POLITECNICO DI TORINO

Master degree course
in Electronic Engineering

Master Thesis

Development of a wearable device for the acquisition of a six-derivations ECG for self-monitoring of cardiovascular diseases.

Supervisor
Prof. Eros Gian Alessandro PASERO
Co-supervisor
Eng. Vincenzo RANDAZZO, Ph.D

Candidates
Marco SENTO
Gianmarco DOGLIANI

Academic year 2022-2023

"The beauty and the scent of roses can be used as a medicine and the sun rays as a food."
— Nikola Tesla

Contents

Abstract

Developed over a century ago, the electrocardiogram provides invaluable information about the state of health of the heart, aiding the diagnosis and management of various cardiovascular conditions.
Cardiovascular diseases have significant impact on health, quality of life and economics, being the major cause of death and disability worldwide. In Europe, CVDs are the leading cause of death, accounting for 32% of all deceases, with an estimated cost of €210 billion annually.
Addressing CVDs through monitoring and prevention is crucial to improve health expectancy and reduce the economic burden of these diseases.

It is in this scenario that smart wearable devices fit. In fact, they let self-monitoring of heart rate and activity. These information, together with other vital parameters, provide feedback to cardiologists, allowing them to remotely monitor patients, eventually detecting early warning signs of CVDs, and to intervene with cures before complications might occur.
There are currently several devices that allow self monitoring of heart activity, but they are not designed to provide a comprehensive diagnosis of cardiac conditions. They commonly provide just numerical information about heart activity, and eventually report a single-lead ECG.

The aim of this thesis is thus to study and to realise an innovative wearable device capable of overcoming these limitations acquiring a six-leads medical ECG in a non invasive way.

While single-lead ECGs offer limited diagnostic utility as the electrical activity of the heart comes from one direction only, a six-leads ECG records information from six different electrodes placed on specific locations of the chest and of the limbs. This way, it allows diagnosis of cardiac conditions such as arrhythmia, ischaemia and heart blocks.
The device conceived in this thesis project uses an ultra-compact design with just three electrodes, leveraging on the mathematical relationship between the six leads.

Going into the details of the signal elaboration system. The device simultaneously acquires two differential signals from the input electrodes.
Common mode noise is reduced using an instrumentation amplifier and different

analog filters have been tested to reduce other sources of noise. Signals are amplified and sampled with a high resolution 24-bit ADC.
Acquired data are sent to a microcontroller and transmitted via Bluetooth to a smartphone application which displays acquisitions in real time.
Furthermore, elaboration includes processing of the signals through digital Finite Impulse Response filters.

Practically, the development of the thesis project has gone through the following steps:

- Pen and paper design of the different functional blocks of the circuit.
- Schematic diagram and PCB layout realization using Cadence OrCAD.
- Firmware development.
- Implementation of DSP filtering techniques.

The combination of digital and analog filtering of the acquired signals made it possible to create a device capable of acquiring a high quality six-leads electrocardiogram maintaining an extremely compact design.
By offering an accessible and user-friendly solution, this device potentially simplifies and changes the current conception of ECG monitoring, improving prevention strategies, diagnosis, and treatments of cardiovascular diseases.

Chapter 1

Introduction

1.1 Cardiovascular diseases nowadays

Despite the improvements in medical research, technology, and surgical interventions, cardiovascular diseases remain a significant global health concern causing a substantial burden on individuals, communities, and healthcare systems.
Especially in recent decades, the prevalence of cardiovascular diseases has raised. CVDs have become the leading cause of mortality worldwide, mainly due to changes in people's lifestyle and to an increasingly elder population.
According to the World Health Organization, CVDs are responsible for approximately 17.7 million deaths each year, accounting for 37% of all premature deaths. Ischemic heart diseases, strokes, and heart failures are the most common cardiovascular conditions [20].
Several modifiable and non-modifiable risk factors contribute to the development of cardiovascular diseases. Non-modifiable risk factors include age, gender, and genetics, while modifiable risk factors encompass unhealthy diets, physical inactivity, tobacco use, obesity, hypertension, diabetes and high cholesterol levels.

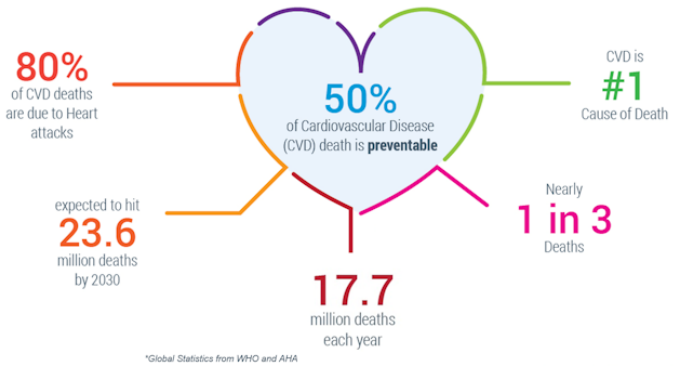


Figure 1.1: Infographic about CVDs.

Early and accurate diagnosis of cardiovascular diseases is essential for effective prevention and for the management of potential complications. Traditional diagnostic methods such as electrocardiography (ECG), echocardiography and stress tests continue to play a fundamental role. In addition, technological advancements have introduced innovative diagnostic features for cardiac imaging processing, just as computed tomography and magnetic resonance. And wearable devices, together with smartphone apps, have gained popularity as well.
Clearly, the management of cardiovascular diseases includes several aspects spanning from lifestyle modifications to pharmacological aid and invasive procedures. In any case, lifestyle behaviours including a heart-healthy diet, regular exercise, smoking cessation and stress reduction are essential to prevent and manage CVDs.

1.2 Prevention and technology

Technological development continues to revolutionize cardiovascular medicine.
Wearable devices, such as smartwatches and fitness trackers, provide real-time monitoring of heart rate, activity, intensity and other data, enabling early detection of abnormalities.
Telemedicine and remote patient monitoring have gained popularity, facilitating access to specialized cares and improving patient outcomes in rural or underserved areas.
Despite these progresses, several challenges persist. Access to quality healthcare, especially in remote areas, remains a concern. Additionally, the rising prevalence of risk factors such as obesity and aging, focuses the attention on disease prevention. Identifying and addressing risk factors early on, the incidence and severity of CVDs can be significantly reduced.

Prevention includes both population-wide strategies and individual-level actions. Public health campaigns and education aim to promote healthier lifestyles, while personalized risk assessments and targeted interventions provide tailored approaches to higher-risk individuals.

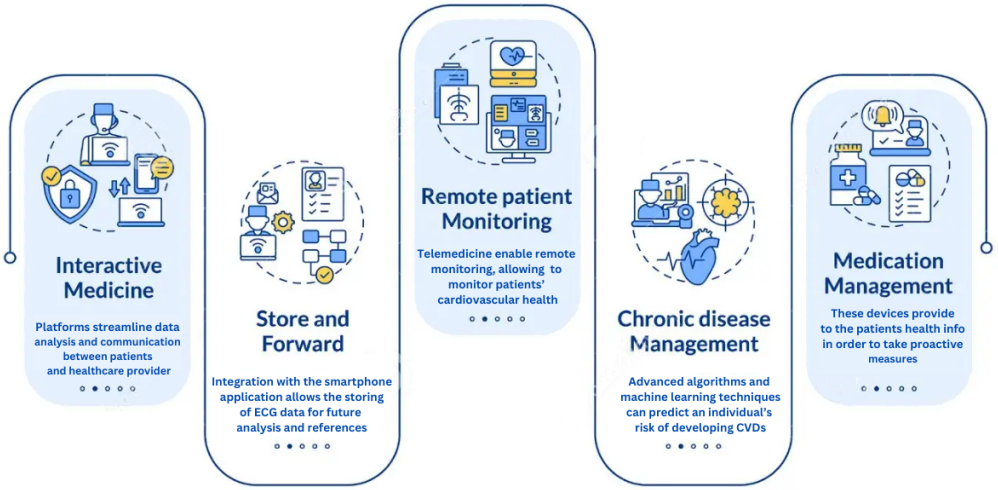


Figure 1.2: Infographic about telemedicine.

There are many fields in which technology is having an enormous impact, profoundly affecting the treatment for cardiovascular diseases:

a. **Risk Prevention**. Algorithms and machine learning techniques can analyze large amounts of data just as genetic information, medical history, and lifestyle factors, to predict an individual's risk of developing CVDs. This customized risk assessment enables early interventions and targeted prevention strategies.

b. **Wearable Devices**. Smartwatches, fitness trackers, and other wearable devices allow continuous monitoring of heart rate activity and intensity, sleep patterns, and other valuable data. These devices provide insights into an individual's cardiovascular health, encouraging informed decisions and proactive measures.

c. **Telemedicine and Remote Monitoring**. Telemedicine platforms allow remote monitoring, letting healthcare providers the monitor of a patient's health, and providing guidance for adjusting treatment plans without the

need for in-person visits. Remote patient monitoring further enhances this approach by continuously collecting and transmitting vital parameters and other relevant data, ensuring timely interventions and reducing the risk of complications [13].

d. **Mobile Applications**. Smartphone applications offer various functionalities, such as exercise tracking, medication reminders, and disease management insights. These apps suggest lifestyle modifications, facilitate self-monitoring and provide educational resources promoting cardiovascular health.

e. **Digital Health Platforms**. Integrated digital health platforms merge together various technologies, including wearable devices, mobile apps, and electronic health records. These platforms streamline data collection, analysis, and communication between patients and healthcare providers, facilitating customized prevention plans and enhancing care coordination.

In such a scenario the wearable device studied in this thesis project fits well. It allows instantaneous monitoring of the health status of a subject's heart through the real-time acquisition of six cardiac signals. Acquired data are managed by a smartphone application so that the subject's state of health can be immediately visualized and shared with doctors, paving the way for a new method of telemedicine.

The development of the product has been divided into different phases, each of which involved a specific set of operations and choices. Below, a list of the steps taken during the device development:

1. Definition of project name and scope.
2. Definition of project specifications.
3. Market analysis of potential competitors.
4. Signal conditioning design.
5. Digital section design.
6. Selection of components.
7. Schematic diagram realization with OrCAD Capture.
8. BOM-Based evaluation of costs.
9. PCB design using OrCAD PCB Editor.
10. Development of firmware for the STM32WB55RG evaluation board.
11. Processing of the signals acquired with the evaluation board.
12. Prototype development.
13. Electrical test of interconnections.
14. Mock-up development.
15. Communication with the Android App.

Chapter 2

Electrocardiography: an overview

Electrocardiography has a long history that begins two centuries ago. It was at the end of the 19^{th} century that the first studies concerning the electrical activity of the heart were conducted. In 1887, the British physiologist Augustus Waller recorded the first human electrocardiogram using a capillary electrometer[1]. He observed that the heartbeat produced an electrical signal measurable as a potential difference on the skin.
Those results were approximate, but the development of new techniques increased considerably, and only fourteen years later, in 1901, the Dutch physiologist Willem Einthoven developed an ECG machine which produced extremely accurate results. Einthoven's machine was based on a galvanometer capable of amplifying electrical signals and returning them on paper. This new technology allowed the clinical study of the electrical activity of the heart, and brought a new method to classify heart diseases such as arrhythmia and heart blocks.
Einthoven also worked on a solution to standardize the electrocardiogram, and realized what has then become the universally adopted ECG representation.
Throughout the following decades, the electrocardiogram has been further developed and improved. Still today, it is an essential tool for identifying and studying heart diseases.

[1] A capillary electrometer uses changes in the surface tension of an electrolyte inside a capillary to detect small variations of electric potential.

2.1 The ECG signal

The electrocardiogram is a graphical representation of the heart's electrical activity over time. It represents variations of electrical potential difference between two points of the human body.
The ECG waveform consists of various segments, each of which represents a different aspect of the heart's electrical activity. The main segments of an ECG are:

- The P wave, which represents depolarization of the atria.
  Typical duration $< 80ms$.
- The QRS complex, which represents depolarization of the ventricles.
  Typical duration $80ms - 100ms$.
- The T wave, which represents repolarization of the ventricles.
  Typical duration $\approx 160ms$.
- The U wave, which represents papillary muscle repolarization.
  Typical duration $< 60ms$.

Note that the U wave is hard to be appreciated and it is typically ignored.

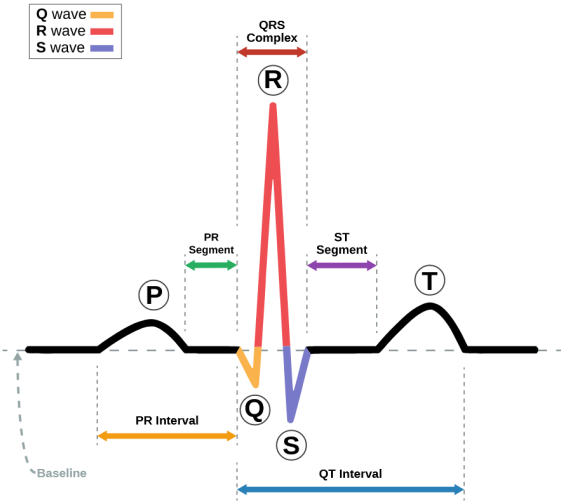
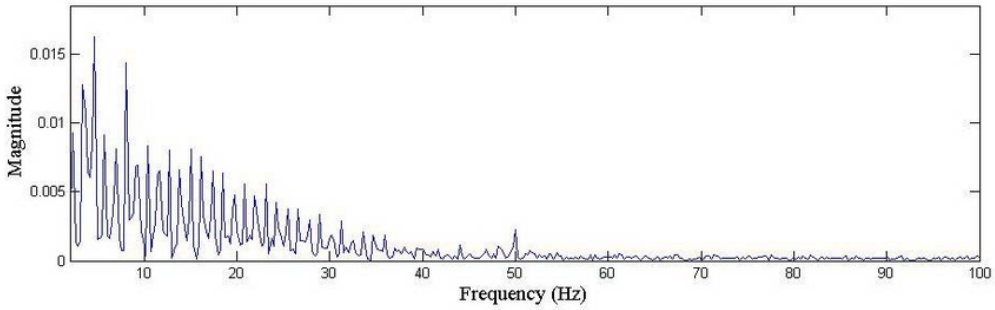


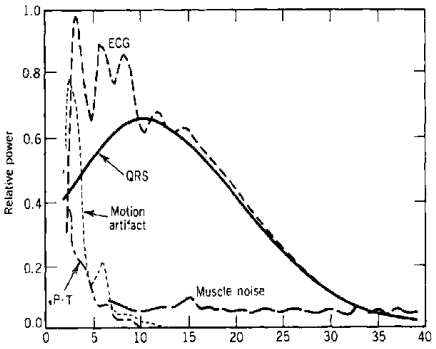
Figure 2.1: ECG waveform.

In addition to the segments mentioned above, there might be other contributions symptomatic of abnormal activity of the heart.

To better appreciate the information carried out by the ECG signal, it is useful to exploit the Fourier transform. Such transform transposes the time-domain signal into its frequency-domain counterpart. This way, it is possible to determine the extension of the frequency spectrum of the ECG signal.
Typically, the frequency range of the ECG signal goes from 0.05Hz up to 100Hz, with most of the energy concentrated in the range from 0.5Hz to 40Hz.
The lower-frequency contributions, below 0.5Hz, are associated with respiratory or other physiological movements, while the higher-frequency contributions, above 40Hz, are mostly noise and interference [30].



(a) Full frequency range[1].



(b) Useful frequency range.

Figure 2.2: Frequency spectrum of the ECG signal.

[1]The spike at 50Hz is an example of powerline interference.

From a practical point of view, the ECG signal is displayed on a grid, with time along the horizontal axis and voltage on the vertical axis.
The standard time reference for this grid is 1 second every 25 millimeters, that is $25\frac{mm}{s}$. The grid is interpreted as follows:

- A small box of $1mm \times 1mm$ represents $0.1mV \times 0.04s$.
- A large box of $5mm \times 5mm$ represents $0.5mV \times 0.20s$.

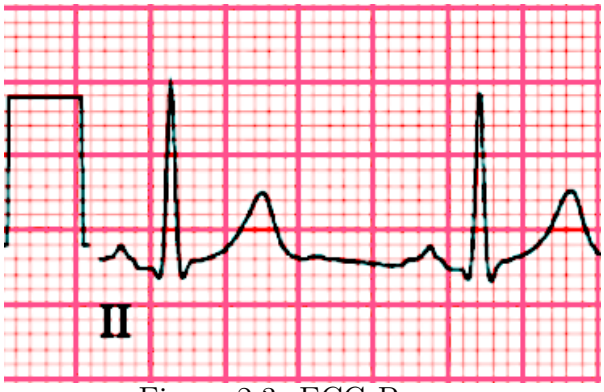


Figure 2.3: ECG Paper.

Interpreting the information with the right scale is crucial to identify abnormal changes in the cardiac rhythm or other conditions such as ventricular hypertrophy. However, multiple conditions can be identified by just observing the signal profile. Pathological variations in the signal profile might show an absence of the P wave, a sinusoidal signal pattern, or even a "saw tooth" profile.

The classical representation of the ECG shows the variation of electric potential between right and left arms, but different waveforms can be appreciated depending on the angle of observation of the cardiac activity. These different waveforms are obtained by simply placing the electrodes on different points of the body.
Notice that the complete electrocardiogram analysis makes use of ten electrodes and is known as 12-leads ECG. In this setup, the subject lies supine, and the ten adhesive electrodes are placed on his chest and on his limbs. The changes in the heart's electrical potential are thus measured from twelve different angles, and recorded for a certain period of time, usually ten seconds. In this way, a complete overview of the heart's activity is captured during each phase of the cardiac cycle.

2.1.1 Limb & Precordial Leads

A difference in the electrical potential between two points of the human body is called derivation, or lead in the medical field. Clearly, using 2 electrodes only, a single derivation can be obtained. And a single waveform can be displayed.
As anticipated in the previous section, combining together 10 electrodes it is possible to obtain 12 different derivations. These derivations are divided into two groups: limb leads and precordial leads.
The main difference between this two categories stands in the angle from which the heart is inspected. Supposing to divide the human body into two planes, as shown in fig. 2.4, limb leads observe cardiac activity in the vertical (or frontal) plane, while precordial leads, also known as chest leads as they are located anteriorly to the chest, detect signals traveling in the horizontal (or transverse) plane. Notice that the horizontal plane is perpendicular to the vertical one [5].

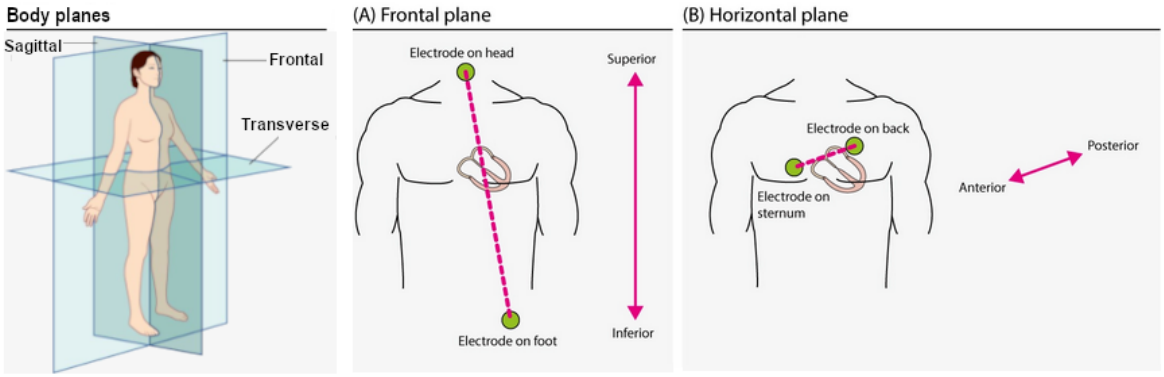


Figure 2.4: Cardiac planes.

While the limb leads provide a general overview of the cardiac condition, the precordial leads are particularly useful for assessing the status of the left ventricle, which is in charge of pumping oxygenated blood to the body.
Moreover, precordial leads can be used to detect changes in the ST segment and in the T wave, which are indicative of a variety of cardiac conditions including myocardial infarction, left ventricular hypertrophy and cardiomyopathies.
However, considering again the limb leads, it is necessary to make a further distinction between Einthoven's and Goldberger's leads.

2.1.2 Einthoven and Goldberger's leads

The limb leads are made up of six different signals located in the frontal plane, namely I, II, III, aVF, aVR and aVL.

The first three leads, I, II and III, are Einthoven's original leads, and can be displayed using the Einthoven's triangle. The remaining leads, aVR, aVL and aVF, introduced by the American cardiologist Lewis Goldberger, are called unipolar leads, as they refer to the average value between two electrodes.
With respect to the unipolar leads, the letter "a" stands for augmented, while "V" stands for voltage. "R" stands for right arm and "L" for left arm. Instead, "F" stands for foot. Notice that usually, lead aVR is converted into lead -aVR, as this conversion may facilitate the interpretation.

Leads I, II and III compare electrical potential differences between two electrodes.
Lead I compares the electrode on the left arm with the electrode on the right arm, where the former, observing from 0°, is at a lower potential.
Lead II compares the left leg with the right arm, with the leg electrode being at a higher potential. Lead II observes the heart from an angle of 60°.
Lead III compares the left leg with the left arm, with the leg electrode being the exploring one. It observes the heart from an angle of 120°.
The spatial organization of these leads forms a triangle known as Einthoven's triangle (fig. 2.5).

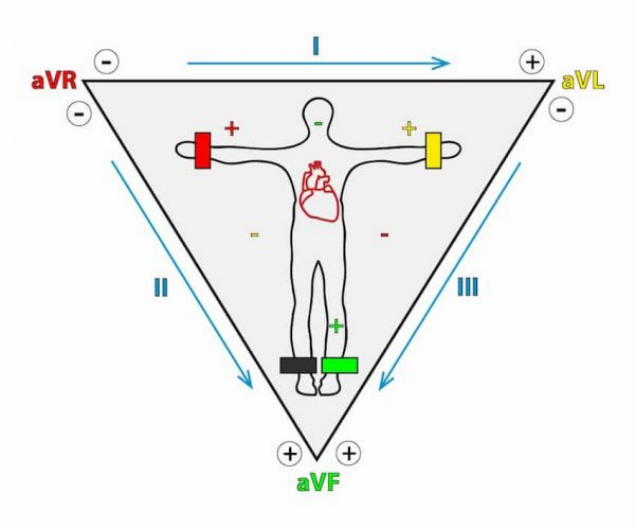
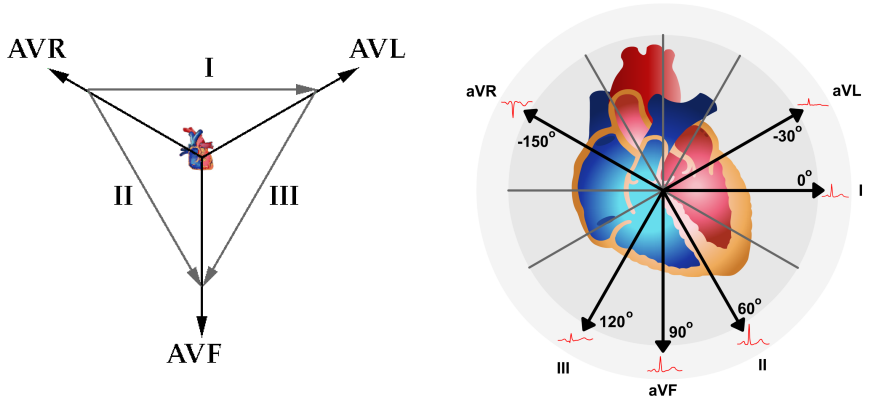


Figure 2.5: Einthoven's triangle.

To better understand the relationships between Einthoven's and Goldberger's leads, the six limb leads are usually represented in a circular Hexaxial coordinate system, as shown in figure 2.6b.
In this coordinate system, it is assumed that Lead I defines 0° in the frontal plane. This also means that lead I "views" the heart from an angle of 0°. Accordingly, leads II and III observe the heart from 60° and 120° respectively.
However, it should be noted that, in this model, the reference is provided by the Goldberger's central terminal, conceptually located at the center of the heart.
Looking at the representation of fig.2.6a, it is possible to interpret the Goldberger's leads. They are vectors starting from the barycentre of Einthoven's triangle. Conceptually, they are bisectors of its angles, even if with a greater amplitude. They divide the angles of the Einthoven's triangle into equal parts, splitting the frontal plane into areas of 30° each [4].
At this point, it is evident that there must be a relationship between the different limb leads.



(a) Limb vectors. (b) Hexaxial coordinate system.

Figure 2.6: Circular Hexaxial coordinate system representation.

2.2 Implementation of the theory

The goal of the thesis project is to realize a portable device able to get the 6 limb leads out of three electrodes. Therefore, this section analyzes the mathematical relationship between the six limb leads. Instead, the study of the precordial leads is out of the scope of the project.
As anticipated, using two electrodes only, it is possible to obtain a single derivation.

However, exploiting the mathematical relationship between the limbic signals, all limb leads can be derived using just three electrodes placed on the right arm, the left arm and the left leg.
The Einthoven's leads (leads I, II and III) are derived using only two derivations. The remaining 3 leads use a reference given by the average of the other three derivations, which is called the Goldberger's central terminal.
With reference to the Einthoven's triangle (fig.2.5), it is possible to define two vectors, derivations I and II, measuring the potential difference between the electrode placed on the right hand (RA), used as reference, and the electrodes of the left hand (LA) and the left leg (LL).

As the electrocardiogram measures variations of the electrical signals as they propagate in the human body, the Einthoven's triangle can be viewed as an electrical circuit.
According to Kirchhoff's current law, the sum of all currents in a closed circuit must be zero. Thus:

$$I + III = II \tag{2.1}$$

This means that the sum of the vectors I and III equals the potential difference of lead II. In clinical electrocardiography, this means that the amplitude of, for example, the R-wave in lead II, is equal to the sum of the R-wave amplitudes in leads I and III.
It follows that we only need the information of two leads in order to obtain the remaining one, and thus there is no need to directly measure the potential difference between LA and LL.

For what concerns Godlberger's leads, it is not surprising that these derivations have a mathematical relation with the Einthoven's one. Indeed, the augmented voltages are obtained from a simple vector sum.
Considering, by definition, the reference potential at the center of the heart, we can derive the Goldberger's leads: the vectors of leads I, II and III are 60° out of phase and at half amplitude compared to the heart center.
With these considerations, graphically shown in fig.2.6a, it is possible to derive aVR, aVL and aVF.

$$\begin{aligned} aVL &= \frac{I - III}{2} \\ aVR &= -\frac{I + II}{2} \\ aVF &= \frac{II - III}{2} \end{aligned} \tag{2.2}$$

This shows that leads aVR, aVL and aVF can be calculated using just leads I, II and III. Therefore, these leads (aVF, aVR/–aVR, aVL) do not offer any new information, but view the same information from different angles [5].
For the sake of completeness, defining the vectors from the origin to the electrodes as EA, EB, EC, the derivations are expressed as:

$$\begin{aligned} I &= EB - EA \\ II &= EC - EA \\ III &= EC - EB \end{aligned} \tag{2.3}$$

It can be shown that the Goldberger's leads are called augmented because they cannot be inscribed in the circular triaxial diagram formed by the leads of Einthoven, as they have a greater amplitude. In fact:

$$\begin{aligned} aVL = \frac{I - III}{2} = \frac{1}{2}[(EB - EA) - (EC - EB)] = \\ = EB - \frac{1}{2}(EA + EC) = \frac{3}{2}EB \end{aligned} \tag{2.4}$$

Thus, their amplitude is 0.5 higher with respect to the vector referred to the electrode.

Anyway, it has been proved that with only 3 electrodes it is possible to obtain all six limb leads.

2.3 Common ECG noise sources

So far, only noise-free ideal signals have been analyzed. In practice, a variety of noise sources impact ECG recordings, affecting the reliability of the interpretation of the ECG signal itself. In order to provide accurate cardiac disease diagnosis and therapy, it is essential to understand the typical sources of ECG noise. There are mainly four types of artifacts that can affect ECG signals:

1. Baseline Wander.
2. Powerline Interference.
3. Electromyography Noise.
4. Electrode motion artifacts.

The following sections briefly describe these undesired contributions for the purpose of understanding how to eliminate them.

2.3.1 Baseline Wander (BW)

Baseline Wander is the effect due to the which the base reference seems to oscillate up and down with a very slow period rather than being straight. This makes the entire signal shifting from its baseline.
The Baseline Wander might be caused by improper electrodes (electrode-skin impedance), subject's movements and breaths (respiration) [12].
Figure 2.7 shows a typical ECG signal affected by Baseline Wander.

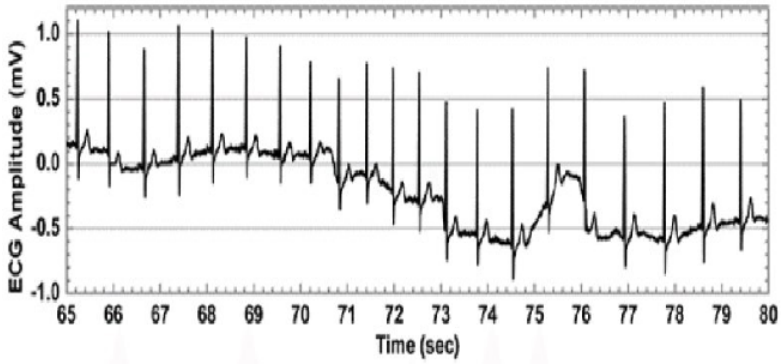


Figure 2.7: ECG signal affected by Baseline Wander.

The frequency content of the Baseline Wander is normally concentrated around 0.5Hz. However, it increases together with greater body movements as during exercise or stress tests.
Since the baseline signal is a low frequency signal, digital FIR filters and high-pass filters with a cut-off frequency of 0.5Hz can be used to estimate and remove the Baseline Wander from the ECG signal.

2.3.2 Powerline Interference (PLI)

Electromagnetic fields produced by powerlines represent a common noise source in ECG, as well as in any other electrical signal recorded from the body surface. Such noise consists in a 50Hz or 60Hz sinusoidal interference, eventually accompanied by a number of harmonics as shown in fig. 2.8.
It is necessary to remove powerline interference from ECG signals as it completely superimposes the low frequency ECG waves, especially the P and T waves.

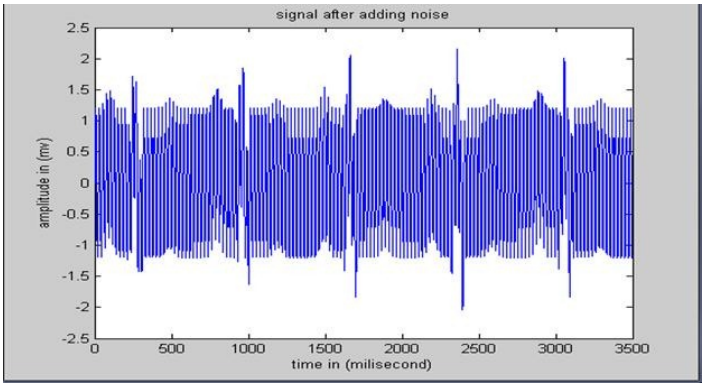


Figure 2.8: ECG with Powerline Interference at 50/60Hz.

A very simple approach to reduce Powerline Interference is to implement a notch filter characterized by a complex-conjugated pair of zeros that lie on the unit circle at the interfering frequency [12].
This type of filter highly rejects disturbances in a very selective way but, due to its response, it also introduces a ringing artifact in the output signal that is visible after the transient. This is a further source of distortion as it introduces a new contribution that overlaps to the QRS complex and the to T wave of the ECG signal.
For this reason, in practice, it has been preferred to opt for a cascade of low-pass filters with a cut-off frequency below 50Hz.

2.3.3 Electromyography Noise (EMG)

Muscle noise is a significant issue in many ECG applications, especially in recordings taken during physical activity, where low amplitude waveforms may entirely disappear.
Unlike Baseline Wander and Powerline Interference, muscle noise is not removed by narrowband filtering. Indeed, it provides a harder filtering challenge as the spectral content of muscle activity significantly overlaps to that of the PQRST complex.
Numerical techniques can be used to reduce muscle noise by taking advantage of the fact that the ECG is a repeating signal. However, the amount of effective noise reduction is insufficient. Hence, there is the need for signal processing techniques that can effectively reduce the influence of muscle noise [12].

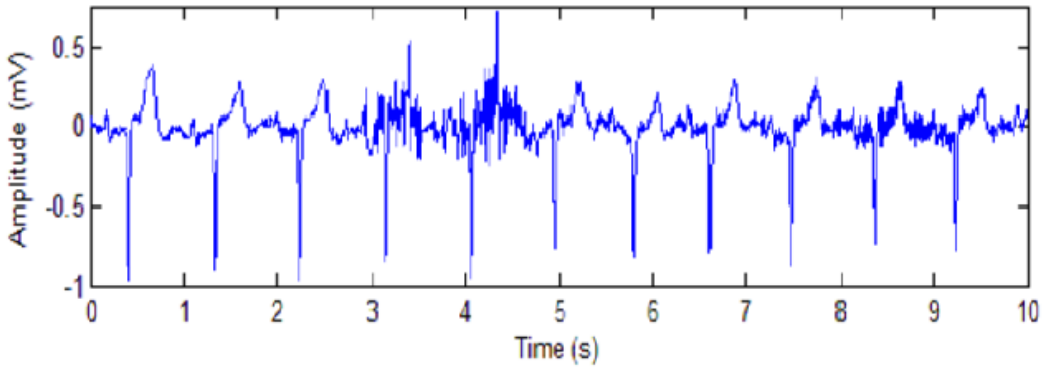


Figure 2.9: Electromyographic (EMG) noise.

2.3.4 Electrode motion artifacts

Electrode motion artifacts are mainly caused by skin stretching altering the impedance of the skin around the electrode.
The spectral content of motion artifacts significantly overlaps to that of the PQRST complex. This makes them particularly difficult to be removed.
Electrode motion artifacts show a large-amplitude waveform and are mostly prominent in the frequency range from 1Hz up to 10Hz. The following figure represents an example of such artifact [12].

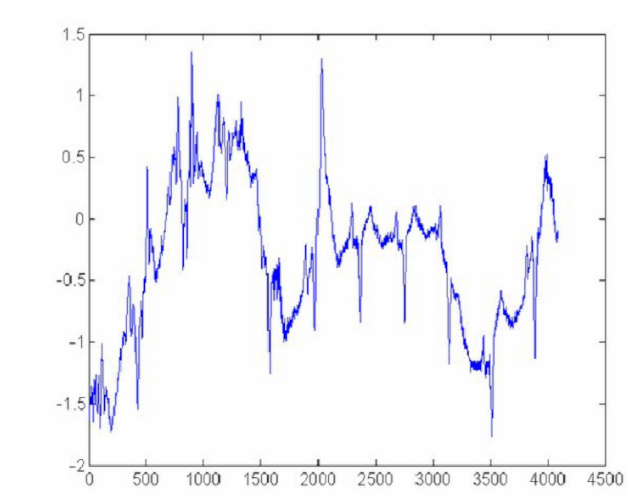


Figure 2.10: ECG affected by electrode motion artifacts.

Chapter 3

Analog Signal Conditioning

To acquire significant limb leads, it has been necessary to develop an analog front end capable of filtering and amplifying input signals, attenuating noise sources and improving quantization. An optimal front end cleans up the ECG signal without removing its useful portion.
Firstly, a modular front end, made up of independent low-pass filters, has been developed. This setup, shown in figure 3.1, allowed the testing of different filtering configurations.
This first solution was single-lead. It took a single differential signal between two electrodes. Such signal supplied an instrumentation amplifier in charge of reducing unintended common-mode contributions, thus attenuating any disturbance acting on both input connections. In addition, the high input impedance of the amplifier prevented the unknown impedance of the electrodes from creating a partition attenuating the signal strength.
The instrumentation amplifier used the voltage supplied by an integrator as a reference. This choice realized a high-pass filter with a cutoff frequency of approximately 0.5Hz, eliminating the Baseline Wander noise contribution.
The amplified signal passed through the modular network of low-pass filters. Among these filters, there were two first-order RC filters and two second-order Sallen-Key configurations. They could have been bypassed, if required.
The purpose of the filter network was to find a valid alternative to the use of a notch filter to eliminate the Powerline Interference noise.
As mentioned before, a notch filter would have eliminated the power supply noise, but its high selectivity, represented by a high quality factor, would have implied a low damping which would have made signals oscillating in the step response. And these oscillations would have represented an additional error contribution that, superimposed to the ECG signal, might have mislead the interpretation of the U-wave.
At the output of the low-pass filter network, an RC low-pass filter followed by a

gain stage has been introduced. It provided an amplified signal at the input of the ADC in order to increase its signal-to-noise ratio.

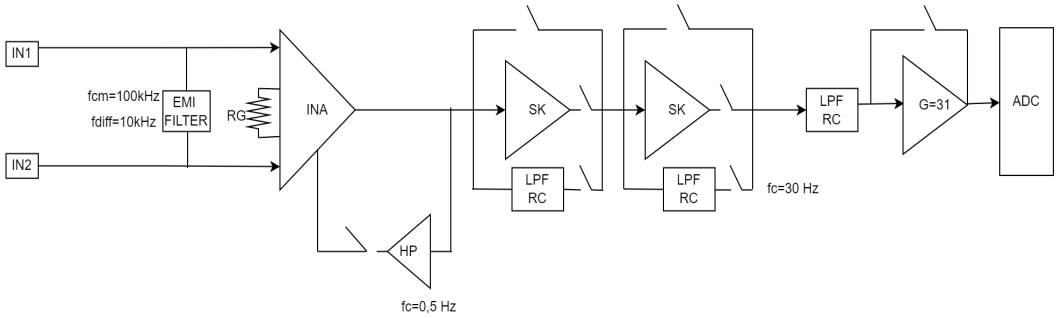


Figure 3.1: Modular testing front end.

3.1 Filters Analysis

This section studies the design of the hardware filters making up the analog front-end. Each filter transfer function has been simulated with MATLAB to determine the correct size of the actual electrical components. The study relies on the typical ECG's power spectral density and noise level covered in the previous chapter.

3.1.1 High Pass Filter

The high pass filter is in charge of removing all disturbances at very low frequencies around the continuous. These distortions might be caused by little muscle movement occurring during the respiration phase.
As it can be seen in figure 3.2, HP filtering is accomplished using a first order integrator fed back to the instrumentation amplifier [17].

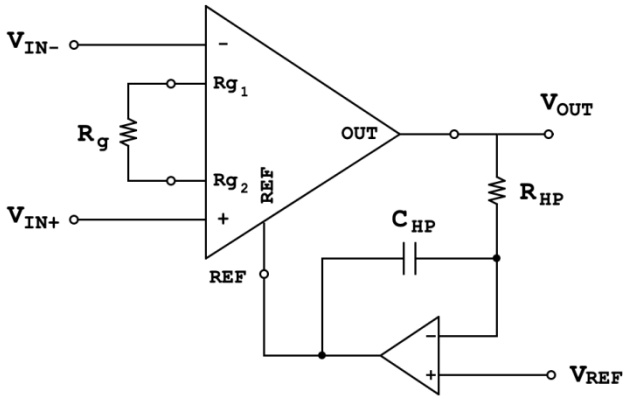


Figure 3.2: High Pass Filter.

The transfer function of this high-pass filter is:

$$H_{in} = \frac{K}{1 + \frac{1}{(s \cdot R_{HP} \cdot C_{HP})}} \tag{3.1}$$

Where K is the gain factor of the instrumentation amplifier, set to 10 to provide a higher CMRR. But notice that its actual value depends on the final device that is used. Instead, the cut-off frequency is given by:

$$f_{HP} = \frac{1}{2\pi \cdot R_{HP} \cdot C_{HP}} \tag{3.2}$$

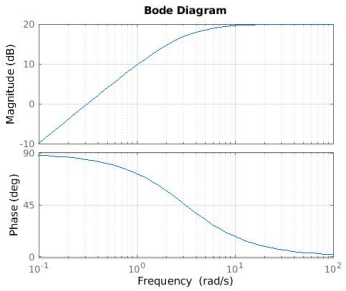
Targeting a cut-off frequency of 0.5Hz, the components turn out to be:

$$\begin{aligned} R_{HP} &= 68k\Omega \\ C_{HP} &= 4.7\mu F \end{aligned} \tag{3.3}$$

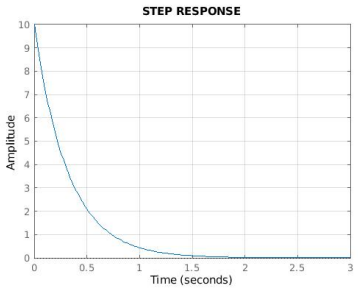
The following figures display the results of a simulation performed reporting the formulas in a MATLAB script. Real component values from the E24 series have been used. In this way, the real cut-off frequency has been evaluated.

3.1.2 Low Pass Filter Network

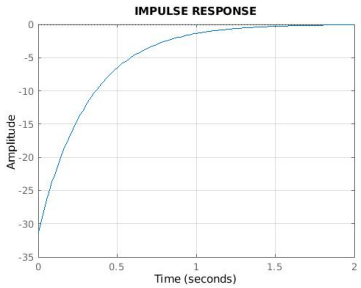
Power supply noise is an electromagnetic disturbance that, in case of very weak signals just as the ECG, might be comparable to the signal itself. To reduce, or better, to eliminate this noise contribution, LP filters can be used.
There are various types of low pass filters, each characterized by different features and performances. This section analyzes the nine different low pass filter configurations allowed by the low pass filter network. The analysis of these setups is useful



(a) Bode Diagram



(b) Step Response



(c) Impulse Response

Figure 3.3: High pass filter simulation results.

for the design of the final device, as it suggests the best way to reduce the noise of the 50Hz power supply. With respect to the low pass filter network, reported again in figure 3.4, it can be seen that a configuration can be built starting from three alternative paths:

1. Stage Bypass via 0Ω resistor.
2. Fourth order Sallen Key Butterworth-Bessel filter.
3. Second order RC filter.

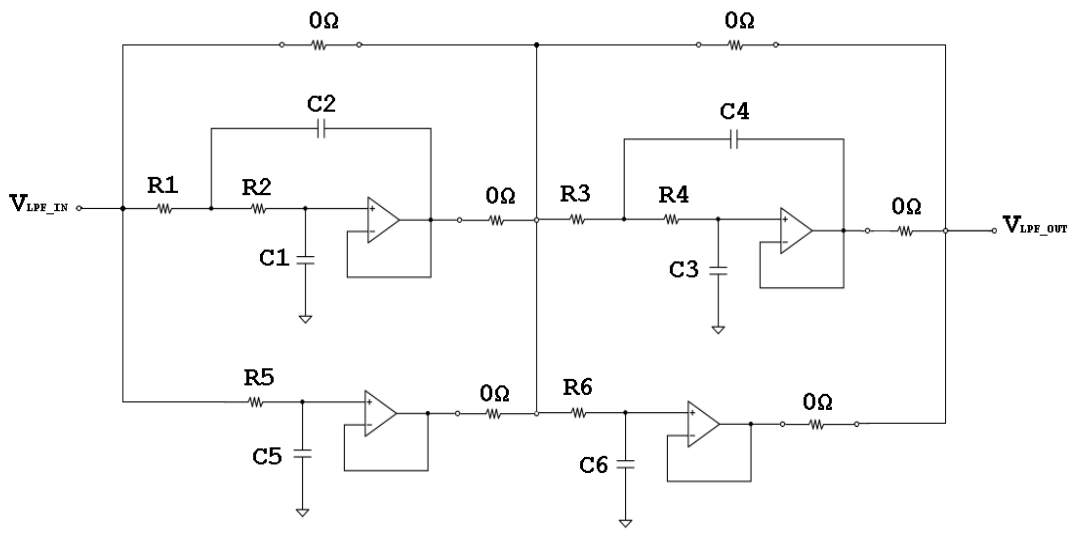


Figure 3.4: Low Pass Filter Network.

At this point, it is worth revising the main characteristics of the different available low pass filter responses:

- Butterworth Filters. These filters show maximally flat magnitude response and gain flatness in pass-band. However, the transient response to a pulse input may generate overshoot and ringing.

- Chebyshev Filters. These filters have ripple in pass-band. For this reason, their cut-off frequency is defined as the frequency in which the response falls below this ripple band. Their response to a pulse input is worse than the one of Butterworth filters.

- Bessel Filters. These filters are designed to have a maximally flat time delay so that they have a linear phase and a good transient response to a pulse input. Attenuation is -3dB at the cutting frequency.

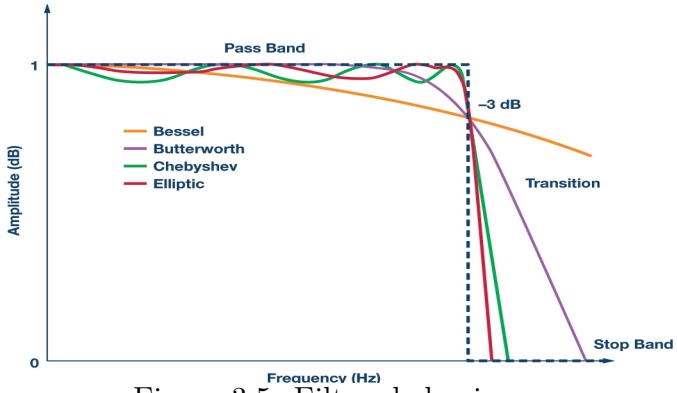


Figure 3.5: Filters behaviours.

In the following, the 4^{th}-order low-pass-active filter based on two cascaded Sallen Key cells, and the 2^{nd}-order low-pass-passive RC filter are designed. Specifically, the second order filter consists of two cascaded first order filters.
In addition, the transfer functions of the nine possible combinations are implemented and analyzed. Having decoupled each filter with an operational amplifier, the transfer function of the series of interconnected filters is given by the product of the individual transfer functions.

1. Sallen Key Filter design

To realize a higher-order filter, it is possible to combine a suitable number of first and second order filters. Generally, the order of the overall filter is equal to the sum of the orders of the individual filters, unless their connection results in zero-pole cancellation or it places, in series or in parallel, two or more reactive components of the same nature, making them non-independent.
It should be noted that in low pass filters of order n, the slope of the attenuated band asymptote is $-n \cdot 20\frac{dB}{dec}$. Thus, higher orders allow a better discrimination between passband and attenuated band.
The 4^{th}-order Sallen Key filter has been designed with reference to the values reported in the following table.

Ordine n°	stadio	Butterworth		Bessel		Chebyshev ($R = 0{,}5$ dB)		Chebyshev ($R = 2{,}0$ dB)	
		f_c	ξ	f_c	ξ	f_c	ξ	f_c	x
2	1	1	0,707	1,274	0,866	1,231	0,579	0,907	0,433
3	1	1	-	1,325	-	0,626	-	0,369	-
	2	1	0,500	1,445	0,724	1,069	0,293	0,941	0,196
4	1	1	0,924	1,432	0,958	0,597	0,709	0,471	0,538
	2	1	0,383	1,606	0,621	1,031	0,170	0,964	0,109
5	1	1	-	1,505	-	0,362	-	0,218	-
	2	1	0,809	1,559	0,888	0,690	0,424	0,627	0,282
	3	1	0,309	1,758	0,546	1,018	0,110	0,976	0,069
6	1	1	0,965	1,606	0,980	0,396	0,731	0,316	0,554
	2	1	0,707	1,692	0,818	0,768	0,276	0,730	0,176
	3	1	0,259	1,908	0,489	1,012	0,077	0,983	0,048
7	1	1	-	1,687	-	0,256	-	0,155	-
	2	1	0,901	1,719	0,340	0,504	0,458	0,461	0,304
	3	1	0,623	1,825	0,756	0,823	0,154	0,797	0,122
	4	1	0,223	2,053	0,444	1,008	0,059	0,987	0,035
8	1	1	0,980	1,781	0,988	0,297	0,740	0,238	0,561
	2	1	0,832	1,835	0,894	0,599	0,310	0,572	0,197
	3	1	0,556	1,956	0,703	0,861	0,144	0,842	0,090
	4	1	0,195	2,132	0,408	1,006	0,043	0,090	0,027

Figure 3.6: Filters behaviours.

In general, to size a low pass filter of order higher than the second, it is possible to proceed as follows [16]:

1. Choose the filter response (Butterworth, Chebyshevor Bessel), the filter order (n), the cut-off pulsation (ωt) and the centre-band gain (A).

2. Based on the type of response and the filter order, identify, for each stage, the values for the normalized frequency f_{ci} and the damping ξ according to table 3.6.

3. For each stage,

   - Derive the cut-off frequency of the filter.
   - Choose C_i so that it is possible to fix $C_{i+1} = n \cdot C_i$.
   - Fix $R_{i+1} = m \cdot R_i$ obtaining $R_i = \frac{1}{(\omega \cdot \sqrt{n \cdot m \cdot C_i^2})}$.
   - Derive the gain: $A = 3 - 2\xi$.
   - The values for the remaining gain resistors are determined fixing one of them.

In this specific case, to achieve unity gain and a good attenuation at the power supply noise frequency, a cut-off frequency of 30Hz and a roll-off factor of $80\frac{dB}{dec}$ have been chosen.
The filter has been designed with an intermediate behaviour between Butterworth and Bessel responses because the Bessel filter is suitable for applications where minimal distortion over the entire bandwidth is required, but it does not provide a roll-off as fast as the Butterworth one.
In the end, a 0.5 Butterworth-Bessel filter of order 4 is designed averaging the

behaviour of the two types. Referring to table 3.6, intermediate values for the pulsations of the two stages are:

$$\begin{aligned} \omega_1 &= \frac{1+1.43}{2} = 1.22 rad/s \\ \omega_2 &= \frac{1+1.61}{2} = 1.31 rad/s \end{aligned} \tag{3.4}$$

For the first stage, it has been chosen:

$$C1 = 36nF \textbf{ and } n = 10\textbf{. So } C2 = 360nF \tag{3.5}$$

Being,

$$f_{c1} = 1.22 \cdot f_c = \frac{1}{2\pi\sqrt{nC_1^2 m R_1^2}} \tag{3.6}$$

and having set m=17, it turns out that the value for R1 is:

$$R1 = \frac{1}{2\pi f_{c1}\sqrt{nC_1^2 m}} = 9.2k\Omega \tag{3.7}$$

Finally, $R1 = 9.2k\Omega$, $R2 = 157.5k\Omega$ and the gain of the stage is one ($A = 1$).
For the second stage, a similar procedure is followed:

$$C3 = 43nF \textbf{ and } n = 10\textbf{. } C2 = 430nF \tag{3.8}$$

Keeping m=17,

$$R3 = 7.2k\Omega \textbf{ and } R4 = 122k\Omega \tag{3.9}$$

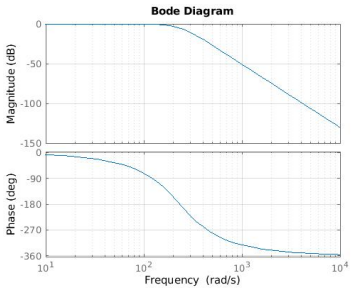
This stage is chosen with unity gain as well.
At this point, the values for resistors and capacitors are set on the basis of the ones commercially available:

$$\begin{aligned} R1 &= 8.67k\Omega & R2 &= 154k\Omega \\ C1 &= 390nF & C2 &= 39nF \\ R3 &= 6.98k\Omega & R4 &= 110k\Omega \\ C3 &= 470nF & C4 &= 47nF \end{aligned} \tag{3.10}$$

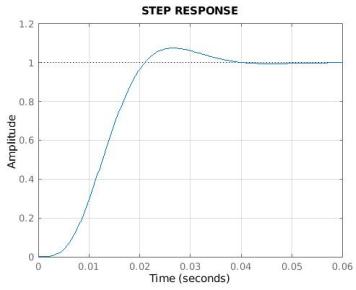
With these values, the actual cut-off frequency of the filter can be evaluated. The overall transfer function is given by the product of the transfer function of the two individuals Sallen Key cells:

$$H = \frac{1}{[s^2(R1R2C1C2) + s((R1 + R2)C2) + 1]} \cdot \cdot \frac{1}{[(s^2(R3R4C3C4) + s((R3 + R4)C4) + 1]} \tag{3.11}$$

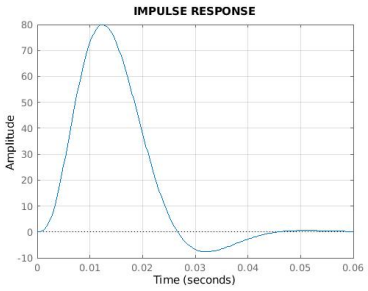
It shows four complex denominator roots which, on the Bode diagram, result in two distinct poles at frequencies f_1=36Hz and f_2=39Hz. The actual cutoff frequency at -3dB is f_c=29.5Hz.



(a) Bode Diagram



(b) Step Response



(c) Impulse Response

Figure 3.7: Sallen Key filter simulation results.

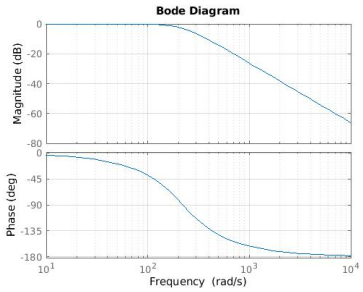
The individual transfer functions of the two cells are presented below.

1.1 First Sallen Key

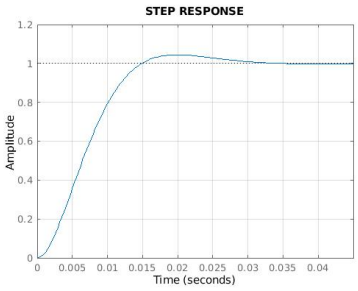
The transfer function of the first Sallen Key filter is:

$$H_{SK1} = \frac{1}{s^2 \cdot (R1 \cdot R2 \cdot C1 \cdot C2) + s \cdot ((R1 + R2) \cdot C2) + 1} \tag{3.12}$$

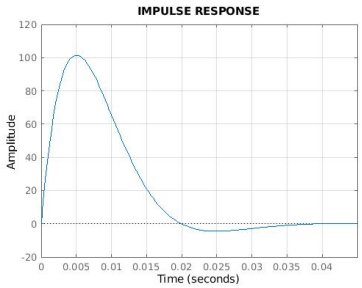
The cut-off frequency is f_{SK1} =36Hz. The attenuation at 50Hz is $6.5dB$. It results in a factor of 2.1



(a) Bode Diagram



(b) Step Response



(c) Impulse Response

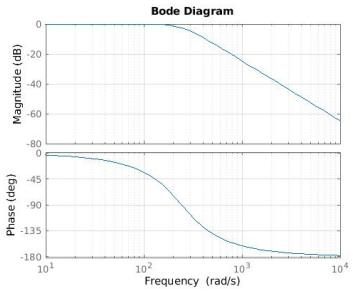
Figure 3.8: First Sallen filter simulation results.

1.2 Second Sallen Key

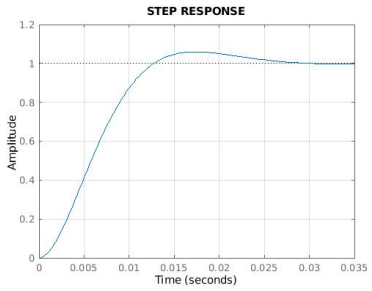
The transfer function of the second Sallen Key filter is:

$$H_{SK2} = \frac{1}{s^2 \cdot (R3 \cdot R4 \cdot C3 \cdot C4) + s \cdot ((R3 + R4) \cdot C4) + 1} \tag{3.13}$$

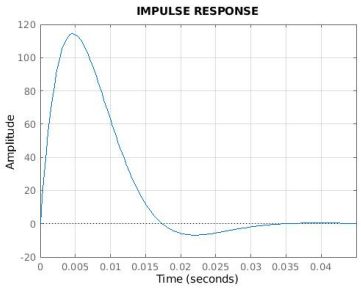
The cut-off frequency is f_{SK2} =39Hz. The attenuation at 50Hz is $6.1dB$. It results in a factor of 2.



(a) Bode Diagram



(b) Step Response



(c) Impulse Response

Figure 3.9: Second Sallen filter simulation results.

2. RC Filter design

The second type of filter which has been designed is a second-order-passive RC filter. For this type of filter, a higher order filter can be obtained simply combining in series a suitable number of first order RC filters. However, as the number of concatenated stages increases, the actual cut-off frequency deviates from the theoretical one. For this reason, it is better to make the impedance of each stage 10 times that of the previous one, thus, R2=10·R1 and C2=$\frac{1}{10}$·C1. Alternatively, it is possible to use op-amps to decouple the various stages.
To design the second order RC filter, it is sufficient to define the desired cut-off frequency. The transfer function of the filter is given by the product of the individual transfer functions of the two stages:

$$H_{RCs} = H_{RC1} \cdot H_{RC2} = \frac{1}{s^2(R_1C_1R_2C_2) + s(R_1C_1 + R_2C_2) + 1} \tag{3.14}$$

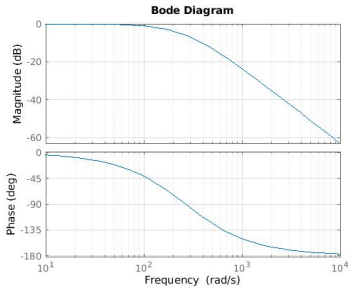
The cut-off frequency of a 2^{nd}-order passive low pass filter is:

$$f_c = \frac{1}{2\pi\sqrt{R_1 \cdot C_1 \cdot R_2 \cdot C_2}} \tag{3.15}$$

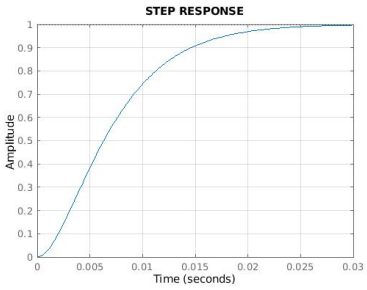
Also in this case, a trade-off between attenuation at 50Hz and cut-off frequency is desired. Considering that there is an attenuation of $40dB$ per decade, it has been decided to cut at 30Hz.
Having set the cut-off frequency, equation 3.15 gives three degrees of freedom for the values of resistors and capacitors. These values have been chosen according to the E24 series to obtain two balanced RC stages. The chosen values are:

$$\begin{aligned} R_1 &= 3.92k\Omega \\ C_1 &= 1\mu F \\ R_2 &= 3.6k\Omega \\ C_2 &= 1\mu F \end{aligned} \tag{3.16}$$

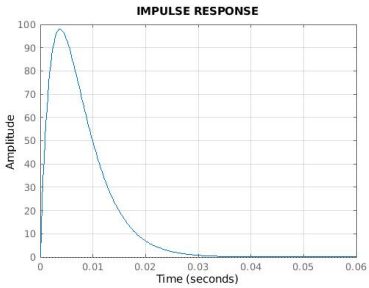
In this way, the cut off frequency is f_{RCs} =30Hz. The attenuation at 50Hz is $8dB$. It results in a factor of 2.5.



(a) Bode Diagram



(b) Step Response



(c) Impulse Response

Figure 3.10: RC filter simulation results.

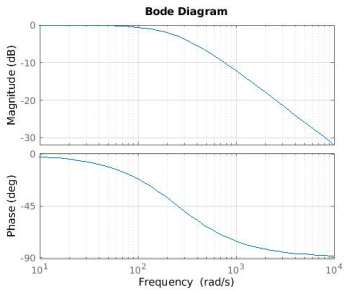
The individual transfer functions of the two first order low pass cells are presented below.

2.1 First RC

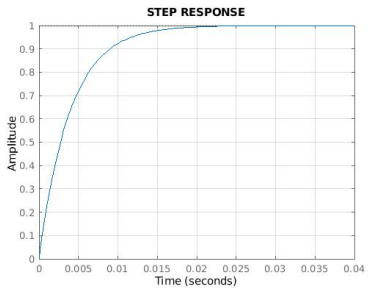
The transfer function of the first RC filter is:

$$H_{RC1} = \frac{1}{s \cdot R_5 \cdot C_5 + 1} \tag{3.17}$$

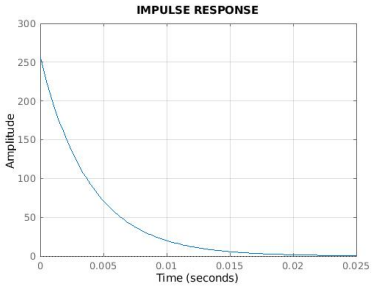
The cut-off frequency is f_{RC1} =41Hz. The attenuation at 50Hz is $3.8dB$. It results in a factor of 1.6.t



(a) Bode Diagram



(b) Step Response



(c) Impulse Response

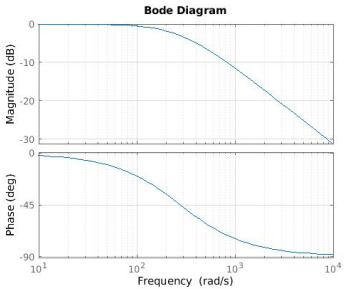
Figure 3.11: First RC filter simulation results.

2.2 Second RC

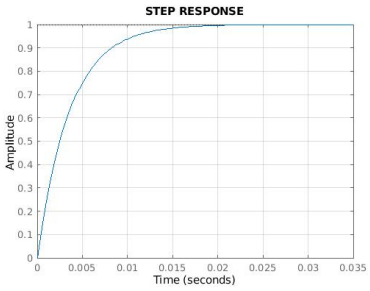
The transfer function of the second RC filter is:

$$H_{RC2} = \frac{1}{s \cdot R_6 \cdot C_6 + 1} \tag{3.18}$$

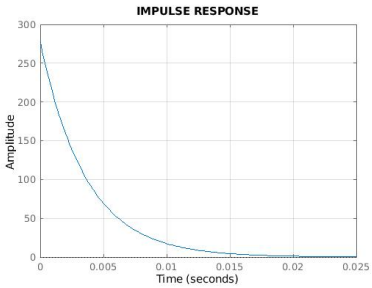
The cut-off frequency is f_{RC1} =44.5Hz, the attenuation at 50Hz is $3.5dB$. It results in a factor of 1.5. Having decoupled the two RC filters with inverting stages, it



(a) Bode Diagram



(b) Step Response



(c) Impulse Response

Figure 3.12: Second RC filter simulation results.

is possible to combine them with the two Sallen Key filters to achieve alternative solutions.

3. Further combinations

Further combinations can be achieved either connecting in series RC and Sallen Key filters or bypassing stages, increasing or decreasing the order of the overall filter. Also in these cases, it is possible to evaluate the responses multiplying the single transfer functions, obtaining a transfer function whose poles are given by those of the individual filters. Numerous different combinations have been tested, and the results are reported in section 3.2.

3.1.3 Output stage

The output stage consists of an additional low pass RC filter together with a gain factor. The cut-off frequency of the filter is set to 50Hz to further attenuate any unwanted signal contribution outside of the band of interest. Notice that this stage could also be excluded if necessary.

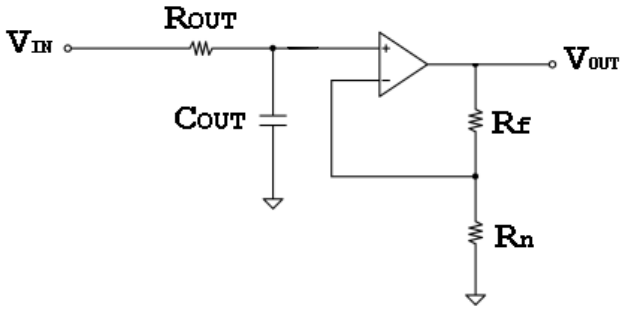


Figure 3.13: Output stage.

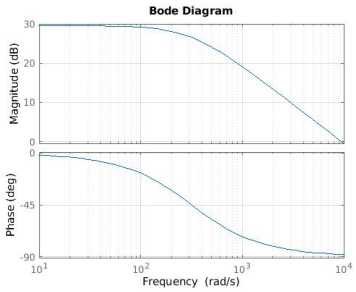
The transfer function of the output stage is:

$$H_{out} = \frac{Rf/Rn}{s \cdot Rout \cdot Cout + 1} \tag{3.19}$$

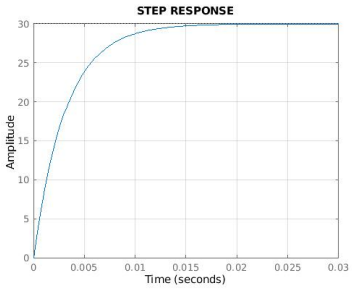
The values for resistors and capacitors, chosen from the E48 series, are:

$$\begin{aligned} Rout &= 3.16k\Omega \\ Cout &= 1\mu F \\ Rf &= 30k\Omega \\ Rn &= 1k\Omega \end{aligned} \tag{3.20}$$

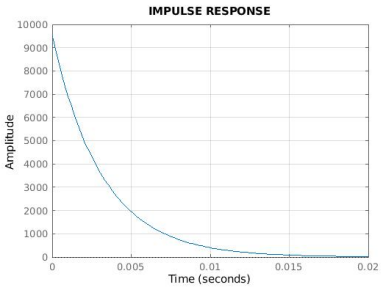
The real pole is located at 50Hz and a gain of 30 is introduced.



(a) Bode Diagram



(b) Step Response



(c) Impulse Response

Figure 3.14: Output stage filter simulation results.

3.2 Preliminary results

The modular front end served as a prototype useful to test the different filter configurations. Different setups could have been made simply soldering 0Ω resistors. Together with the analog front end, the prototype included a 24-bit Delta Sigma ADC and a development board. Acquired samples, converted into digital streams, were transferred to the development kit via SPI and, from there, they were printed on standard console using UART. Figure 3.15 shows the top view of prototype layout

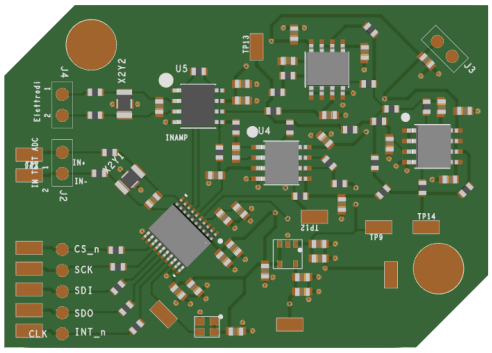
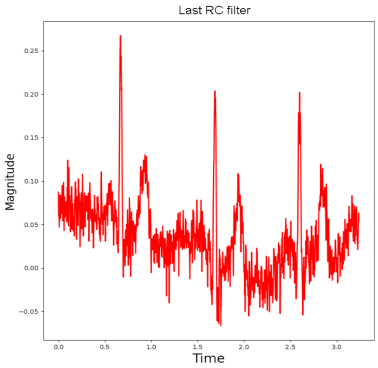
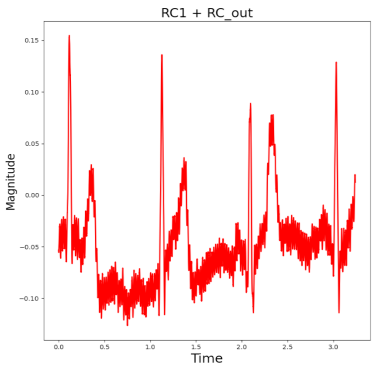


Figure 3.15: Front end testing board top layout.

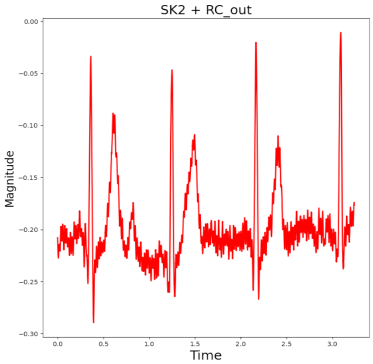
Instead, the following images report the results of the acquisitions performed with six different configurations of low pass filters.



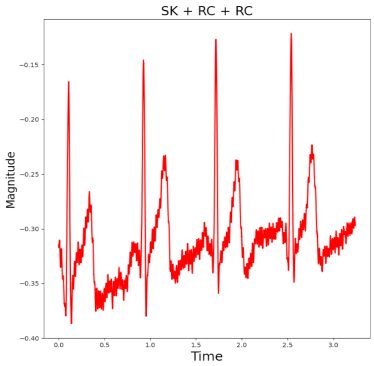
(a) Only output RC filter.

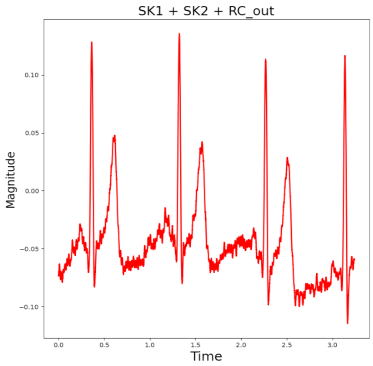


(b) First and output RC filters.

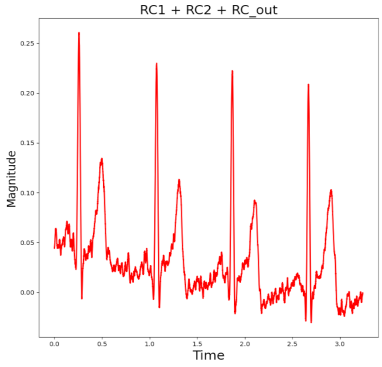


(c) Second SK and output RC filters.





(a) First SK, second RC and out RC filters. (b) First SK, second SK and out RC filters.



(c) First RC, second RC and out RC filters.

3.3 Final configuration

Experimental results showed that the best configuration consisted in a cascade of 2 RC filters followed by the RC filter of the gain stage. This configuration, paired with DSP filtering techniques, brought results that were almost comparable to those of a professional electrocardiograph.

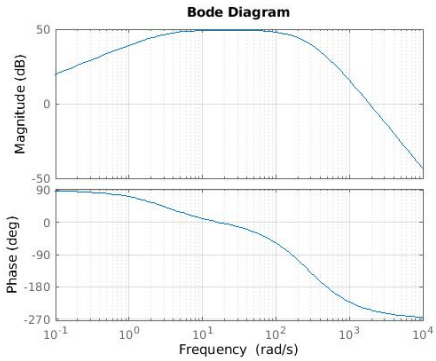
3.3.1 Full front end transfer function

The transfer function of the complete front end, given again by the product of the transfer functions of the individual blocks, is:

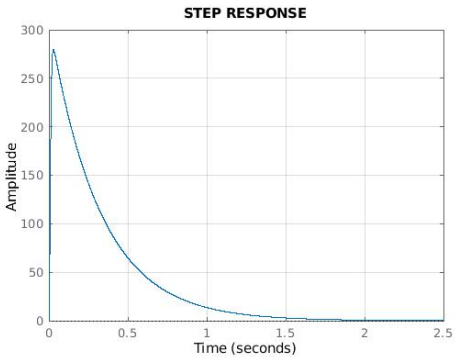
$$H = H_{in} \cdot H_{RCs} \cdot H_{out} \tag{3.21}$$

Figure **??** shows the theoretical responses of the transfer function. Instead, figure **??** compares the experimental ECG acquired with the best configuration for the

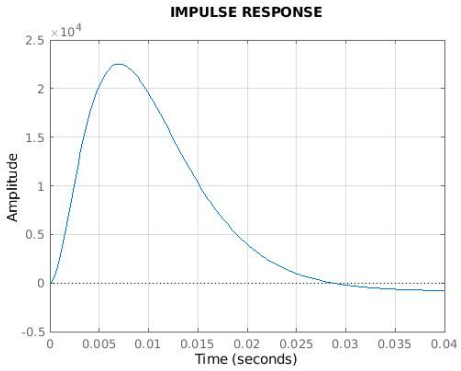
prototype to a professional electrocardiogram registered with a medical electrocardiograph. Experimental results are presented with and without filtering.



(a) Bode Diagram.



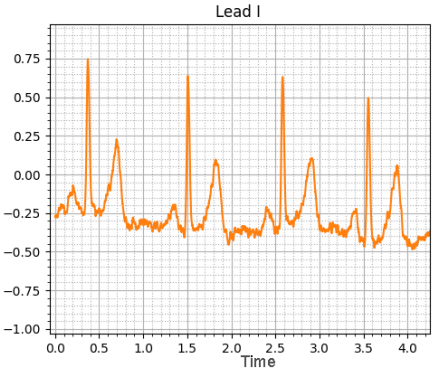
(b) Step Response.



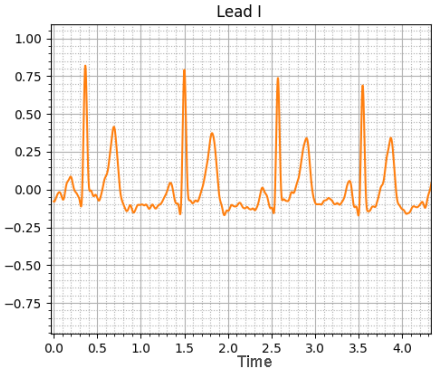
(c) Impulse Response.

Figure 3.18: Full front end simulation results.

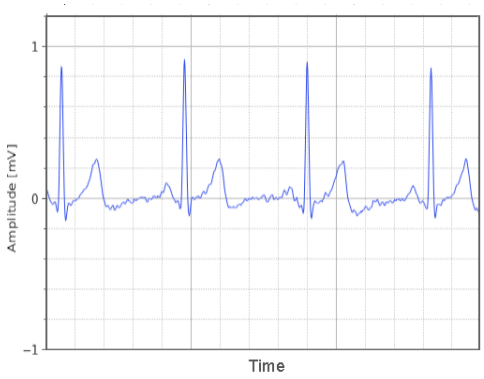
The best configuration shows a gain of $50dB$ in pass-band and an optimal rejection of power interferences. However, the cutoff frequency is very low, about 30Hz. A low cutoff frequency may remove portions of useful signal in ECG waveforms acquired from subjects with larger spectral contents with respect to those subjected to the preliminary tests.
For this reason, it is necessary to tune the values of resistors and capacitors in the low-pass filters to increase the cutoff frequency, leaving the rejection of powerline noise to digital filtering.
Another mandatory modification is the duplication of the test front end, as the final device needs to simultaneously acquire two differential signals between right arm, left arm and left leg. In this case, there are two input paths flowing from the input electrodes to the ADC, as shown in the following scheme.



(a) No digital filtering.



(b) Ddigital filtering.



(c) Reference electrocardiograph.

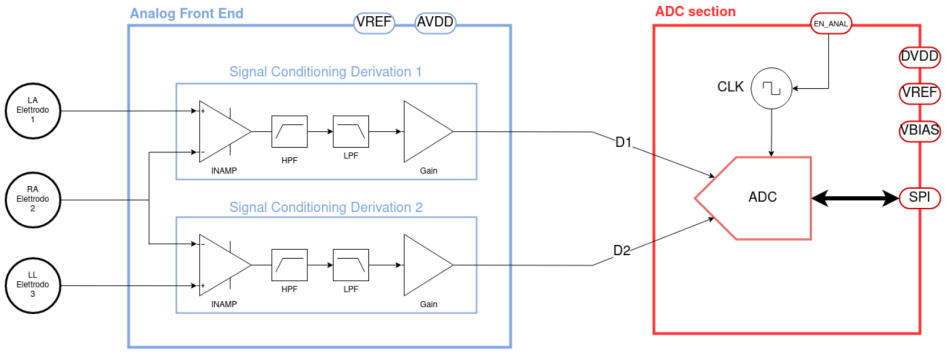


Figure 3.20: Schematic of the final front end implementation.

Chapter 4

Board design

The aim of the project is to acquire a 6-leads electrocardiogram by means of a low-power, ultra small, portable, wireless device. It has been decided to shape the device as a wearable wrist watch.
One electrode lays at the bottom of the device, making contact with the left wrist. Another electrode is located on the top side of the device, so that the right index finger can be placed on it. The third electrode is embedded in the strap to be comfortably rested on the left ankle when needed.

4.1 Device overview

The first step of the development consists in the definition of the specifications. The following list reports the requirements and the main features of the device:

- Deliver of a 6-leads ECG.
- Ultra small design (Radius=2 cm, Thickness=0.6cm).
- Portable and wearable on the wrist.
- Low power.
- Rechargeable through separate type-C USB charger.
- Low energy Bluetooth 5.3.
- Four layer PCB.
- Double side SMT.

Figure 4.1 shows a sketch of the device, together with its main functional blocks. Functional blocks are partitioned between bottom and top sides. This way, it is possible to have a visual understanding of the device structure.

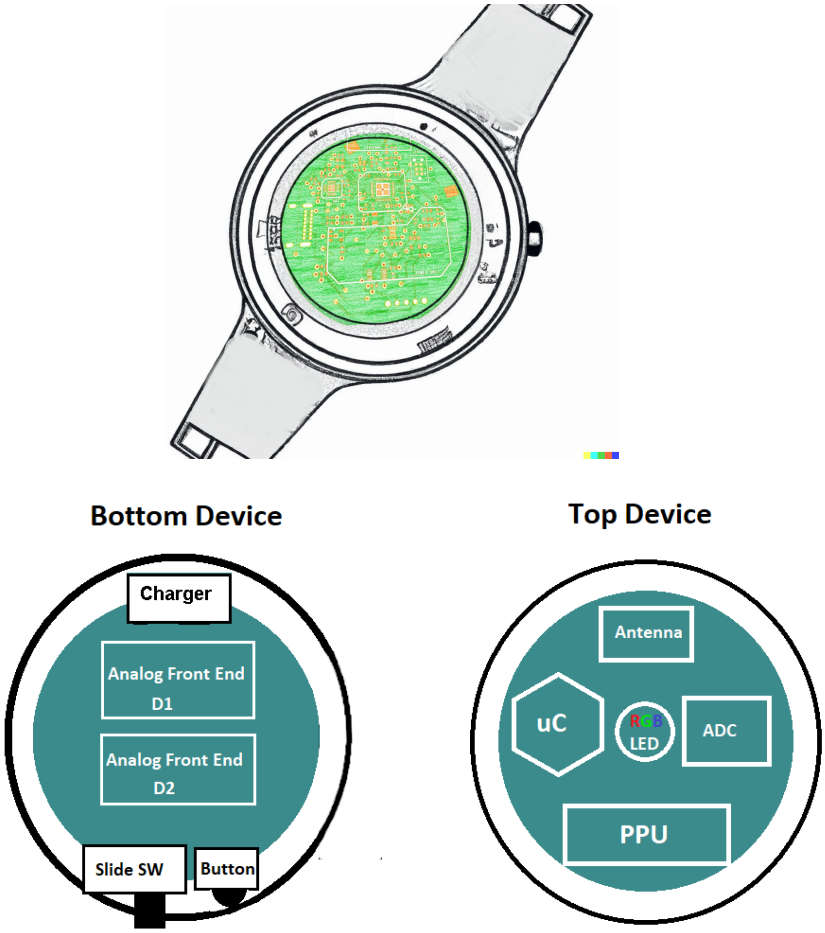


Figure 4.1: Board Sketch and main functional blocks.

4.1.1 Block Diagram

To better understand how the various functional blocks work in relation to each other, it is useful to take a look at figure 4.2, which represents the block diagram of the device. Each block of the figure is a hardware unit devoted to a specific function. It is possible to identify five main functional blocks:

1. **Analog Front End**. It makes use of three electrodes (LA, RA and LL) to acquire the ECG signal in a differential way. Common mode noise is reduced using an instrumentation amplifier (INAMP) on each signal. Signals are then band-pass filtered through the serial combination of a HP filter at 0.5Hz and a LP filter with a cut-off frequency of 30Hz. Finally, they are amplified (GAIN) to optimize SNR due to quantization.

2. **ADC section**. It discretizes the analog signals feeding digital data to the Microprocessor. It uses an 8MHz clock (CLK) coming out of the Microprocessor itself to reach 24-bit resolution.

3. **Microcontroller**. It manages the delivery of control signals to the different functional blocks and communicates with slave peripherals using SPI and I2C interfaces. It monitors battery level and sends data to the smartphone through the Bluetooth antenna.

4. **Peripherals**. They consists in the interface between the user and the hardware circuit.

5. **Power Management**. It provides the $3.3V$ supply voltage to the whole system, as well as the bias ($2.5V$) and reference ($1.25V$) voltages. It drives the turn on and shutdown of the integrated circuits to optimize power savings. It turns off the device during charging, as required by law.

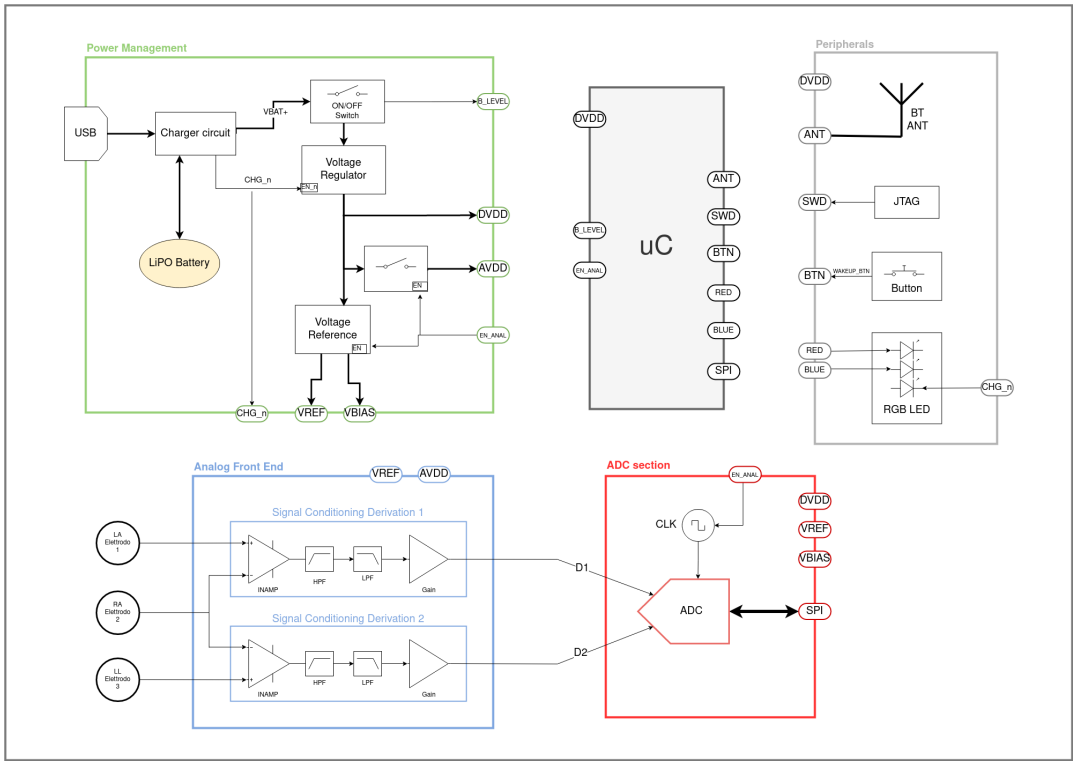


Figure 4.2: Block Diagram.

4.2 Schematic diagram

Schematic diagram has been realized using Cadence OrCAD Capture. Again, the diagram is organized in five sections:

- Analog Front End.
- ADC Section.
- Microcontroller.
- Peripherals.
- Power Management.

4.2.1 Analog Front End

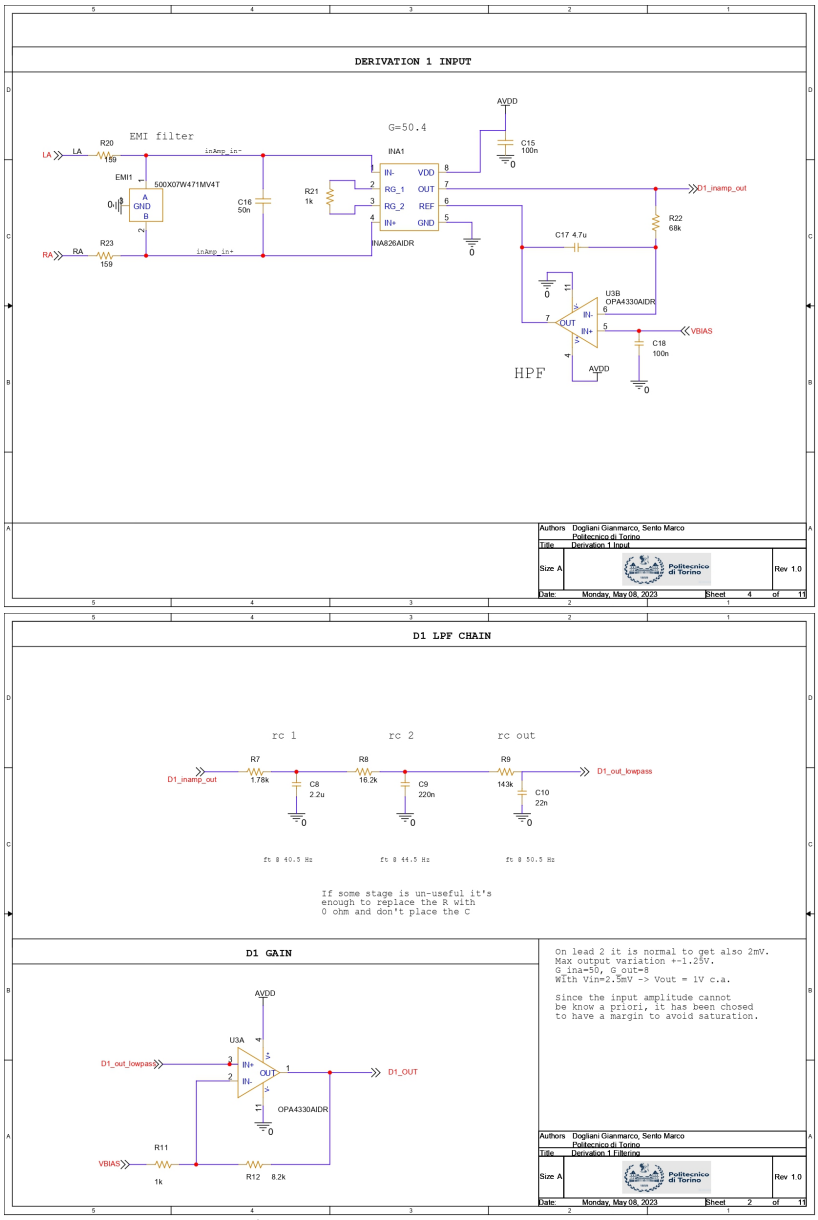


Figure 4.3: 1^{st} Derivation of analog front end section.

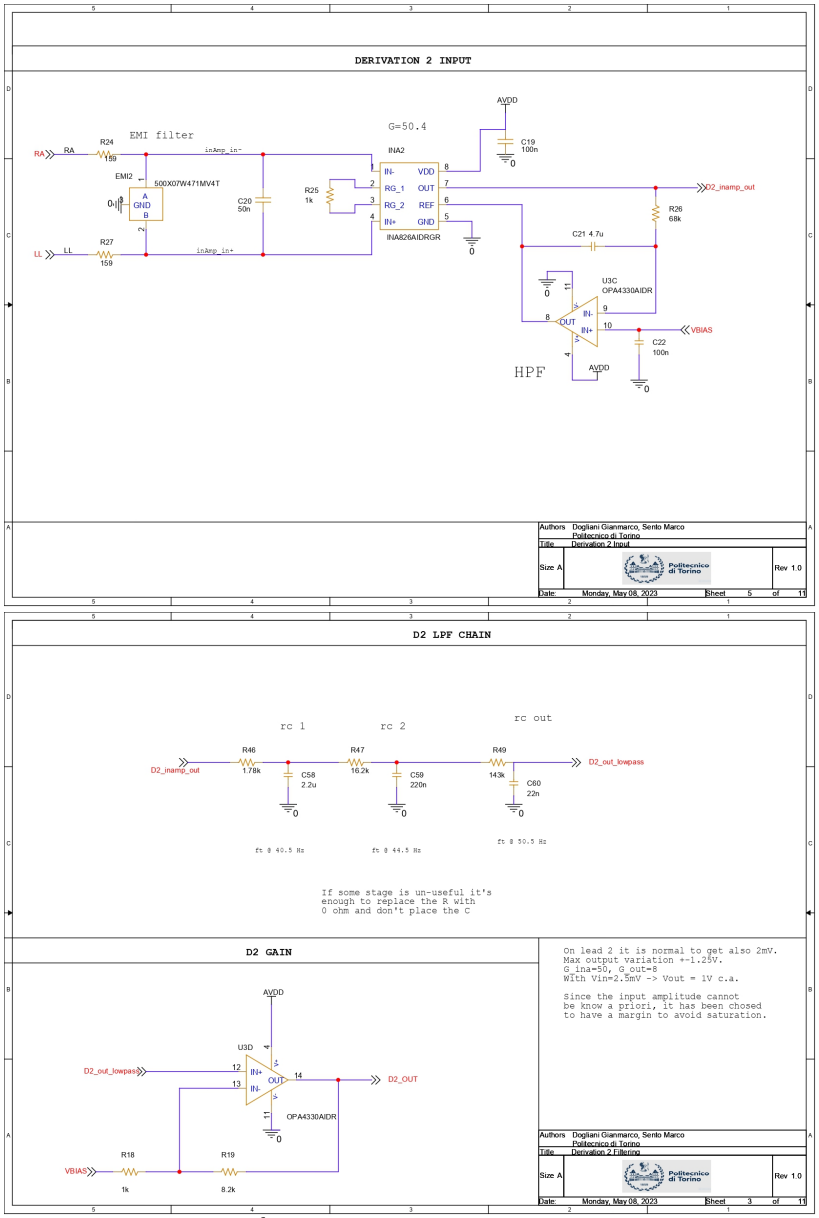


Figure 4.4: 2^{nd} Derivation of analog front end section.

Figures 4.3 and 4.4 show the analog path followed by the differential signals acquired from the input electrodes. The first derivation represents lead I while the second derivation represents lead II. It can be seen that the signals go through:

- An EMI filter.
- An instrument amplifier with a gain of 50.
- An integrator acting as a high pass filter.
- A chain of 3 RC low-pass filters with an overall cut-off frequency of 30Hz.
- A gain stage amplifying of a factor 8.

The overall gain has been set to optimize the sampling SNR without making signals too much wide than exceeding the dynamics of the ADC.

4.2.2 ADC Section

This section contains the 24-bit Sigma Delta ADC and its clock. Using an 8MHz clock, the ADC is capable of simultaneously sampling the two differential channels with a sampling rate of $1300sps$. With 24 bits and the reference voltage set to $2.5V$, it offers a resolution of:

$$Q = \frac{V_{ref}}{2^N} = \frac{2.5V}{2^{24}bit} = 149nV \tag{4.1}$$

As the CMOS clock has a high power consumption, it embeds an enable pin.

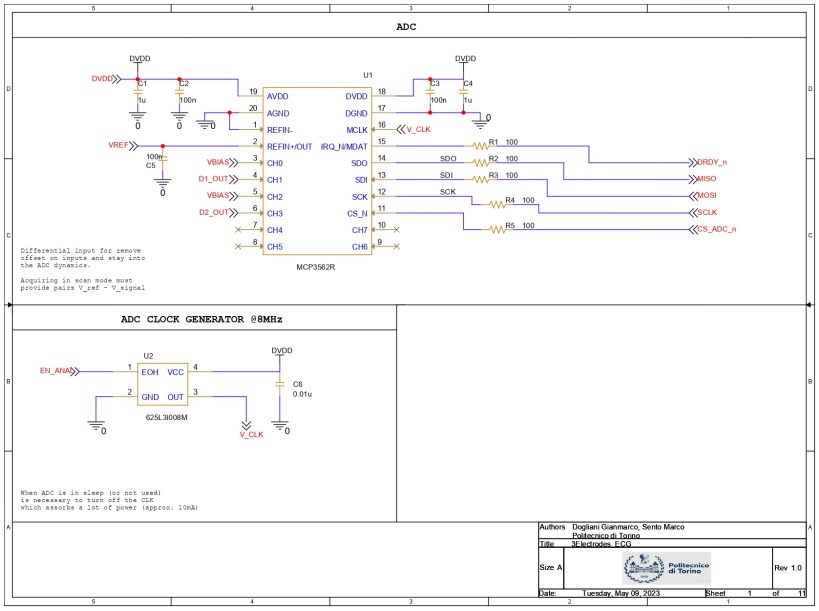


Figure 4.5: ADC Section.

4.2.3 Microcontroller

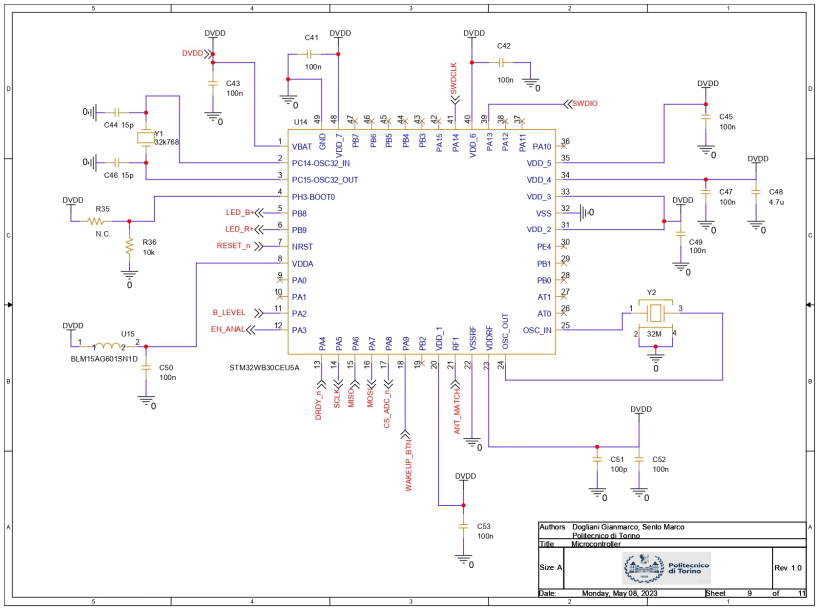


Figure 4.6: Microcontroller Section.

It follows a resume of the main pins of the microcontroller section:

- The ANT_MATCH pin is connected to the matching circuit of the patch antenna.
- Pins 13 to 17 are used for SPI communication with the ADC.
- Pin 11 is connected to the internal ADC to sample the battery level and obtain the battery percentage.
- Pins 5 and 6 are used to change the duty cycle of the LEDs reducing their power consumption according to the requirements.
- Pin 18 is used to execute an interrupt to exit from low power states.
- Pin 12 is used to enable or disable ICs when necessary. It is useful to turn off the high-power-consumption 8MHz clock of the ADC.
- SWDIO, SWDCLK and RESET_n are used to connect the microcontroller to the JTAG connector.

Several capacitors are used to decouple the pins of the microcontroller and two external clock sources are used. Specifically, a 32MHz and a 32.768kHz crystals are required.

4.2.4 Peripherals

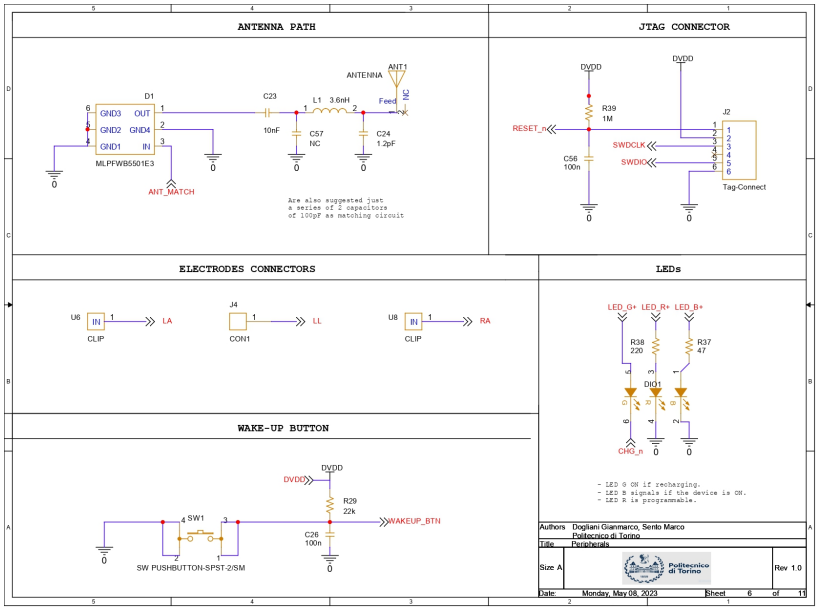


Figure 4.7: Peripherals Section.

This section includes all the communication elements:

- A push button to wake-up the microcontroller from low power states.
- A JTAG connector to program and debug the circuit.
- An RF matching path and a patch antenna.
- Different electrodes connectors.
- Three LEDs.

LEDs values have been calculated with the following formula:

$$R_{LED} = \frac{V_{max} - V_{nominal}}{I_{nominal}} \tag{4.2}$$

4.2.5 Power Management Section

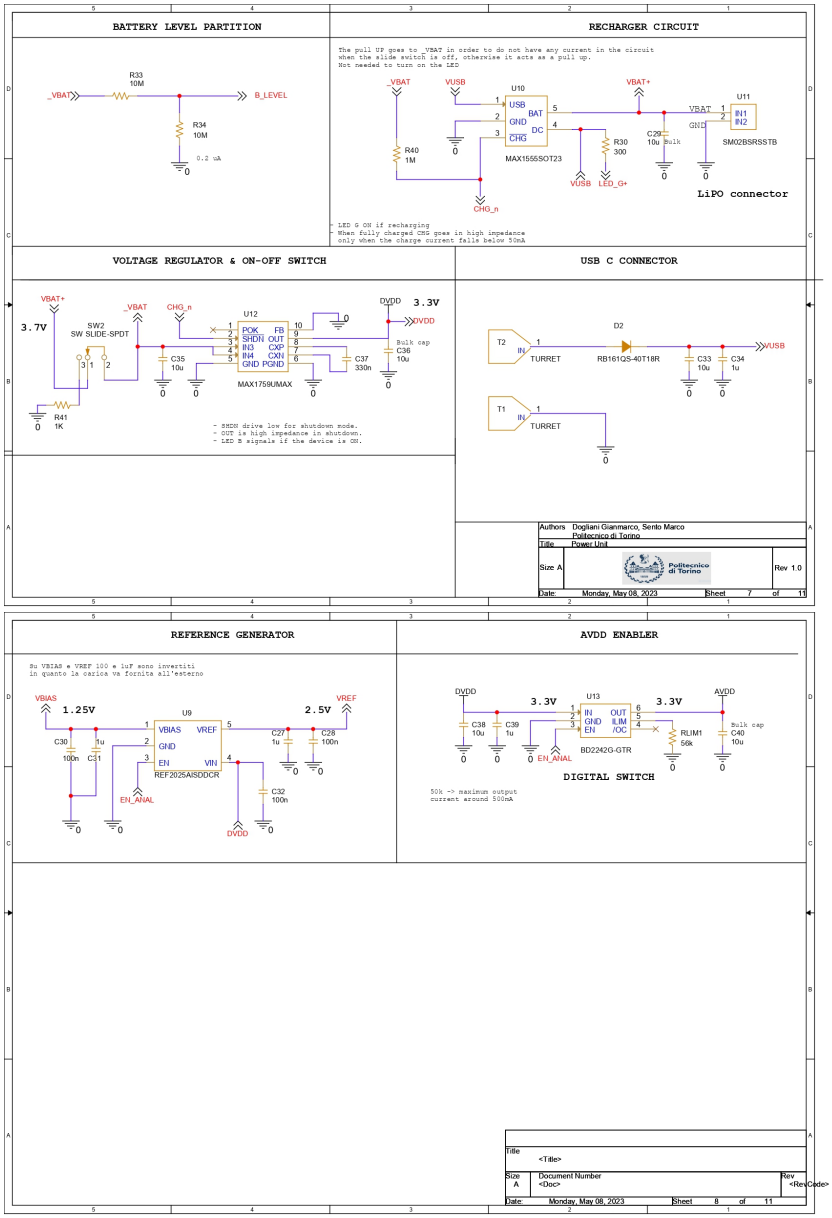


Figure 4.8: Power management section.

This last section consists of different components:

- The MAX1555 battery charger connected to the power connectors.
- The MAX1759 voltage regulator to provide a stable voltage to the circuit.
- A reference generator to provide stable $2.5V$ and $1.25V$ to the analog sub-part.
- An High Side MOS to disable the analog sub-part to save energy.
- A connector to attach the power source to recharge the battery.
- A slide button to detach the battery.
- A green LED.

The green Led is connected between the voltage provided by the USB charger and the CHG pin of the MAX1555 to signal the user that the battery is charging.

4.3 Components selection

4.3.1 Microcontroller

The μC is the core of the system. It is in charge of:

- Reading ADC data through SPI.
- Monitoring battery status via I2C.
- Communicating with the smartphone using the Bluetooth antenna.
- Managing power savings driving sleep and shutdown modes of the various ICs.
- Interfacing the user by means of light indicators (LEDs), push-buttons and the JTAG interface.

The project initially relied on the nRF52833 μC from Nordic Semiconductor. However, it has been replaced with the ST Microelectronic's STM32WB30RG, as this microcontroller offers equivalent features with a more user-friendly and flexible IDE that supports the integration of an Artificial Neural Network developed with TensorFlow.

STM32WB30RG

The STM32WB30RG multiprotocol, wireless device embeds a powerful and ultra-low power radio compliant with Bluetooth® LE 5.3 specifications. It is designed to be extremely low power, and it relies on the high-performance 32-bit Arm® Cortex®-M4 RISC processor operating at a frequency up to 64MHz. The device embeds a high-speed 512KB Flash and a 96KB SRAM memory. Direct data transfers memory-peripherals and memory-to-memory are supported by fourteen DMA channels.
Compared to the nRF52833 SoC, the STM32WB30RG offers a lower power consumption and a more user-friendly IDE that allows the firmware integration of an ANN developed with TensorFlow. All this at the expense of a slightly larger occupied area on the PCB layout [28].

Main features

- 64MHz Arm Cortex-M4.
- 1MB Flash, 256KB SRAM.
- 6 different Low Power Modes.
- Bluetooth Low Energy 5.3.
- 32MHz SPI, I2C, UART, DMA, NFC.
- 12-bit ADC.
- 6x timers (1x RTC, 3x General Purpose, 2x Low Power).
- 7x7 mm QFN48 Package.

Electrical characteristics (T @ $25°C$)

- Supply Voltage: $1.7V$ to $3.6V$.
- Shutdown mode current: $13nA$.
- Standby mode current: $600nA$.
- Stop mode current: $2.1\mu A$.
- Active mode current: up to $53\mu A/MHz$.
- RF current: Rx $4.5mA$ / Tx (@$0dBm$) $5.2mA$.

- Up to 64MHz from a 32 MHz external crystal oscillator.
- External 32.768 kHz crystal oscillator.
- Working Temperature: -40 to $+105°C$.

Pinout and Package Outline

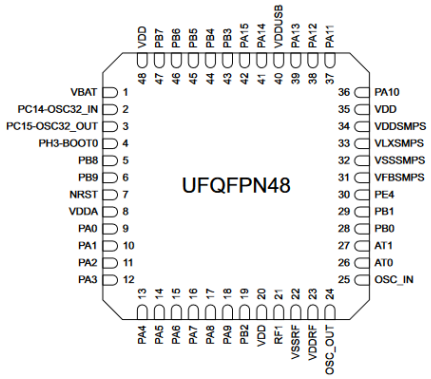


Figure 4.9: STM32WB55RG UFQFPN pinout.

Reference Layout

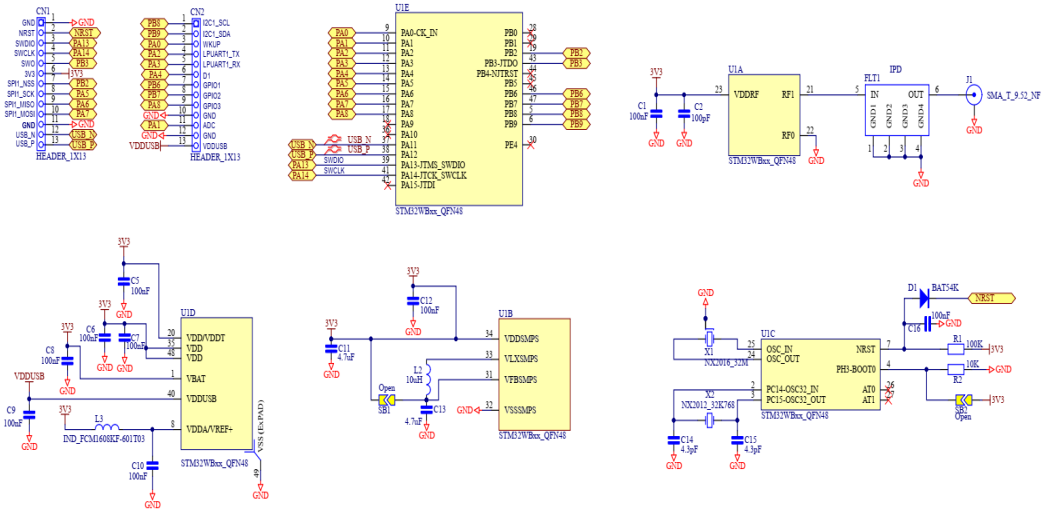


Figure 4.10: STM32WB30RG reference layout.

4.3.2 Analog front end

Instrumentation Amplifier - INA826AIDRGT

The Texas Instruments INA826AIDRGT is a single rail instrumentation amplifier suitable for medical applications. The gain is set by means of a single feedback resistor [7]. It is in charge of:

- Reducing the common mode noise coming from the input electrodes.
- Amplifying the ECG signal.

Main features

- Low Power Consumption.
- High Common-Mode Rejection.
- Single-Supply Operation.
- Reverse Current Protection when Power Switch Off.
- 8-pin SOP package.

Electrical characteristics (T @ $25°C$)

- Supply Voltage: $2.7V$ to $7.5V$.
- Supply Current: $150\mu A$.
- CMRR: $110dB$ ($G = 10V/V$, 50Hz).
- GBW: 34KHz ($G = 10V/V$).
- Output swing: $V_{cc} - 1.1V$.

Pinout and Package Outline

Pin Functions

PIN NAME	PIN NO. SOIC, VSSOP	PIN NO. WSON	I/O	DESCRIPTION
–IN	1	1	I	Negative (inverting) input
+IN	4	4	I	Positive (noninverting) input
REF	6	6	I	Reference input. This pin must be driven by low impedance.
R_G	2	2	—	Gain setting pin. Place a gain resistor between pin 2 and pin 3.
	3	3		
V_{OUT}	7	7	O	Output
$-V_S$	5	5	—	Negative supply
$+V_S$	8	8	—	Positive supply
Thermal pad	—	—	—	Exposed thermal die pad is internally connected to $-V_S$. Connect externally to $-V_S$ or leave floating.

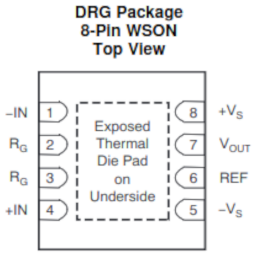


Figure 4.11: INA826AIDRGT SOP pinout.

Operational Amplifier - OPA4330

The OPA4330 from Texas Instruments is a low power, rail-to-rail output swing amplifier. As it is a four parts-per-package operational amplifier, it compacts 4 different ICs in a single 8 pin VQFN package [9]. It is in charge of:

- Taking part in active filters (integrators).
- Amplifying the signals.
- Buffering different stages.

Main features

- Low Power Consumption.
- Low Quiescent Current.
- Single-Supply Operation.
- 8-pin VQFN package.

Electrical characteristics (T @ 25°C)

- Supply Voltage: $1.8V$ to $5.5V$.
- Quiescent Current for amplifier: $35\mu A$.
- CMRR: $110dB$ $(10Hz)$.
- GBW: $350KHz$ $(G = 1V/V)$.
- Output swing: $V_{cc} - 0.1V$.

Pinout and Package Outline

PIN				I/O	DESCRIPTION
NAME	SOIC	TSSOP	VQFN		
–IN A	2	2	2	I	Negative (inverting) input signal, channel A
+IN A	3	3	3	I	Positive (noninverting) input signal, channel A
–IN B	6	6	6	I	Negative (inverting) input signal, channel B
+IN B	5	5	5	I	Positive (noninverting) input signal, channel B
–IN C	9	9	9	I	Negative (inverting) input signal, channel C
+IN C	10	10	10	I	Positive (noninverting) input signal, channel C
–IN D	13	13	13	I	Negative (inverting) input signal, channel D
+IN D	12	12	12	I	Positive (noninverting) input signal, channel D
OUT A	1	1	1	O	Output channel A
OUT B	7	7	7	O	Output channel B
OUT C	8	8	8	O	Output channel C
OUT D	14	14	14	O	Output channel D
V–	11	11	11	—	Negative (lowest) power supply
V+	4	4	4	—	Positive (highest) power supply

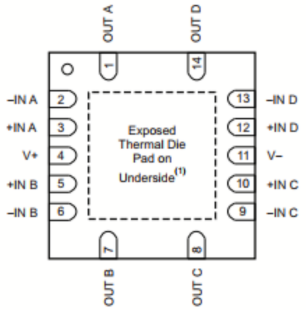


Figure 4.12: OPA4330 VQFN pinout.

Electrodes connectors

Different alternatives have been taken into account for electrodes connectors. Electrodes are hard to be soldered, that's why it has been chosen to fix them on the device and to create a contact with a spring connector.
The alternatives are listed below, but in the end, the choice felt on the shield finger electrode.

Main features

- Harwin S1791-42R Finger Gold Plated [6].
- Wurth 31161452070 Gold.

Pinout and Package Outline

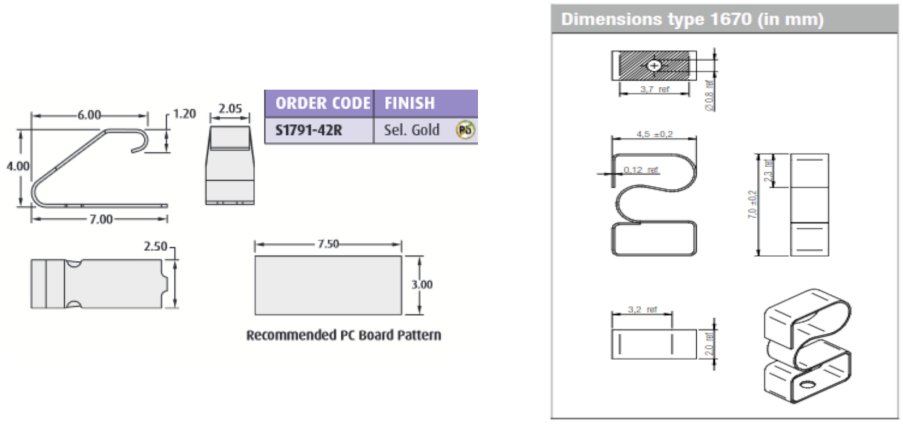


Figure 4.13: Harwin S1791-42R and Wurth 331161452070 layout.

4.3.3 ADC Section

ADC - MCP3564

The Microhip MCP3564 is an 8 channels, 24-bit, delta-sigma analog-to-digital converter with a programmable data rate up to $153ksps$. Its acquisitions can be performed either in single-ended (8 channels) or differential (4 differential channels) mode. Acquisitions might be looped over a user-defined channel sequence. This ADC is available in an ultra-small $3mm$ x $3mm$ UQFN-20 package and can operate over an extended temperature range, from $-40°C$ to $+125°C$. The MCP3564 is in charge of acquiring signals coming from the two input derivations with a μV resolution and a sampling frequency of $1300sps$. Acquired data are then sent to the μC via SPI [14].

Main features

- Four Differential or eight SE input channels.
- 24-bit resolution.
- Programmable Data Rate up to 153.6$ksps$.
- Programmable Gain: 0.33x to 64x.
- Ultra-low Full Shutdown Current consumption.
- Internal Conversions Sequencer (SCAN mode) for automatic multiplexing of input channels.
- Dedicated IRQ pin for easy synchronization.
- 20MHz SPI interface.
- 3mm x 3mm 20-Lead UQFN package.

Electrical characteristics ($T \in [-40{,}125]°C$, CLK Frequency = 4.9152MHz)

- Operating Voltage (AVDD, DVD) up to 3.6V.
- Differential input Voltage range: $\pm\frac{V_{ref}}{Gain}V$.
- Reference Voltage range: $V_{ref}^{+} - V_{ref}^{-}$ up to AVDD.
- Operating Current,
  - Analog: 1.3mA.
  - Digital: 0.37mA.
- Full Shutdown Current,
  - Analog: 0.83μA
  - Digital: 2.4μA.
- Maximum Data Rate at full resolution: 4800Hz.

Pinout and Package Outline

MCP3561	MCP3562	MCP3564	MCP3561	MCP3562	MCP3564	Symbol	Description
20-Lead UQFN			20-Lead TSSOP				
1			3			REFIN-	Inverting Reference Input Pin
2			4			REFIN+	Noninverting Reference Input Pin
3			5			CH0	Analog Input 0 Pin
4			6			CH1	Analog Input 1 Pin
—	5	5	—	7	7	CH2	Analog Input 2 Pin
—	6	6	—	8	8	CH3	Analog Input 3 Pin
—	—	7	—	—	9	CH4	Analog Input 4 Pin
—	—	8	—	—	10	CH5	Analog Input 5 Pin
—	—	9	—	—	11	CH6	Analog Input 6 Pin
—	—	10	—	—	12	CH7	Analog Input 7 Pin
11			13			$\overline{CS}$	Serial Interface Chip Select Digital Input Pin
12			14			SCK	Serial Interface Digital Clock Input Pin
13			15			SDI	Serial Interface Digital Data Input Pin
14			16			SDO	Serial Interface Digital Data Output Pin
15			17			$\overline{IRQ}$/MDAT	Interrupt Output Pin or Modulator Output Pin
16			18			MCLK	Master Clock Input or Analog Master Clock Output Pin
17			19			D_{GND}	Digital Ground Pin
18			20			DV_{DD}	Digital Supply Voltage Pin
19			1			AV_{DD}	Analog Supply Voltage Pin
20			2			A_{GND}	Analog Ground Pin
5, 6, 7, 8, 9, 10	7, 8, 9, 10	—	7, 8, 9, 10, 11, 12	9, 10, 11, 12	—	NC	Not Connected
21			—	—	—	EP	Exposed Thermal Pad, internally connected to A_{GND}

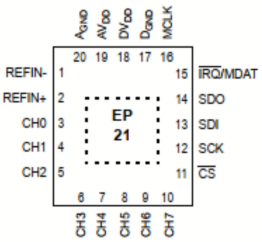


Figure 4.14: MCP3564 UQFN pinout.

Clock generator - 625L3I008M

The CTS 625L3I008M is a low cost, ultra-low voltage clock oscillator supporting CMOS output. It is in charge of providing a stable reference clock for the MCP ADC. When active, the clock generator sinks a lot of current. It is important to keep it shut-down if not used [3].

Main features

- Fundamental and 3rd overtone crystal designs.
- Operating temperature between -40°C and +85°C
- Output-Enable pin.

Electrical characteristics (T @ 25°C)

- Maximum operating Voltage: $4V$.
- Maximum supply current,
  - Active: 15mA.
  - Standby: $10\mu A$
- Quality factor: 10000 - 50000.

Pinout and Package Outline

Pin	Symbol	Function
1	EOH	Enable
2	GND	Circuit & Package Ground
3	Output	RF Output
4	V_{CC}	Supply Voltage

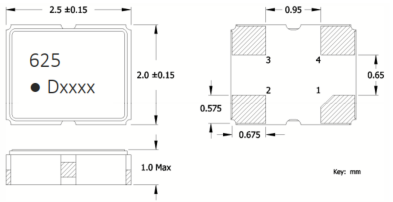


Figure 4.15: 625L3I008M pinout.

4.3.4 Peripherals

JTAG connector - TC2030

The JTAG Connector is used to program the microcontroller. To keep the layout as compact as possible, the JTAG connector consists in six simple conductive contact pads. This way, it covers an area of only 5x5mm^2. The programmer cable is known as PLUG-OF-NAILS cable.

Pinout and Package Outline

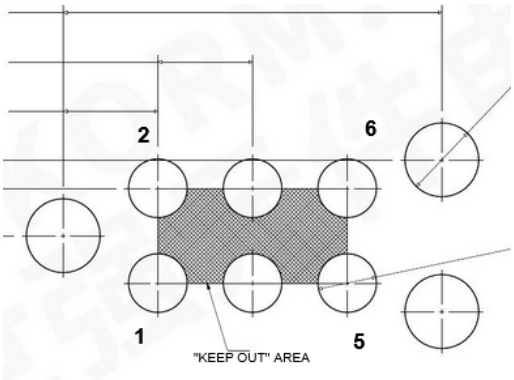


Figure 4.16: TC2030 pinout.

Antenna layout

In order to avoid matching problems related to the use of delicate chip antennas, it has been chosen to implement a patch antenna. Such antenna ensures reliable Bluetooth communication maintaining a compact layout [29].

Main features

- Compact design.
- Single feed.

Electrical characteristics

- Operating Frequency: 2.4-2.5GHz.
- Impedance: 50Ω.
- Averange Gain: 1.95dBi.

Reference Layout

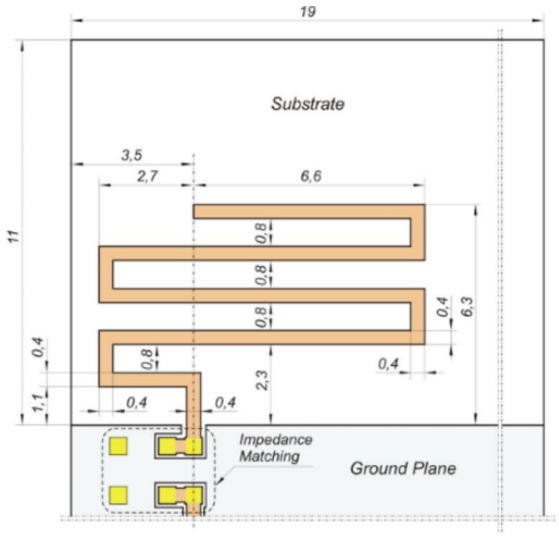


Figure 4.17: Antenna Reference layout

Push button - EVQPUK02K

The push button is in charge of waking up the controller from the deep sleep mode [**?**].

Pinout and Package Outline

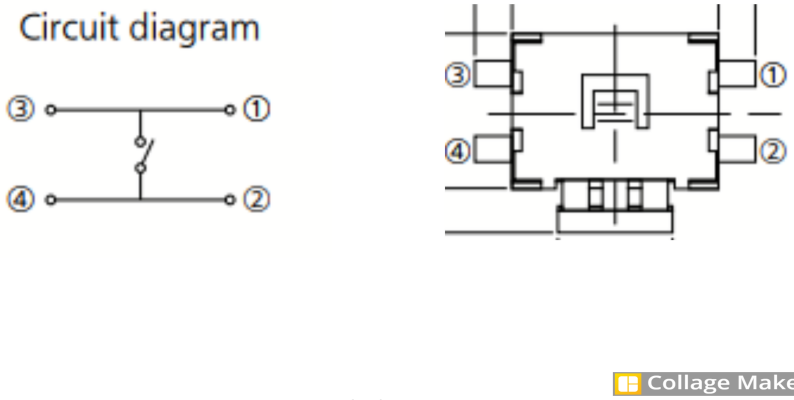


Figure 4.18: Push button pinout.

RGB LED - EAST1616RGBA4

The RGB LED is in charge of highlighting the charge status and operating conditions of the μC. Its anodes are directly managed by the μC to control the power consumption by varying the duty cycle.
Even if the RGB LEDs have different forward operating voltages and currents, it has been assumed as maximum continuous current the lowest value of $6mA$. Consequently, the series resistors have been chosen considering the different forward voltages of each colour and the different supply voltages applied to them [**?**]. The mappings colour-status are:

- BLUE: is ON when the device is active.
- GREEN: is ON when the battery is charging.
- RED: is ON when the battery has low charge.

Main features

- Small 6 pin package
- Individual anodes and cathodes

Electrical characteristics (T @ $25°C$)

- Maximum forward Voltage,
  - R $2.4V$.
  - G $3.8V$.
  - B $3.8V$.
- Maximum forward Current,
  - R $6mA$.
  - G $10mA$.
  - B $10mA$.

  - - -
- Dominant Wavelength,
  - R $624nm$.
  - G $525nm$.
  - B$470nm$.

Pinout and Package Outline

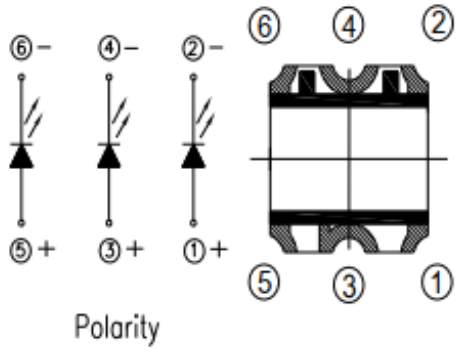


Figure 4.19: RGB LED pinout.

4.3.5 Power Management Section

LiPo battery - HPL402323-2c-190mAh

The MikroE HPL402323 is a Li-Polymer Battery for low power portable devices providing $3.7V$ and $190mA$ [15].

Electrical characteristics (T @ $25°C$)

- Connector: JST-SHR-02V-S.
- Nominal Voltage: $3.7V$.
- Voltage at the end of discharge: $3.0V$.
- Charging voltage: $4.2 \pm 0.03V$.
- Capacity: $190mAH$.
- Standard charge $0.5CA$.
- Fast charge $1CA$.
- Standard discharge $0.2CA$.

Pinout and Package Outline

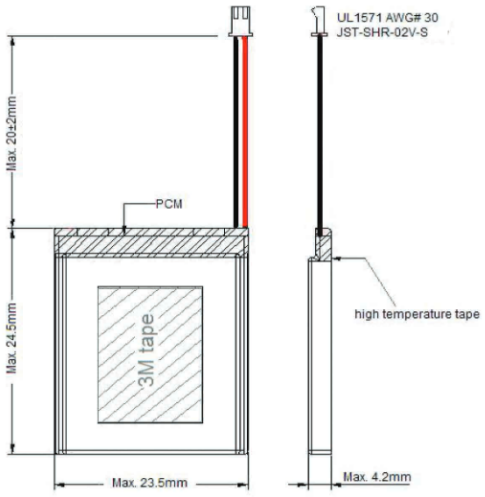


Figure 4.20: Li-Po Battery pinout.

Battery Charger - MAX1555

The Maxim MAX1555 is in charge of charging the single-cell LiPo battery through the USB source. It operates with input voltages up to $7V$ and it embeds the CHG pin used to detach the supply when in charge. When CHG goes to a high-impedance state, the battery is fully charged and the charge current falls below $50mA$ [10].

Main features

- Charge from USB and/or AC Adapter.
- Automatic switch over when AC Adapter is plugged IN.
- On-Chip thermal limiting simplifying board design.
- Charge status indicator.
- 5-pin Thin SOT23 package.

Electrical characteristics ($T \in [0,85]°C$)

- DC Voltage range: $3.7V$ to $7.0V$.
- USB Voltage range: $3.7V$ to $6.0V$.
- DC to BAT Voltage range: $0.1V$ to $6.0V$.
- BAT regulation Voltage: $4.158V$ to $4.242V$.
- Max DC supply Current: $3mA$.
- Max DC charging Current: $340mA$.

Pinout and Package Outline

PIN	NAME	FUNCTION
1	USB	USB Port Charger Supply Input. USB draws up to 100mA to charge the battery. Decouple USB with a 1µF ceramic capacitor to GND.
2	GND	Ground
3	$\overline{POK}$	Power-OK Active-Low Open-Drain Charger Status Indicator. $\overline{POK}$ pulls low when either charger source is present (MAX1551 only).
	$\overline{CHG}$	Active-Low Open-Drain Charge Status Indicator. $\overline{CHG}$ pulls low when the battery is charging. $\overline{CHG}$ goes to a high-impedance state, indicating the battery is fully charged, when the charger is in voltage mode and charge current falls below 50mA. $\overline{CHG}$ is high impedance when both input sources are low (MAX1555 only).
4	DC	DC Charger Supply Input for an AC Adapter. DC draws 280mA to charge the battery. Decouple DC with a 1µF ceramic capacitor to GND.
5	BAT	Battery Connection. Decouple BAT with a 1µF ceramic capacitor to GND.

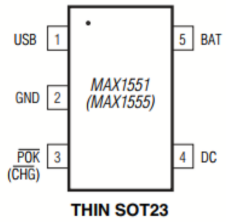


Figure 4.21: MAX1555 SOT23 pinout.

Reference Layout

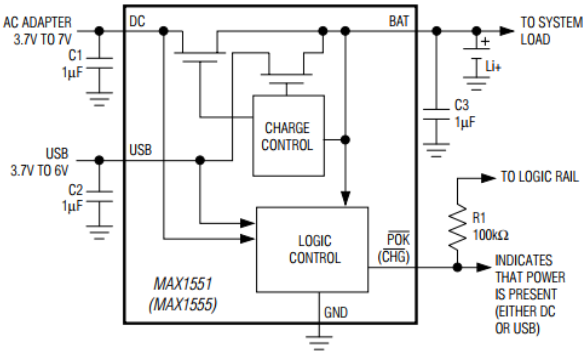


Figure 4.22: MAX1555 reference layout

Voltage Regulator - MAX1759

The Maxim MAX1759 is in charge of generating a regulated output voltage from a single cell LiPo battery. It provides a fixed voltage ($3.3V$) to all the circuit, accepting an input voltage up to $5.5V$. It embeds a shutdown pin to turn off the device during charging [11].

Main features

- Regulated output Voltage,
  - Fixed 3.3V
  - Adjustable 2.5V to 5.5V).
- Shutdown mode.
- Load disconnected from input in shutdown.
- Short-Circuit protection and thermal shutdown.
- Small 10-Pin μMAX package.

Electrical characteristics ($T \in$ [0,85]$^{\circ}C$)

- Input Voltage Range: $1.6V$ to $5.5V$.
- Output Voltage: $3.17V$ to $3.43V$.
- Minimum Output Current: $100mA$.
- Quiescent Supply Current (Vin $> 4V$): $90\mu A$.
- Shutdown Supply Current: $5\mu A$.
- Efficiency: 90%.
- Switching Frequency: 1.2-1.8MHz.

Pinout and Package Outline

PIN	NAME	FUNCTION
1	POK	Open-Drain Power-OK Output. POK is high impedance when output voltage is in regulation. POK sinks current when V_{FB} falls below 1.1V. Connect a 10kΩ to 1MΩ pull-up resistor from POK to V_{OUT} for a logic signal. Ground POK or leave unconnected if not used. POK is high impedance in shutdown.
2	$\overline{SHDN}$	Shutdown Input. Drive high for normal operation; drive low for shutdown mode. OUT is high impedance in shutdown.
3, 4	IN	Input Supply. Connect both pins together and bypass to GND with a ceramic capacitor (see *Capacitor Selection* section).
5	GND	Ground. Connect GND to PGND with a short trace.
6	PGND	Power Ground. Charge-pump current flows through this pin.
7	CXN	Negative Terminal of the Charge-Pump Transfer Capacitor
8	CXP	Positive Terminal of the Charge-Pump Transfer Capacitor
9	OUT	Power Output. Bypass to GND with an output filter capacitor.
10	FB	Dual-Mode Feedback. Connect FB to GND for 3.3V output. Connect to an external resistor divider to adjust the output voltage from 2.5V to 5.5V.

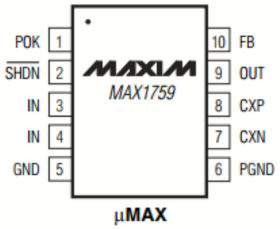


Figure 4.23: MAX1759 μMAX pinout.

Reference Layout

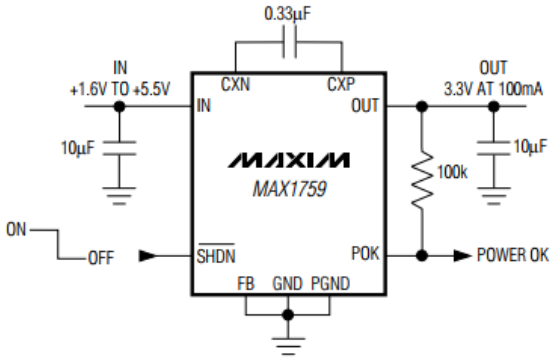


Figure 4.24: MAX1759 Reference layout

Voltage Reference - REF2025

The Texas Instruments REF2025 is in charge of providing two different voltage references, $1.25V$ and $2.5V$, to the analog circuitry. Since it is used as unipolar supply voltage, it is useful to translate the signals around 1.25V. It also provides the Reference voltage to the ADC. It embeds a shutdown pin [8].

Main features

- Two outputs, $VREF$ and $\frac{VREF}{2}$, for convenient use in single-supply systems.
- High initial accuracy: $\pm 0.05\%$ (maximum).
- $VREF$ and $VBIAS$ tracking overtemperature .
- Microsize SOT23-5 package.

Electrical characteristics (T @ $25°C$)

- Supply Voltage: -0.6V to 6V.
- Low dropout voltage: 10mV.
- High output current: $\pm 20mA$.
- Low quiescent current,
  - Active mode: 360μA.
  - Shutdown mode: 5μA.

Pinout and Package Outline

PIN		I/O	DESCRIPTION
NO.	NAME		
1	V_{BIAS}	Output	Bias voltage output (V_{REF} / 2)
2	GND	—	Ground
3	EN	Input	Enable (EN ≥ V_{IN} – 0.7 V, device enabled)
4	V_{IN}	Input	Input supply voltage
5	V_{REF}	Output	Reference voltage output (V_{REF})

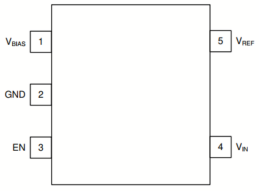


Figure 6-1. DDC Package SOT23-5 (Top View)

Figure 4.25: REF2025 SOT23-5 pinout.

Reference Layout

Figure 12-1 shows an example of a PCB layout for a data acquisition system using the REF2030. Some key considerations are:

- Connect low-ESR, 0.1-µF ceramic bypass capacitors at V_{IN}, V_{REF}, and V_{BIAS} of the REF2030.
- Decouple other active devices in the system per the device specifications.
- Using a solid ground plane helps distribute heat and reduces electromagnetic interference (EMI) noise pickup.
- Place the external components as close to the device as possible. This configuration prevents parasitic errors (such as the Seebeck effect) from occurring.
- Minimize trace length between the reference and bias connections to the INA and ADC to reduce noise pickup.
- Do not run sensitive analog traces in parallel with digital traces. Avoid crossing digital and analog traces if possible, and only make perpendicular crossings when absolutely necessary.

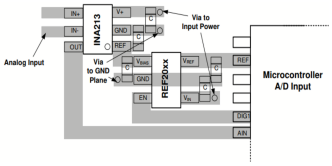


Figure 4.26: REF2025 Reference layout

High side MOSFET switch - BD2242G

The BD2242G is a low on-resistance high-side power switch MOSFET. It embeds over-current detection, thermal shutdown and soft-start. The range of Current limit threshold can be adjusted up to $1.7A$ [25]. It is in charge of reducing power consumption during μC hibernation. An high side switch has been chosen as it must deliver power to all the analog devices.

Main features

- Adjustable Current Limit Threshold: $0.2A$ to $1.7A$.
- Output Discharge Function.
- Thermal Shutdown.
- Reverse Current Protection when Power Switch Off.
- Small 6-pin SSOP package.

Electrical characteristics (T @ $25°C$)

- IN Operating Voltage: $2.8V$ to $5.5V$.
- On Resistance: (VIN=$5V$) $89m\Omega$(Typ).
- Adjustable Current Limit Threshold: $0.2A$ to $1.7A$.
- Standby Current: $0.01\mu A$ (Typ).

Pinout and Package Outline

Pin No.	Symbol	I/O	Function
1	IN	I	Switch input and the supply voltage for the IC.
2	GND	-	Ground.
3	EN	I	Enable input. High-level input turns on the switch (BD2222G, BD2242G) Low-level input turns on the switch (BD2243G)
4	/OC	O	Over-current notification terminal. Low level output during over-current or over-temperature condition. Open-drain fault flag output.
5	ILIM	O	Current limit threshold set Pin. External resistor used to set Current limit threshold. Recommended 11.97 kΩ ≤ R_{LIM} ≤ 106.3 kΩ
6	OUT	O	Power switch output.

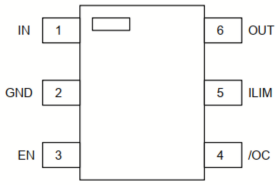


Figure 4.27: BD2242G SSOP pinout.

Reference Layout

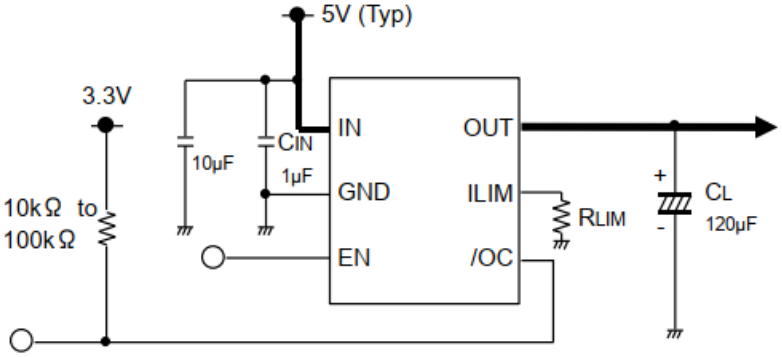


Figure 4.28: BD2242G Reference layout

Mechanical switch - TE Connectivity 1825232-1

The TE is a slide Switch in charge of turning on or off the device. It is a SPDT Through Hole, Right Angle switch [2].

Pinout and Package Outline

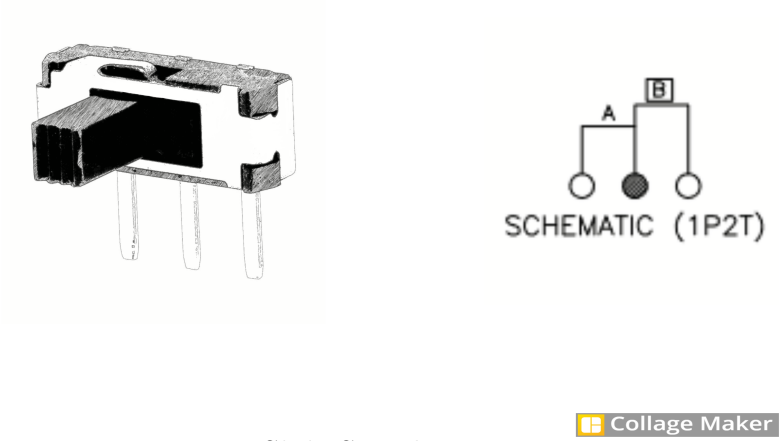


Figure 4.29: Slide Switch pinout.

4.3.6 Bill of Materials

Schematic parts have been characterized with custom properties in order to speed up BOM generation. Specifically, the following properties have been added:

- Description;
- Distributor and Distributor Part Number;
- Manufacturer and Manufacturer Part Number;
- Unit Price;

The resulting total price of the bill of material is 70€ including tax.

4.4 Printed Circuit Board design

The PCB is very small as it must fit inside a wristwatch. They layout consists in a circle with a radius of $2cm$. Considering that the battery is placed under the PCB, the thickness is $6mm$.
Component's density is high, and the analog part is cumbersome. Future developments may rely on a completely digital filtering.
The implementation of a double layer PCB would have been difficult. A traditional four layer design with ground and supply planes in the middle has not been possible as well, as there is a high number of ground vias. In addition, the analog part references to a different supply net, AVDD.
For these reasons, it has decided to make the PCB according to the following stackup:

- Top Layer: analog front end and power unit.
- Layer 1: digital GND.
- Layer 2: analog GND.
- Bottom Layer: digital part (μC, ADC and peripherals)

Note that in practice the top layer actually faces down, right above the battery.

The antenna has been put on the upper part of the circuit, as close as possible to the microcontroller to avoid any parasitics. Below the antenna, there is no ground plane.

The microcontroller has been placed at the center of the circuit to facilitate interconnections to other components.
The number of traces in the GND planes has been kept at minimum to avoid high current density points. The two GND planes are elettrically connected.
The charger connector and the buttons have been positioned at the outline of the layout to facilitate their use.
A large housing has been dedicated in front of the LiPo connector to ease the connection.
The following pages show:

- Top layer: complete view and routing only.
- Layer 1: digital GND.
- Layer 2: analog GND.
- Bottom Layer: complete view and routing only.
- PCB renders Top and Bottom.

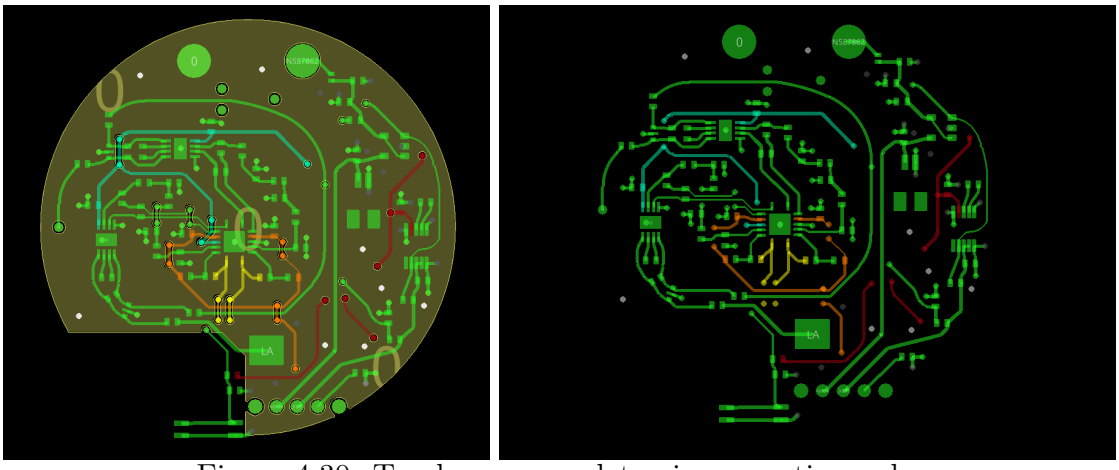


Figure 4.30: Top layer: complete view - routing only.

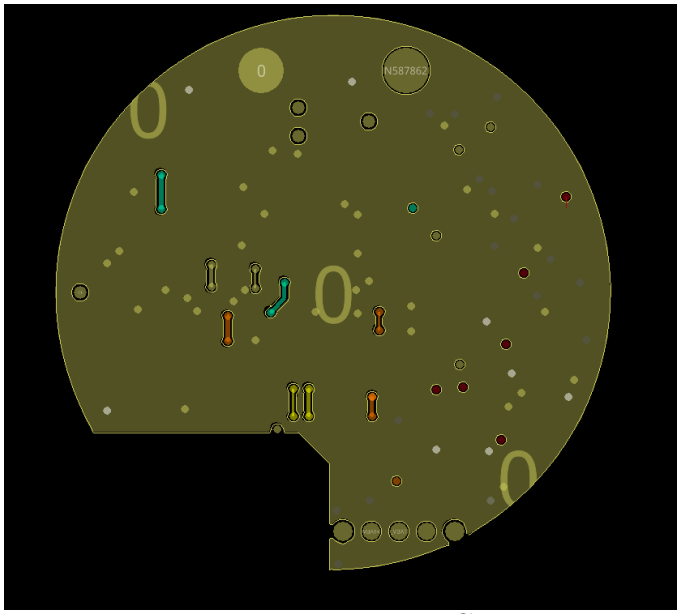


Figure 4.31: Layer 1: GND.

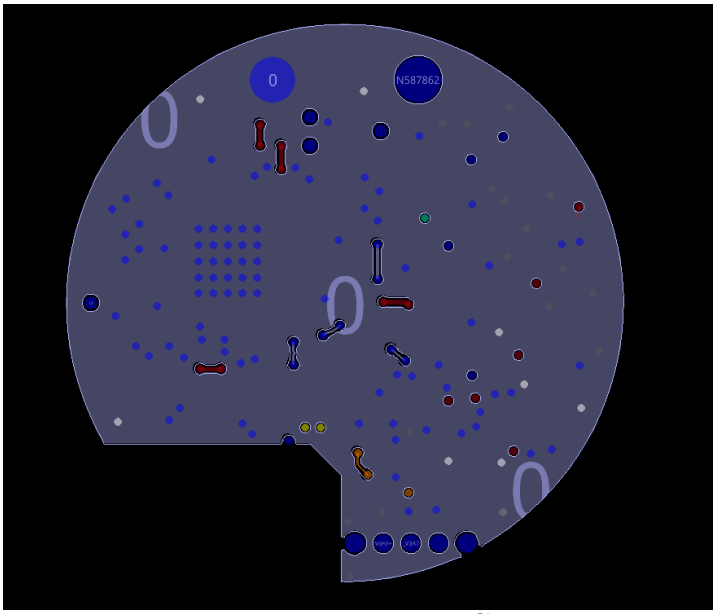


Figure 4.32: Layer 2: GND.

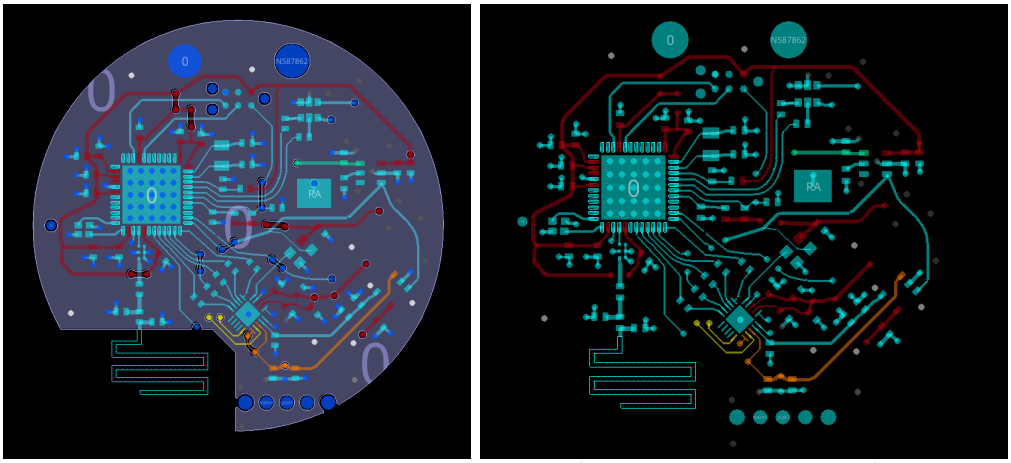


Figure 4.33: Bottom Layer: complete view - routing only

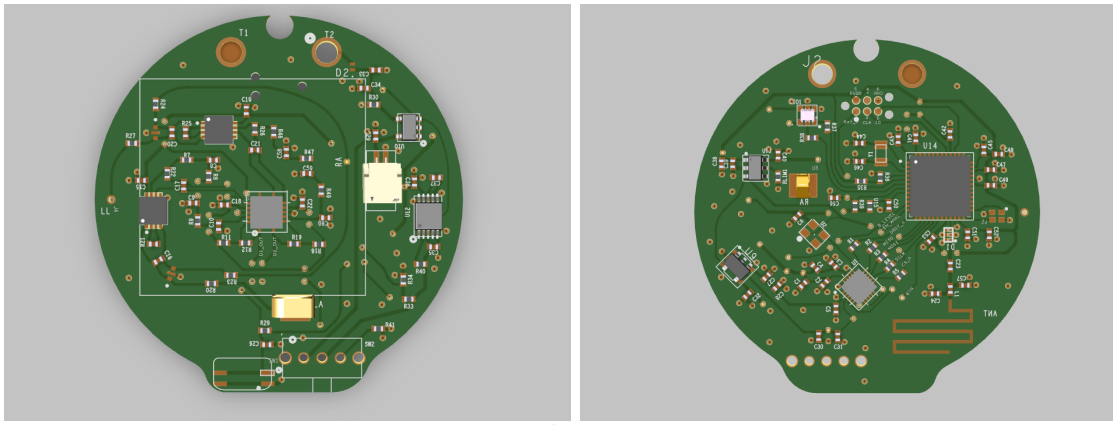


Figure 4.34: Renders: Top - Bottom

4.5 Mockup

This six-leads wearable device represents a breakthrough in ECG monitoring, revolutionizing the way cardiovascular health is assessed outside of traditional clinical settings. Its ergonomic design prioritizes user comfort and ease of use, ensuring optimal adherence and long-term wearability.
After the PCB design, the mock-up is going to be realized. As anticipated, an electrode will be placed in contact with the wrist, another one on the upper shell and the last one under the strap, to be placed on the left leg for a complete six-leads ECG.
Below is the final device concept made using Autodesk's fusion360 CAD.

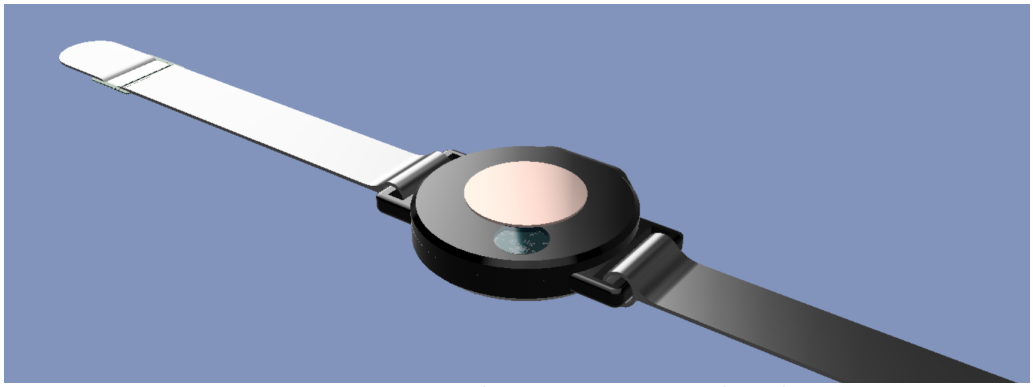


Figure 4.35: Mock-up: smart wristband

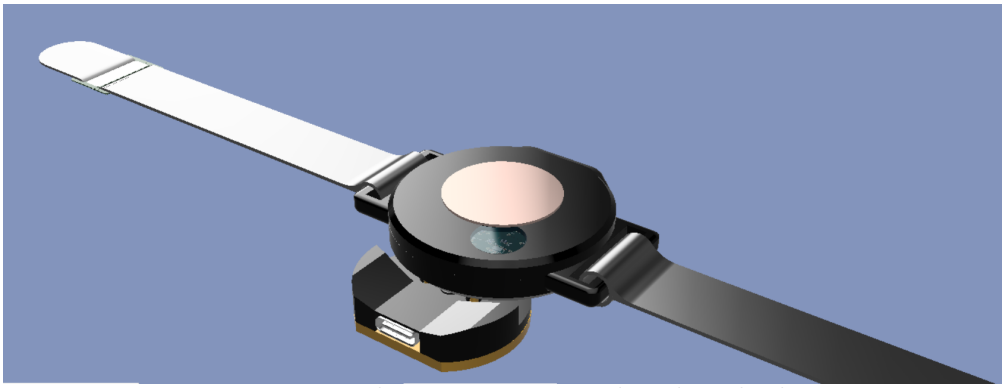


Figure 4.36: Mock-up: smart wristband with charger

Chapter 5

Firmware

This chapter focuses on firmware, written for the STM32WB30 microcontroller in the C programming language.
The primary objective of this firmware is to facilitate the acquisition of high-quality ECG signals, and to establish a reliable communication link with the smartphone. To accomplish this, the firmware must effectively manage the communication between the analog front end, and the digital components of the device.

One of the key components is the 24-bit analog-to-digital converter (ADC), which is responsible for the conversion of the analog ECG signal into a digital one. The communication between the ADC and the μC occurs via Serial Peripheral Interface protocol (SPI). The firmware must implement the necessary SPI functionalities to enable efficient data transfer, ensuring accurate and reliable signal acquisitions.

Furthermore, the firmware plays a critical role in establishing and managing the Bluetooth connection with the smartphone. This involves implementing the Bluetooth Low Energy protocol stack (BLE), which enables energy-efficient communication over short distances. In addition, the firmware must handle tasks such as device discovery, connection establishment, and data transmission/reception, adhering to the relevant Bluetooth profiles and specifications.
The development of firmware for a Bluetooth medical wearable device presents various challenges, from limited resources (memory and processing power), through support for real-time data visualization on the smartphone, to data storage and retrieval.

This chapter aims to dig into the technical aspects of firmware development, emphasizing the integration of Bluetooth communication, the management of the analog front end via SPI, and the insurance of accurate and reliable acquisition of ECG signals.

It is through the development of efficient firmware that it is possible to advance in the field of wearable healthcare technology, ultimately improving subject monitoring and healthcare outcomes.

5.1 Code overview

As the device relies on the ST's STM32WB30 microcontroller, a large part of the initialization of hardware peripherals is performed using the STMCubeIDE, an integrated development environment (IDE) provided by ST Microelectronics that offers a wide range of features and tools to simplify the initialization of hardware components.
STMCubeIDE provides a graphical interface and a toolchain (compiler and linker) for the cross-compilation of embedded systems. It provides support to write the code, compile, program and debug/run the target application.
This development environment also offers a device configuration tool called MX, which takes care of generating the project (Makefile), the libraries, and the hardware initialization procedures.
The source files of the generated project are then available in the environment, and the resulting firmware is compatible with bare metal systems.
In these systems, the application is the only software on board. Therefore, a series of start-up procedures are linked with the user application to initialize the hardware.
In the FLASH, there are the interrupt vectors, the application code, and the constants. In the RAM, there are the stack and the variables.
As there is no bootloader, all hardware initialization tasks must be performed by the application itself, which performs the initialization of GPIO, SPI, Timer, and of the internal ADC. In addition, the application is in charge of generating the code for handling interrupts and microcontroller resources, and for configuring the Bluetooth stack by means of special APIs enabling a fast management of Bluetooth connections, services, characteristics, and data exchange between the board and connected devices.

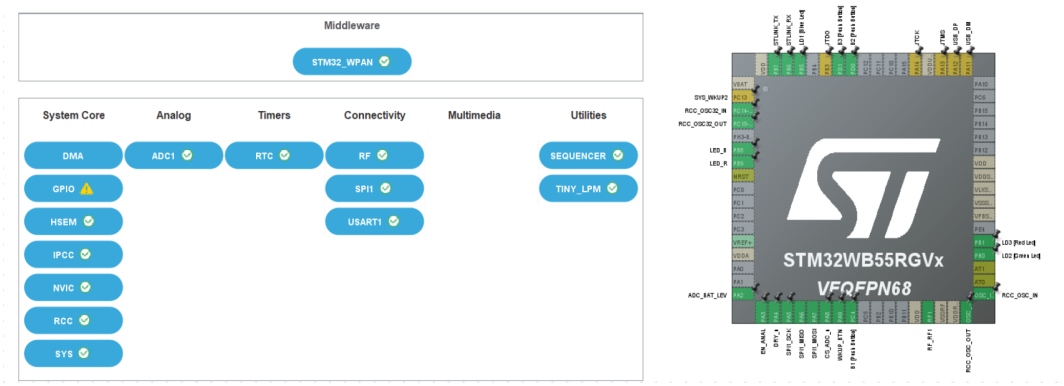


Figure 5.1: STMCubeMX configuration and peripherals.

The chapter focuses more on the description of the functions and application-dependent modules, implemented to obtain an harmonious operation of the system's various sub-parts.
It is important to understand that the user code is managed via a scheduler, called SEQ (sequencer), which takes care of sequentially executing the tasks set by the user according to different priorities [26].

The core modules that make up the firmware deal with 3 main aspects:

1. Management of the core operations and events.
2. Management of the ADC via SPI.
3. Management of the Bluetooth.

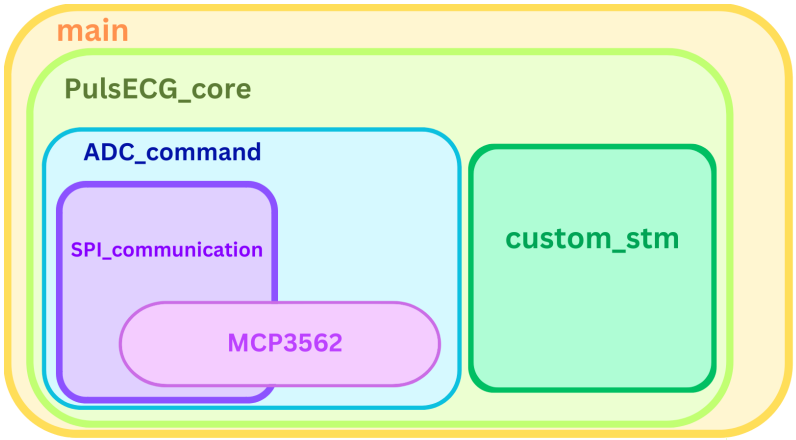


Figure 5.2: Modules hierarchy.

5.2 Application core

While enabling of peripheral is performed by the STMCubeIDE development tool, peripheral initialization takes places in the **main.c** file. The following figure lists the init procedures making up the main file.

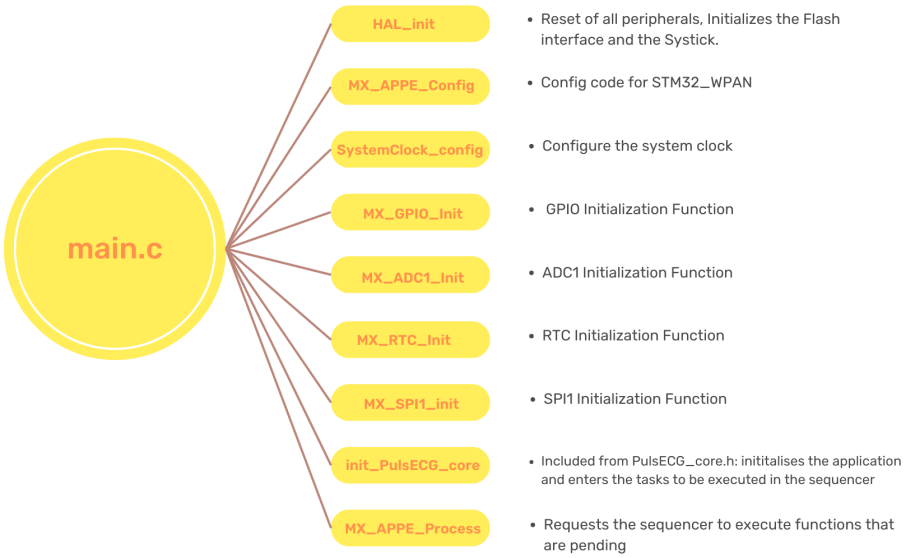


Figure 5.3: Main file functions.

At power-up, the firmware takes care of setting up the hardware and the basic functionalities. Variables are initialized, and tasks that will be executed are fed into the scheduler. It is at this time that the *init_PulsECG_core* function is executed.

The **PulsECG_core** module contains the body of the application. This module takes care of orchestrating the operation of the device. There are basically two tasks to be performed:

- Monitoring of the battery charge state.
- Starting and stopping the acquisition of ECG signals.

Upon reception of an operation from the central device via Bluetooth, just as a write or a read, the module drives the sequencer to execute the correct function.
If the request is a battery read, the sequencer executes the *send_BLevel* function once. This function sends via Bluetooth the battery charge percentage.
Notice that this value is periodically updated by initiating a battery voltage sampling through the microcontroller's internal ADC, as defined in the *update_BLevel* function.
If the request is a read of the ECG signal, then a sequence of functions is executed: *start_stop_acquisition* receives as input the time frame for which the acquisition lasts. It enables the ADC's clock and analog power supply, it adjusts the status LED and it starts the acquisition of ECG signals by sending the appropriate opcode to the ADC. This causes the *acquire* and *sendSample* functions to repeatedly enter the scheduler for the entire duration of the acquisition.
As the name suggests, *acquire* receives the converted sample from the ADC via SPI, while *sendSample* sends the samples, almost in real time, to the Central device.
Functions to be executed by the sequencer must be declared in the **app_conf.h** file. In this way, an ID that maps the function in the scheduler can be defined.
There are two lists of IDs, one for tasks that require bluetooth and one for those that do not communicate with it.
IDs allow to use the API provided by the scheduler to insert or remove functions from the list of tasks to be executed.

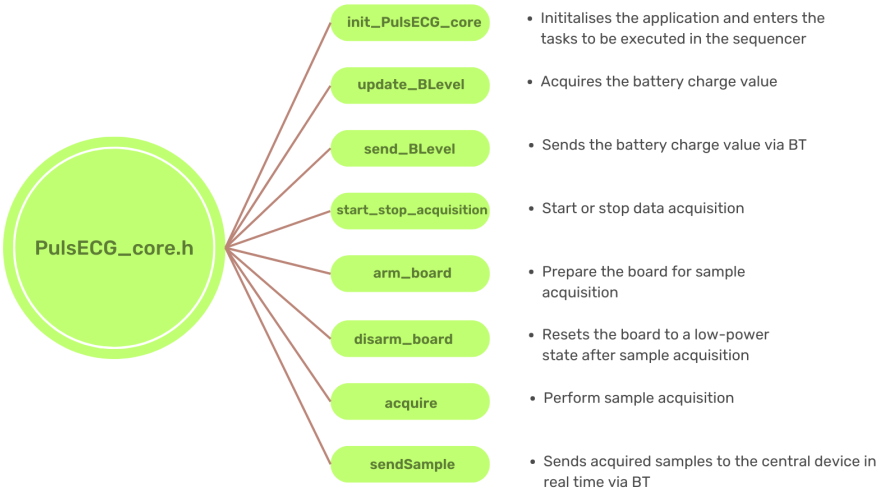


Figure 5.4: PulsECG_core file functions.

This module handles the operations that are performed by the device, but to obtain the desired operations from the peripherals, it is necessary to format the requests correctly. This is why the different operations pass through different modules: **SPI_communication**, **ADC_command**, **MCP3562** and **custom_stm**.

5.3 SPI interface

The Serial Peripheral Interface is one of the most common communication protocols. It is widely adopted due to its simplicity, versatility, and high-speed capabilities. It allows for the exchange of data between a master device (typically a microcontroller or a microprocessor) and one or multiple slave devices (peripheral devices such as sensors, displays, ADCs, DACs, memory chips, etc.).
SPI operates then in a master-slave architecture, where the master device initiates and controls the communication. The master device generates clock pulses and selects the slave device with which it wishes to communicate. The data transfer occurs simultaneously in both directions (full-duplex), meaning that the master can send data to the slave while receiving data from it, or vice versa.

The SPI interface consists of four main lines:

- SCLK (Serial Clock). This line provides the clock signal generated by the master device. The clock rate is configurable, and it determines the speed of data transfer.
- MOSI (Master Out Slave In). This line is used by the master to send data to the selected slave device. It carries the data bits serially, with each bit synchronized with the clock signal.
- MISO (Master In Slave Out). This line is used by the master to receive data from the slave device. Again, the slave sends data bits serially on this line, synchronized with the clock signal.
- SS/CS (Slave Select/Chip Select). This line is used by the master to select a specific slave device with which it wants to communicate. By activating the SS/CS line for a particular slave, the master establishes a communication channel with that device, deactivating all other slave devices.

SPI supports different configurations and settings, which can be tailored according to the specific requirements of the embedded application. Some of the commonly configurable parameters include:

- Clock Polarity (CPOL). It determines the idle state of the clock line. It can be set either high or low, indicating whether the clock is idle when high or low respectively.
- Clock Phase (CPHA). It determines the timing and shifting of data sampling. It can be set either to the first or the second edge of the clock cycle.
- Bit Order. SPI supports two modes of data transmission: MSB (Most Significant Bit) first or LSB (Least Significant Bit) first. The selection depends on the specific device data format.
- Data Frame Size. The frame size specifies the number of bits of each data frame. Usually, 8 bits are transmitted, but it depends on the capabilities of the devices which are involved in the communication.

The following figure reports the settings used for the the SPI communication.

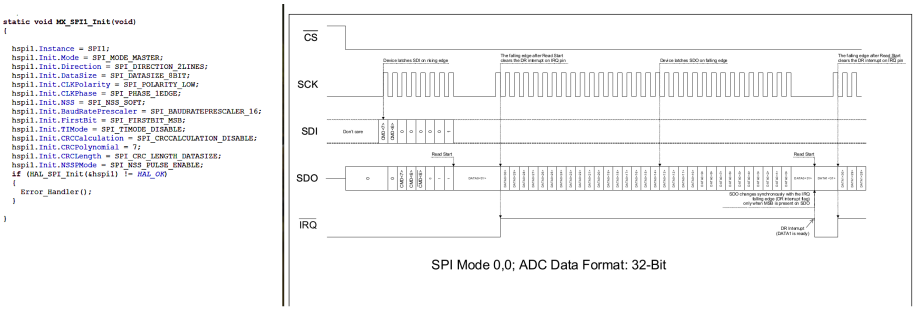


Figure 5.5: SPI configuration and timing diagram.

The module in charge of managing the SPI communication with the ADC is the **SPI_communication** module. It implements the functions needed to send bit sequences, read the bits received as a response or acknowledge and, if necessary, convert them into a format understandable to the developer.

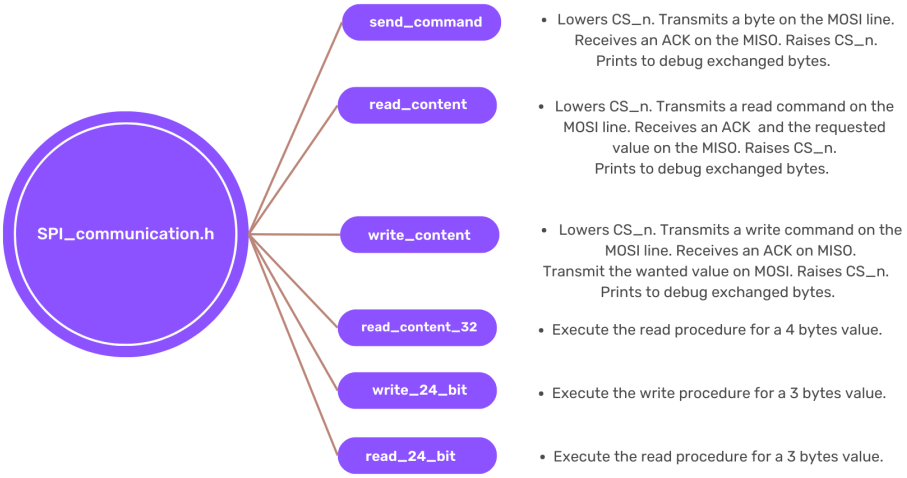


Figure 5.6: SPI_communication file functions.

5.3.1 ADC configuration

Correct communication with the ADC is essential, but it is necessary to know how to format the bit sequences sent via SPI, and the meaning of the sequences of 1s and 0s sent. This information can be found on the device datasheet. By carefully reading the specifications, it is possible to know which registers to write and what to write into them. It follows an overview of the MCP registers and their configuration.

- Configuration Registers (CONFIGx). They are used to configure various settings of the ADC. The microcontroller uses them to set parameters such as gain selection, operating mode, clock source, conversion mode, and filter options. There are 4 configuration registers, each of which is 8 bits wide.

- Scan Register (SCAN). In SCAN mode, the device sequentially and automatically converts a list of predefined differential inputs in a defined order. This mode is useful for applications that require constant monitoring of defined channels. This register is 24-bit.

- Data Register (ADCDATA). The Data Register holds the converted digital sample. After a conversion is completed, the microcontroller can read the Data Register to retrieve the acquired measurement for further processing and analysis. Data can be formatted in 24 or 32 bits modes depending on the DATA_FORMAT.

- Calibration Registers. The MCP3564 includes calibration registers that store calibration values for various parameters just as offset and gain. These registers can be accessed and updated by the microcontroller during the calibration process to ensure accurate and reliable measurements. There are 2 calibration registers of 24 bits each [14].

The **ADC_command** module takes care of properly formatting the commands to be sent and of arranging them in the correct sequence to achieve the desired operation. In addition, the declaration file **MCP3564.h** includes all the opcodes needed for the correct setting of the ADC registers.

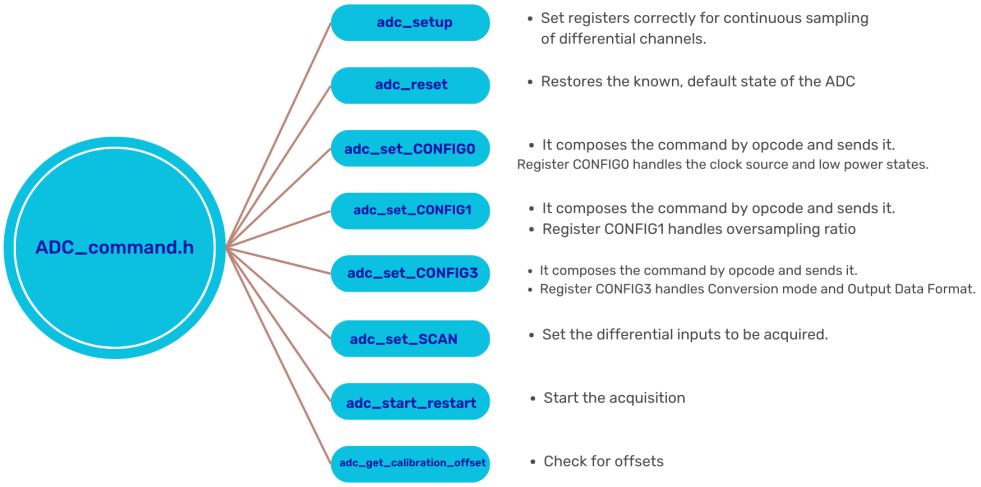


Figure 5.7: ADC_command file functions.

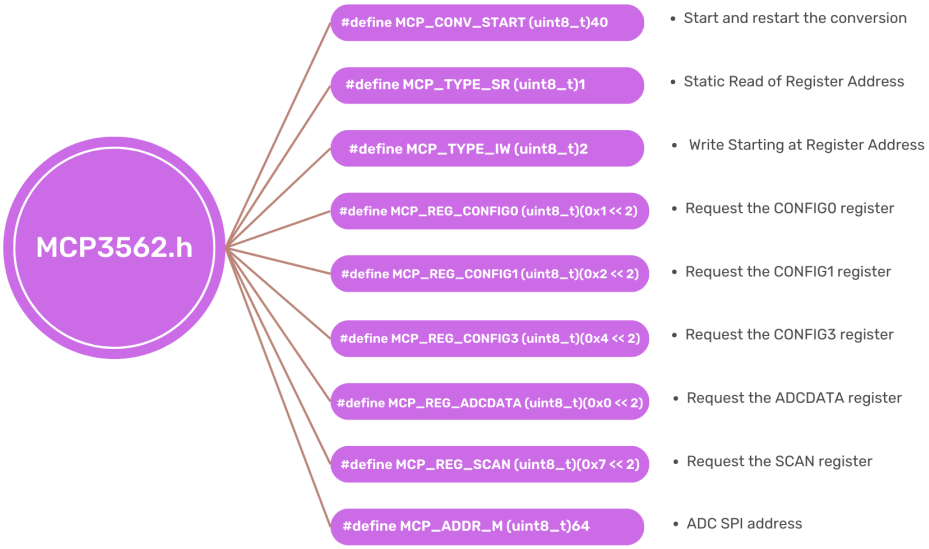


Figure 5.8: MCP3562 opcode.

The following lines report an example of the command structure used to request a write operation to the CONFIG0 register:

CMD[7]	CMD[6]	CMD[5]	CMD[4]	CMD[3]	CMD[2]	CMD[1]	CMD[0]
Device Address Bits		Register Address/Fast Command Bits				Command Type Bits	

```
#define MCP_ADDR_M (uint8_t)64
#define MCP_REG_CONFIG0 (uint8_t)(0x1 << 2)
#define MCP_TYPE_IW (uint8_t)2
uint8_t cmd=0;
cmd = (cmd | MCP_ADDR_M | MCP_REG_CONFIG0 | MCP_TYPE_IW);
```

5.4 Bluetooth LE

The last implemented communication protocol is the Bluetooth low energy. BLE has emerged as a key wireless communication protocol for low-power, short-range applications. It is particularly suited for battery-operated devices, such as wearable medical devices and IoT devices, where energy efficiency and data transmission with smartphones or other compatible devices are essential, offering a seamless connectivity and data rate up to 2 Mbps.
The protocol operates in the 2.4 GHz ISM band, providing low-power connectivity, and it uses frequency-hopping spread spectrum technology to mitigate interference and ensure reliable communication.
The STM32WB architecture separates the BLE profiles and application, running on the CPU1, from the real-time aspects residing in the BLE peripheral.
The BLE peripheral incorporates a CPU2 processor containing the stack which handles the link layers up to the Generic Access Profile (GAP) layer. It also incorporates the physical 2.4 GHz radio. Instead, the application CPU1 collects and computes the data to be transferred to the BLE [27].
BLE devices operate in two primary roles: the central role and the peripheral role. The protocol supports short-range communication, typically up to 10 meters, depending on the environment.

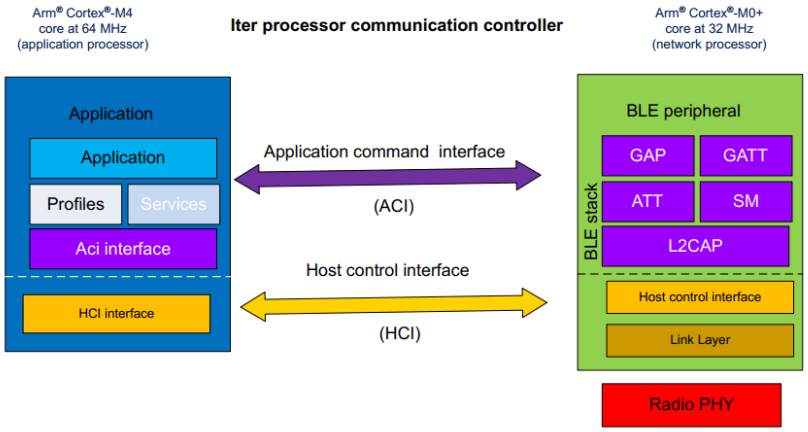


Figure 5.9: Generic Attribute Profile.

Generic Attribute Profile (GATT)

The Generic Attribute Profile (GATT) in BLE manages the communication between a client and a server device. GATT defines a hierarchical structure consisting of services, characteristics, and descriptors to organize and exchange data. The GATT client initiates requests to read or write data or subscribe to notifications/indications, and the GATT server responds accordingly.

Services and Characteristics

Services. Services in GATT represent a collection of related data and behaviors. They define a specific functionality or feature of a device. Services can be predefined by the Bluetooth or custom-defined by developers. Examples of predefined services include the Heart Rate Service, Battery Service, and Device Information Service. Each service has a unique 128-bit Universally Unique Identifier (UUID) that identifies it.

Characteristics. Characteristics are the fundamental data elements within a service. They represent a specific piece of data or a control point for the device. Characteristics have properties, such as read, write, notify, and indicate, which define how they can be accessed or modified. They also have a UUID to uniquely identify them within a service.

Attribute Protocol (ATT). GATT uses the Attribute Protocol to manage the exchange of data between the client and server devices. ATT defines the format and rules for accessing and manipulating characteristics and other attributes. It

uses Attribute Handles and Attribute Types (UUIDs) to identify and differentiate attributes.

Descriptors. Descriptors provide additional information or configuration options for characteristics. They provide metadata about characteristics, such as user-friendly descriptions, measurement units, or client configuration settings. Descriptors are optional and can be associated with a characteristic to enhance its functionality or to provide contextual information.

In this specific case, the device is set up with a Server profile offering a customised experience set as follows.

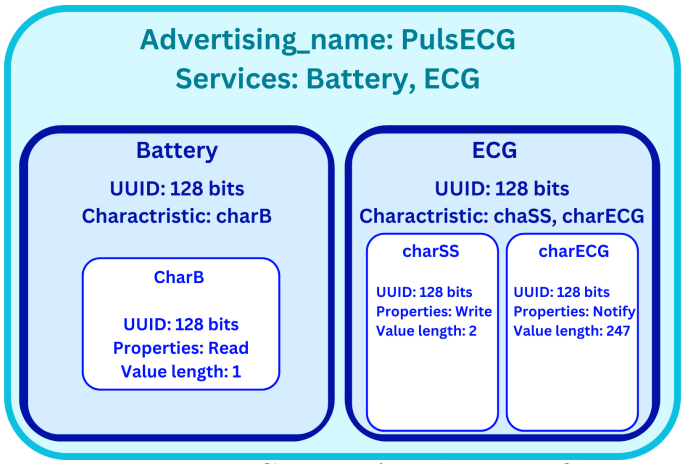


Figure 5.10: Generic Attribute Profile.

As it can be seen from the picture above, there are two services: Battery and ECG.

- The Battery service offers only one characteristic, charB, which is responsible for receiving requests to read the battery status.
- The ECG service offers two characteristic:.
  1. charSS takes care of receiving a write request with the time frame for which the ECG signal is to be acquired.
  2. charECG notifies the acquired samples to the central device and is characterised by a much larger payload than the usual 20 bytes transmitted as notification.

The management of requests from the Bluetooth central device and the exchange

of information is managed by the **custom_stm** module, whose main functions are shown below.

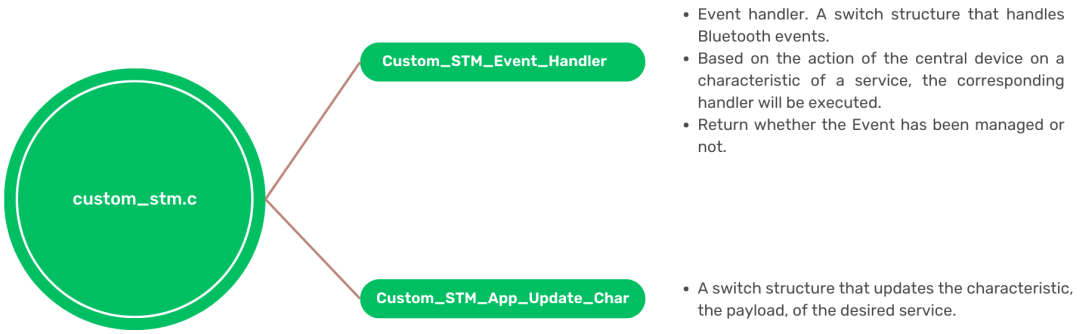


Figure 5.11: custom_stm file functions.

When an event is received, the *Custom_STM_Event_Handler* function is executed. An ID is associated with the event. It maps the service and characteristic to which the event belongs.
The ID is then compared with a switch against the possible verifiable events and, depending on which characteristic generated it, certain functions are executed.
In the firmware, upon reception of a write to the charSS characteristic, the *start_stop_acquisition* function of the **PulsECG_core** module is executed to start or stop the acquisition of ECG signals.
In addition, an opcode is set to generate longer payloads, up to 247bytes. This is done via the following lines of code:

```
uint32_t ACI_GATT_NOTIFICATION_EXT_EVENT = 0x00400000;
aci_gatt_set_event_mask(ACI_GATT_NOTIFICATION_EXT_EVENT);
aci_gatt_exchange_config(attribute_modified->Connection_Handle);
hci_le_set_data_length(attribute_modified->Connection_Handle, 200, (200+14)*8);
                                                    //TxTime = (payload + 14)*8
```

This reduces the overhead transmitted with each packet and provides 10 times more datarate to be able to send the sampled data in realtime without the need to store it on the device. Moreover, the central device must be alerted to the new payload size with the function:

```
hci_le_write_suggested_default_data_length(200, (200+14)*8); //TxTime = (payload + 14)*8
```

On the other hand, if a read request is received on the charB characteristic, the *send_BLevel* handler of the **PulsECG_core** module is triggered, which calls the *Custom_STM_App_Update_Char* function and passes it the updated battery value.
This function takes care of updating the characteristic related to a certain service. In fact, it is the same function that updates the characteristic charECG of the service ECG.
The following lines show how the characteristics related to charB (20 bytes) and charECG (200 bytes) are updated in the function *Custom_STM_App_Update_Char*.

```
tBleStatus Custom_STM_App_Update_Char(Custom_STM_Char_Opcode_t CharOpcode, uint8_t *pPayload)
{
  tBleStatus ret = BLE_STATUS_INVALID_PARAMS;
  int ret2=0;

  switch (CharOpcode)
  {
    case CUSTOM_STM_CHARB:
      ret = aci_gatt_update_char_value(CustomContext.CustomBatteryHdle,
                                       CustomContext.CustomCharbHdle,
                                       0, /* charValOffset */
                                       SizeCharb, /* charValueLen */
                                       (uint8_t *)  pPayload);

     case CUSTOM_STM_CHARECG:
      /* USER CODE BEGIN CUSTOM_STM_App_Update_Service_2_Char_2*/
      ret = aci_gatt_update_char_value_ext( 0x0000, CustomContext.CustomEcgHdle,
                    CustomContext.CustomCharecgHdle,
                    0x00, /* DO NOT NOTIFY payload received*/
                    200, /* charValueTotLen */
                      0, /* charValOffset */
                     20, /* charValueLen */
                    (uint8_t *)  pPayload);

      ret2= aci_gatt_update_char_value_ext( 0x0000, CustomContext.CustomEcgHdle,
                    CustomContext.CustomCharecgHdle,
                    0x01, /* NOTIFY payload received*/
                    200, /* charValueTotLen */
                     20, /* charValOffset */
                    180, /* charValueLen */
                    (uint8_t *)(pPayload + 20));

    default:
      break;
  }
  return ret;
}
```

Chapter 6

Digital signal processing

In the realm of Electrocardiography, Digital Signal Processing techniques have become essential in extracting meaningful information from raw ECG data, aiding the detection and characterization of cardiac abnormalities.
This chapter focuses on the application of DSP techniques for ECG signal reconstruction, starting from data retrieval from the peripheral device, via Bluetooth using Python scripts, to digital filtering algorithms, with a special emphasis on their implementation and effectiveness.
Leveraging on the versatility of the Bash command "bluetoothctl", it is possible to establish a connection with the peripheral, retrieve the ECG data, and store that data for further processing. This initial step of data retrieval sets the basis for the subsequent DSP operations, enabling the reconstruction of the six different ECG signals from the two acquired from the peripheral.
The reconstruction process involves DSP algorithms that employ mathematical techniques to recreate signals, eliminate noise and highlight relevant features for accurate interpretation. In addition, special attention is paid to the aspect of digital filtering, which is crucial for noise reduction and signal enhancement.
Digital filtering techniques serve as key tools for improving the accuracy and reliability of ECG signal reconstruction. By selectively attenuating or eliminating unwanted noise components, while preserving the essential features of the ECG waveform, digital filters contribute significantly to the improvement of diagnostic capabilities. The chapter explores various types of digital filters, including low-pass, high-pass, notch filters, FIR and convolution, discussing their principles and implementation in Python. The effectiveness of these filters in reducing noise artifacts and enhancing the fidelity of the reconstructed ECG signals is demonstrated.
From the retrieval of ECG data via BLE to the digital signal filtering techniques, this chapter aims to provide readers with a comprehensive understanding of the DSP pipeline for ECG signal reconstruction.

6.1 Signals retrieval

To display and process data on the central device, it is necessary to communicate via Bluetooth with the wristband.
At the moment, communication is established via a python program. This program uses the linux shell to call the "bluetoothctl" command and obtain data from the device.
Python has several libraries for Bluetooth communication, but not all of them support BLE, and none of them support notifications reception with payloads greater than 20bytes. For this reason, the bluetoothctl command has been used to obtain the data via pipe and then process it.
The bluetoothctl command scans for nearby devices and, if there is a wristband, a connection is estabilished. Notifications are enabled for the charECG feature of the ECG service and a write command is sent, with payload equal to the desired acquisition time, to the charSS feature of the same service. Afterwards, the program listens and receives the sampled signals from lead I and II.
Of course, it is also possible to request a read to the Battery service to obtain information on the charge state of the battery.
Once the samples have been retrieved, it is possible to proceed as introduced in chapter 2.1.2. Having divided the samples into two distinct signals, lead I and lead II, it is possible to apply a mathematical equation to obtain the remaining signals: lead III, aVL, aVR and aVF.
As shown in the code snippet below, once all six signals have been obtained, filtering can be applied to improve their representation.

```
lead_III, aVR, aVL, aVF=[],[],[],[]

for II, I in zip(lead_II, lead_I): lead_III.append(II-I)

for I, III in zip(lead_I, lead_III): aVL.append((I-III)/2)

for I, II in zip(lead_I, lead_II): aVR.append(-(I+II)/2)

for II, III in zip(lead_II, lead_III):  aVF.append((II+III)/2)

#Filtering ----------------------------------------

lead_I = filtering(lead_I)
lead_II = filtering(lead_II)
lead_III = filtering(lead_III)

aVL = filtering(aVL)
aVR = filtering(aVR)
aVF = filtering(aVF)
```

6.2 Digital filters

As for analog filtering, the primary objective of digital filtering in ECG processing is to remove unwanted noise and artifact contributions while preserving the essential features of the ECG waveform. Various noise sources contribute to ECG signal distortion: Powerline Interference, muscle activity, electrode movement, and motion artifacts, as comprehensively reported in chapter 2.3. These disturbances can significantly degrade the signal quality and hinder accurate interpretation. Digital filtering algorithms offer the ability to selectively attenuate or eliminate unwanted noise components, offering a cleaner and more reliable ECG data.

```
def filtering(lead):
    global samp_freq
    lead = filters.zeroing(lead)
    lead = filters.notch(lead, 50, 5)
    lead = filters.LPF( lead, 30, order=3)
    lead = filters.HPF( lead, 0.5, order=3)
    lead = filters.Hanning_convolution(lead, int(samp_freq/80))
    lead = filters.moving_average(lead, int(samp_freq/80))
    lead = new_filter.smooth_signal(lead, samp_freq,polyorder=5 )
    #lead= filters.moving_average(lead, int(samp_freq/300))
    _mean= filters.moving_average(lead, int(samp_freq))
    lead = lead[:len(_mean)] - _mean
    return lead
```

It follows a detailed discussion of the different types of applied digital filters and their contributions to the raw data of the six different derivations.

6.2.1 Notch filter

The first type of filter which has been applied is a notch filter operating at a frequency of 50Hz. Being a band-elimination filter operating over a very narrow range of frequencies, a quality factor of 5 has been chosen.
This filter is introduced to eliminate residual contributions of interference from the power supply in a frequency range already attenuated by analog filtering.

```
def notch(noisySignal, notch_freq, Q):

    quality_factor = Q # Quality factor

    # Design a notch filter using signal.iirnotch
    b_notch, a_notch = signal.iirnotch(notch_freq, quality_factor, samp_freq)

    return signal.filtfilt(b_notch, a_notch, noisySignal, method="gust")
```

The snippet uses the iirnotch[1] function from the scipy.signal module to design a notch filter operating in the digital domain. The resulting numerator coefficients are assigned to b_notch, and the denominator coefficients are assigned to a_notch. These coefficients define the transfer function of the notch filter [23].
The filtfilt function applies the designed notch filter to the noisySignal. The filtfilt function performs zero-phase digital filtering by applying the filter in both the forward and reverse directions, canceling out phase distortion [22]. The resulting filtered signal is returned as output.

6.2.2 Low pass filter

The second type of filter applied is a third order low pass filter operating at a frequency of 30Hz. This filter has been introduced to reinforce the attenuation of noise outside of the band of interest.

[1]It implements an Infinite Impulse Response, a type of digital filter that has feedback, resulting in an impulse response that extends indefinitely. The output depends not only on the current input but also on past inputs and outputs.

```
def LPF( data, cutoff, order):

    def butter_lowpass_filter(data, cutoff, fs, order=4):

        b, a = butter(order, cutoff, fs=fs, btype='low', analog=False)
        y = signal.filtfilt(b, a, data)
        return y

    fs = samp_freq       # sample rate, Hz

    y = butter_lowpass_filter(data, cutoff, fs, order)
    return y
```

The Butter function is called to design a Butterworth low pass digital filter. The Butterworth filter is characterized by a maximally flat magnitude response in the passband and a gradual roll-off in the stopband [21].
The filtfilt function is used to apply the filter to the input signal [22].

6.2.3 High pass filter

Similarly, a high-pass filter is applied to accentuate the work done by the analog front end. The cut-off frequency is 0.5Hz, the order is 3 and it is a Butterworth filter.

```
def HPF(data, cutoff, order):

    fs = samp_freq       # sample rate, Hz

    b, a = butter(order, cutoff, fs=fs, btype='highpass', analog=False)
    y = signal.filtfilt(b, a, data)

    return y
```

The butter function is called to design a high-pass Butterworth digital filter [21].
The filtfilt function is used to apply the designed filter to the input signal [22].

6.2.4 Hanning convolution

The Hanning window acts as a weighting function that applies a smooth tapering to the input signal. The window coefficients gradually increase from zero at the edges to one in the center, forming a bell-shaped curve. This tapering minimizes the abrupt changes at the edges of the input signal, reducing spectral leakage in frequency analysis and mitigating artifacts caused by edge effects in convolution.
The convolution operation combines each element of the Hanning window with the corresponding elements of the input signal, emphasizing the central portion of the window where the coefficients are higher. This procedure mitigates the effect of sudden, low-amplitude variations, such as noise, and contributes to obtain a cleaner, smoother signal.

```
def Hanning_convolution(vector, windowSize):
    window = np.hanning(windowSize)
    window = window / window.sum()

    # filter the data using convolution
    filtered = np.convolve(window, vector, mode='valid')
    return filtered
```

The 'windowSize' parameter of the 'Hanning_convolution' function's sizes the Hanning window, which determines the length of the filter.
The Hanning window is generated using the 'np.hanning(windowSize)' function. The Hanning window is a symmetric windowing function that tapers the edges of the input signal to reduce artifacts in the frequency domain.
The window coefficients are then normalized to ensures that the overall gain of the filter remains approximately unitary.
The 'np.convolve' function is used to perform a linear convolution between the normalized Hanning window and the input signal [18]. The 'mode='valid" parameter ensures that the output size matches the valid region of the convolution, excluding any edge effects.

6.2.5 Moving average

A moving average filter is a simple yet effective technique used for smoothing signals, removing high-frequency noise and emphasize the underlying trends, useful to extract the ECG signals features.
The moving average filter works by averaging a set of adjacent samples in the input signal to produce the filtered output. This filter also smooths out the signal

by reducing the impact of rapid fluctuations or high-frequency components.
The cumulative sum operation accumulates the values of the input signal, where each element in the resulting array represents the sum of all preceding samples. This cumulative sum is used to efficiently calculate the moving average by taking the difference between cumulative sums of shifted vectors.
Dividing the calculated sum by 'n' provides the average value for the window, effectively smoothing the input signal.

```
def moving_average(vector, n) :
    ret = np.cumsum(vector, dtype=float)
    ret[n:] = ret[n:] - ret[:-n]
    return ret[n - 1:] / n
```

The 'np.cumsum' function calculates the cumulative sum of the input signal. The resulting array contains the cumulative sum values at each index [19].
The second line calculates the difference between the cumulative sums of the vector shifted by 'n' positions. This operation effectively computes the sum of 'n' consecutive samples starting from the 'n'th position.
The last line divides the calculated sum by 'n' to obtain the moving average for each corresponding window.

6.2.6 Savitzky-golay filter

The Savitzky-Golay filter performs a sort of local polynomial regression for smoothing the input signal. It estimates the local trend of the data within each window by fitting a polynomial of the specified order to the data points.
By using least-squares regression, the filter determines the polynomial coefficients that provide the best fit to the local data points. The polynomial coefficients are then used to compute the smoothed output values.
The Savitzky-Golay filter is particularly useful for smoothing signals while preserving important features, such as peaks or edges, within the data. It can effectively reduce noise and remove high-frequency variations while maintaining the overall shape and characteristics of the signal.

```
def smooth_signal(data, sample_rate, window_length=None, polyorder=3):

    #Function that smooths data using savitzky-golay filter using default settings.

    if window_length == None:
        window_length = sample_rate // 10

    if window_length % 2 == 0 or window_length == 0: window_length += 1

    smoothed = savgol_filter(data, window_length = window_length,
                             polyorder = polyorder)

    return smoothed
```

The code first checks if the window length is specified, then, if the length is zero or an even number, the length is incremented by 1 to ensure an odd window length is used, as the Savitzky-Golay filter requires an odd window length.
Next, the savgol_filter function applies the Savitzky-Golay filtering algorithm to the input signal, fitting a polynomial of the specified order to the data within each window. In this way, the polynomial coefficients are determined, using least-squares regression, and then the smoothed output signal is returned [24].
Its contribution is very influential so it will not be implemented definitively but rather can be thought of as a beautification filter.

6.3 Final results

This section presents the acquisitions of the six ECG signals for five different subjects. The acquisitions were carried out with the device studied so far on internal volunteer members of the research team.
The plots show the signals filtered through the analog front end, sampled with the 24-bit ADC, and transmitted real time to the central device via BLE. Signals are then processed in order to obtain all six leads and improve their readability through additional digital filtering.
Signals acquired with the prototype come from different subjects. As it can be observed, some waveforms show electrodes motion artifacts, but the overall achieved results are acceptable.

6.3.1 Subject 1

Male, age 25, good health.

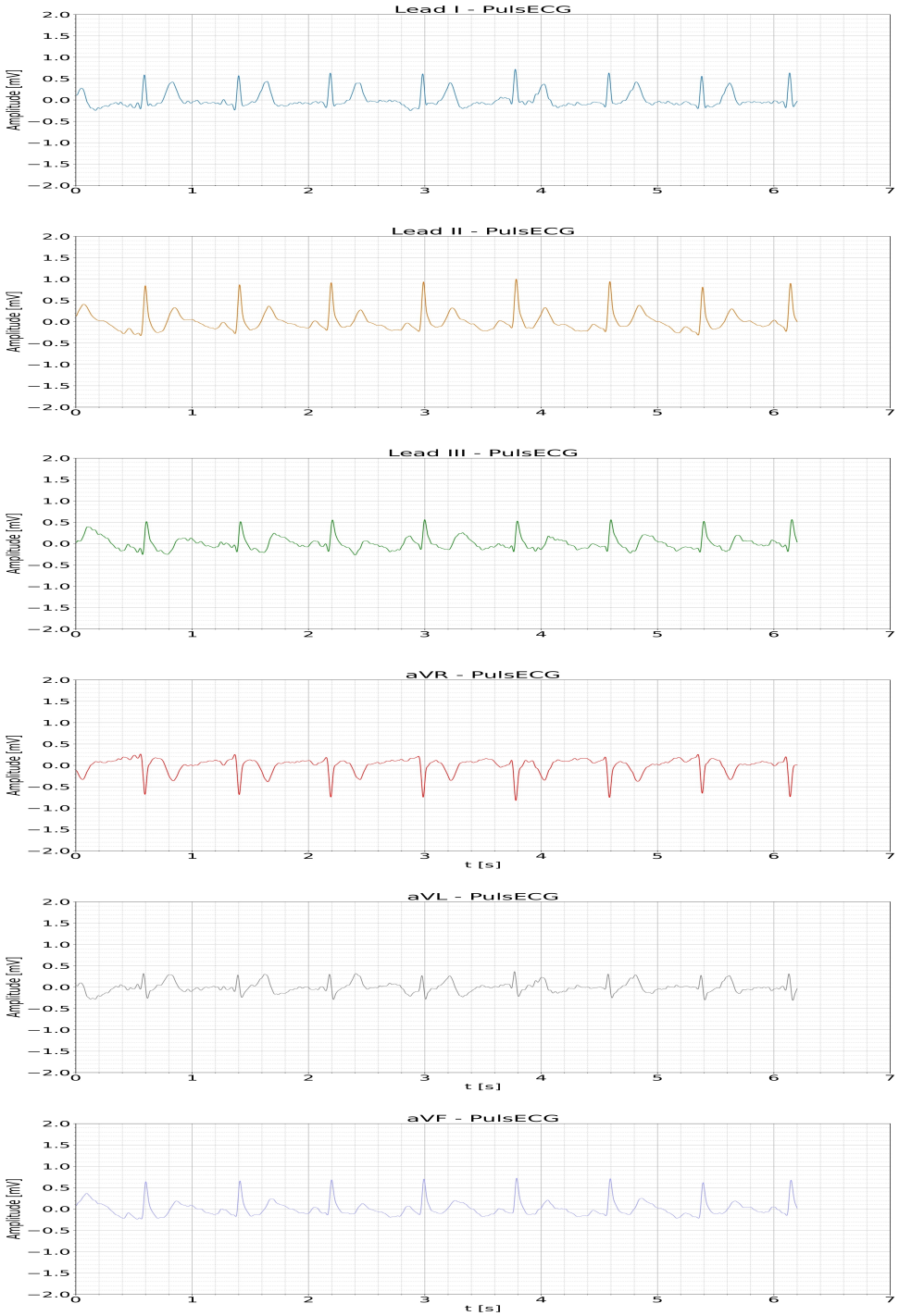


Figure 6.1: Subject 1 ECG.

6.3.2 Subject 2

Female, age 23, good health, suspected extrasystole detected on lead II and III.

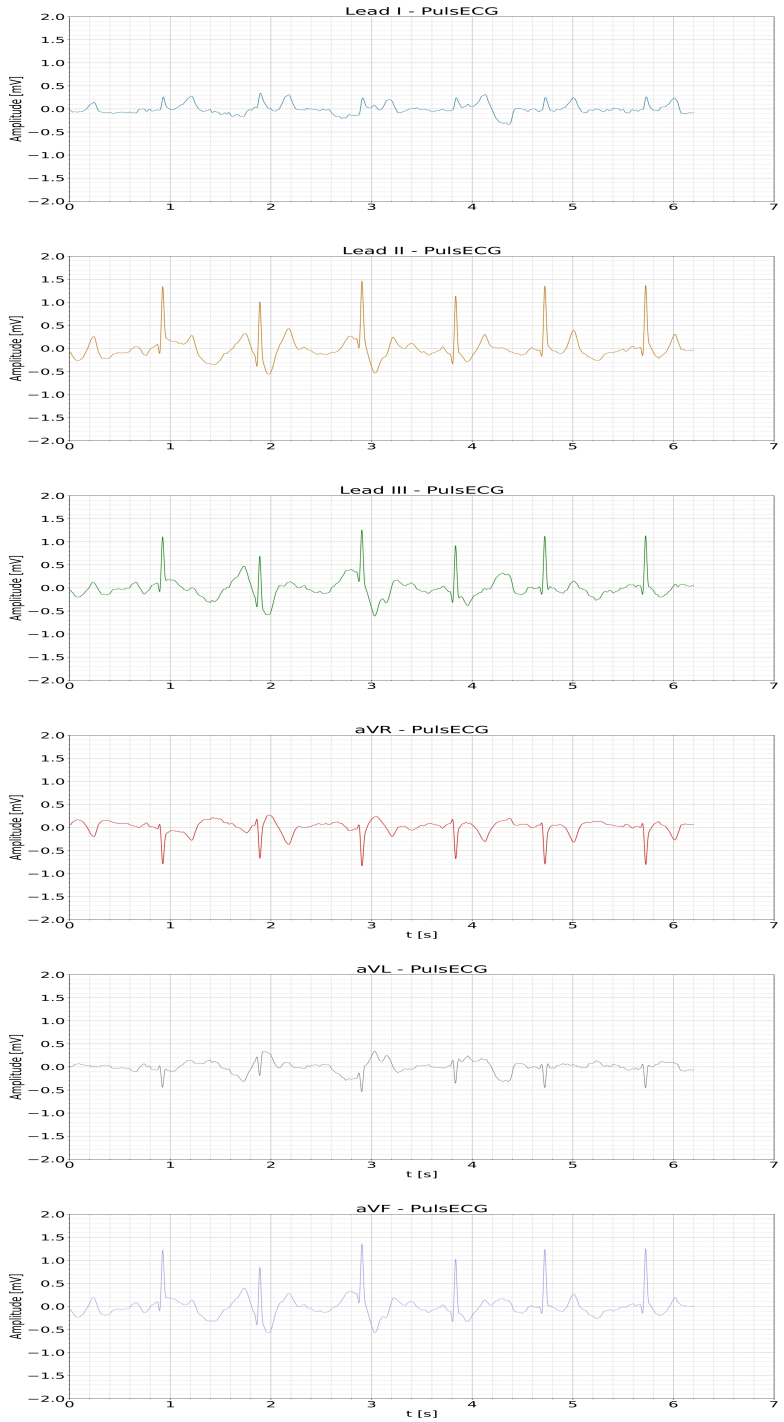


Figure 6.2: Subject 2 ECG.

6.3.3 Subject 3

Male, age 33, good health.

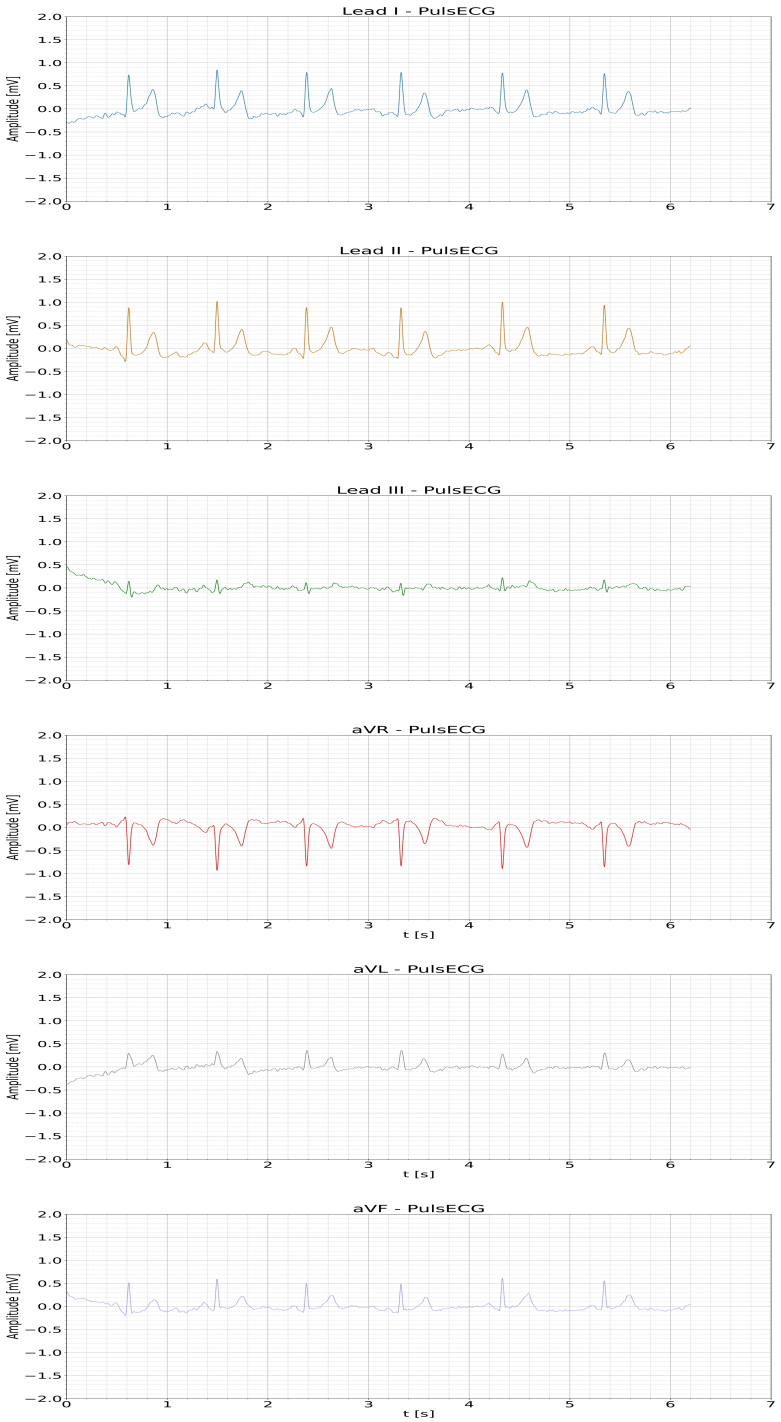


Figure 6.3: Subject 3 ECG.

6.3.4 Subject 4

Female, age 30, good health.

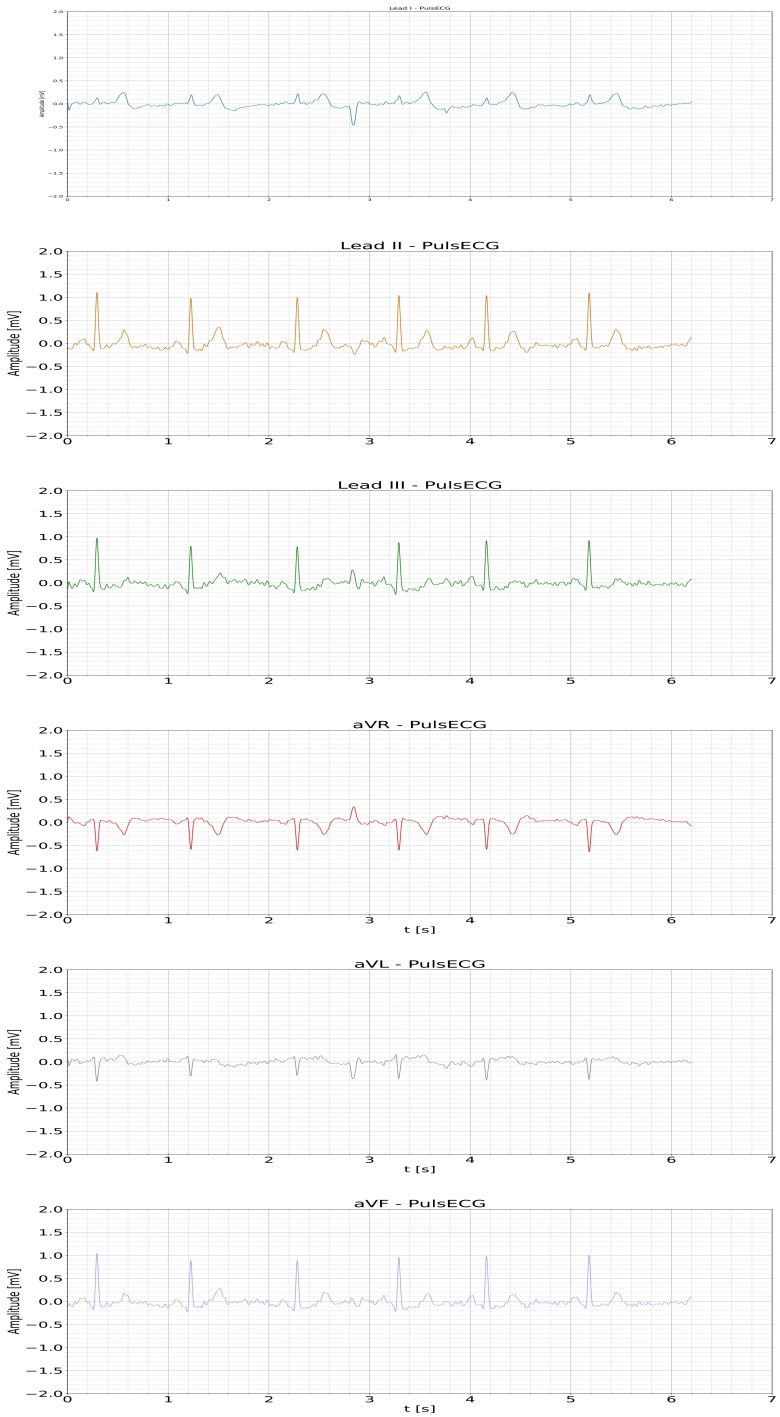


Figure 6.4: Subject 4 ECG.

6.3.5 Subject 5

Female, age 62, good health.

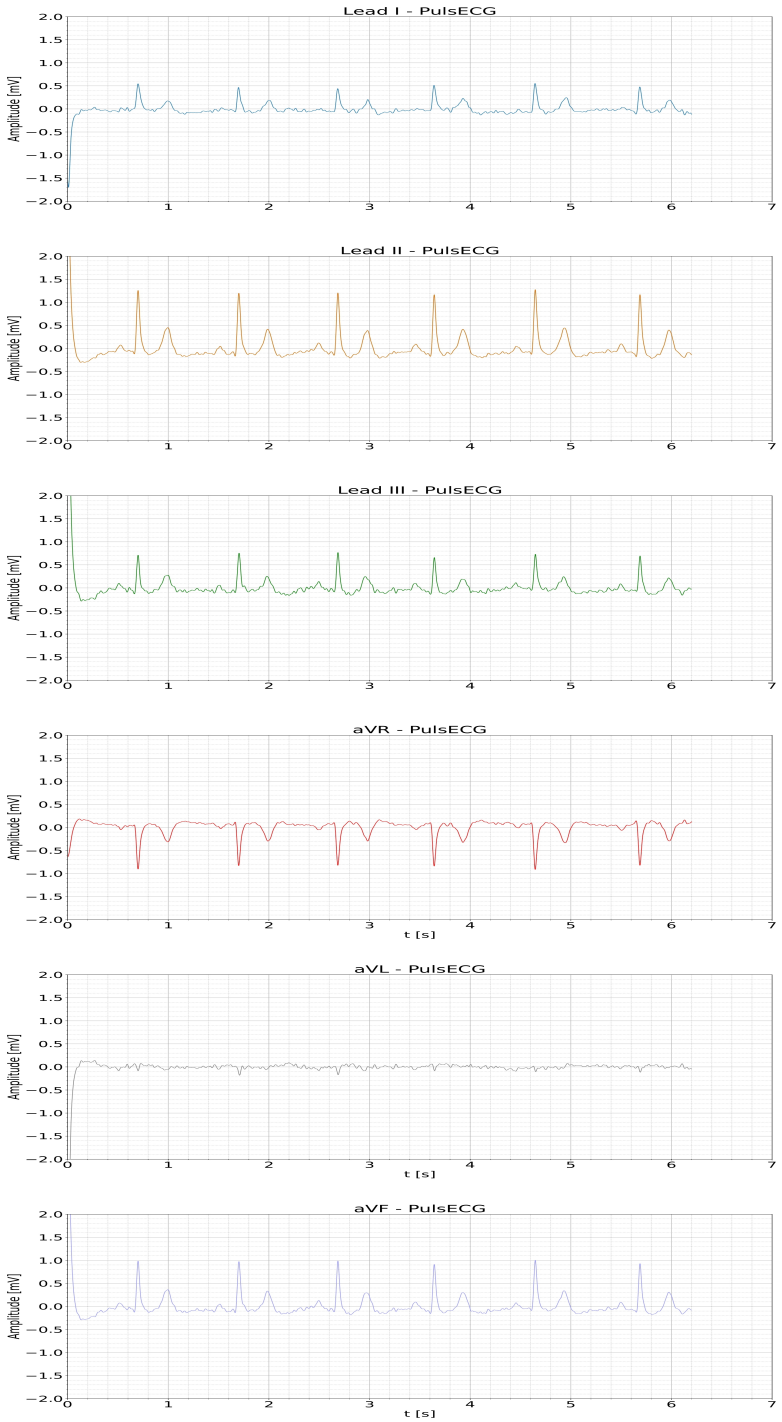


Figure 6.5: Subject 5 ECG.

Chapter 7

Validation and Conclusion

The development of the thesis project traversed several aspects, from the study of the hidden mathematical relationships among the different derivations, to the electronic design of a device capable of acquiring and reconstructing these relations.
An important step in the design has been the realization of a prototype to test the front-end, allowing to study the impact of the different filters on the acquired signal. This made it possible to become acquainted with the Cadence OrCAD design tools, and to develop skills to optimize and speed up subsequent designs.
The prototype board has been used extensively. Having determined the best setup, it has been possible to interconnect two identical front ends to acquire two leads in parallel. This way, it has been possible to test a setup similar to that of the final device, allowing the firmware to be set in advance. Such firmware correctly configured the operation of the ADC to obtain a preview of the six leads, starting from the two acquired by the front-end.
All this has been achieved connecting the front-end to the NUCLEO-WB55RG development board by means of jumpers. The dev kit offered the possibility of testing the Bluetooth connectivity of the STM32WB55 microcontroller, very similar to the one included in the final design of the project.
It has been also possible to test the sending procedure of acquired data to the client, via Bluetooth. This further phase involved the implementation and optimization of the bluetooth firmware code to send the acquired samples in real time, via notifications, with a payload extended to 200 bytes.
Following this phase, it has been necessary to implement a solution for retrieving and displaying the acquired data. Not having an up-to-date application for the visualization of six leads, it has been decided to write a Pyhon script to establish a Bluetooth connection with the device and retrieve the transmitted data. This step posed further difficulties, as the Python libraries do not offer the possibility of handling Maximum Transfer Units (MTUs) larger than 20bytes. But, as stated

earlier, this has been overcome using the Bash 'bluetoothctl' command to obtain and display the complete packets.
At the end of this process, it has been possible to retrieve the data and process and filter it digitally to display the six acquired leads.

7.1 Testing

The testing phase implied the acquisition of cardiac signals from different subjects, volunteer members of the research team. The acquired signals were then compared with those reported by other devices on the market.
Specifically, the signals were compared with a medical electrocardiograph, and with the only device, currently on the market, that can offer a six leads ECG reading, the KardiaMobile [1].
The results shown below were acquired with the provisional test setup. Further improvement in performance are to be expected from the implementation of the final device.
It must kept in mind that the acquisitions were made simultaneously[1] with the medical electrocardiograph, but deferred with Kardia.

[1] With a possible time shift of few second.

• **Comparison of professional electrocadiograph with thesis prototype:**

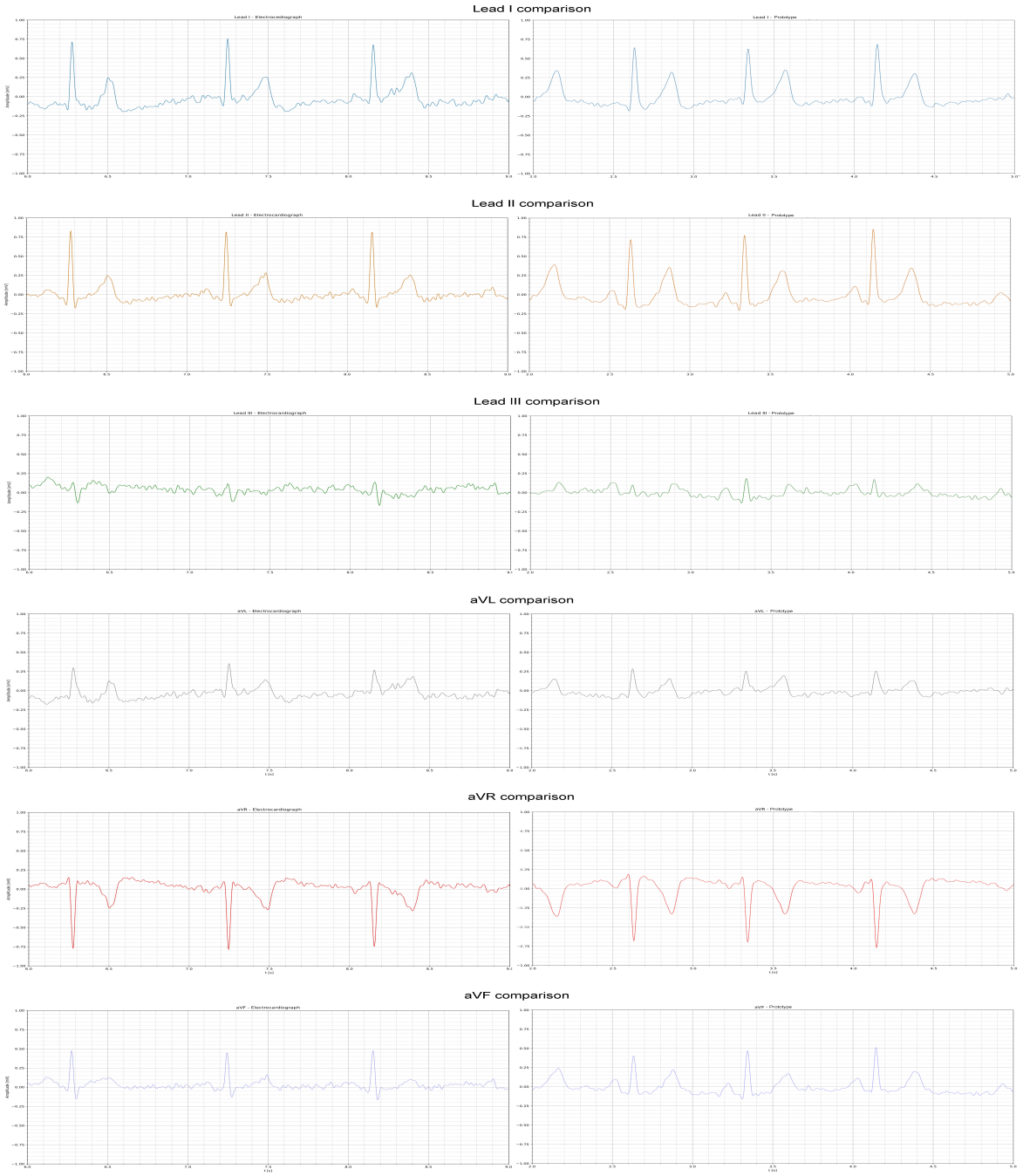


Figure 7.1: Professional electrocadiograph vs thesis prototype.

- **Comparison of KardiaMobile with thesis prototype:**

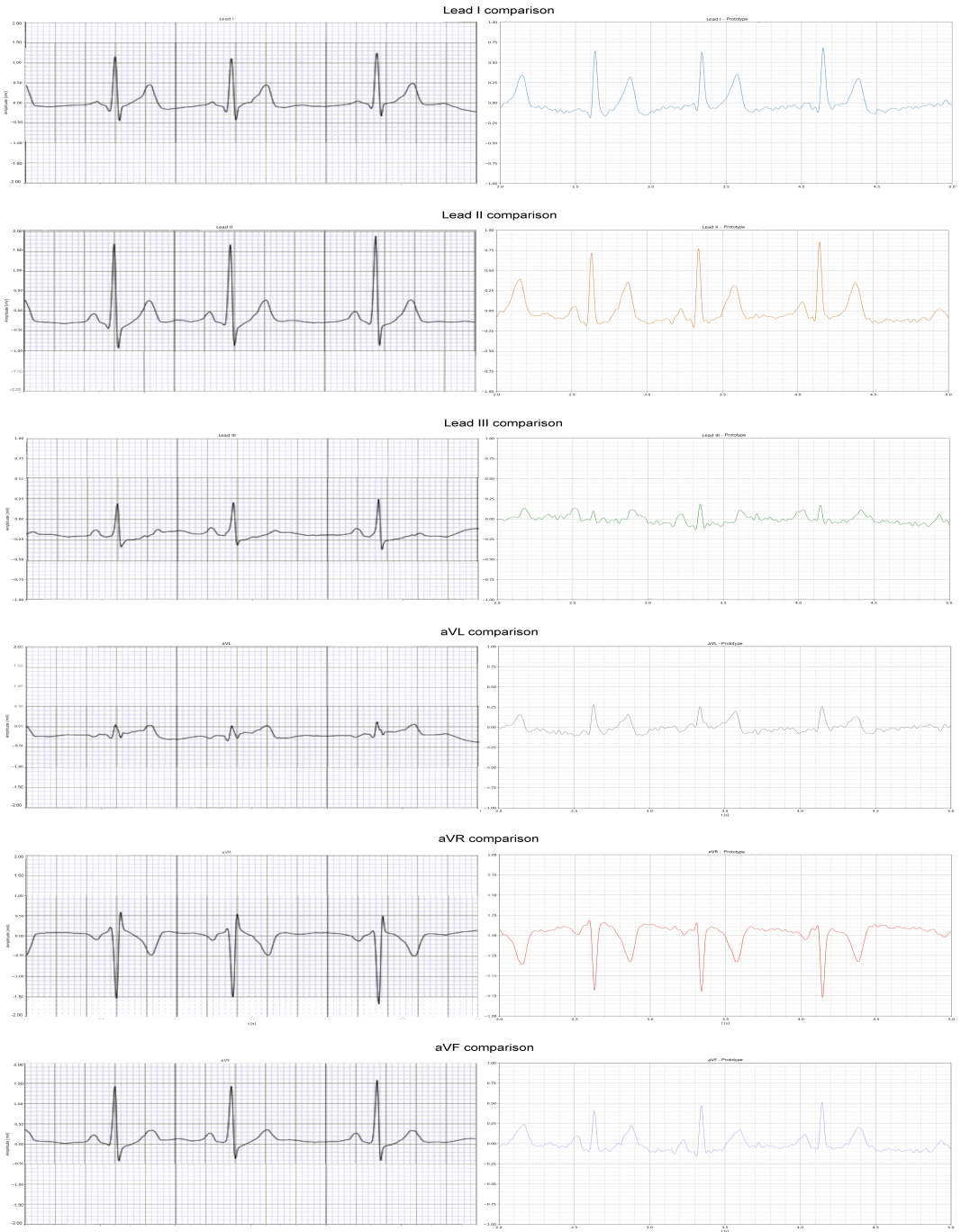


Figure 7.2: KardiaMobile vs thesis prototype.

As it can be seen from a qualitative point of view, results look very similar to those reported by the professional electrocardiograph.

It is possible to proceed with a numerical analysis of the reported results, considering the parameters of primary importance.
The morphology of the QRS in both cases is narrow, lasting less than 100 ms, indicating normal ventricular conduction.
The peak-to-peak amplitude of the QRS complex is consistent, with values reported by the prototype being approximately 5% greater than those of the professional device, on both leads acquired./
The morphology of the T wave is coherent, with QT duration around 360 ms, while the amplitude reported by the prototype is 10% greater than the reference values.
Measuring the distance between two R waves, the medical device reports a heart rate of 67 bpm while the prototype reports 75 bpm, an unusual discrepancy, but the reported window is too short for an accurate definition of heart rate.

Comparison with Kardia yields comparable results, but there are differences.
The graph shows the comparison between Kardia and the prototype while the numerical analysis refers to the signals reported by the professional electrocardiograph.
The morphology of the QRS is narrow in both cases, extremely narrow in the Kardia which shows a duration close to 40 ms.
The peak-to-peak amplitude of the QRS complex is inconsistent, with values reported by Kardia being approximately 100% greater than those of the professional device, on all leads except aVL.
T wave morphology is coherent, with QT duration around 360 ms, while the amplitude reported by Kardia is about 20% greater.
Measuring the distance between two R waves, the medical device reports a heart rate of 67 bpm while Kardia reports 77 bpm, again the discrepancy is insignificant and the reported window is too short for an accurate definition of heart rate.

From this comparison it can be supposed that KardiaMobile introduces a strong signal beautification that can lead to variations in the displayed signal.
Having said this, further comparison among the three instruments need to be carried out in order to draw reliable conclusions. And it should be remembered that the acquisitions with KardiaMobile were made in a later moment.

7.2 Future Perspectives

At this point, not having practically realized the designed device, there is ample room for future improvements.
It is possible to identify at least three main areas in which improvements can be brought into the project:

- Future Hardware developments: leaving aside the realisation of the device itself, other sensors can be implemented to make it more comprehensive. Examples are PPG, temperature or motion sensors.
- Future Firmware developments: firmware can be improved introducing effective management of low power states of the device, and also leveraging on signal processing techniques via AI for advanced diagnosis methodologies, thus establishing the device in the edge AI sector.
- Future App developments: it will be necessary to develop an application capable of handling the reception of two and/or six ECG signals in parallel (depending on whether the reconstruction of the signals is done by the app or the device itself).

Furthermore, it will be possible to implement the calculation of blood pressure with a non-invasive method thanks to other sensors integrated on the device.

7.3 Conclusions

The aim of this thesis has been to create a novel device capable of providing a six-lead medical ECG in a non-invasive way.
During the development board testing phase, the design of the final device has been carried out: it integrates what has been already implemented on the test board, that is the ST microcontroller and the entire I/O and power management subsystem, in a very small area.
Although it has been possible to develop the code, design the board and retrieve the components required for assembly, the development has been halted and it has not been possible to obtain the PCB in time.
However, the results obtained with an extemporaneous setup are no less valid. On the contrary, the results obtained in this way are satisfactory. This allows to assume that they can only improve if obtained by means of a more robust and structured solution.
In conclusion, there are all the prerequisites for the future development and realization of the device.
Such device would offer a new way of monitoring subjects, reducing the pressure

on the hospital system, decreasing the need for hospital visits and enabling continuous remote monitoring by caregivers.
It would represent a new starting point for the optimization and integration of ever more comprehensive and accurate diagnostic technologies in the new and vibrant field of edge AI.

Appendix A

Firmware code

A.1 main.c

main.c mercoledì 31 maggio 2023, 14:56

```c
/* USER CODE BEGIN Header */
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include "custom_app.h"
#include "app_conf.h"
#include "PulsECG_core.h"

ADC_HandleTypeDef hadc1;

IPCC_HandleTypeDef hipcc;

RTC_HandleTypeDef hrtc;

SPI_HandleTypeDef hspi1;

UART_HandleTypeDef huart1;

/* USER CODE BEGIN PV */

/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
void PeriphCommonClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_IPCC_Init(void);
static void MX_RF_Init(void);
static void MX_RTC_Init(void);
static void MX_ADC1_Init(void);
static void MX_SPI1_Init(void);
static void MX_USART1_UART_Init(void);


int main(void)
{

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();
  /* Config code for STM32_WPAN (HSE Tuning must be done before system clock
  configuration) */
  MX_APPE_Config();

  /* Configure the system clock */
  SystemClock_Config();

/* Configure the peripherals common clocks */
  PeriphCommonClock_Config();

  /* IPCC initialisation */
   MX_IPCC_Init();

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_RF_Init();
  MX_RTC_Init();
  MX_ADC1_Init();
  MX_SPI1_Init();
  MX_USART1_UART_Init();
  /* USER CODE BEGIN 2 */
  /* USER CODE END 2 */
```

```
 64
 65   /* Init code for STM32_WPAN */
 66   MX_APPE_Init();
 67
 68   /* Infinite loop */
 69   /* USER CODE BEGIN WHILE */
 70
 71   uint32_t tempo=HAL_GetTick();
 72   uint8_t tmp=1;
 73   char uart_buf[30];
 74   int uart_buf_len;
 75   uart_buf_len =sprintf(uart_buf, "\tHello_v3\t\r\n");
 76   HAL_UART_Transmit(&huart1, (uint8_t *)uart_buf, uart_buf_len,1000);
 77
 78   //All'accensione eseguire lampeggio led di benvenuto es 10 lampeggi veloci per
    notificare accensione
 79   HAL_Delay(1);
 80
 81   init_PulsECG_core();
 82
 83   while (1)
 84   {
 85     /* USER CODE END WHILE */
 86     MX_APPE_Process();
 87
 88     /* USER CODE BEGIN 3 */
 89
 90   }
 91   /* USER CODE END 3 */
 92 }
 93
 94
 95 /**
 96   * @brief ADC1 Initialization Function
 97   * @param None
 98   * @retval None
 99   */
100 static void MX_ADC1_Init(void)
101 {
102
103   ADC_ChannelConfTypeDef sConfig = {0};
104
105   /** Common config
106   */
107   hadc1.Instance = ADC1;
108   hadc1.Init.ClockPrescaler = ADC_CLOCK_ASYNC_DIV1;
109   hadc1.Init.Resolution = ADC_RESOLUTION_8B;
110   hadc1.Init.DataAlign = ADC_DATAALIGN_RIGHT;
111   hadc1.Init.ScanConvMode = ADC_SCAN_DISABLE;
112   hadc1.Init.EOCSelection = ADC_EOC_SINGLE_CONV;
113   hadc1.Init.LowPowerAutoWait = DISABLE;
114   hadc1.Init.ContinuousConvMode = DISABLE;
115   hadc1.Init.NbrOfConversion = 1;
116   hadc1.Init.DiscontinuousConvMode = DISABLE;
117   hadc1.Init.ExternalTrigConv = ADC_SOFTWARE_START;
118   hadc1.Init.ExternalTrigConvEdge = ADC_EXTERNALTRIGCONVEDGE_NONE;
119   hadc1.Init.DMAContinuousRequests = DISABLE;
120   hadc1.Init.Overrun = ADC_OVR_DATA_PRESERVED;
121   hadc1.Init.OversamplingMode = DISABLE;
122   if (HAL_ADC_Init(&hadc1) != HAL_OK)
123   {
124     Error_Handler();
125   }
126
```

```
127   /** Configure Regular Channel
128   */
129   sConfig.Channel = ADC_CHANNEL_7;
130   sConfig.Rank = ADC_REGULAR_RANK_1;
131   sConfig.SamplingTime = ADC_SAMPLETIME_2CYCLES_5;
132   sConfig.SingleDiff = ADC_SINGLE_ENDED;
133   sConfig.OffsetNumber = ADC_OFFSET_NONE;
134   sConfig.Offset = 0;
135   if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
136   {
137     Error_Handler();
138   }
139   /* USER CODE BEGIN ADC1_Init 2 */
140
141   /* USER CODE END ADC1_Init 2 */
142
143 }
144
145
146 /**
147   * @brief SPI1 Initialization Function
148   * @param None
149   * @retval None
150   */
151 static void MX_SPI1_Init(void)
152 {
153
154   hspi1.Instance = SPI1;
155   hspi1.Init.Mode = SPI_MODE_MASTER;
156   hspi1.Init.Direction = SPI_DIRECTION_2LINES;
157   hspi1.Init.DataSize = SPI_DATASIZE_8BIT;
158   hspi1.Init.CLKPolarity = SPI_POLARITY_LOW;
159   hspi1.Init.CLKPhase = SPI_PHASE_1EDGE;
160   hspi1.Init.NSS = SPI_NSS_SOFT;
161   hspi1.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_16;
162   hspi1.Init.FirstBit = SPI_FIRSTBIT_MSB;
163   hspi1.Init.TIMode = SPI_TIMODE_DISABLE;
164   hspi1.Init.CRCCalculation = SPI_CRCCALCULATION_DISABLE;
165   hspi1.Init.CRCPolynomial = 7;
166   hspi1.Init.CRCLength = SPI_CRC_LENGTH_DATASIZE;
167   hspi1.Init.NSSPMode = SPI_NSS_PULSE_ENABLE;
168   if (HAL_SPI_Init(&hspi1) != HAL_OK)
169   {
170     Error_Handler();
171   }
172
173 }
174
175 /**
176   * @brief GPIO Initialization Function
177   * @param None
178   * @retval None
179   */
180 static void MX_GPIO_Init(void)
181 {
182   GPIO_InitTypeDef GPIO_InitStruct = {0};
183
184   /* GPIO Ports Clock Enable */
185   __HAL_RCC_GPIOC_CLK_ENABLE();
186   __HAL_RCC_GPIOB_CLK_ENABLE();
187   __HAL_RCC_GPIOA_CLK_ENABLE();
188   __HAL_RCC_GPIOD_CLK_ENABLE();
189
190   /*Configure GPIO pin Output Level */
```

```
191   HAL_GPIO_WritePin(GPIOB, LED_B_Pin|LED_R_Pin|LD2_Pin|LD3_Pin
192                            |LD1_Pin, GPIO_PIN_RESET);
193
194   /*Configure GPIO pin Output Level */
195   HAL_GPIO_WritePin(GPIOA, EN_ANAL_Pin|CS_ADC_n_Pin, GPIO_PIN_RESET);
196
197   /*Configure GPIO pins : LED_B_Pin LED_R_Pin LD2_Pin LD3_Pin
198                            LD1_Pin */
199   GPIO_InitStruct.Pin = LED_B_Pin|LED_R_Pin|LD2_Pin|LD3_Pin
200                         |LD1_Pin;
201   GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
202   GPIO_InitStruct.Pull = GPIO_NOPULL;
203   GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
204   HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
205
206   /*Configure GPIO pins : EN_ANAL_Pin CS_ADC_n_Pin */
207   GPIO_InitStruct.Pin = EN_ANAL_Pin|CS_ADC_n_Pin;
208   GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
209   GPIO_InitStruct.Pull = GPIO_NOPULL;
210   GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
211   HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
212
213   /*Configure GPIO pin : DRY_n_Pin */
214   GPIO_InitStruct.Pin = DRY_n_Pin;
215   GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
216   GPIO_InitStruct.Pull = GPIO_PULLUP;
217   HAL_GPIO_Init(DRY_n_GPIO_Port, &GPIO_InitStruct);
218
219   /*Configure GPIO pin : WKUP_BTN_Pin */
220   GPIO_InitStruct.Pin = WKUP_BTN_Pin;
221   GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
222   GPIO_InitStruct.Pull = GPIO_NOPULL;
223   HAL_GPIO_Init(WKUP_BTN_GPIO_Port, &GPIO_InitStruct);
224
225   /*Configure GPIO pin : B1_Pin */
226   GPIO_InitStruct.Pin = B1_Pin;
227   GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
228   GPIO_InitStruct.Pull = GPIO_PULLUP;
229   HAL_GPIO_Init(B1_GPIO_Port, &GPIO_InitStruct);
230
231   /*Configure GPIO pins : PA11 PA12 */
232   GPIO_InitStruct.Pin = GPIO_PIN_11|GPIO_PIN_12;
233   GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
234   GPIO_InitStruct.Pull = GPIO_NOPULL;
235   GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
236   GPIO_InitStruct.Alternate = GPIO_AF10_USB;
237   HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
238
239   /*Configure GPIO pins : B2_Pin B3_Pin */
240   GPIO_InitStruct.Pin = B2_Pin|B3_Pin;
241   GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
242   GPIO_InitStruct.Pull = GPIO_NOPULL;
243   HAL_GPIO_Init(GPIOD, &GPIO_InitStruct);
244
245 }
246
247
```

A.2 PulsECG_core.h


```c
/*
 * PulsECG_core.h
 *
 *  Created on: Apr 27, 2023
 *      Author: marco
 */

#ifndef APP_PULSECG_CORE_H_
#define APP_PULSECG_CORE_H_

#include <ADC_command.h>
#include <SPI_communication.h>
#include "CallBack.h"
#include "custom_stm.h"
#include "custom_app.h"
#include "stm32_seq.h"


#define ADC_BYTES 1000
#define NOTIFY_PAYLOAD 200

static uint32_t duration, t_start, BTime;
static uint8_t BLevel, overflow;
static uint8_t bool ECGreading, bool BATreading, bool already read;
static uint8_t ADCDATA[ADC_BYTES];
static uint16_t index_1, index_2, tot_sample;
extern uint16_t tmp_BT;
static uint32_t NotifyECGval;


void init_PulsECG_core(void);
void update_BLevel(void);
void send_BLevel(void);
void start_stop_acquisition(uint8_t tempo);
void arm_board(void);
void disarm_board(void);
void acquire(void);
void sendSample(void);

void BT_config(void);


#endif /* APP_PULSECG_CORE_H_ */

```

A.3 PulsECG_core.c


```
/*
 * PulsECG_core.c
 *
 *  Created on: Apr 27, 2023
 *      Author: marco
 */
#include "PulsECG_core.h"


void init_PulsECG_core(void){

    BLevel=0;
    BTime=0;
    NotifyECGval=0;
    bool_BATreading=0;
    bool_ECGreading=0;
    bool_already_read=0;
    index_1=0, index_2=0, overflow=0, tot_sample=0;
    duration=0;
    t_start=0;
    for(int i=0; i<ADC_BYTES;i++) ADCDATA[i]=0;

    //Register and start GET_BLEVEL task
    UTIL_SEQ_RegTask(1<<CFG_TASK_GET_BLEVEL, CFG_SCH_PRIO_0, get_BLevel);
    UTIL_SEQ_SetTask(1<<CFG_TASK_GET_BLEVEL, CFG_SCH_PRIO_0);

    //Register ACQUIRE task
    UTIL_SEQ_RegTask(1<<CFG_TASK_ACQUIRE, CFG_SCH_PRIO_0, acquire);

    //Register SENDSAMPLE task
    UTIL_SEQ_RegTask(1<<CFG_TASK_SENDSAMPLE, CFG_SCH_PRIO_0, sendSample);
}

/*
 * Va inserita e attivata subito nello scheduler
 */
void get_BLevel(void){

    if(BTime==0 || HAL_GetTick() - BTime > 60000){
        bool_BATreading=1;
        BTime = HAL_GetTick();
        HAL_ADC_Start(&hadc1);

    }
    if(bool_BATreading && HAL_ADC_PollForConversion(&hadc1,1)==HAL_OK){
        bool_BATreading =0;
        BLevel=(uint8_t)HAL_ADC_GetValue(&hadc1);
    }
    UTIL_SEQ_SetTask(1<<CFG_TASK_GET_BLEVEL, CFG_SCH_PRIO_0);
}


/*
 * L'acquisizione viene fatta periodicamente nel SEQUENCER con get_BLevel
 *  Es: all'avvio, all'uscita da sleep mode, ogni minuto se non in sleep ecc..
 * Quando invece, asincronamente, si riceve una richiesta di lettura della
  batteria,
 * viene richiamata la funzione send_BLevel e si invia il valore.
 */
void send_BLevel(void){
    //Per inviare valore batteria
    Custom_STM_App_Update_Char(CUSTOM_STM_CHARB, &BLevel);
}
```

```
 64
 65
 66/*
 67 * AVVIO:
 68 * -Abilitare alimentazione analogica e clock ADC
 69 * -Regolare LED stato
 70 * -Avviare acquisizione ADC
 71 * -Solo all'avvio resetto l'index (Allo stop no perchè il BT deve ancora inviare
    ultimi dati)
 72 * STOP:
 73 * -Disabilitare alimentazione analogica e clock ADC
 74 * -Regolare LED stato
 75 * -Terminare ADC
 76 *
 77 * La funzione è eseguita alla ricezione di una richiesta di acquisizione da BT.
 78*/
 79void start_stop_acquisition(uint8_t tempo){
 80
 81    //Se ricevo scrittura mentre non sto acquisendo avvio acquisizione
 82    //Se ricevo scrittura mentre sto acquisendo, stoppo
 83
 84    if(!bool_ECGreading){
 85        //BT_config();
 86
 87        //Avvio acquisizione
 88        //Inserisco task nel scheduler
 89        duration=tempo*1000;
 90        arm_board();
 91        t_start=HAL_GetTick();
 92        adc_continous_read();
 93
 94    }else{
 95
 96        //Stoppo acquisizione
 97        //Rimuovo task dal scheduler
 98        disarm_board();
 99
100    }
101}
102
103
104void arm_board(){
105
106    tmp_BT=0, tot_sample=0;
107
108    bool_ECGreading=1;
109    index_1=0, index_2=0, overflow=0;
110    HAL_GPIO_WritePin(EN_ANAL_GPIO_Port, EN_ANAL_Pin, GPIO_PIN_SET);
111    HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_SET);
112
113    adc_setup();
114
115    UTIL_SEQ_SetTask(1 << CFG_TASK_ACQUIRE, CFG_SCH_PRIO_0);
116    UTIL_SEQ_ResumeTask(1 << CFG_TASK_ACQUIRE);
117    UTIL_SEQ_SetTask(1 << CFG_TASK_SENDSAMPLE, CFG_SCH_PRIO_0);
118
119    char uart_buf[30];
120    int uart_buf_len;
121    uart_buf_len =sprintf(uart_buf, "\tBoard armed.\t\r\n");
122    HAL_UART_Transmit(&huart1, (uint8_t *)uart_buf, uart_buf_len,1000);
123
124}
125
126
```

```
127 void disarm_board(){
128     bool_ECGreading=0;
129     HAL_GPIO_WritePin(CS_ADC_n_GPIO_Port, CS_ADC_n_Pin, GPIO_PIN_SET);
130     HAL_GPIO_WritePin(EN_ANAL_GPIO_Port, EN_ANAL_Pin, GPIO_PIN_RESET);
131     HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_RESET);
132 
133     UTIL_SEQ_PauseTask( 1 << CFG_TASK_ACQUIRE );
134 
135     char uart_buf[50];
136     int uart_buf_len;
137 
138     // CFG_TASK_SENDSAMPLE is stopped automatically when has finished the buffer
139 
140 }
141 
142 
143 
144 // This function doesn't use the HCI
145 void acquire(void){
146 
147 
148     if(HAL_GetTick() - t_start < duration){
149 
150         //Controllo se ci sono dati pronti da essere acquisiti (e inviati via BT)
151 
152         if(HAL_GPIO_ReadPin(DRY_n_GPIO_Port, DRY_n_Pin)==0 && bool_already_read==0)
    {
153 
154             if(index_1>=ADC_BYTES){
155                 overflow++;
156                 index_1=0;
157                 HAL_GPIO_TogglePin(LD2_GPIO_Port, LD2_Pin);
158             }
159 
160             tot_sample++;
161             HAL_SPI_Receive(&hspi1, (uint8_t *)&ADCDATA[index_1], 4, 100);
162             index_1 +=4;
163             bool_already_read=1;
164         }
165 
166         if(HAL_GPIO_ReadPin(DRY_n_GPIO_Port, DRY_n_Pin)==1 && bool_already_read==1
    ) bool_already_read=0;
167 
168     }
169     else disarm_board();
170 
171     UTIL_SEQ_SetTask(1<<CFG_TASK_ACQUIRE, CFG_SCH_PRIO_0);
172 }
173 
174 
175 // This function uses the HCI
176 // CFG_TASK_SENDSAMPLE is stopped automatically when has finished to send the
    buffer
177 void sendSample(void){
178 
179     //Ad ogni iterazione invio un dato via BT e incremento index_2
180 
181     if (index_1!=0 && index_1%NOTIFY_PAYLOAD==0){
182             (uint8_t)Custom_STM_App_Update_Char( CUSTOM_STM_CHARECG, (uint8_t
    *)&ADCDATA[(index_1/NOTIFY_PAYLOAD-1)*NOTIFY_PAYLOAD] );    //UPDATE CHAR IS USED
    TO NOTIFY
183 
184     }
185 
```

```
186     if( bool_ECGreading==1) UTIL_SEQ_SetTask(1 << CFG_TASK_SENDSAMPLE,
    CFG_SCH_PRIO_0);
187
188 }
189
190
191
192
```

A.4 ADC_command.h


```
 1 /*
 2  * ADC.h
 3  *
 4  *  Created on: Nov 19, 2022
 5  *      Author: marco
 6  */
 7
 8 #ifndef INC_ADC_COMMAND_H_
 9 #define INC_ADC_COMMAND_H_
10
11 #include <MCP3562.h>
12 #include <SPI_communication.h>
13 #include "CallBack.h"
14
15 void adc_setup(void);
16
17 void adc_reset();
18
19 void adc_set_CONFIG0(int w_val);
20
21 void adc_set_CONFIG3(int w_val);
22
23 void adc_read_CONFIG1();
24
25 void adc_set_CONFIG1(int w_val);
26
27 void adc_set_MUX();
28
29 void adc_set_SCAN();
30
31 void adc_get_value(int mode, int seconds);
32
33 void adc_start_restart();
34
35 void adc_get_calibration_offset();
36
37 #endif /* INC_ADC_COMMAND_H_ */
38
```

A.5 ADC_command.c


```
/*
 * ADC.c
 *
 *  Created on: Nov 19, 2022
 *      Author: marco
 */

#ifndef SRC_ADC_C_
#define SRC_ADC_C_

#include <ADC_command.h>

static uint8_t cmd=0, write_val=0;


void adc_continous_read(void){

    uint8_t cmd = (cmd | MCP_ADDR_M | MCP_REG_ADCDATA | MCP_TYPE_SR );

    HAL_GPIO_WritePin(CS_ADC_n_GPIO_Port, CS_ADC_n_Pin, GPIO_PIN_RESET);
    HAL_SPI_Transmit(&hspi1, (uint8_t *)&cmd, 1, 100);

}


void adc_setup(void){
    adc_reset();
    //adc_start_restart();

    adc_set_CONFIG0(-1);
    adc_set_CONFIG1(-1);    // OSR = 1024 = 0x14
    //adc_read_CONFIG1();

    adc_set_CONFIG3(0xD0); //0xD2 for continous read
    //adc_set_MUX();
    adc_set_SCAN();
    adc_start_restart();
}


void adc_reset(){
    char cmdName[32]="RESET";

    cmd=0;
    cmd = (cmd | MCP_ADDR_M | MCP_RESET); // COMMAND 01.111000

    send_command(cmdName,32,cmd);
}



void adc_set_CONFIG0(int w_val){

    char cmdName[32]="CONFIG0";
    cmd=0; write_val=0;

    //write the register
    cmd = (cmd | MCP_ADDR_M | MCP_REG_CONFIG0 | MCP_TYPE_IW );

    if (w_val!=-1) write_val= (write_val | (uint8_t) w_val );
    else write_val= (write_val | (uint8_t) 0xC3 );  //default value

    write_content(cmdName,32,cmd, write_val);

```

```
 65     //verify the new content
 66     cmd=0;
 67     cmd = (cmd | MCP_ADDR_M | MCP_REG_CONFIG0 | MCP_TYPE_SR );
 68     read_content(cmdName,32,cmd);
 69
 70 }
 71
 72
 73
 74 void adc_set_CONFIG3(int w_val){
 75
 76     char cmdName[32]="CONFIG3";
 77     cmd=0; write_val=0;
 78
 79     //write the register
 80     cmd = (cmd | MCP_ADDR_M | MCP_REG_CONFIG3 | MCP_TYPE_IW );
 81
 82     if (w_val!=-1) write_val= (write_val | (uint8_t) w_val );
 83     else write_val= (write_val | (uint8_t) 0x90  ); //default value (single read)
 84     // aka 10 01 0 0 0 0
 85
 86     write_content(cmdName,32,cmd, write_val);
 87
 88     //verify the new content
 89     cmd=0;
 90     cmd = (cmd | MCP_ADDR_M | MCP_REG_CONFIG3 | MCP_TYPE_SR );
 91     read_content(cmdName,32,cmd);
 92
 93 }
 94
 95 void adc_read_CONFIG1(){
 96
 97
 98     char cmdName[32]="CONFIG1";
 99     cmd=0; write_val=0;
100
101     cmd = (cmd | MCP_ADDR_M | MCP_REG_CONFIG1 | MCP_TYPE_SR );
102     read_content(cmdName,32,cmd);
103
104
105
106 }
107
108 //If OSR = -1 is 256 (default)
109 void adc_set_CONFIG1(int w_val){
110
111
112     char cmdName[32]="CONFIG1";
113     cmd=0; write_val=0;
114
115     //write the register
116     cmd = (cmd | MCP_ADDR_M | MCP_REG_CONFIG1 | MCP_TYPE_IW );
117
118     if (w_val!=-1) write_val= (write_val | (uint8_t) w_val );
119     else write_val= (write_val | (uint8_t) 0xC  );  //default value (MCLK=ACLK e
    OSR=256)
120
121     write_content(cmdName,32,cmd, write_val);
122
123 }
124
125
126
127
```

```
128 void adc_set_MUX(){
129 
130     char cmdName[32]="MUX";
131     cmd=0; write_val=0;
132 
133     //write the register
134     cmd = (cmd | MCP_ADDR_M | MCP_REG_MUX | MCP_TYPE_IW );
135 
136     write_val= (write_val | MUX_CH0 <<4 | MUX_CH1 );
137 
138     write_content(cmdName,32,cmd, write_val);
139 
140     //verify the new content
141     cmd=0;
142     cmd = (cmd | MCP_ADDR_M | MCP_REG_MUX | MCP_TYPE_SR );
143     read_content(cmdName,32,cmd);
144 
145 }
146 
147 void adc_start_restart(){
148 
149     char cmdName[32]="Start acquisition";
150     cmd=0; write_val=0;
151 
152     //start the acquisition the register
153     cmd = (cmd | MCP_ADDR_M | MCP_CONV_START );
154 
155     send_command(cmdName,32,cmd);
156 
157 }
158 
159 
160 
161 //The stream of data (single/continous) depends on the 7-8 bit in the CONFIG3 reg
162 void adc_get_value(int mode, int seconds){
163 
164     adc_start_restart();
165 
166     //read the acquired data stored in ADCDATA
167     char cmdName[32]="ADCDATA";
168     cmd=0;
169     cmd = (cmd | MCP_ADDR_M | MCP_REG_ADCDATA | MCP_TYPE_SR );
170 
171     if (mode==0) read_content(cmdName,32,cmd);
172     else if(mode==1) continous_read(seconds, cmd);
173     else if(mode==2) {
174         int t_start = HAL_GetTick();
175         while(HAL_GetTick() - t_start < 5000)  read_content_32(cmdName, 32, cmd);
176     }
177 
178 
179 }
180 
181 void adc_get_calibration_offset(){
182 
183     char cmdName[32]="OFFSETCAL";
184     cmd=0; write_val=0;
185 
186     //verify the offset
187     cmd=0;
188     cmd = (cmd | MCP_ADDR_M | MCP_REG_OFFSETCAL | MCP_TYPE_SR );
189     read_content(cmdName,32,cmd);
190 
191 }
```

```
192
193
194
195 void adc_set_SCAN(){
196
197     char cmdName[32]="SCAN";
198     cmd=0;
199     uint8_t write_val[3]={0};
200
201     //write the register
202     cmd = (cmd | MCP_ADDR_M | MCP_REG_SCAN | MCP_TYPE_IW );
203
204     //write_val[1]= (write_val[1] | (uint8_t) 6 );
205     write_val[1]= (write_val[1] | (uint8_t) 5 );
206     //write_val[1]= (write_val[1] | (uint8_t) 2 );
207
208     write_24_bit(cmdName,32,cmd, write_val);
209
210     //verify the new content
211     cmd=0;
212     cmd = (cmd | MCP_ADDR_M | MCP_REG_SCAN | MCP_TYPE_SR );
213
214     read_24_bit(cmdName, 32, cmd);
215
216     cmd=0;
217
218 }
219
220
221
222 #endif /* SRC_ADC_C_ */
223
```

A.6 SPI_communication.h


```
 1 /*
 2  * communication.h
 3  *
 4  *  Created on: Nov 19, 2022
 5  *      Author: marco
 6  */
 7
 8 #ifndef INC_SPI_COMMUNICATION_H_
 9 #define INC_SPI_COMMUNICATION_H_
10
11
12 #include <MCP3562.h>
13 #include "CallBack.h"
14
15 void toBinary(int num, char binary[]);
16
17 void print_content(int num, int type);
18
19 //utility function: send the command and read the ack
20 void u_send_command(uint8_t *cmd, uint8_t *bin_send);
21
22 void send_command(char *cmdName, size_t lengthCmdName, uint8_t cmd);
23
24 //Get the content of a register
25 void read_content(char *cmdName, size_t lengthCmdName, uint8_t cmd);
26
27 void read_content_32(char *cmdName, size_t lengthCmdName, uint8_t cmd);
28
29 void write_24_bit(char *cmdName, size_t lengthCmdName, uint8_t cmd, uint8_t reg_val
   []);
30
31 void read_24_bit(char *cmdName, size_t lengthCmdName, uint8_t cmd);
32
33 //Write something to a register
34 void write_content(char *cmdName, size_t lengthCmdName, uint8_t cmd, uint8_t new_val
   );
35
36
37 #endif /* INC_SPI_COMMUNICATION_H_ */
38
```

A.7 SPI_communication.c


```c
/*
 * communication.c
 *
 *  Created on: Nov 19, 2022
 *      Author: marco
 */


#ifndef INC_COMMUNICATION_C_
#define INC_COMMUNICATION_C_

#include <SPI_communication.h>

static char uart_buf[90];
static int uart_buf_len;


void toBinary(int num, char binary[])
{
    for (int i=0; i<8; i++) {

        binary[7-i] = (char)((num >> i & 1) + '0');
        binary[8]='\0';
    }
}


//type: 1 = ack | 2 = read value | 3 = wrote value
void print_content(int num, int type){
    char binary[34]={'\0'};
    toBinary(num, binary);

    if(type==1)      uart_buf_len = sprintf(uart_buf, "\tACK received:\t %s aka %u \r\n", binary, num);
    else if(type==2) uart_buf_len = sprintf(uart_buf, "\tContent read:\t %s aka %u \r\n", binary, num);
    else if(type==3) uart_buf_len = sprintf(uart_buf, "\tContent wrote:\t %s aka %u \r\n", binary, num);

    HAL_UART_Transmit(&huart1, (uint8_t *)uart_buf, uart_buf_len, 100);
}



//Utility function: send the command and return the ack
//First function called for every communication
void u_send_command(uint8_t *cmd, uint8_t *recev){

    char bin_send[32]={'\0'};
    HAL_Delay(1);

    toBinary(*cmd, bin_send);

    uart_buf_len = sprintf(uart_buf, "\tbyte sent:\t %s aka %u \r\n", bin_send, *cmd);
    HAL_UART_Transmit(&huart1, (uint8_t *)uart_buf, uart_buf_len, 100);

    HAL_SPI_TransmitReceive(&hspi1, (uint8_t *)cmd, (uint8_t *)recev, 1, 100);

}



//For commands that don't require additional read or write
```

```
61 void send_command(char *cmdName, size_t lengthCmdName, uint8_t cmd){
62
63     uint8_t ack_num=0;
64
65     uart_buf_len = sprintf(uart_buf, "\nSending cmd %s.\r\n", cmdName);
66     HAL_UART_Transmit(&huart1, (uint8_t *)uart_buf, uart_buf_len, 100);
67
68     //turn on the communication
69     HAL_GPIO_WritePin(CS_ADC_n_GPIO_Port, CS_ADC_n_Pin, GPIO_PIN_RESET);
70
71     //send the command and receive the ack
72     u_send_command(&cmd, &ack_num);
73
74     //turn off the communication
75     HAL_GPIO_WritePin(CS_ADC_n_GPIO_Port, CS_ADC_n_Pin, GPIO_PIN_SET);
76
77     //print the received bits
78     print_content(ack_num, 1);
79
80 }
81
82
83 //For commands that receive also one bytes other than the ack
84 void read_content(char *cmdName, size_t lengthCmdName, uint8_t cmd){
85
86     uint8_t ack_num=0, content=0;
87
88     uart_buf_len = sprintf(uart_buf, "\nReading %s.\r\n", cmdName);
89     HAL_UART_Transmit(&huart1, (uint8_t *)uart_buf, uart_buf_len, 100);
90
91     //turn on the communication
92     HAL_GPIO_WritePin(CS_ADC_n_GPIO_Port, CS_ADC_n_Pin, GPIO_PIN_RESET);
93
94     //send the command and receive the ack
95     u_send_command(&cmd, &ack_num);
96
97     //read the returned content
98     HAL_SPI_Receive(&hspi1, (uint8_t *)&content, 1, 100);
99
100    //turn off the communication
101    HAL_GPIO_WritePin(CS_ADC_n_GPIO_Port, CS_ADC_n_Pin, GPIO_PIN_SET);
102
103    //print the received bits
104    print_content(ack_num, 1);
105    print_content(content, 2);
106
107
108 }
109
110
111
112 void write_content(char *cmdName, size_t lengthCmdName, uint8_t cmd, uint8_t
    new_val ){
113
114    uint8_t ack_num=0;
115
116    uart_buf_len = sprintf(uart_buf, "\nWriting %s.\r\n", cmdName);
117    HAL_UART_Transmit(&huart1, (uint8_t *)uart_buf, uart_buf_len, 100);
118
119    //turn on the communication
120    HAL_GPIO_WritePin(CS_ADC_n_GPIO_Port, CS_ADC_n_Pin, GPIO_PIN_RESET);
121
122    //send the command and receive the ack
123    u_send_command(&cmd, &ack_num);
```

```
124
125     //write the new content
126     HAL_SPI_Transmit(&hspi1, (uint8_t *)&new_val, 1, 100);
127
128     //turn off the communication
129     HAL_GPIO_WritePin(CS_ADC_n_GPIO_Port, CS_ADC_n_Pin, GPIO_PIN_SET);
130
131     //print the ack received
132     print_content(ack_num, 1);
133
134     //print the wrote bits
135     print_content(new_val, 3);
136
137 }
138
139
140
141 void read_content_32(char *cmdName, size_t lengthCmdName, uint8_t cmd){
142
143         uint8_t ack_num=0;
144         uint32_t content=0;
145
146
147         //turn on the communication
148         HAL_GPIO_WritePin(CS_ADC_n_GPIO_Port, CS_ADC_n_Pin, GPIO_PIN_RESET);
149
150         //send the command and receive the ack
151         //u_send_command(&cmd, &ack_num);
152         HAL_SPI_TransmitReceive(&hspi1, (uint8_t *)&cmd, (uint8_t *)&ack_num, 1,
    100);
153         //read the returned content
154         HAL_SPI_Receive(&hspi1, (uint8_t *)&content, 4, 100);
155
156         //turn off the communication
157         HAL_GPIO_WritePin(CS_ADC_n_GPIO_Port, CS_ADC_n_Pin, GPIO_PIN_SET);
158
159         //print the received bits
160         //print_content(ack_num, 1);
161         uart_buf_len = sprintf(uart_buf, "\t%lu\r\n", content);
162         HAL_UART_Transmit(&huart1, (uint8_t *)uart_buf, uart_buf_len, 100);
163
164 }
165
166
167
168 void write_24_bit(char *cmdName, size_t lengthCmdName, uint8_t cmd, uint8_t reg_val
    []){
169
170         uint8_t ack_num=0;
171         uint32_t content=0;
172
173         uart_buf_len = sprintf(uart_buf, "\nWriting %s.\r\n", cmdName);
174         HAL_UART_Transmit(&huart1, (uint8_t *)uart_buf, uart_buf_len, 100);
175
176         //turn on the communication
177         HAL_GPIO_WritePin(CS_ADC_n_GPIO_Port, CS_ADC_n_Pin, GPIO_PIN_RESET);
178
179         //send the command and receive the ack
180         u_send_command(&cmd, &ack_num);
181
182         //write the 24 bit content
183         HAL_SPI_Transmit(&hspi1, (uint8_t *)&reg_val[0], 3, 100);
184
185         //turn off the communication
```

```
186         HAL_GPIO_WritePin(CS_ADC_n_GPIO_Port, CS_ADC_n_Pin, GPIO_PIN_SET);
187
188         //print the ack received
189         print_content(ack_num, 1);
190
191
192 }
193
194
195 void read_24_bit(char *cmdName, size_t lengthCmdName, uint8_t cmd){
196
197     uint8_t ack_num=0;
198     uint8_t content[3]={0};
199
200     uart_buf_len = sprintf(uart_buf, "\nReading %s.\r\n", cmdName);
201     HAL_UART_Transmit(&huart1, (uint8_t *)uart_buf, uart_buf_len, 100);
202
203     //turn on the communication
204     HAL_GPIO_WritePin(CS_ADC_n_GPIO_Port, CS_ADC_n_Pin, GPIO_PIN_RESET);
205
206     //send the command and receive the ack
207     u_send_command(&cmd, &ack_num);
208
209     //read the returned content
210     HAL_SPI_Receive(&hspi1, (uint8_t *)&content[0], 3, 100);
211
212     //turn off the communication
213     HAL_GPIO_WritePin(CS_ADC_n_GPIO_Port, CS_ADC_n_Pin, GPIO_PIN_SET);
214
215     //print the received bits
216     //print_content(ack_num, 1);
217     uart_buf_len = sprintf(uart_buf, "\t24 byte content: %u | %u | %u\r\n", content
    [2], content[1], content[0]);
218     HAL_UART_Transmit(&huart1, (uint8_t *)uart_buf, uart_buf_len, 100);
219 }
220
221
222
223
224 #endif
225
```

A.8 MCP3562.h


```
1 // command structure: 2 bit device addr + 4 bit command/reg address + 2 command type
2
3 // Example 1:   MCP_ADDR + MCP_REG + MCP_TYPE
4 // Example 2:   MCP_ADDR + MCP_FAST_CMD
5
6 //########## UNIQUE INDIVIDUAL ADDRESS 2 bit ###############
7
8 #define MCP_ADDR_M (uint8_t)64
9
10
11 //########## FAST COMMANDS 6 bit ############################
12
13 // The following are fast commands [ 4 command + 2 type bits, all together ]
14
15 #define MCP_NOPE (uint8_t)0
16
17 #define MCP_CONV_START (uint8_t)40
18
19 #define MCP_STANDBY (uint8_t)44
20
21 #define MCP_SHUTDOWN (uint8_t)48
22
23 #define MCP_FULL_DOWN (uint8_t)52
24
25 #define MCP_RESET (uint8_t)56
26
27
28 //########## GENERAL COMMANDS 4 bit addr + 2 bit type ######
29
30 // --> TYPE 2 bit ########################################
31
32 #define MCP_TYPE_SR (uint8_t)1  //Static Read of Register Address
33
34 #define MCP_TYPE_IW (uint8_t)2  //Incremental Write Starting at Register Address
35
36 #define MCP_TYPE_IR (uint8_t)3  //Incremental Read Starting at Register Address
37
38 // --> ADDRESSES 4 bit ###################################
39
40 //8 bit, IRQ Status bits and IRQ mode settings;
41 //enable for Fast commands andfor conversion start pulse
42 #define MCP_REG_IRQ (uint8_t)(0x5 << 2)
43
44 #define MCP_REG_CONFIG3 (uint8_t)(0x4 << 2) //8 bit
45
46 #define MCP_REG_CONFIG1 (uint8_t)(0x2 << 2)
47
48 #define MCP_REG_CONFIG0 (uint8_t)(0x1 << 2)
49
50 #define MCP_REG_MUX (uint8_t)(0x6 << 2)
51
52 #define MCP_REG_ADCDATA (uint8_t)(0x0 << 2)
53
54 #define MCP_REG_OFFSETCAL (uint8_t)(0x9 << 2)
55
56 #define MCP_REG_SCAN    (uint8_t)(0x7 << 2)
57
58
59 // SCAN REGISTER CHANNEL SELECTION
60
61 #define SCAN_CH_B (uint8_t)9
62 #define SCAN_CH_C (uint8_t)10
63
64
```

A.9 custom_stm.h


```
 1
 2/* Define to prevent recursive inclusion -------------------------------------*/
 3#ifndef __CUSTOM_STM_H
 4#define __CUSTOM_STM_H
 5
 6#ifdef __cplusplus
 7extern "C" {
 8#endif
 9
10/* Includes ------------------------------------------------------------------*/
11/* USER CODE BEGIN Includes */
12
13/* USER CODE END Includes */
14
15/* Exported types ------------------------------------------------------------*/
16typedef enum
17{
18  /* Battery */
19  CUSTOM_STM_CHARB,
20  /* ECG */
21  CUSTOM_STM_CHARSS,
22  CUSTOM_STM_CHARECG,
23} Custom_STM_Char_Opcode_t;
24
25typedef enum
26{
27  /* charB */
28  CUSTOM_STM_CHARB_READ_EVT,
29  /* charSS */
30  CUSTOM_STM_CHARSS_WRITE_EVT,
31  /* charECG */
32  CUSTOM_STM_CHARECG_READ_EVT,
33  CUSTOM_STM_CHARECG_NOTIFY_ENABLED_EVT,
34  CUSTOM_STM_CHARECG_NOTIFY_DISABLED_EVT,
35
36  CUSTOM_STM_BOOT_REQUEST_EVT
37} Custom_STM_Opcode_evt_t;
38
39typedef struct
40{
41  uint8_t * pPayload;
42  uint8_t   Length;
43} Custom_STM_Data_t;
44
45typedef struct
46{
47  Custom_STM_Opcode_evt_t       Custom_Evt_Opcode;
48  Custom_STM_Data_t             DataTransfered;
49  uint16_t                      ConnectionHandle;
50  uint8_t                       ServiceInstance;
51} Custom_STM_App_Notification_evt_t;
52
53/* USER CODE BEGIN ET */
54
55/* USER CODE END ET */
56
57/* Exported constants --------------------------------------------------------*/
58extern uint8_t SizeCharb;
59extern uint8_t SizeCharss;
60extern uint8_t SizeCharecg;
61
62/* USER CODE BEGIN EC */
63
64/* USER CODE END EC */
```

A.10 custom_stm.c


```
 1
 2 /* Includes ------------------------------------------------------------------*/
 3 #include "common_blesvc.h"
 4 #include "custom_stm.h"
 5
 6 /* USER CODE BEGIN Includes */
 7 #include "main.h"
 8 #include "PulsECG_core.h"
 9 /* USER CODE END Includes */
10
11 /* Private typedef -----------------------------------------------------------*/
12 typedef struct{
13   uint16_t  CustomBatteryHdle;                   /**< Battery handle */
14   uint16_t  CustomCharbHdle;                   /**< charB handle */
15   uint16_t  CustomEcgHdle;                   /**< ECG handle */
16   uint16_t  CustomCharssHdle;                   /**< charSS handle */
17   uint16_t  CustomCharecgHdle;                   /**< charECG handle */
18 }CustomContext_t;
19
20
21 /* Private macros ------------------------------------------------------------*/
22 #define CHARACTERISTIC_DESCRIPTOR_ATTRIBUTE_OFFSET         2
23 #define CHARACTERISTIC_VALUE_ATTRIBUTE_OFFSET              1
24 /* USER CODE BEGIN PM */
25
26 /* USER CODE END PM */
27
28 /* Private variables ---------------------------------------------------------*/
29 uint8_t SizeCharb = 1;
30 uint8_t SizeCharss = 2;
31 uint8_t SizeCharecg = 247;
32
33 /**
34  * START of Section BLE_DRIVER_CONTEXT
35  */
36 PLACE_IN_SECTION("BLE_DRIVER_CONTEXT") static CustomContext_t CustomContext;
37
38 /**
39  * END of Section BLE_DRIVER_CONTEXT
40  */
41
42 /* USER CODE BEGIN PV */
43 uint16_t tmp_BT=0;
44 /* USER CODE END PV */
45
46 /* Private function prototypes -----------------------------------------------*/
47 static SVCCTL_EvtAckStatus_t Custom_STM_Event_Handler(void *pckt);
48
49 static SVCCTL_EvtAckStatus_t Custom_STM_Event_Handler(void *Event)
50 {
51   SVCCTL_EvtAckStatus_t return_value;
52   hci_event_pckt *event_pckt;
53   evt_blecore_aci *blecore_evt;
54   aci_gatt_attribute_modified_event_rp0 *attribute_modified;
55   aci_gatt_read_permit_req_event_rp0    *read_req;
56   Custom_STM_App_Notification_evt_t     Notification;
57   /* USER CODE BEGIN Custom_STM_Event_Handler_1 */
58
59   /* USER CODE END Custom_STM_Event_Handler_1 */
60
61   return_value = SVCCTL_EvtNotAck;
62   event_pckt = (hci_event_pckt *)(((hci_uart_pckt*)Event)->data);
63
64   switch (event_pckt->evt)
```

```
 65  {
 66    case HCI_VENDOR_SPECIFIC_DEBUG_EVT_CODE:
 67      blecore_evt = (evt_blecore_aci*)event_pckt->data;
 68      switch (blecore_evt->ecode)
 69      {
 70        case ACI_GATT_ATTRIBUTE_MODIFIED_VSEVT_CODE:
 71          /* USER CODE BEGIN EVT_BLUE_GATT_ATTRIBUTE_MODIFIED_BEGIN */
 72
 73          /* USER CODE END EVT_BLUE_GATT_ATTRIBUTE_MODIFIED_BEGIN */
 74          attribute_modified = (aci_gatt_attribute_modified_event_rp0*)blecore_evt-
   >data;
 75          if (attribute_modified->Attr_Handle == (CustomContext.CustomCharecgHdle +
   CHARACTERISTIC_DESCRIPTOR_ATTRIBUTE_OFFSET))
 76          {
 77            return_value = SVCCTL_EvtAckFlowEnable;
 78            /* USER CODE BEGIN CUSTOM_STM_Service_2_Char_2 */
 79
 80            /* USER CODE END CUSTOM_STM_Service_2_Char_2 */
 81            switch (attribute_modified->Attr_Data[0])
 82            {
 83
 84              case (!(COMSVC_Notification)):
 85                /* USER CODE BEGIN CUSTOM_STM_Service_2_Char_2_Disabled_BEGIN */
 86
 87                /* USER CODE END CUSTOM_STM_Service_2_Char_2_Disabled_BEGIN */
 88                Notification.Custom_Evt_Opcode =
   CUSTOM_STM_CHARECG_NOTIFY_DISABLED_EVT;
 89                Custom_STM_App_Notification(&Notification);
 90                /* USER CODE BEGIN CUSTOM_STM_Service_2_Char_2_Disabled_END */
 91
 92                /* USER CODE END CUSTOM_STM_Service_2_Char_2_Disabled_END */
 93                break;
 94
 95              /* Enabled Notification management */
 96              case COMSVC_Notification:
 97                /* USER CODE BEGIN
   CUSTOM_STM_Service_2_Char_2_COMSVC_Notification_BEGIN */
 98
 99                /* USER CODE END
   CUSTOM_STM_Service_2_Char_2_COMSVC_Notification_BEGIN */
100                Notification.Custom_Evt_Opcode =
   CUSTOM_STM_CHARECG_NOTIFY_ENABLED_EVT;
101                Custom_STM_App_Notification(&Notification);
102                break;
103
104              default:
105              break;
106            }
107          }  /* if (attribute_modified->Attr_Handle ==
   (CustomContext.CustomCharecgHdle + CHARACTERISTIC_DESCRIPTOR_ATTRIBUTE_OFFSET))*/
108
109          else if (attribute_modified->Attr_Handle ==
   (CustomContext.CustomCharssHdle + CHARACTERISTIC_VALUE_ATTRIBUTE_OFFSET))
110          {
111            return_value = SVCCTL_EvtAckFlowEnable;
112            /* USER CODE BEGIN
   CUSTOM_STM_Service_2_Char_1_ACI_GATT_ATTRIBUTE_MODIFIED_VSEVT_CODE */
113
114            uint8_t *ptr = attribute_modified->Attr_Data;
115            uint8_t duration=0;
116
117            duration = (ptr[0]-'0')*10 + (ptr[1]-'0');
118
119            uint32_t ACI_GATT_NOTIFICATION_EXT_EVENT = 0x00400000;
```

```
120             aci_gatt_set_event_mask(ACI_GATT_NOTIFICATION_EXT_EVENT);
121             aci_gatt_exchange_config(attribute_modified->Connection_Handle);
122             hci_le_set_data_length(attribute_modified->Connection_Handle, 200,
    (200+14)*8);
123             hci_le_write_suggested_default_data_length(200, (200+14)*8);
124 
125             start_stop_acquisition(duration);
126 
127           } /* if (attribute_modified->Attr_Handle ==
    (CustomContext.CustomCharssHdle + CHARACTERISTIC_VALUE_ATTRIBUTE_OFFSET))*/
128 
129           break;
130 
131         case ACI_GATT_READ_PERMIT_REQ_VSEVT_CODE :
132           /* USER CODE BEGIN EVT_BLUE_GATT_READ_PERMIT_REQ_BEGIN */
133 
134           /* USER CODE END EVT_BLUE_GATT_READ_PERMIT_REQ_BEGIN */
135           read_req = (aci_gatt_read_permit_req_event_rp0*)blecore_evt->data;
136           if (read_req->Attribute_Handle == (CustomContext.CustomCharbHdle +
    CHARACTERISTIC_VALUE_ATTRIBUTE_OFFSET))
137           {
138             return_value = SVCCTL_EvtAckFlowEnable;
139             /*USER CODE BEGIN
    CUSTOM_STM_Service_1_Char_1_ACI_GATT_READ_PERMIT_REQ_VSEVT_CODE_1 */
140 
141              send_BLevel();
142 
143             /*USER CODE END
    CUSTOM_STM_Service_1_Char_1_ACI_GATT_READ_PERMIT_REQ_VSEVT_CODE_1*/
144             aci_gatt_allow_read(read_req->Connection_Handle);
145           }
146           break;
147 
148         case ACI_GATT_WRITE_PERMIT_REQ_VSEVT_CODE:
149           break;
150         default:
151           break;
152       }
153       break; /* HCI_VENDOR_SPECIFIC_DEBUG_EVT_CODE */
154 
155 
156     default:
157       break;
158   }
159 
160   return(return_value);
161 }/* end Custom_STM_Event_Handler */
162 
163 /**
164  * @brief  Characteristic update
165  * @param  CharOpcode: Characteristic identifier
166  * @param  Service_Instance: Instance of the service to which the characteristic
    belongs
167  *
168  */
169 tBleStatus Custom_STM_App_Update_Char(Custom_STM_Char_Opcode_t CharOpcode, uint8_t
    *pPayload)
170 {
171   tBleStatus ret = BLE_STATUS_INVALID_PARAMS;
172   /* USER CODE BEGIN Custom_STM_App_Update_Char_1 */
173 
174   int ret2=0;
175 
176   /* USER CODE END Custom_STM_App_Update_Char_1 */
```

```
177
178   switch (CharOpcode)
179   {
180
181     case CUSTOM_STM_CHARB:
182       ret = aci_gatt_update_char_value(CustomContext.CustomBatteryHdle,
183                                        CustomContext.CustomCharbHdle,
184                                        0, /* charValOffset */
185                                        SizeCharb, /* charValueLen */
186                                        (uint8_t *)  pPayload);
187       if (ret != BLE_STATUS_SUCCESS)
188       {
189         APP_DBG_MSG("  Fail   : aci_gatt_update_char_value CHARB command, result :
    0x%x \n\r", ret);
190       }
191       else
192       {
193         APP_DBG_MSG("  Success: aci_gatt_update_char_value CHARB command\n\r");
194       }
195       /* USER CODE BEGIN CUSTOM_STM_App_Update_Service_1_Char_1*/
196
197       /* USER CODE END CUSTOM_STM_App_Update_Service_1_Char_1*/
198       break;
199
200     case CUSTOM_STM_CHARSS:
201       ret = aci_gatt_update_char_value(CustomContext.CustomEcgHdle,
202                                        CustomContext.CustomCharssHdle,
203                                        0, /* charValOffset */
204                                        SizeCharss, /* charValueLen */
205                                        (uint8_t *)  pPayload);
206       if (ret != BLE_STATUS_SUCCESS)
207       {
208         APP_DBG_MSG("  Fail   : aci_gatt_update_char_value CHARSS command, result :
    0x%x \n\r", ret);
209       }
210       else
211       {
212         APP_DBG_MSG("  Success: aci_gatt_update_char_value CHARSS command\n\r");
213       }
214
215       break;
216
217     case CUSTOM_STM_CHARECG:
218
219       ret = aci_gatt_update_char_value_ext( 0x0000, CustomContext.CustomEcgHdle,
    CustomContext.CustomCharecgHdle,
220                   0x00,
221                   200,
222                    0,
223                   20,
224                   (uint8_t *)  pPayload);
225       ret2= aci_gatt_update_char_value_ext( 0x0000, CustomContext.CustomEcgHdle,
    CustomContext.CustomCharecgHdle,
226                   0x01,
227                   200,
228                   20,
229                   180,
230                   (uint8_t *)(pPayload + 20));
231
232             if (ret != BLE_STATUS_SUCCESS || ret2 != BLE_STATUS_SUCCESS)
233             {
234               APP_DBG_MSG("  Fail   : aci_gatt_update_char_value CHARECG command,
    result : 0x%x \n\r", ret);
235             }
```

```
236             else
237             {
238               APP_DBG_MSG("  Success: aci_gatt_update_char_value CHARECG command\n
    \r");
239             }
240       if (ret == BLE_STATUS_SUCCESS && ret2 == BLE_STATUS_SUCCESS) tmp_BT++;
241 
242       break;
243 
244     default:
245       break;
246   }
247 
248   return ret;
249 }
250 
```

Bibliography

[1] AliveCor. Ecg anywhere, anytime | alivecor, 2023. URL https://www.alivecor.co.uk/.

[2] T. connectivity. *Slide Switches, Single Pole - Double Throw Configuration*, 2023. URL https://www.te.com/usa-en/product-1825232-1.datasheet.pdf.

[3] CTS. *HCMOS Clock Oscillator*, 2023. URL https://www.mouser.it/datasheet/2/96/008-0329-0-786365.pdf.

[4] R. Dahl and R. Berg. Trigonometry of the ecg: A formula for the mean electrical axis of the heart. *Physiology News*, 12 2020.

[5] ecgwaves.com. The ecg leads: electrodes, limb leads, chest (precordial) leads, 12-lead ecg (ekg) – ecg echo, 2023. URL https://ecgwaves.com/topic/ekg-ecg-leads-electrodes-systems-limb-chest-precordial/.

[6] Harwin. *S1791-42R - S1791-42R.pdf*, 2023. URL https://cdn.harwin.com/pdfs/S1791-42R.pdf.

[7] T. Instrument. *INA826 Precision, 200-µA Supply Current, 3-V to 36-V Supply Instrumentation Amplifier With Rail-to-Rail Output*, 2023. URL https://www.ti.com/lit/ds/symlink/ina826.pdf?HQS=dis-mous-null-mousermode-dsf-pf-null-wwe&ts=1687497114757&ref_url=https%253A%252F%252Fwww.mouser.de%252F.

[8] T. Instruments. *REF20xx Low-Drift, Low-Power, Dual-Output, VREF and VREF / 2 Voltage References datasheet (Rev. E) - ref2025.pdf*, 2023. URL https://www.ti.com/lit/ds/symlink/ref2025.pdf?ts=1687508017650&ref_url=https%253A%252F%252Fwww.ti.com%252Fproduct%252FREF2025.

[9] T. Instrumentt. *OPAx330 50-µV VOS, 0.25-µV/°C, 35-µA CMOS Operational Amplifiers Zero-Drift Series datasheet (Rev. G) - opa4330.pdf*, 2023. URL https://www.ti.com/lit/ds/symlink/opa4330.pdf?ts=1687518220241&ref_url=https%253A%252F%252Fwww.ti.com%252Fproduct%252FOPA4330.

[10] M. Integrated. *Dual-Input USB/AC Adapter 1-Cell Li+ Battery Chargers*, 2023. URL https://www.mouser.it/datasheet/2/609/MAX1551_MAX1555-3126749.pdf.

[11] M. Integrated. *Buck/Boost Regulating Charge Pump in μMAX*, 2023. URL https://www.mouser.it/datasheet/2/609/MAX1759-3127571.pdf.

[12] R. Kher. Signal processing techniques for removing noise from ecg signals. *Journal of Biomedical Engineering and Research*, 2019.

[13] R. N. . D. K. Malasinghe, L.P. Remote patient monitoring: a comprehensive study. *Journal of Ambient Intelligence and Humanized Computing*, pages 57–76, 2019.

[14] Microchip. *Two/Four/Eight-Channel, 153.6 ksps, Low-Noise 24-Bit Delta-Sigma ADCs*, 2023. URL https://www.mouser.it/datasheet/2/268/MCP3561_2_4_Family_Data_Sheet_DS20006181C-2257924.pdf.

[15] MikroElektronika. *Li-ion Polymer Battery Specification Model:HPL402323-2C-190mAh*, 2023. URL https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&cad=rja&uact=8&ved=2ahUKEwiakIvCm9n_AhVF_rsIHZz-ALcQFnoECBYQAQ&url=https%3A%2F%2Fdownload.mikroe.com%2Fdocuments%2Fdatasheets%2FHypercell%2520HPL402323-2C-3.7V-190mAh%2520-%2520datasheet.pdf&usg=AOvVaw2ehPxlz23L4GKCa7SLYJ-I&opi=89978449.

[16] S. Mirandola. *Progettazione di filtri attivi passa-basso e passa-alto di ordine superiore*. Zanichelli Editore, 2012. URL https://online.scuola.zanichelli.it/mirandola-files/Elettronica_V03/Capitolo_02/Mirandola_V3_02-3_Filtri_ordine_superiore.pdf.

[17] D. Motta. Design and development of a multi-parameter wearable medical device. Master's thesis, Politecnico di Torino, 2019.

[18] numpy. numpy.convolve — numpy v1.25 manual, 2023. URL https://numpy.org/doc/stable/reference/generated/numpy.convolve.html.

[19] numpy. numpy.cumsum — numpy v1.25 manual, 2023. URL https://numpy.org/doc/stable/reference/generated/numpy.cumsum.html.

[20] W. H. Organization. Cardiovascular diseases | who | regional office for africa, 2023. URL https://www.afro.who.int/health-topics/cardiovascular-diseases.

[21] scipy. scipy.signal.butter — scipy v1.10.1 manual, 2023. URL https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.butter.html.

[22] scipy. scipy.signal.filtfilt — scipy v1.10.1 manual, 2023. URL https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.filtfilt.html.

[23] scipy. scipy.signal.iirnotch — scipy v1.10.1 manual, 2023. URL https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.iirnotch.html.

[24] scipy. scipy.signal.savgol_filter — scipy v1.10.1 manual, 2023. URL https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.savgol_filter.html.

[25] R. semiconductor. *BD2222G BD2242G BD2243G : Power Management - bd2242g-e.pdf*, 2023. URL https://fscdn.rohm.com/en/products/databook/datasheet/ic/power/power_switch/bd2242g-e.pdf.

[26] STMicroelectronics. *Building wireless applications with STM32WB series MCUs - Application note - an5289-building-wireless-applications-with-stm32wb-series-mcus-stmicroelectronics.pdf*, 2023. URL https://www.st.com/resource/en/application_note/an5289-building-wireless-applications-with-stm32wb-series-mcus-stmicroelectronics.pdf.

[27] STMicroelectronics. *Guidelines for Bluetooth® Low Energy stack programming on STM32WB/ STM32WBA MCUs*, 2023. URL https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&cad=rja&uact=8&ved=2ahUKEwjW_7vgzdn_AhU7hv0HHS2LAvIQFnoECA0QAQ&url=https%3A%2F%2Fwww.st.com%2Fresource%2Fen%2Fprogramming_manual%2Fpm0271-guidelines-for-bluetooth-low-energy-stack-programming-on-stm32wb-stm32wba-mcus-stmicroelectronics.pdf&usg=AOvVaw2vWrbdcCq0kSocU_t9cflS&opi=89978449.

[28] STMicroelectronics. *Datasheet - STM32WB50CG STM32WB30CE - Multiprotocol wireless 32-bit MCU Arm®-based Cortex®-M4 with FPU, Bluetooth® 5.3 or 802.15.4 radio solution - stm32wb50cg.pdf*, 2023. URL https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&ved=2ahUKEwj9h8SAmtn_AhUphP0HHQKtCAgQFnoECBUQAQ&url=https%3A%2F%2Fwww.st.com%2Fresource%2Fen%2Fdatasheet%2Fstm32wb50cg.pdf&usg=AOvVaw3NHEUxHJXxYp1piVJ-vR31&opi=89978449.

[29] STMicroelectronics. *AN5129 Low cost PCB antenna for 2.4GHz radio*, 2023. URL https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&cad=rja&uact=8&ved=2ahUKEwjb4b24m9n_AhVE_bsIHRZNAa8QFnoECAoQAQ&url=https%3A%2F%2Fwww.st.com%2Fresource%2Fen%2Fapplication_note%2Fan5129-low-cost-pcb-antenna-for-24ghz-radio-meander-d

esign-for-stm32wb-series-stmicroelectronics.pdf&usg=AOvVaw1TLCuik68QIgQ41Wwwmtq6&opi=89978449.

[30] N. V. Thakor, J. G. Webster, and W. J. Tompkins. Estimation of qrs complex power spectra for design of a qrs filter. *IEEE Transactions on Biomedical Engineering*, BME-31:702–706, 1984.