
28.  dsp->biquads[1].B[0] = MAKE_FP_FIXED(NoF_B[0]);
29.  dsp->biquads[1].B[1] = MAKE_FP_FIXED(NoF_B[1]);
30.  dsp->biquads[1].B[2] = MAKE_FP_FIXED(NoF_B[2]);
31.  dsp->biquads[1].z[0] = 0;
32.  dsp->biquads[1].z[1] = 0;
33.
34.  dsp->biquads[2].A[0] = MAKE_FP_FIXED(LPF_A[0]);
35.  dsp->biquads[2].A[1] = MAKE_FP_FIXED(LPF_A[1]);
36.  dsp->biquads[2].B[0] = MAKE_FP_FIXED(LPF_B[0]);
37.  dsp->biquads[2].B[1] = MAKE_FP_FIXED(LPF_B[1]);
38.  dsp->biquads[2].B[2] = MAKE_FP_FIXED(LPF_B[2]);
39.  dsp->biquads[2].z[0] = 0;
40.  dsp->biquads[2].z[1] = 0;
```

5.2.3 Moving average and median

Turning to the moving average, on the other hand, a proper algorithm was developed to replace each sample with the mathematical average of the surrounding valued (Listing 5.11). In this respect, a variable was created to accumulate all the values of the window and this latter was updated for each new input sample. In so doing, every new sample is replaced by the value of the accumulation variable, divided by the window size. As a result, it is possible to implement moving averages with any window size, thus good to be used both for a low-pass and a high-pass filtering (i.e. to extract the baseline from the relevant signal).
Similarly, an algorithm wad implemented for the moving median, aimed to receive a data array and to replace each data with the middle value among the surrounding ones (Listing 5.12).

Listing 5.11. movmean() function implementation.

```
1. void movmean(int_fixed_t * data_in, int_fixed_t * data_out,
   int_fixed_t * mean_window_samples, int block_size, int window_size)
2. {
3.    int i,j;
4.
5.    int_fixed_t mean_acc;
6.
7.    for(i=0; i<block_size; ++i){
8.
9.       mean_acc = 0;
```

```
10.
11.         for(j=1; j<window_size; ++j){
12.
13.            mean_window_samples[j-1] = mean_window_samples[j];
14.         }
15.
16.         mean_window_samples[window_size-1] = data_in[i];
17.
18.         for(j=0; j<window_size; ++j){
19.
20.            mean_acc += mean_window_samples[j];
21.         }
22.
23.         data_out[i] = mean_acc/window_size;
24.      }
25.   }
```

Listing 5.12. movmedian() function implementation.

```
1. void movmedian(int_fixed_t * data_in, int_fixed_t * data_out,
2. int_fixed_t * median_window_samples, int block_size, int window_size)
3. {
4.    int i,j,sorting;
5.
6.    int_fixed_t sorting_buffer;
7.
8.    int_fixed_t median_buffer[20];
9.
10.    for(i=0; i<block_size; ++i){
11.
12.       for(j=1; j<window_size; ++j){
13.
14.           median_window_samples[j-1] = median_window_samples[j];
15.       }
16.
17.       median_window_samples[window_size-1] = data_in[i];
18.
19.       for(j=0; j<window_size; ++j){
20.
21.           median_buffer[j] = median_window_samples[j];
22.       }
23.
24.       for(j=0; j<window_size-1; ++j){
25.
26.          for(sorting=0; sorting<window_size-1; ++sorting){
27.
28.              if(median_buffer[sorting] > median_buffer[sorting+1]){
29.
30.                  sorting_buffer = median_buffer[sorting+1];
31.                  median_buffer[sorting+1] = median_buffer[sorting];
32.                  median_buffer[sorting] = sorting_buffer;
33.              }
34.          }
35.       }
36.
37.       data_out[i] = median_buffer[window_size/2];
38.    }
```

```
39.  }
```

5.3 Future scenarios

The presented study on the methods to denoise an ECG signal can lead the way to several possibilities in the context of signal processing. The combination of the techniques described above, together with those developed down the line, can increasingly improve the signal quality. For instance, a denoising test is shown in Fig. 5.18, obtained by combining a 35 Hz Butterworth low-pass filter and a moving average with a window of 10 samples for the high frequencies filtering, notch filer for the 50 Hz noise and a moving median with a window of 250 samples to eliminate the baseline drift.

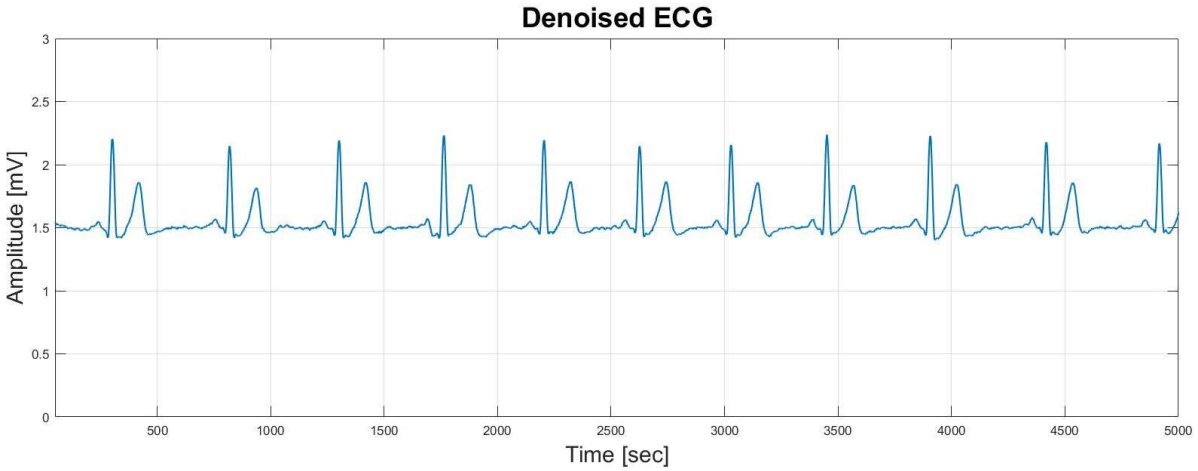

Figure 5.18: ECG signal after notch, low-pass and high-pass filtering.

Some of the implemented techniques, such as the moving average and median for the high-pass filtering, had trouble being included in the final device firmware because it was not possible to allocate in the RAM such a big buffer to collect all the 250 16-bit data of the window. In any case, the baseline drift removing was performed by the mobile application, in order to achieve the best outcome.
However, this project shows that it is possible to move the bulk of the ECG signal filtering to the digital domain, by means of a well-structured firmware. Moreover, all the signal elaboration and the mathematical computations have been performed in the fixed-point format. Such result allows not to be constrained by powerful and expensive microprocessors. The device thereby becomes much more flexible and open to further improvements which can be implemented without touching the hardware filters, which are instead much harder to design and implement. Hence, the foundations towards the final signal processing algorithm have been laid, in order to enhance the quality of the device and to allow it to be used as a real monitoring tool.

6. Mobile application development

The final phase of the project was devoted to developing a proper mobile application in JavaScript for an Android system, to let the user to interface the device via Bluetooth and visualize the detected ECG signal on the screen, upper appropriate adjustments. The full application development was handled to Vincenzo Randazzo, from the NeuronicaLabs.
Based on the previously implemented protocol stack, the app recognizes the device name and the related services and characteristics to start sharing commands and information. After pairing between device and app is enabled, this latter tries to discover the device by its name (i.e. ECG-CustDevice) among those available, in order to start the connection (Listing 6.1). All the services and characteristics of the profile are thus memorized among those previously used.
The communication begins after activating the *Payload* characteristic (described in the Chapter 4) and so enabling the *Notify* property, to be able to receive notifications. Then, after detecting that the starting button was pressed, the app sends the boot command to the device, through the *command* characteristic (Listing 6.2) and waits till the first data are received (Fig. 6.1).

Listing 6.1. Connection protocol implementation.

```
1. public boolean connect(final String address) {
2.   if (mBluetoothAdapter == null || address == null) {
3.     Log.w(TAG, "BluetoothAdapter not initialized or unspecified
4. address.");
5.     return false;
6.  }
7.
8. // Previously connected device.  Try to reconnect.
9.  if (mBluetoothDeviceAddress != null &&
10. address.equals(mBluetoothDeviceAddress)
11.       && mBluetoothGatt != null) {
12.
13.     Log.d(TAG, "Trying to use an existing mBluetoothGatt for
14. connection.");
15.
16.    if (mBluetoothGatt.connect()) {
17.        mConnectionState = STATE_CONNECTING;
18.        return true;
19.    } else {
20.        return false;
21.    }
22.}
23.    final BluetoothDevice device =
24. mBluetoothAdapter.getRemoteDevice(address);
25.    if (device == null) {
26.        Log.w(TAG, "Device not found.  Unable to connect.");
27.        return false;
28.    }
```

```
// We want to directly connect to the device, so we are setting the
autoConnect
// parameter to false.
mBluetoothGatt = device.connectGatt(this, false, mGattCallback);
Log.d(TAG, "Trying to create a new connection.");
mBluetoothDeviceAddress = address;
mConnectionState = STATE_CONNECTING;
return true;
}
```

Figure 6.2: App front page.

A collection of 5000 samples is thus performed (Listing 6.3), through 8-sample payloads (which are 625 payloads). The receipt of the data is controlled by means of the payload last byte, i.e. a code which informs the application on the transmission status (e.g. “0x03 for each properly received payload, “0xC0” for the last payload and “0x55” in case of an error occurs. Once collected all the payloads, these are decomposed in 16-bit signed integer values, in little endian format. The full signal is now possibly adjusted and plotted on the screen (Fig. 6.3). In addition, the app provides information on the patient heart rate, by applying the Pan-Tompkins Algorithm [2].
The 5000-sample array is finally stored in the internal memory of the mobile device, still available to be further visualized or sent to the pc.

Listing 6.2. Command transmission.

```
byte[] command = new byte[1];
command[0] = BleConstants.COMMAND_WRITE_START_ACQ;
mWriteCommandCharacteristic.setValue(command);
boolean result =
mBluetoothLeService.getBluetoothGatt().writeCharacteristic(mWriteCom
mandCharacteristic);
```

Listing 6.3. Data receiving.

```
1. private void broadcastUpdate(final String action,
2.                                final BluetoothGattCharacteristic
3. characteristic) {
4.    Intent intent = new Intent(action);
5.    double value;
6.
7.    byte[] payload = characteristic.getValue();
8.
9.    switch (characteristic.getUuid().toString()){
10.   case BleConstants.ECG_PAYLOAD_CHAR_UUID:
11.
12.      int i = 0;
13.      double ecgSample;
14.      this.seqNum = MainActivity.expectedSeqNum;
15.      int packetNumSeq =
16.characteristic.getIntValue(BluetoothGattCharacteristic.FORMAT_UINT1,
17.BleConstants.payloadSize);
18.      byte controlCode =
19.characteristic.getValue()[BleConstants.payloadSize  + 2];
20.      Log.i(TAG, "Packet number received: " + packetNumSeq + " , app
21.seq number: " + seqNum + " , final code =" + controlCode);
22.// Log.i(TAG, "Payload" + payload[15]);
23.
24.   if (controlCode == BleConstants.FIRSTPACK_CODE) {
25.      Log.i(TAG, "First packet received and skipped");
26.      MainActivity.expectedSeqNum++;
27.// broadcastUpdate(ACTION_DATA_AVAILABLE);
28.      return;
29.   }
30.   if (controlCode == BleConstants.NEXTPACK_CODE || controlCode ==
31.BleConstants.TERMINATION_CODE) {
32.      if (seqNum != packetNumSeq) {
33.         String msg = "Numero sequenza non valido, packetNumSeq = "
34.+ packetNumSeq + " , last expectedSeqNum = " + seqNum;
35.         Log.e(TAG, msg);
36.         intent = new Intent(ACTION_PACKET_SEQ_ERR);//todo add error
37.management
38.         intent.putExtra(EXTRA_UUID,
39.characteristic.getUuid().toString());
40.         intent.putExtra(EXTRA_DATA, msg);
41.         sendBroadcast(intent);
42.       } else {
43.            MainActivity.expectedSeqNum++;
44.
```

```
45.           while (i < BleConstants.payloadSize) {
46.
47.//ecgSample = (int)( payload[i] |payload[i + 1]<<8 );//Little endian
48.codification
49.               ecgSample =
50.characteristic.getIntValue(BluetoothGattCharacteristic.
51.FORMAT_SINT1, i);
52.               i += 2;
53.               MainActivity.mECG[MainActivity.offset++] = ecgSample;
54.//storeEcgSample(ecgSample);
55.             }
56.          }
57.          if (controlCode == BleConstants.TERMINATION_CODE ||
58.MainActivity.offset == MainActivity.noOfSamplesECG)
59.
60.             broadcastUpdate(ACTION_DATA_MEMORIZE_ECG);
61.       }
62.    default:
63.       value = characteristic.getValue()[0];
64.       break;
65.    }
66.
67.    /*  else {
68.          intent.putExtra(EXTRA_UUID,
69.          characteristic.getUuid().toString());
70.          intent.putExtra(EXTRA_DATA, value);
71.          sendBroadcast(intent);
72.       }*/
73. }
74.
```

Figure 6.3: ECG acquisition from app.

7. Project results and conclusions

The obtained results in the different sections of the project are summarized below. First, the improvements regarding the size of the device and the hardware and firmware choices are outlined. Moreover, a follow-up is provided on the quality of the detected and elaborated signal, compared with the ones obtained by the device in its previous version and by professional devices for monitoring and diagnostic purposes. It was not possible to perform an analysis on the device energy performances, since a final and completely assembled version yet does not exist. Nonetheless, considering prolonged use of the lithium polymer battery during the development phases, a several days self-sufficiency was assumed, in a way to execute hundreds of acquisition cycles.

7.1 Reduced board size

As abovementioned, a reduction of the device overall size was achieved thanks to a careful design of the PCB and an accurate component selection. These choices allowed to obtain a half-sized board compared to the previous one, in order to be inserted (together with a proper case) inside a common plastic wristband (Fig. 7.1). In this regard, small components were selected, without giving up on good performances. Additionally, a 4-layer board was designed, in a matter to manage the spaces at best. Lastly, a small sized lithium polymer battery was selected to provide 250mAh without taking too much space.

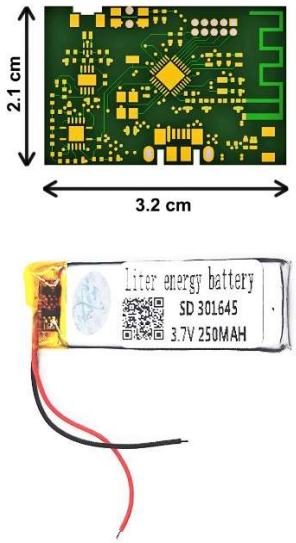

Figure 7.1: PCB outline and 3.7V 250mAh lithium polymer battery

In this stage of development, a great attention was also paid to the costs, to keep down the total expenditure of the device, if produced. The bill of the necessary materials to assemble one single board is thus provided in the Appendix.

7.2 Signal analysis

Finally, the signal quality is discussed, showing the ECGs collected from a heterogeneous group of volunteers by means of several devices aimed on patient monitoring. Thereby, the quality of the digital filtering, performed by the here presented device (version 2) can be examined, by comparing it with the ECG watch in its previous version (version 1), where all the signal processing is performed in the analog domain and through the proper mobile application.
A comparison between an ECG detected by the B105 Patient Monitor (provided by GE Healthcare) and the same ECG obtained from the device in its version 1 and version 2 is shown in the Fig. 7.2. In the version 1 the signal is processed by means of hardware filters and subsequently through the mobile app, which removes the baseline and the high-frequency noise. It is clear how much the analog filtering is responsible for the signal distortion by introducing undershoots, which are fake waveforms caused by a too aggressive high-pass filtering.

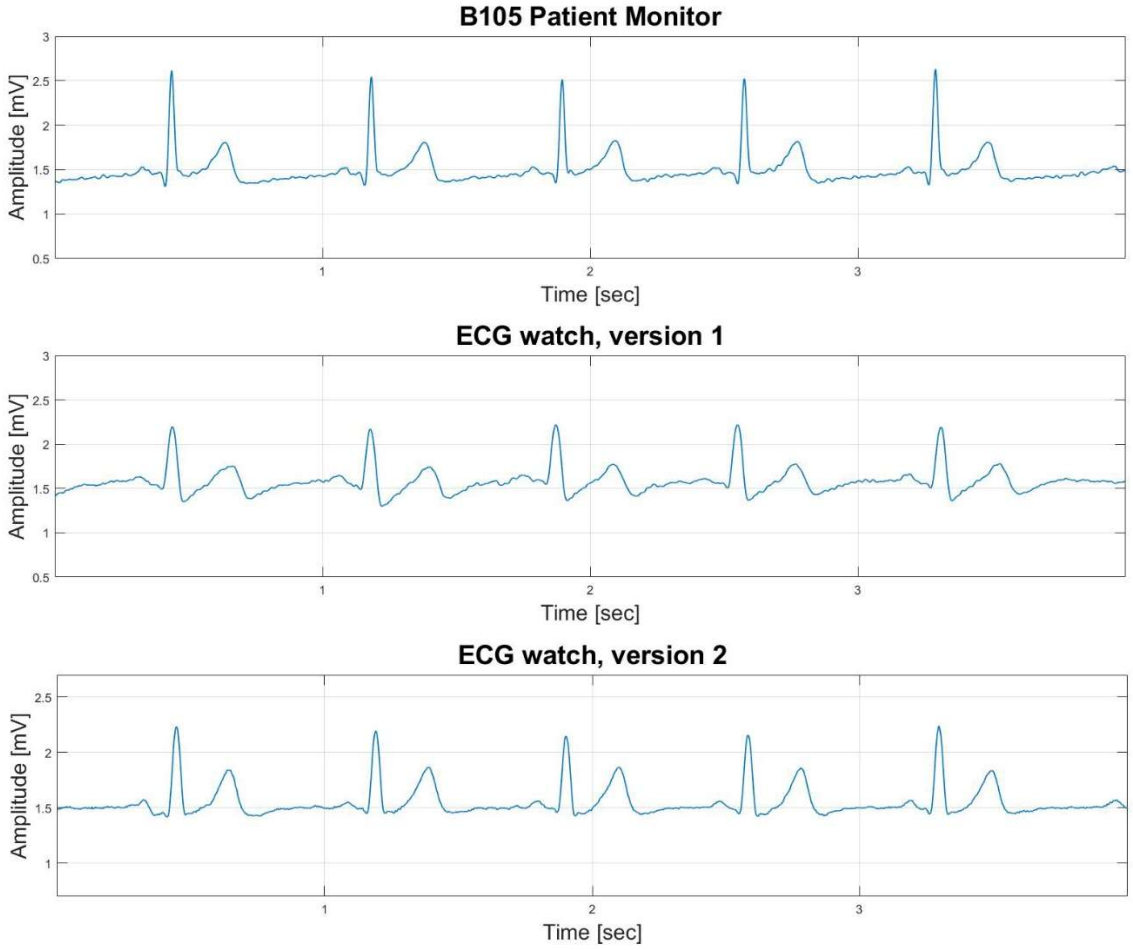

Figure 7.2: Comparison among the professional monitoring ECG and the version 1 and 2 of the ECG watch; male patient, 29 years old.

The signal detected by the ECG watch in its version 2 (that is the one presented in this thesis project) is instead processed in the digital domain and adjusted through the mobile app (which is responsible for the baseline drift removing). The signal appears almost uncontaminated, except for an attenuation of the steepest spikes caused by the low-pass filtering. It is, however,

possible to observe each of its significant parts (e.g. the P wave, the QRS complex and the T wave). The resulting signal, since it suffered almost no analog filtering, does not present any significant artefact.

Furthermore, two other ECGs concurrently detected by three different devices are reported in Fig. 7.4 and Fig. 7.5. The firsts were obtained from the B105 Patient Monitor, provided by GE Healthcare; the seconds from the ECG watch in its version 1; the thirds were detected by the device developed in this project, i.e. the version 2.

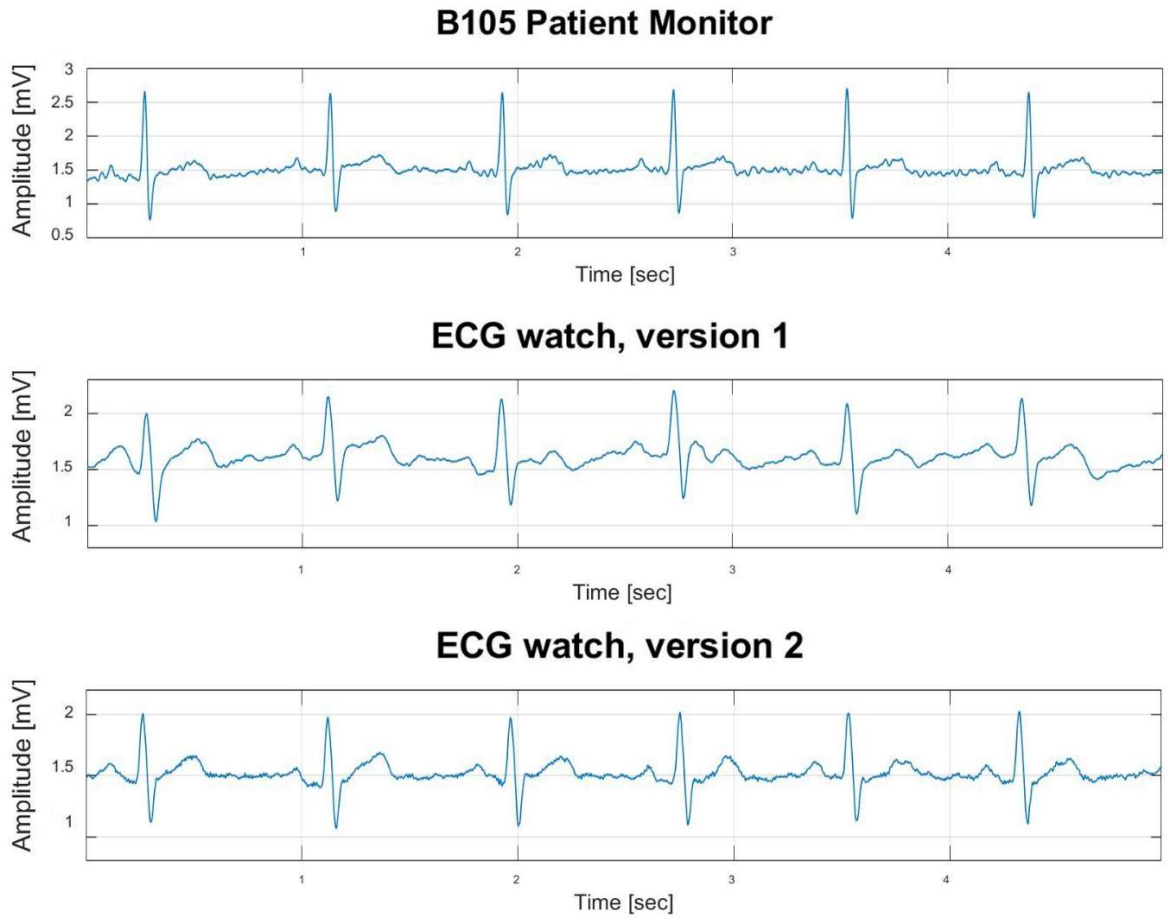

Figure 7.4: Comparison among the professional monitoring ECG and the version 1 and 2 of the ECG watch; male patient, 28 years old.

In this respect, it is recalled that the latter device processes the signal in the analog domain by means of a simple 1st order high-pass active filter (used to attenuate the DC component) and a 4th order low-pass active filter (to remove a part of the high-frequency disturbance). Additionally, the digital signal processing consists in an IIR notch, a moving average with a window of 10 samples for the high frequencies and a moving median with a window of 250 samples to eliminate the baseline drift.

Figure 7.5: Comparison among the professional monitoring ECG and the version 1 and 2 of the ECG watch; female patient, 24 years old.

From these measurements it can be observed how the here presented device fits within the standard used for health monitoring devices, providing a clean signal and non-contaminated by any type of artefact. The signal total amplitude results attenuated by 40-45 %, because of the overall low-pass filtering. However, such attenuation does not modify excessively the shape of the signal. Moreover, the baseline drift due to respiration results virtually absent. Lastly, a comparison between an ECG detected the version 2 of the device and an actual diagnostic ECG is presented (Fig. 7.6). Here again, it is possible to observe that the signal outline remains basically unchanged after filtering and all the cardiac information keep being clearly visible, despite the significant amplitude attenuation.

7.3 Concluding remarks

The project was closed with the achievement of the starting aims. The methods to implement a heart monitoring device were deepened in many aspects, from the hardware and PCB design to the firmware development, with the related digital signal processing. It was shown how to obtain satisfactory results, such as a small sized board to be inserted into a common wristband,

with reduced costs and a low power consumption. Last but not the least, the signal quality, similar to that of a professional heart monitoring system, could be exploited to identify the most common diseases, in the hope that this study will be further developed and improved in some respects, such as:

- to optimize the firmware as regards the data memory allocation;
- to implement a technique of lead-off detection, i.e. to recognize whether the patient's fingers are not well positioned on the electrode surface;
- to develop an algorithm to take advantage of the battery gauge, in order to provide information on the battery voltage level;
- to add an accelerometer and a gyroscope to the circuit, to detect an excess of movement by the patient.

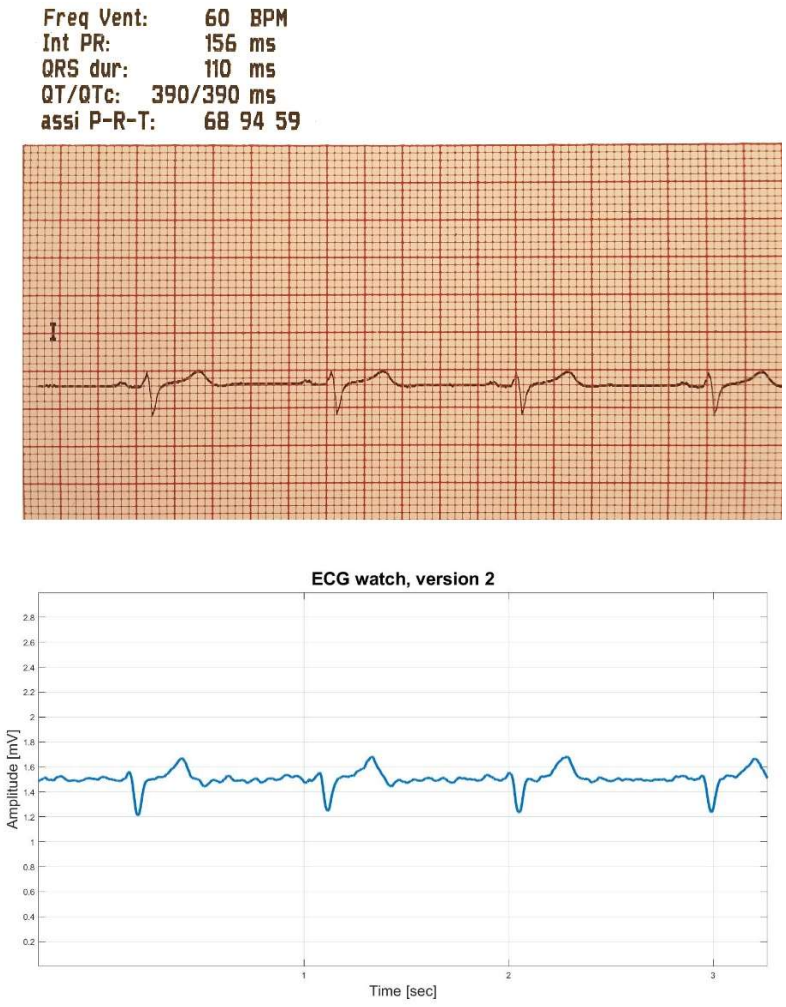

Figure 7.6: Comparison between a diagnostic ECG and the version 2 of the ECG watch; male patient, 27 years old.

Appendix

Bill of materials

Component	Qty	Manufactorer code	Unit cost [€]	Total Cost [€]
Connector (male)	2	02SUR-32S	0.45	0.9
Connector (female)	2	BM02B-SURS-TF(LF)(SN)	0.39	0.78
Capacitor SMD 1µF	4	CL05A105KP5NNNC	0.035	0.14
Capacitor SMD 330nF	1	CL05A334KP5NNNC	0.11	0.11
Capacitor SMD 10 µF	4	GRM155R60G106ME44D	0.19	0.76
Capacitor SMD 100nF	15	CL05F104ZO5NNNC	0.0089	0.1335
Capacitor SMD 33pF	2	CL05C330JB5NNNC	0.019	0.38
Capacitor SMD 330pF	1	GRM1555C1H331JA01D	0.09	0.09
Capacitor SMD 15nF	3	CL05B153KO5NNNC	0.023	0.069
Capacitor SMD 5.6nF	1	CL05B562KA5NNNC	0.019	0.019
Capacitor SMD 12nF	1	C0402C123K4RACTU	0.09	0.09
Capacitor SMD 1.2pF	2	GJM1555C1H1R2BB01D	0.113	0.226
Capacitor SMD 12pF	3	CL05C120JB5NNNC	0.022	0.066
Ferrite bead SMD 1.5kΩ	1	BLM18HE152SN1D	0.18	0.18
Cortex Debug Connector	1	FTSH-105-01-L-D-K	2.64	2.64
RED LED	1	LTST-C193KRKT-5A	0.36	0.36
GREEN LED	1	LTST-C193KGKT-5A	0.36	0.36
Inductor SMD 10nH	1	LQG15HS10NJ02D	0.09	0.09
Inductor SMD 15nH	1	LQG15HS15NJ02D	0.09	0.09
Inductor SMD 2nH	1	LQG15HS2N0S02D	0.053	0.053
Resistor SMD 100kΩ	3	ERA-2AED104X	0.19	0.57
Resistor SMD 24.9kΩ	1	ERA-2AEB2492X	0.281	0.281
Resistor SMD 0 Ω	2	RMCF0402ZT0R00	0.01	0.02
Resistor SMD 20MΩ	1	RC0402JR-0720ML	0.09	0.09
Resistor SMD 121kΩ	1	RT0402BRD07121KL	0.34	0.34
Resistor SMD 57.6kΩ	1	RC0402FR-0757K6L	0.564	0.564
Resistor SMD 130kΩ	1	CPF0402B130KE1	0.44	0.44
Resistor SMD 90.9kΩ	1	ERA-2AEB9092X	0.39	0.39
Resistor SMD 1kΩ	1	ERA-2AED102X	0.19	0.19
Resistor SMD 140kΩ	1	RC0402FR-07140KL	0.014	0.014
Resistor SMD 33kΩ	1	ERA-2AEB333X	0.39	0.39
Resistor SMD 137 Ω	2	RC0402FR-07137RL	0.014	0.028
Resistor SMD 3.3kΩ	1	ERA-2AEB332X	0.39	0.39
Resistor SMD 10kΩ	2	ERA-2AED103X	0.164	0.328
Test point	1	RCU-0C	0.185	0.185

USB connector	1	MOLEX 1051640001	0.8	0.8
CC2640R2F microcontroller	1	CC2640R2FRSMR	4.08	4.08
MAX1555 battery charger	1	MAX1555EZK+T	1.99	1.99
MAX1759 voltage regulator	1	MAX1759EUB+	6.26	6.26
INA333 instrumentation ampl.	1	INA333AIDRGR	3.78	3.78
OPA4330 operational ampl.	1	OPA4330AIRGYT	3.51	3.51
Side Push Button	1	EVQ-P7C01P	0.31	0.31
REF2033 ref. voltage provider	1	REF2033AIDDCR	2.94	2.94
DVIULC6-2x6 ESD protection	1	DVIULC6-2M6	1.11	1.11
MAX17048 battery gauge	1	MAX17048G+T10	2.55	2.55
32.768 kHz crystal oscillator	1	SC20S-12.5PF20PPM	0.65	0.65
24 MHz crustal oscillator	1	TSX-3225 24.0000MF15X-AC3	0.31	0.31
TOTAL				38.1265

Bibliography

[1] F. Lacirignola and E. Pasero, “Hardware design of a wearable ECG-sensor: Strategies implementation for improving CMRR and reducing noise,” 2017.

[2] J. Pan and W. J. Tompkins, “A Real-Time QRS Detection Algorithm,” *IEEE Transactions on Biomedical Engineering,* Vols. BME-32, no. 3, pp. 230-236, 3 March 1985.

[3] V. Acharya, *Improving Common-Mode Rejection Using the Right-Leg Drive Amplifier,* Texas Instruments, 2011.

[4] S. Braida, *Risposta in frequenza di un elettrocardiografo a scopo diagnostico,* A. i. i. clinici, Ed., 2014.

[5] STMICROELECTRONICS, *Ultra low capacitance ESD protection,* 2015.

[6] Texas Instruments, Incorporated, *Micro-Power (50μA), Zerø-Drift, Rail-to-Rail Out Instrumentation Amplifier datasheet (Rev. C),* 2017.

[7] Texas Instruments, Incorporated, *50-μV VOS, 0.25-μV/°C, 35-μA CMOS Operational Amplifiers Zero-Drift Series datasheet (Rev. G),* 2019.

[8] Texas Instruments, Incorporated, *Active Low-Pass Filter Design (Rev. B),* 2002.

[9] Texas Instruments, Incorporated, *Chapter 16 - Active Filter Design Techniques,* 2008.