UNIVERSITÀ DI PISA
Laurea Magistrale in Ingegneria Biomedica

Design of an open source AED in the framework of the UBORA project

Supervisors

Prof. Ahluwalia Arti Devi
Ing. De Maria Carmelo

Candidate

Jacopo Ferretti

2017

Contents

List of Tables

List of Figures

Introduction

Amongst the most impending health challenges of modern society, the Sudden Cardiac Arrest (SCA) is probably one of the most deadly. SCA alone is responsible every year for over 300'000 deaths in the United States. As a mean of comparison, his number is greater than the combined number of those who die from Alzheimer's disease, assault with firearms, breast cancer, cervical cancer, colorectal cancer, diabetes, HIV, house fires, motor vehicle accidents, prostate cancer, and suicides.

SCA is a condition in which the heart suddenly and unexpectedly stops beating in an ordered fashion way, and instead start exhibiting a chaotic behaviour. When this happens, blood stops flowing to the brain and other vital organs. In these conditions the patient may be considered to all effects dead, and will remain so unless someone help him/her immediately. The most effective way to treat a SCA is with defibrillation, namely a therapeutic dose of electrical energy. The necessity of performing defibrillation as soon as possible -in order to significantly raise the chance of resuscitation- has led to the development of Automated External Defibrillators (AED).

AEDs are portable devices capable of automatically diagnose whether a patient is experiencing SCA or not, and treat him/her through defibrillation when needed. The critical aspect of AEDs, is the ease of use that allow their operation with a very basic training. This, united with a capillary diffusion, could drastically reduce the number of deaths for SCA. For these reasons, it is in my opinion that the open source approach is the optimal solution. An open source approach lead to reduced development costs, improvement on the operations, training on the engineering aspects of defibrillation, and finally a more sensitization toward the -mostly underrated- subject of SCA.

The goal of this thesis therefore, was the development of an Open-source Automated External Defibrillator, named OAED. The device was designed keeping in mind: compliance with the regulatory standards required to obtain the CE marking; the necessity of using it as a teaching instrument; the low costs required for a greater diffusion; and competitiveness with the existing devices, in terms of efficiency, reliability, and safety.

The thesis will be structured in three different subsequent parts: the background, the hardware, and the software. The first chapter of the background contain: a brief introduction on the UBORA project, and some generalities about SCA, defibrillators, and AEDs. The second chapter then, will deal about the regulatory required to obtain the CE marking, and its related

standards. Furthermore, at the end of chapter two there will be a conceptual design for OAED.

In the second part there will be an in-depth analysis of the hardware design of OAED. The chapter three will contain the schematics and models used to design the circuitry that perform defibrillation. While the fourth chapter contain those used to assert SCA and to control the device.

The third and final part will contain the software of OAED. The fifth chapter will deal the basic aspects of the firmware and OAED operations. Finally, in the sixth chapter there will be a description of the algorithms used to assert SCA.

Part I

Background

CHAPTER 1

An overview on the UBORA project and the Automated External Defibrillator

1.1 Project UBORA

The revolutionary breakthrough of technology in the past two decades have had repercussion on the world in many different ways. One of the most important of those effect is the improvement of healthcare system which increased life expectancy. [1]
This higher life expectation is in turn shifting world's population mean age towards increasing values. Therefore the priority of both pharmaceutical and biomedical market/research are changing accordingly.
While these changes can be seen as a positive drift of the world's condition, this whole scenario is accompanied by a huge increase in economic inequality – the world's 62 richest people now own more than its poorest half of 3.6 billion [2].

Besides the explosives growth of eastern Asia, Europe and Africa are the continents which are going through the most significant changes. The recent geopolitical events are causing one of the greatest migrations towards European Union since World War II and it is Europe social and moral responsibility at this time to offer safe, yet affordable, medical treatment to immigrants. On the other hand, in Africa, a growing urban middle class is willing to pay for better treatment, and governments are now beginning to provide better healthcare facilities and increased access to medicine, at least in urban areas. Even thought much of its rural population still does not have access to quality healthcare.

UBORA: Euro-African Open Biomedical Engineering e-Platform for Innovation through Education. In this evolving scenario, UBORA ("excellence" in Swahili) project addresses the shifts in the economy and those related to changes in population dynamics by supporting a more equilibrated distribution of health and wealth through the development of an e-Infrastructure oriented towards open innovation in biomedical device de-

sign.

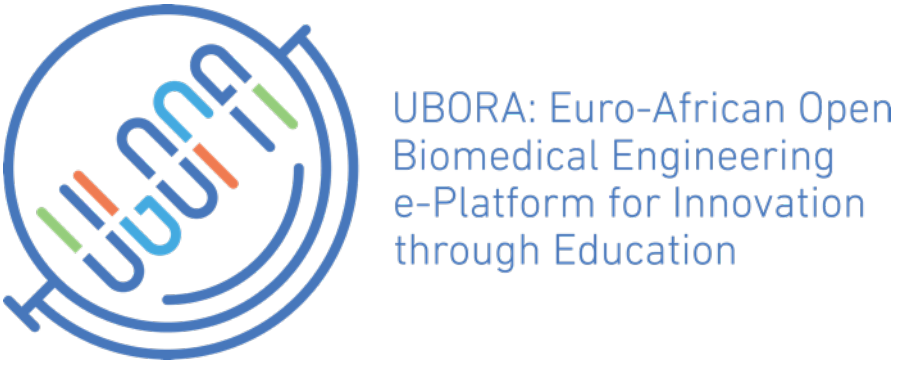


Figure 1.1: UBORA logo

1.1.1 What is UBORA?

The UBORA project aims at creating an European Union-Africa e-Infrastructure, for open source co-design of new solutions to face the current and future healthcare challenges of both continents, by exploiting networking, knowledge on rapid prototyping of new ideas, and sharing of safety criteria and performance data. The e-Infrastructure will foster advances in education and the development of innovative solutions in Biomedical Engineering, both of which are flywheels for European and African economies.

It is conceived as a virtual platform for generating, exchanging, improving and implementing creative ideas in Biomedical Engineering (BME) underpinned by a solid safety assessment framework.

Through the UBORA e-Infrastructure, the biomedical community will generate and share open data and blueprints of biomedical devices, accompanied by the required procedures for respecting quality assurance, and assessing performance and safety. When properly implemented, as guaranteed by authorised Notified Bodies, these biomedical devices can safely be used in hospitals and on patients.

In a nutshell, UBORA couples the open design philosophy with Europe's leadership in quality control and safety assurance, guaranteeing better health and new opportunities for growth and innovation.

Specific objectives of the project are:

- **to create UBORA, a web-based platform for co-design, and sharing data and blueprints of biomedical devices after vetting by trained biomedical engineers;**
- **to generate and upload a complete set of open access projects, data on performance and safety, and designs of biomedical devices to the platform;**
- **to empower innovations in healthcare, sustained by a solid academic background in BME.**

UBORA's e-Infrastructure will be aimed at stimulating innovation in the field of BME through knowledge distribution, promoting harmonisation of biomedical device requirements, design and manufacturing, with subsequent impacts on healthcare services and ultimately on patient safety.

Furthermore, UBORA as a tool for co-design and sharing ideas will be a point of reference for those academic institutions which train in BME (and related training courses), fostering harmonization of teaching curricula, contributing to breaking down barriers for highly qualified workers. Through UBORA both Europe and Africa will develop the capacity to create and exploit new technologies, products and services in the context of global competition.

Development of Open Source medical devices

While, a couple of years ago, the development of biomedical devices was essentially linked to companies and universities, now several examples of open source biomedical devices have appeared on the web [3], but seldom are they designed to be compliant with safety standards (e.g. the Gammasoft Open electrocardiogram [4]).

Although, at present, some of these instruments are not accurate or safe enough to be inserted in the clinical routine, their use can probably save a life more than a damaged, unused (e.g., for high cost) or useless (e.g., because no one knows how to operate) Magnetic Resonance Imaging machine. Indeed, software-reliant devices have also brought on new types of potential risks for patients. Given that medical software (and hardware) is proprietary and patent-protected, thus veiled in secrecy [5, 6], there are difficulties in exposing specific problems with these products. The open-source approach could, in theory, make it easier to fix, or even avoid, dangerous flaws before they hurt or kill hundreds or thousands of patients.

Today, thanks to crowd-thinking and crowd-sourcing, the design of several products has an intrinsic revision process, driven by a virtual community, composed of a heterogeneous and large population (from highly skilled designers to laymen), which has become an active player, and no longer a passive element [7, 8, 9, 10, 11]. This community is the best analyst in terms of quality, reliability and feasibility. While this philosophy is now well accepted in the "software" world, there is still an unjustified unbelief in open "hardware", because many people are anchored to the consolidated production processes, in which product development is affected by high costs due to the inflexibility of high throughput fabrication technologies (e.g., injection molding). As described in the seminal work of Chris Anderson "In the next industrial revolution, atoms are the new bits" [12, 13], additive manufacturing technologies (known also as 3D printing) are giving everyone, companies, makers, and inventors, the tools that were the exclusive prerogative of a few companies less than ten years ago.

According to UBORA consortium, academia, and specifically biomedical engineers in higher education, must embrace these new tools, and pass on the message that an Open Source product, developed by a community, without a multinational brand is not equal to un-reliable.

A note of caution however; the freedom given by the Web, and by the possibility to share, fork and re-implement projects, which characterises the Open Source Software, Electronic, and Hardware world, has one major drawback: organizing information (schematics, blueprints) and quality control are the boring parts that are not always pursued in a passion-driven, and self-assembled community. **In the context of BME however, this latter aspect is critical for ensuring safety and efficacy of biomedical devices, and must go hand in hand with the adoption of open resources for medical applications [14].**

1.1.2 Impacts

UBORA promotes the open-access, low-cost, 3D printing-oriented design approach for medical devices for industries and institutions in Africa as well as in emerging countries and Europe, in order to increase their safety, reliability but also for finding new solutions to existing health problems.

Impacts on healthcare

Establishing an e-Infrastructure of research and innovation able to respond to real local issues, in order to provide clinics and hospitals with more effective, safe and efficient devices for the diagnosis and treatment of diseases, will have positive impacts on healthcare.

Impact on education

UBORA, as a tool for design biomedical device, will encourage upgrading of curricula based on solid engineering principles with courses on new fabrication technologies, and with new ways of thinking and problem solving.

Impacts on innovation and research

The UBORA e-Infrastructure foresees the cooperation of scientists and students, technologists, clinicians, technicians, regulators and designers who can set up design and research initiatives in the fields of BME, creating a strong, academic research community in Europe and Africa.
Positive impacts will be the reinforcement of research on sustainable healthcare and the fostering of an innovative and sharing spirit among young engineers, who are our future leaders.

The blueprints of the devices developed in UBORA, which comprise not only the designs, but also the proper guidelines and data for needs, quality and safety assessment will be shared by and through the UBORA community, thus consolidating the ties between the European Union and Africa.

Impact on harmonization of Medical Devices directives

To ensure safety and health in a country it is essential to have regulations especially in the field of medical products. A widely held belief is that African countries do not have any regulations on medical devices, but this is not completely correct. About 10 years ago, the WHO performed a study on the presence of National Regulatory Authorities (NRA) in the 46 sub-Saharan African countries. The responsibility of NRAs is to regulate and control besides medicines, vaccines and blood products also medical devices. Only 7% of these countries had a National Regulatory Authority (NRA) in place, while 63% had minimal and 30% no regulations [15]. Thanks to the international efforts and collaborations with African organizations (like the African Union or the African Organization for Standardization) as well as African governments and physicians, single guidelines and regulations were implemented in some countries.

Generally speaking, African countries orient their regulatory processes on the European system. As an example, Egypt implemented the European Medical Device Directive with a European classification of medical products, while Ethiopia or South Africa established their own laws that are harmonized with the European regulations.

Other countries were recommended to use other countries' approved equipment supplier lists, e.g. South Africa's public documents, while working towards establishing a regulatory authority for medical equipment [16]. Nowadays, all ABEC countries have a NRA to regulate and control medical devices, but in several cases they are also responsible for food and medicines control: medical device regulations require a different approach than those for food or medicines. Furthermore, the majority of those countries still have limited capacities to implement these regulations. According to a study performed in 2012 by Rugera et al. [17], those limited capacities might include lack of investment and training to improve and maintain knowledge and skills of personnel.

UBORA, with its focus on safety and regulations and with its dissemination activities, both on the ground (i.e. students and academic personnel) and through policy making at government level, will promote and push harmonzation on medical devices throughout the ABEC countries and beyond. European regulations will be the benchmark, facilitating trade of devices from European manufacturers in Africa.

Impact on economic growth and cooperation

A well-established healthcare sector is an indicator of the economic status of a country. Odrakiewicz [18] examined the existence of a two-way relationship between health and income.

The UBORA e-Infrastructure will contribute to economic growth both indirectly by providing more reliable and safer medical devices, which will have impact on healthcare, and directly, by playing a role into the medical device market whose value was 328 B$ in 2013 and is growing at about 6% annually [19]. These trends are even faster in Africa, with Kenya, for example, which has a rate of 9.2% (Figure 1.2).

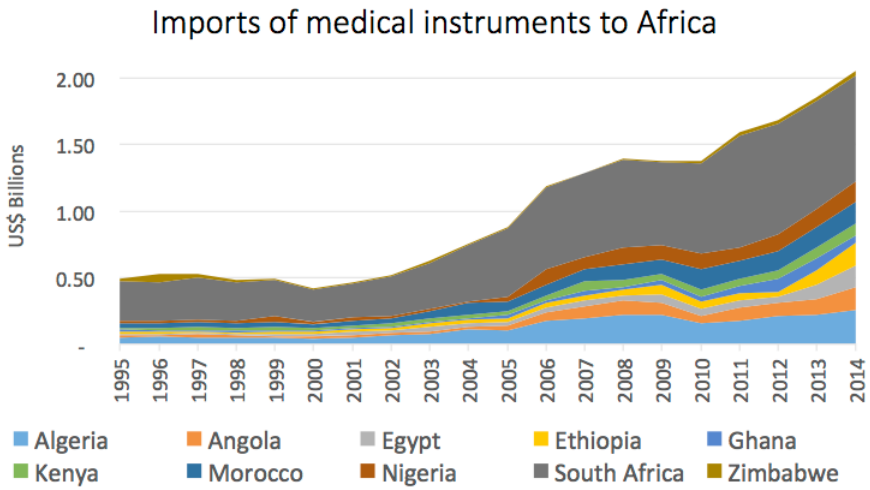


Figure 1.2: Imports of medical instruments to Africa

The Open Source approach, followed in UBORA, will represent a further added value [20] and is compliant with different business models [21]. In fact, contrary to the prevailing popular conception, Open does not mean free but implies the accessibility to project blueprints, giving people the freedom to control their technology while sharing knowledge and encouraging commerce through the open exchange of designs.

Summarizing, UBORA promotes economic benefit and capacity building: the e-Infrastructure will be a point of reference for creating a substrate of highly-skilled personnel and for transforming ideas by an open, cloud and crowd approach into products.

1.2 Automated External Defibrillator

The main goal of this thesis is to design an Open-source Automated External Defibrillator (OAED) as part of the UBORA project.

Figure 1.3: AED symbol

An Automated External Defibrillator (AED) (symbol on Figure 1.3) is a portable electronic device that automatically diagnoses the life-threatening Sudden Cardiac Arrest (SCA); and is able to treat him/her through defibrillation, that is the application of an electrical therapy which stops the arrhythmia, allowing the heart to re-establish an effective rhythm [22].

Sudden Cardiac Arrest is one of the leading cause of death in the world. SCA alone is responsible every year for over 300'000 deaths in the United States. This number is greater than the combined number of those who die from Alzheimer's disease, assault with firearms, breast cancer, cervical cancer, colorectal cancer, diabetes, HIV, house fires, motor vehicle accidents, prostate cancer, and suicides [23].

What makes SCA so lethal, is the immediate need for assistance combined with the possibility to happen at any time. Without intervention the victim will die within a few minutes. Even in the best possible scenario, is highly improbable that a medical team is able to reach the victim in this short time, and thus his life is in the hands of the bystanders.

1.2.1 Sudden Cardiac Arrest

Sudden Cardiac Arrest is a condition in which the heart suddenly and unexpectedly stops beating. When this happens, blood stops flowing to the brain and other vital organs. In these conditions the patient may be considered to all effects dead, and will remain so unless someone help him/her immediately with cardiopulmonary resuscitation (CPR), defibrillation, advanced cardiac life support, and/or mild therapeutic hypothermia.

Although cerebral neurons can tolerate up to 20 minutes of normothermic ischemic anoxia, cerebral recovery from more than 5 minutes of cardiac arrest is hampered by complex secondary derangements of multiple organ systems after re-perfusion [24] (The so called post-resuscitation syndrome). Therefore even if the victim could survive enough time for the medical team to arrive, the chances of post-resuscitation syndrome and further health issues -such as brain damages- drastically increases with time.

Causes

In SCA, as result of a bio-electrical dysfunction, the heart stops beating in an efficient, organized manner, and instead start manifesting a chaotic

behaviour. The pumping action thereby, becomes ineffective and the blood flow stops. It also often occurs in active people who seem to be healthy and have no known medical conditions. In fact, for some patients SCA is the first indication of a heart condition.

SCA can manifest in two different heart rhythm alterations. Ventricular Tachycardia (VT), which is the early stage of SCA, and Ventricular Fibrillation (VF), which is the advanced stage of SCA -and the real life threatening condition. Both these conditions can be diagnosed by watching the ECG.

Electrocardiography

Electrocardiogram (ECG) is the process of recording the electrical activity of the heart over a period of time using electrodes placed on the skin. These electrodes detect the tiny electrical changes on the skin that arise from the heart muscle's electro-physiologic pattern of depolarizing during each heartbeat. It is a very commonly performed cardiology test.
During each heartbeat, a healthy heart has an orderly progression of depolarization that starts with pacemaker cells in the sinoatrial node, spreads out through the atrium, passes through the atrioventricular node down into the bundle of His and into the Purkinje fibers, spreading down and to the left throughout the ventricles. This orderly pattern of depolarization gives rise to the characteristic ECG tracing (Figure 1.4). To the trained clinician, an ECG conveys a large amount of information about the structure of the heart and the function of its electrical conduction system. Among other things, an ECG can be used to measure the rate and rhythm of heartbeats, the size and position of the heart chambers, the presence of any damage to the heart's muscle cells or conduction system, the effects of cardiac drugs, and the function of implanted pacemakers.

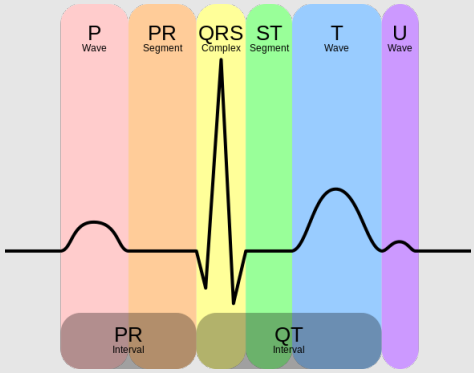


Figure 1.4: Schematic representation of normal ECG

Interpretation of the ECG is ultimately that of pattern recognition.

Normal rhythm produces four entities (see Figure 1.4) that each have a fairly unique pattern:

- the **P wave** represents atrial depolarization;
- the **QRS complex** represents ventricular de-polarization;
- the **T wave** represents ventricular re-polarization;
- the **U wave** represents papillary muscle re-polarization.

Normal analysis of ECG tracks involves investigation of the shape, length, rate, and width of every segment.

Ventricular Tachycardia (VT) is defined as a sequence of three or more ventricular beats with a frequency higher than 100bpm -usually between 110 and 250bpm. Although a few seconds may not result in problems, longer periods are dangerous and may deteriorate quickly enough into ventricular fibrillation. The cardiac output during sustained VT is often strongly reduced resulting in hypotension and loss of conciousness.
Depending on its duration or morphology Ventricular Tachycardia can be classified in:

- **Non-sustained VT :** a condition where three or more ventricular contractions happen within a maximum duration of thirty seconds.
- **Sustained VT :** a VT which extends to more than 30 seconds of duration.
- **Monomorphic VT :** a VT where all ventricular beats have the same configuration (Figure 1.5).
- **Polymorphic VT :** a VT in which ventricular beats have a changing configuration.
- **Biphasic VT :** a VT in which QRS complex alternates from beat to beat. (usually associated with digoxin intoxication)

Ventricular Tachycardia it's usually caused when the cardiac pulse is not generated in the sino-atrial node, but instead in irritated ventricular cell, which are significantly faster ($\sim 120bpm$). Another cause is the presence of non conductive scar tissue which may cause the formation of a re-entrance circuit. Hence, once the pulse is generated, it start circulating around the heart causing spontaneous uncontrolled contraction.

Ventricular Fibrillation (VF) is defined as the chaotic and uncoordinated depolarisation of the ventricles. This result in cardiac arrest, with consequent loss of consciousness and pulse. Followed by death in few minutes if no action is taken.
Ventricular Fibrillation causes are not always entirely clear. It may due to:

- heart tissue stress or damages caused by structural or electrical alterations;

- tissue heterogeneity with anomalous behaviour;
- drugs, and drugs overdose;
- electrochemical imbalances;
- ischemia;
- others.

There are however reports of cases in absence of other heart pathology or evident cause.

Diagnosis

Apart from the obvious symptoms, such as unconsciousness and absence of pulse, visual inspection of ECG is the main way of diagnosis for both VT and VF.

VT is relatively tricky to diagnose. The ECG track appears as a sequence of high rate, wide QRS complexes (see Figure 1.5). Morphology depends on VT type, and the recognition difficulties vary from different types.

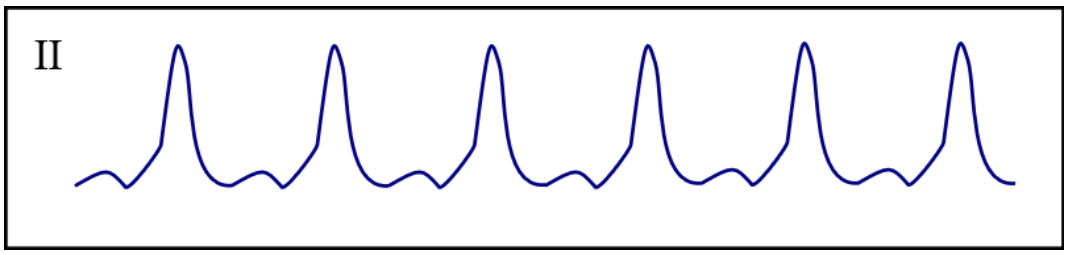


Figure 1.5: Example of Ventricular Tachycardia ECG

VF unlike VT -in which it's possible to see a regular albeit altered rhythm- is characterized by a chaotic morphology, with irregular unformed QRS complexes and the absence of clear P waves (Figure 1.6).

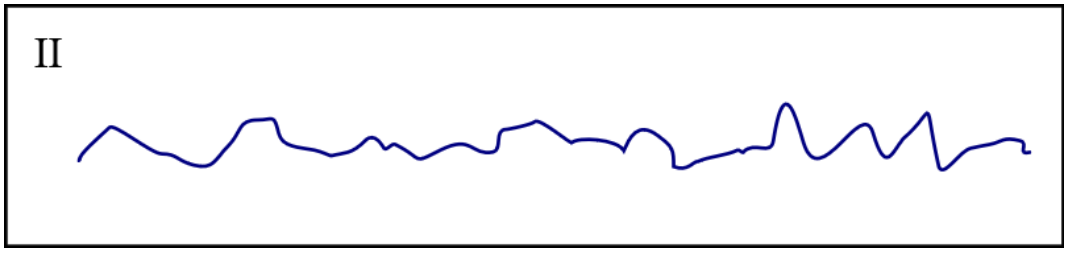


Figure 1.6: Example of Ventricular Fibrillation ECG

Treatment

Although VT can also effectively be treated with cardioversion, medication, and surgery; defibrillation remain the preferable option. Good timing is vital to avoid further damage and/or the degeneration to VF.

Things are different for VF, which is only treatable with immediate defibrillation.

1.2.2 Defibrillation

By definition, defibrillation consist of delivering a therapeutic dose of electrical current to the heart. This depolarizes a critical mass of the heart muscle, terminates the arrhythmia and allows normal sinus rhythm to be re-established by the heart's natural pacemaker, in the sino-atrial node.

In simple terms, defibrillation "reset" the heart bio-electrical functionalities. By its nature, the heart has different fail-safe measures which ensure that once the synchronization is restored (e.g. with the defibrillation), then its pacing capabilities re-start in a ordered fashion way.

History of defibrillation

Defibrillators were first demonstrated in 1899 by Jean-Louis Prévost and Frédéric Batelli, two physiologists from University of Geneva, Switzerland. They discovered that small electrical shocks could induce ventricular fibrillation in dogs, and that larger charges would reverse the condition [25].

In 1933, Dr. Albert Hyman, heart specialist and C. Henry Hyman, an electrical engineer, looking for an alternative to injecting powerful drugs directly into the heart, came up with an invention that used an electrical shock in place of drug injection. This invention was called the Hyman Otor where a hollow needle is used to pass an insulated wire to the heart area to deliver the electrical shock. The hollow steel needle acted as one end of the circuit and the tip of the insulated wire the other end. Whether the Hyman Otor was a success is unknown [26].

The external defibrillator as known today was invented by Electrical Engineer William Kouwenhoven in 1930. William studied the relation between the electric shocks and its effects on human heart when he was a student at Johns Hopkins University School of Engineering. His studies helped him to invent a device for external jump start of the heart. He invented the defibrillator and tested on a dog, like Prévost and Batelli. The first use on a human was in 1947 by Claude Beck [27], professor of surgery at Case Western Reserve University. Beck's theory was that ventricular fibrillation often occurred in hearts which were fundamentally healthy, in his terms "Hearts that are too good to die", and that there must be a way of saving them. Beck first used the technique successfully on a 14-year-old boy who was being operated on for a congenital chest defect. The boy's chest was surgically opened, and manual cardiac massage was undertaken for 45 minutes until the arrival of the defibrillator. Beck used internal paddles on either side of

the heart, along with procainamide, an antiarrhythmic drug, and achieved return of normal sinus rhythm.

These early defibrillators used the alternating current from a power socket, transformed from the 110–240 volts available in the line, up to between 300 and 1000 volts, to the exposed heart by way of "paddle" type electrodes. The technique was often ineffective in reverting VF while morphological studies showed damage to the cells of the heart muscle post mortem. The nature of the AC machine with a large transformer also made these units very hard to transport, and they tended to be large units on wheels.

Until the early 1950s, defibrillation of the heart was possible only when the chest cavity was open during surgery. The closed-chest defibrillator device which applied an alternating voltage of greater than 1000 volts, conducted by means of externally applied electrodes through the chest cage to the heart, was pioneered by Dr V. Eskin with assistance by A. Klimov in Frunze, USSR (today known as Bishkek, Kyrgyzstan) in the mid-1950s [28].

A major breakthrough was the introduction of portable defibrillators used out of the hospital. In Soviet Union, a portable defibrillator, named DPA-3, was reported in 1959. In the western countries this was pioneered in the early 1960s by Prof. Frank Pantridge in Belfast. Today portable defibrillators are among the many very important tools carried by ambulances. They are the only proven way to resuscitate a person who has had a cardiac arrest unwitnessed by Emergency Medical Services (EMS) who is still in persistent ventricular fibrillation or ventricular tachycardia at the arrival of pre-hospital providers.

Gradual improvements in the design of defibrillators, partly based on the work developing implanted versions, have led to the availability of AEDs.

More technical features of defibrillators and AEDs will be discussed in detail during the course of this thesis.

1.2.3 AED operation guideline

As already mentioned before, an AED is a device capable of automatically diagnose whether a patient is experiencing SCA or not, and treat him/her through defibrillation when needed. Since most of the operation is automated and performed by the device, operators only need few or no medicine knowledge, making possible its use by virtually anybody.

An AED only has a pair of electrodes to be correctly placed on the patient (Figure 1.7), and this is the only difficulty of use. Once the electrodes are placed the AED automatically perform ECG analysis on the patient to evaluate the necessity of defibrillation. In positive cases the AED notify the operator that defibrillation is indeed required, and wait for him/her to push the "defibrillation" button. The defibrillation release is left to the operator, in order to avoid possible hazardous situations -e.g. the operator should not touch the patient during defibrillation.

The relative ease of use, coupled with a capillary diffusion of AED could

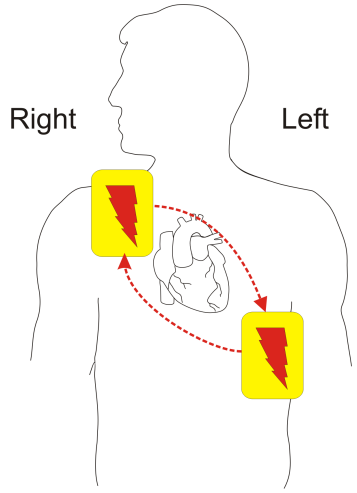


Figure 1.7: Placement of electrodes for defibrillation

significantly rise the chances of surviving a Sudden Cardiac Arrest.

1.3 Objectives

The main goal of this thesis is designing an open source Automated External Defibrillator (named OAED) to help the diffusion of these devices. To achieve this, the proposed device has to be designed keeping in mind:

- compliancy with the conformity standards,
- safety,
- reliability,
- ease of use and fabrication, and
- cost containment.

1.3.1 Open source

As previously said the open source approach has different benefits, in this peculiar case we can summarize them in:

- it provides lower development costs;
- it give the possibility of customizing the device according to particular needs of different scenarios;
- it offer a practical example of a functional device for teaching purposes; and
- all its functionalities can be improved by anyone -given enough knowledges.

At the time of writing there are no other open-source AED -probably due to the critical aspects of this particular device.

Despite many commercial product already exists, the costs and maintenance of these devices slow their spread, limiting their presence only to the main cities. Another concern about commercial AED is that in the past most of them lacked in reliability. In the last few years many devices were withdrawn from the market from every manufacturer over the world due to various technical malfunctioning which led to over 750 deaths between 2004 and 2009 [29]. An open source approach could have avoided those malfunctioning and saved those peoples.

Both OAED's source code and electronic schematics will be available via on-line repositories to be used without limitations.

1.3.2 Impacts

OAED's aim to help the widespread diffusion of AEDs in pursuit of preventing death from Sudden Cardiac Arrest. Given the almost total unpredictability of SCA occurrence, anybody is a valid target for this device, thereby the expected impacts on life are high.

The open source approach can facilitate the production and distribution of AEDs thanks to lower costs, and to less time required for the development and adaptation.

In particular -regarding adaptation- the open source approach make possible the complete customization of the device with relative ease. This further facilitate the diffusion of AEDs by adjusting the device according to the particular needs of different scenarios.

For instance, in an isolated service station, an AED could include a solar panel to charge the battery, and/or a gsm module to notify its use and malfunctioning at the nearest emergency centre simplifying maintenance, or speeding the rescue.
Another example is a high-density metropolitan area where the presence of many devices with bluetooth/wifi functionalities can notify use, malfunctioning, or battery depletion.

As regards the teaching, given the strong multidisciplinary nature of the project, OAED can be used to train engineers and technicians about:

- regulatory of electro-medical devices;
- electronic and PCB design;
- high-voltage electronics design and applications;
- bio-electrical signal acquisition;
- signal processing;
- programming;
- embedded systems design; and
- 3D printing.

A final expected collateral effect of OAED, are the sensitization towards SCA, and the importance of CPR training. Anyone can find themselves

in need to perform resuscitation, and the importance of an AED in that situation -united with the proper preparation- can *really*make a difference.

Summarizing, so far the first chapter illustrated the root and the motivations behind this project. On chapter 2 there will be a detailed description of the regulations necessary to design a compliant AED. From these the technical specifications and a conceptual design will be outlined. Subsequently, in chapter 3, and chapter 4 will be described the electronic circuits on high-voltage board and control board, respectively. Then, on chapter 5 there will be the firmware and signal analysis discussion, and on chapter 6 there will be a description of SCA recognition algorithms. Finally, chapter 7 will contain conclusions and possible future developments of OAED.

Bibliography

[1] World Health Organization. *Global Health Observatory (GHO) data*. http://www.who.int/gho/mortality_burden_disease/life_tables/situation_trends/en/.

[2] Fuentes-Nieva R. Hardoon D Ayele S. "An Economy for the 1%". In: (2016). https://www.oxfam.org/en/research/economy-1.

[3] Thimbleby H. Niezen G. Eslambolchilar P. "Open-source hardware for medical devices." In: (2016).

[4] Gamma Cardio Soft S.r.l. "http://www.gammacardiosoft.it/openecg/". In: ().

[5] The Economist. "Open-source medical devices: When code can kill or cure." In: (). http://www.economist.com/node/21556098.

[6] Pallikarakis N. Bliznakov Z. Mitalas G. "Analysis and Classification of Medical Device Recalls". In: (2007).

[7] Wikimedia Foundation Inc. *http://www.wikipedia.org*.

[8] Makerbot Industries LLC. *http://www.thingiverse.com*.

[9] Youmagine 3D BV. *http://www.youmagine.com*.

[10] Grafman L. *http://www.appropedia.org/*.

[11] Jakubowski M. *http://opensourceecology.org/*.

[12] Anderson C. "Makers: the new industrial revolution." In: (2012).

[13] Anderson C. "In the next industrial revolution, atoms are the new bits." In: (2014).

[14] Ahluwalia A. De Maria C. Mazzei D. "Improving African Healthcare through Open Source Biomedical Engineering". In: *International Journal on Advances in Life Sciences* (2015).

[15] Lamph S. "Regulation of medical devices outside the European Union". In: (2012).

[16] Musonda G. Mullally S. Bbuku T. "Medical equipment maintenance personnel and training in Zambia". In: (2012).

[17] Kajumbula H. Akimana G. Mariki R.F. "Regulation of medical diagnostics and medical devices in the East African community partner states. BMC Health Services Research". In: (2014).

[18] Odrakiewicz D. "The Connection Between Health and Economic Growth: Policy Implications Re-examined". In: (2012).

[19] Madete J. "The BME market in Africa, Expert Group Meeting on the development and management of a regional innovation platform". In: (2016).

[20] Pearce JM. "Quantifying the Value of Open Source Hardware Development." In: (2015). http://dx.doi.org/10.4236/me.2015.61001.

[21] McAloone T.C. Achiche S. Özkil A.G. "Open Design and Crowdsourcing: Maturity, Methodology and Business Models". In: (2012).

[22] Wikipedia. *Automated external defibrillator — Wikipedia, The Free Encyclopedia*. https://en.wikipedia.org/wiki/Automated_external_defibrillator. 2016.

[23] Dariush Mozaffarian, Emelia J. Benjamin, and et al. Go. "Heart Disease and Stroke Statistics—2015 Update". In: *Circulation* (2014). ISSN: 0009-7322. DOI: 10.1161/CIR.0000000000000152. URL: http://circ.ahajournals.org/content/early/2014/12/18/CIR.0000000000000152.

[24] P Safar. "Cerebral resuscitation after cardiac arrest: a review". In: *Circulation* 74.6 Pt 2 (Dec. 1986). http://europepmc.org/abstract/MED/3536160, pp. IV138–53. ISSN: 0009-7322.

[25] N. Lockyer. *Nature*. v. 61. Macmillan Journals Limited, 1900.

[26] "Self Starter For Dead Man's Heart". In: *Popular Science* (1933).

[27] Claude Beck. "Defibrillation and CPR". In: ().

[28] Lysenko VI. "[Some results with the use of the DPA-3 defibrillator (developed by V. Ia. Eskin and A. M. Klimov) in the treatment of terminal states]." In: (1965).

[29] Mark Harris. "The Shocking Truth About Defibrillators". In: (2012). URL: http://spectrum.ieee.org/biomedical/devices/the-shocking-truth-about-defibrillators.

CHAPTER 2

Regulations

Biomedical devices -just like any other device- need to adhere certain conformity standard in order to be sold, or even just be used.

Although it is of vital importance, the regulatory process of a device is often overlooked, or not given enough prominence. It is not in common use to teach the regulation in universities, and this brings new engineers to underestimate the problem, or to worry about it only after it becomes inevitable -which is often too late. A "*perfect*" device who can perform amazing diagnosis it's useless if it does not comply with the regulatory standards, because it cannot be used -not to mention of being sold.

Regulations is not only required for "*bureaucratic*" purpose, but also to make sure the device is safe, functional, and actually capable of offering a benefit. E.g. A scalpel which cuts its user when gripped in certain way has a bad design, and it should not be used or sold. But how to decide if a design is bad or not? Here is where regulation comes into play.

Figure 2.1: CE logo

In European Union (EU), a device compliant with the required standards -namely a device which passed all conformity tests- is recognized by the CE marking (Figure 2.1), as described in EEC medical devices directive 93/42 (MDD 93/42).

In this chapter we are going to see the necessary steps towards the regulation of a medical device -needed to obtain the CE marking- with particular regards on medical electrical devices and specifically AEDs

The first step towards the device's regulation then, is to find the correct nomenclature. Even an object simple as a needle has to be accurately described in order to avoid confusion: what kind of needle is it? Is it a suture needle? A syringe needle? Or perhaps a biopsy one? There are many types of needles, all for different purposes, and their regulation may vary from type to type.

Fortunately, databases are available with all the information necessary to find the unique nomenclature for each device (one of those is GMDN, described on section 2.1.1).

After establishing the unique name of the device, it is time to define its

class. Classes of biomedical devices are described in detail in EEC medical devices directive 93/42. In this document there is also a description of the Essential Requirements (ERs) necessary to apply the CE marking. Compliance to these Essential Requirements is ensured by designing the device following directives described in specific norms, different for every device. Thereafter, the technical and safety specifications are obtained from the norms in form of Essential Requirements and Essential Performance.

Those are the mandatory guidelines necessary to properly design the device.

When the design process is finished, a prototype of the device shall be built and tested in order to ensure that the design is actually compliant with all the required standards. These tests are also provided in every norm as part of the design guideline.

2.1 Regulation of biomedical devices

2.1.1 Global Medical Device Nomenclature

As previously said, GMDN is a list of generic names used to identify all medical device products. Such products include those used in the diagnosis, prevention, monitoring, treatment or alleviation of disease or injury in humans.

The main purpose of the Global Medical Device Nomenclature (GMDN) is to provide health authorities and regulators, health-care providers, manufacturers and others with a naming system that can be used to exchange medical device information and support patient safety.
The GMDN is used for:

- Data exchange between manufacturers, regulators and healthcare authorities;
- Exchange of post-market vigilance information;
- Supporting inventory control in hospitals;
- Purchasing and supply chain management.

Medical device experts from around the world (manufacturers, healthcare authorities and regulators) compiled the GMDN according to the international standard ISO 15225.
The GMDN is recommended by the International Medical Device Regulators Forum (IMDRF) and is now used by over 70 national medical device regulators to support their activity. The GMDN is managed by the GMDN Agency, a registered charity, which has a Board of Trustees, which represent regulators and industry.

The GMDN is updated by member change requests. New and updated GMDN terms are published on the member website, the GMDN

Database. Information in the form of a 5 digit numeric GMDN Code is cross-referenced to a precisely defined Term Name and Definition.

Access to the GMDN Database is priced according to organization size [1].

UMDNS A valid alternative to GMDN is the Universal Medical Device Nomenclature System (UMDNS). UMDNS is a standard, free of charge, monthly updated, international nomenclature and computer coding system to help you better manage medical devices. UMDNS has been officially adopted by many nations.

UMDNS facilitates identifying, processing, filing, storing, retrieving, transferring, and communicating data about medical devices. The nomenclature is used in applications ranging from hospital inventory and work-order controls to national agency medical device regulatory systems and from e-commerce and procurement to medical device databases [2].

2.1.2 EEC medical devices directive 93/42 (MDD 93/42)

In the introduction of this chapter we said that a medical device *cannot be used or sold* without the CE marking. And this is only applied after following the EEC medical devices directive 93/42 (MDD 93/42). In this section we are going to see in detail what does that mean.

MDD 93/42 is a document which sets out the general criteria to be used in the design and construction of certain medical devices categories. It was first published in the Official Journal of the European Union in June 1993 [3], and most recently reviewed and amended by the 2007/47/EC [4] with a number of changes. Compliance with the revised directive became mandatory on March 21, 2010.

Contents

The directive scope is to propose provisions that are common to all member states of the European Community for the certification of medical devices. Despite of this, the directive *is not* a list of requirements for two main reasons:

1. the prescriptive standards require periodic revision, and due to the rapid technological progress these days in this area, the rules should be revised frequently (every 2 or 3 years), which would lead to high costs;
2. and thanks to the progress we can rely on a broader variety of instrumentation. Therefore it is possible (if not very likely) to obtain a safe device by following many different approaches having the same validity, and it would be impossible to consider them all.

Another effect of the former considerations, is that MDD 93/42 only has a descriptive form for the ERs. This means that ERs are expressed in a

verbal form, i.e. *"The device shall not harm the user"*. The technical aspect of how pursuit the ERs is left to specific norms (some of which are treated in the next section of this chapter), i.e. IEC 60601.

MDD 93/42 contains twenty-five articles and thirteen annexes, resumed respectively in Table 2.1 and Table 2.2.

Article 1 contain the main scope and all the definitions used in the classification of different medical devices. The main scope is in paragraph 1 and is reported here.

> This Directive shall apply to medical devices and their accessories. For the purposes of this Directive, accessories shall be treated as medical devices in their own right. Both medical devices and accessories shall hereinafter be termed *devices*.

Among the definitions in article 1, the most interesting is the actual definition of a medical device:

> **Medical device** means any instrument, apparatus, appliance, material or other article, whether used alone or in combination, including the software necessary for its proper application intended by the manufacturer to be used for human beings for the purpose of:
>
> - diagnosis, prevention, monitoring, treatment or alleviation of disease;
> - diagnosis, monitoring, treatment, alleviation of or compensation for an injury or handicap;
> - investigation, replacement or modification of the anatomy or of a physiological process;
> - control of conception;
>
> and which does not achieve its principal intended action in or on the human body by pharmacological, immunological or metabolic means, but which may be assisted in its function by such means.

Article 2 deals with the scope of the directive. Although in the first article there is already a paragraph dedicated to the scope, article two is used to remove all manner of doubt on the applicability. A meaningful quote is the following.

> 3. **This Directive *does not* apply to:**
>
> a) in vitro diagnostic devices;
>
> b) active implantable devices covered by Directive 90/385/EEC;
>
> c) medicinal products covered by Directive 65/65/EEC;

Table 2.1: Articles of EEC medical devices directive 93/42

Article	Content
1	Definitions, scope
2	Field of application
3	Placing on the market and putting into service
4	Essential requirements
5	Free movement, devices intended for special purposes
6	Reference to standards
7	Safeguard clause
8	Classification
9	Information on incidents occuring following placing of devices on the market
10	-Repealed-
11	Conformity assessment procedures
12	Particular procedure for systems and procedure packs
13	Registration of persons responsible for placing devices on the market
14	Clinical investigation
15	Notified bodies
16	CE marking
17	Market surveillance and compliance testing
18	Decision in respect of refusal or restriction
19	Confidentiality
20	Trade of devices
21	Advertising
22	Electrical equipment used in medicine
23	Sanctions
24	Transitional and final provisions
25	Reference norm

Table 2.2: Annexes of EEC medical devices directive 93/42

Annex	Content
I	General requirements
II	Declaration of conformity
III	CE certification
IV	CE verification
V	CE declaration of conformity (production quality)
VI	CE declaration of conformity (product quality)
VII	CE declaration of conformity
VIII	Statement concerning special purposes devices
IX	Classification criteria
X	Clinical evaluation
XI	Criteria for the designation of notified bodies
XII	Method and content of applications for requesting authorization to certification
XIII	CE marking

d) cosmetic products covered by Directive 76/768/EEC;

e) human blood, human blood products, human plasma or blood cells of human origin or to devices which incorporate at the time of placing on the market such blood products, plasma or cells, with the exception of devices referred to in paragraph 2-bis;

f) transplants or tissues or cells of human origin nor to products incorporating or derived from tissues or cells of human origin, with the exception of devices referred to in paragraph 2-bis;

g) transplants or tissues or cells of animal origin, unless a device is manufactured utilizing animal tissue which is rendered non-viable or non-viable products derived from animal tissue.

Article 3 asserts that the devices can be placed on the market -or put into service- **only** if they meet all the requirements of this directive when properly installed, maintained and used in accordance with their intended purpose.
In other words, a device not complying with this directive is outlaw.

Article 4 states that all devices must meet the Essential Requirements set out in Annex I which apply to them, taking account of the intended purpose of the devices concerned. For this scope, Annex I contain a list of descriptive ERs -as stated before. An example is the following extract.

1. The devices must be designed and manufactured in such a way that, when used under the conditions and for the purposes intended, they will not compromise the clinical condition or the safety of patients, or the safety and health of users or, where applicable, other persons, provided that any risks which may be associated with their use constitute acceptable risks when weighed against the benefits to the patient and are compatible with a high level of protection of health and safety.
   This involves:
   - reducing, as far as possible, the risk of misuse due to the ergonomic features of device and environment in which it is provided that the device is used (design for patient safety),
   - the consideration of the technical knowledge, experience, education and training and, as appropriate, the medical and physical conditions of users where the device is intended (designed for common, professional, disabled or other users).

2. The solutions adopted by the manufacturer for the design and construction of the devices must conform to safety principles, taking account of the generally acknowledged state of the art.
   In selecting the most appropriate solutions, the manufacturer must apply the following principles in the following order:
   - eliminate or reduce risks as far as possible (inherently safe design and construction),
   - where appropriate take adequate protection measures including alarms if necessary, in relation to risks that cannot be eliminated,
   - inform users of the residual risks due to any shortcomings of the protection measures adopted.

Article 8 contain a reference to Annex IX regarding the classification of devices. There are four distinct classes in which devices are divided.

- **Class I** is the less critical. For example this class contains -but they are not limited to- all non-invasive devices that does not enter in contact with body fluids or wounded skin.
- **Class IIa** is an intermediate step and contain the less critical invasive devices, such as the ones intended for temporary use.
- **Class IIb** represent the first high-risk class. I.e. it includes temporary invasive devices intended to release ionizing radiations.
- **Class III** is the highest-risk class. It contains the most dangerous and critical devices such as invasive devices intended to stay in close contact with the central nervous system.

Annex IX contain all the rules necessary to locate a device in his specific class. Furthermore, it contains all the definitions omitted in the first article, i.e. the definition of *invasive device*.

> **Invasive device -** Device that penetrates partially or entirely into the body through a body orifice or body surface.

Article 11 contain all the necessary procedures for conformity assessment.

- **Class I:** Given the nature of this class, the procedures of conformity assessment are facilitated, and may be carried out under the sole responsibility of the manufacturer (self-certification).
- **Class IIa:** The procedure of conformity assessment, in this case, contemplates that a notified body carries out certain controls during the manufacturing stage.
- **Class IIb:** In this case, the notified body is necessary in both the design phase and the production phase.
- **Class III:** Conformity assessment is the same of Class IIb devices, but the commercialization requires an explicit authorization of prelimi-

nary compliance.

In Article 11 there are also direct references to various annexes, among which Annex XI, that defines role and designation of *notified body*. For example:

> 1. The *notified body*, its Director and the assessment and verification staff shall not be the designer, manufacturer, supplier, installer or user of the devices which they inspect, nor the authorized representative of any of these persons. They may not be directly involved in the design, construction, marketing or maintenance of the devices, nor represent the parties engaged in these activities. This in no way precludes the possibility of exchanges of technical information between the manufacturer and the body.
>
> [...]
>
> 4. The *notified body* **must** have:
>
> - sound vocational training covering all the assessment and verification operations for which the body has been designated,
> - satisfactory knowledge of the rules on the inspections which they carry out and adequate experience of such inspections,
> - the ability required to draw up the certificates, records and reports to demonstrate that the inspections have been carried out.

Article 15 harmonizes regulation of notified bodies and modality of certification. Treated in Annexes XI and XII respectively.

Article 16 states that the devices -excluding those of custom-made or intended for clinical investigations- considered to meet all the essential requirements referred to in Article 3 must bear, upon placing on the market, CE conformity marking. Article 16 also refer to annex XII, regarding shape and positioning of CE marking.

Article 23 provides an in-depth description of all the possible sanctions to those whom not follow this directive. These can be monetary -up to hundreds of thousands euro- and/or imprisonment.

2.2 Conformity standards

As stated in the last section, albeit EEC medical devices directive 93/42 define the Essential Requirements, it does not provide any mean to pursuit

them. In fact, those requirements should be interpreted and applied in order to take account of existing technology and practice during the design phase. The easiest way to do it is by following specific conformity standards.

The conformity standards are unofficial documents that contain all the design procedures and conformity tests that ensure compliance with MDD 93/42. Albeit conformity standards are unofficial, and thereafter their use is not mandatory, they are acknowledged by the EU to guarantee compliance with MDD 93/42. Therefore, it is highly recommended using them since it can save considerable loss of time.

2.2.1 European standards bodies

European standards bodies are organizations officially recognized by the EU to release and harmonize European standards. These bodies are the European equivalent of International Organization for Standardization (ISO), and they usually work in conjunction with them.

The main European standards bodies are: the European Committee for Standardization (CEN), the European Committee for Electrotechnical Standardization (CENELEC), and European Telecommunications Standards Institute (ETSI).

The standardization system in Europe plays an important role in the development and consolidation of the European Single Market. The fact that each European Standard is recognized across the whole of Europe, and automatically becomes the national standard in European countries, makes it much easier for businesses to sell their goods or services to customers throughout the European Single Market.

European Committee for Standardization (CEN)

The European Committee for Standardization (CEN from french Comité Européen de Normalisation) is a public standards organization whose mission is to foster the economy of European Union in global trading, the welfare of European citizens, and the environment by providing an efficient infrastructure to interested parties for the development, maintenance and distribution of coherent sets of standards and specifications. The CEN was founded in 1961, and its thirty-four national members work together to develop European Standards (ENs) in various sectors to build a European internal market for goods and services, and to position Europe in the global economy.

The standardization system is based on the national pillars, which are the National Standardization Bodies or the members of CEN. A National Standardization Body is the one stop shop for all stakeholders, and is the

main focal point of access to the concerted system, which comprises regional (European) and international (ISO) standardization. It is the responsibility of the CEN National Members to implement ENs as national standards. The National Standardization Bodies distribute and sell the implemented European Standard, and have to withdraw any conflicting national standards.

European Telecommunications Standards Institute (ETSI)

European Telecommunications Standards Institute (ETSI) is a non-profit, independent standardization organization founded in 1988 in France. ETSI is officially responsible for standardization of Information and Communication Technologies (ICT) within Europe, with special regards in the telecommunication field, including fixed, mobile, radio, converged, broadcast, and Internet technologies.

ETSI have over 800 members drawn from 67 countries across five continents including manufacturers, network operators, service and content providers, national administrations, universities and research bodies, user organizations, and consultancy companies and partnerships.

European Committee for Electrotechnical Standardization (CENELEC)

If CEN is the European specific equivalent of ISO, CENELEC is the same for International Electrotechnical Commission (IEC).

IEC is a non-profit, non-governmental organization founded in 1906 with the goal of developing and publishing international standards for all electrical, electronic, and related technologies. The IEC counts on 84 national members (of which 61 are full member) from all over the world, among which: Argentina, Belgium, Canada, China, Egypt, France, Germany, Greece, Italy, Japan, Mexico, Morocco, Pakistan, Qatar, Russian Federation, South Africa, Switzerland, United Kingdom, and United States of America. IEC standards have numbers in the range 60000–79999 -indicated e.g. as IEC 60601- and are harmonized as European standards by the CENELEC and indicated with the EN prefix -e.g. IEC 60601 is harmonized in EN 60601.

The European Committee for Electrotechnical Standardization (CENELEC from french Comité Européen de Normalisation Électrotechnique) is an international non-profit organization. CENELEC develop and harmonizes standards for electronic and related technologies in cooperation with ISO and IEC, with the difference with the latter that CENELEC standards are intended specifically for -but not limited to- the European Union.

As a last observation, it should be noted that although CENELEC, ETSI and CEN are all organizations recognized and in close cooperation with the European Union, they *are not* EU institutions.

2.2.2 Biomedical standards

The overwhelming quantity of standards available makes sound the first impact of standardization as a confusing job. In fact, the necessity to standardize every single aspect regarding devices, plants, components, procedures, codes, etc; requires a huge amount of different standards, each of them recognized by a prefix and a 5-digit number (as mentioned before). In this notation the number represent the actual standard code while the prefix indicate whether the standard is international (ISO) or European specific (EN). Even though the codes may seems to indicate same aspects in ISOs and in ENs, they usually don't have any direct correspondence. To further complicate the matter, in some case the prefix of a standard can instead indicate the organization which is the direct responsible for its publication -i.g. IEC or CEN.

In these condition, adding the absence of specific biomedical sections where to look for, the whole procedure may seem disheartening. Contrariwise, it is usually only required to find a single applicable standard to find all the others. Related standards often have similar codes and names, furthermore all the additional related and/or required standards are *always* indicated on the document itself. This simplifies enormously the research job to finding a single related standard, and given the power of research tool we have available nowadays, this is objectively a simple task.

Once found and *bought* all the required standards, it starts the actual regulatory analysis. This involves carefully reading all the clauses from each standard to extract the applicable rules and tests.

A special note must be taken on the price. In fact, standards are copyright protected and -with the exception of a small number of isolated cases- sold for a specific fee. What is worse, is that when the standard is eventually updated, the new version shall be bought again. Some organizations offer convenient limited access to a library of standards for a yearly subscription fee, usually large enough that only big institutions such as universities or multinational can afford to pay without problems. In most of cases anyway, the price alone can discourage hackers, open source enthusiasts, and even small business -such as start-up or third world companies- from developing new devices.

2.3 AED regulations

We saw in the last section what is the purpose of regulatory analysis, and why is so important. We also introduced the generic first steps to take in order to start, but since the real analysis is case specific, in this section we are going to see a complete regulatory analysis for Automated External Defibrillators -after all this thesis is about the design of a compliant AED.

2.3.1 Identification of the device

Before we get started with the norm analysis, it is very important to find the correct nomenclature and description for the device. This is not our case, but some devices can differ from others by very subtle differences. In any case, is a good practice having a solid idea on the device in order to avoid confusion. Therefore, we start with a formal definition of our device -this will also ease the future steps.

> An **Automated External Defibrillator (AED)** is a lightweight, portable device that *delivers an electric shock* through the chest to the heart in order to restore normal rhythm and contractile function in patients who are experiencing a **Sudden Cardiac Arrest (SCA)** [5].

For an AED the GMDN codes (see section 2.1.1) are in the following table. There are two different codes depending whether the battery is rechargeable or not [5].

Table 2.3: GMDN codes for AEDs

GMDN code	Description
48049	Non-rechargeable professional semi-automated external defibrillator
48048	Rechargeable professional automated external defibrillator

Now that the device have a definition and its unique codes, the only thing still needed to finish the identification, is defining the device class as described in Article 8 of EEC medical devices directive 93/42 (see section 2.1.2). For this purpose the Annex IX (of MDD 93/42) help us with a series of definitions and rules. The most relevant for our purpose are here reported [3].

> I. Definitions
>
> [...]
>
> **Active medical device -** Any medical device operation of which depends on a source of electrical energy or any source of power other than that directly generated by the human body or gravity and which acts by converting this energy.
> Medical devices intended to transmit energy, substances or other elements between an active medical device and the patient, without any significant change, are not considered to be active medical devices.
>
> **Active therapeutical device -** Any active medical device, whether used alone or in combination with other medical devices, to support, modify, replace or restore biological functions

or structures with a view to treatment or alleviation of an illness, injury or handicap.

[...]

III. Classifications

[...]

Rule 9 All active therapeutic devices intended to administer or exchange energy are in Class IIa unless their characteristics are such that they may administer or exchange energy to or from the human body in a potentially hazardous way, taking account of the nature, the density and site of application of the energy, in which case they are in Class IIb.
All active devices intended to control or monitor the performance of active therapeutic devices in Class IIb, or intended directly to influence the performance of such devices are in Class IIb.

Thereafter, we can conclude that an Automated External Defibrillator is a class IIb device. As mentioned before (see Article 11 of MDD 93/42 in section 2.1.2) class IIb devices needs a *notified body* in both design and production phase.

In the next section we are going to identify and analyse the required norms in order to extract the ERs.

2.3.2 EN 60601

AEDs are electronic device, therefore a good point to start the norm's analysis is the EN 60601 [6]. EN 60601 is a series of standards regarding safety and effectiveness of medical electrical (ME) equipment.

The 60601 series was first published by the IEC in 1977, and is regularly updated and restructured in order to follow the evolution of technologies and techniques. The European Standards derived from IEC 60601 have been prepared under a mandate given to CENELEC by the European Commission and the European Free Trade Association, and covers essential requirements of EC Directives 90/385/EEC and 93/42/EEC. Currently we are at the third edition of the norm (released on 2005).

Brief history of EN 60601 series

In 1976, IEC published the first edition of IEC/TR 60513 -Basic aspects of the safety philosophy for electrical equipment used in medical practice. The first edition of IEC/TR 60513 provided the basis for developing:

- **the first edition of IEC 60601-1** (the parent safety standard for medical electrical equipment);
- **the IEC 60601-1-xx** series of collateral standards for ME equipment;
- **the IEC 60601-2-xx** series of particular standards for particular types of ME equipment; and
- **the IEC 60601-3-xx** series of performance standards for particular types of ME equipment.

Aware of the need and the urgency for a standard covering electrical equipment used in medical practice, the majority of National Committees voted in 1977 in favor of the first edition of IEC 60601-1, based on a draft that at the time represented a first approach to the problem. The extent of the scope, the complexity of the equipment concerned, and the specific nature of some of the protective measures and the corresponding tests for verifying them, required years of effort in order to prepare this first standard, which can now be said to have served as a universal reference since its publication.

However, the frequent application of the first edition revealed room for improvement. These improvements were all the more desirable in view of the considerable success that this standard has enjoyed since its publication. The careful work of revision subsequently undertaken and continued over a number of years, resulted in the publication of the second edition in 1988. This edition incorporated all the improvements that could be reasonably expected up to that time.

The original IEC approach was to prepare separate basic safety and performance standards for medical electrical equipment. This was a natural extension of the historical approach taken at the national and international level with other electrical equipment standards (e.g. those for domestic equipment), where basic safety is regulated through mandatory standards but other performance specifications are regulated by market pressure. In this context, it has been said that, "The ability of an electric kettle to boil water is not critical to its safe use!"

It is now recognized that this is not the situation with many items of ME equipment, and responsible organizations have to depend on standards to ensure Essential Performance (EP), as well as basic safety. Such areas include the accuracy with which the equipment controls the delivery of energy or therapeutic substances to the patient, or processes and displays physiological data that will affect patient management.

This recognition means that separating basic safety and performance is somewhat inappropriate in addressing the hazards that result from inadequate design of ME equipment. Many particular standards in the IEC 60601-2-xx series address a range of Essential Performance requirements that cannot be directly evaluated by the responsible organization without

applying such standards.

In anticipation of a third edition of IEC 60601-1, IEC prepared a second edition of IEC/TR 60513 in 1994. It was intended that the second edition of IEC/TR 60513 [7] would provide guidance for developing this edition of IEC 60601-1, and for the further development of the IEC 60601-1-xx and IEC 60601-2-xx series.

In order to achieve consistency in international standards, address present expectations in the health care community and align with developments in IEC 60601-2-xx, the second edition of IEC/TR 60513 includes two major new principles:

- the first change is that the concept of "safety" has been broadened from the basic safety considerations in the first and second editions of IEC 60601-1 to include EP matters, (e.g. the accuracy of physiological monitoring equipment). Application of this principle leads to the change of the title of this publication from "Medical electrical equipment, Part 1: General requirements for safety" in the second edition, to "Medical electrical equipment, Part 1: General requirements for basic safety and Essential Performance";
- the second change is that, in specifying minimum safety requirements, provision is made for assessing the adequacy of the design process when this is the only practical method of assessing the safety of certain technologies such as programmable electronic systems. Application of this principle is one of the factors leading to introduction of a general requirement to carry out a risk management process. In parallel with the development of the third edition of IEC 60601-1, a joint project with ISO/TC 210 resulted in the publication of a general standard for risk management of medical devices. Compliance with this edition of IEC 60601-1 requires that the manufacturer have a risk management process complying with ISO 14971 [8] in place.

This standard contains requirements concerning basic safety and Essential Performance that are generally applicable to medical electrical equipment. For certain types of ME equipement, these requirements are either supplemented or modified by the special requirements of a collateral or particular standard. Where particular standards exist, the standard should not be used alone [6].

EN 60601 series organization

EN 60601 series has a hierarchical structure. The main document is the EN 60601-1, which is about the general requirements for basic safety and EP of ME equipments and systems. This standard contain all the common prescription necessary for any ME equipment.

The second layer is represented by the collateral norms, recognized by

the EN 60601-1-xx codes -where xx indicates the particular collateral norm. Collateral norms specifies general requirements for basic safety and EP applicable to:

- a subgroup of ME equipment (e.g. radiological equipment);
- a specific characteristic of all ME equipment not fully addressed in the standard.

The third layer are the particular standards, indicated with the codes EN 60601-2-xx -where xx indicates the particular norm. These are device-specific and may modify, replace or delete requirements contained in the standard as appropriate for the particular ME equipment under consideration, and may add other basic safety and Essential Performance requirements.

Table 2.4: List of EN 60601 collateral standards

60601-1-xx	**Scope**
1	Safety Requirements for Medical Electrical Systems
2	Electromagnetic Compatibility –Requirements and Tests
3	General Requirements for Radiation Protection in Diagnostic X-Ray Equipment
4	Programmable Electrical Medical Systems
6	Usability
7	General Requirements, Tests and Guidance for Alarm Systems in Medical Electrical Equipment and Medical Electrical Systems
8	Requirements for Environmentally Conscious Design
9	Requirements for the Development of Physiologic Closed-Loop Controllers
10	Requirements for Medical Electrical Equipment and Medical Electrical System Used in Home Care Applications

Table 2.5: List of EN 60601 particular standards

60601-2-xx	**Scope**
1	Particular Requirements for the Safety of Electron Accelerators in the Range 1 Mev to 50 Mev
2	Particular Requirements for the Safety of High Frequency Surgical Equipment
3	Particular Requirements for the Safety of Short-Wave Therapy Equipment
4	Particular Requirements for the Safety of Cardiac Defibrillators and Cardiac Defibrillators Monitors
5	Particular Requirements for the Safety of Ultrasonic Physiotherapy Equipment
6	Particular Requirements for the Safety of Microwave Therapy Equipment
7	Particular Requirements for the Safety of High-Voltage Generators of Diagnostic X-Ray Generators
8	Particular Requirements for the Safety of Therapeutic X-Ray Equipment Operating in the Range 10 Kv to 1 Mv

9	Particular Requirements for the Safety of Patient Contact Dosemeters Used in Radiotherapy With Electrically Connected Radiation Detectors
10	Particular Requirements for the Safety of Nerve and Muscle Stimulators
11	Particular Requirements for the Safety of Gamma Beam Therapy Equipment
12	Particular Requirements for the Safety of Lung Ventilators for Medical Use
13	Particular Requirements for the Safety of Anaesthetic Workstations
14	Particular Requirements for the Safety of Electroconvulsive Therapy Equipment
15	Particular Requirements for the Safety of Capacitor Discharge X-Ray Generators
16	Particular Requirements for the Safety of Haemodialysis Equipment
17	Particular Requirements for the Safety of Remote-Controlled Automatically Driven Gammaray After-Loading Equipment
18	Particular Requirements for the Safety of Endoscopic Equipment
19	Particular Requirements of Safety of Baby Incubators
20	Particular Requirements for the Safety of Transport Incubators
21	Particular Requirements for the Safety of Infant Radiant Warmers
22	Particular Requirements for the Safety of Diagnostic and Therapeutic Laser Equipment
23	Particular Requirements for the Safety, Including Essential Performance, of Transcutaneouspartial Pressure Monitoring Equipment
24	Paritcular Requirements for the Safety of Infusion Pumps and Controllers
25	Particular Requirements for the Safety of Electrocardiographs
26	Particular Requirements for the Safety of Electroencephalographs
27	Particular Requirements for the Safety of Electrocardiographic Monitoring Equipment
28	Particular Requirements for the Safety of X-Ray Source Assemblies and X-Ray Tube Assemblies for Medical Diagnosis
29	Particular Requirements for the Safety of Radiotherapy Simulators
30	Particular Requirements for the Safety, Including Essential Performance, of Automatic Cycling Non-Invasive Blood Pressure Monitoring Equipment
31	Particular Requirements for the Safety of External Cardiac Pacemakers With Internal Power Source
32	Particular Requirements for the Safety of Associated Equipment of X-Ray Equipment
33	Particular Requirements for the Safety of Magnetic Resonance Equipment for Medical Diagnosis
34	Particular Requirements for the Safety, Including Essential Performance, of Invasive Blood Pressure Monitoring Equipment
35	Particular Requirements for the Safety of Blankets, Pads and Mattresses, Intended for Heating in Medical Use
36	Particular Requirements for the Safety of Equipment for Extracorporeally Induced Lithotripsy
37	Particular Requirements for the Basic Safety and Essential Performance of Ultrasonic Medical Diagnostic and Monitoring Equipment
38	Particular Requirements for the Safety of Electrically Operated Hospital Beds
39	Particular Requirements for the Safety of Peritoneal Dialysis Equipment

40	Particular Requirements for the Safety of Eletromyographs and Evoked Response Equipment
41	Particular Requirements for the Safety of Surgical Luminaires and Luminaires for Diagnosis
43	Particular Requirements for the Safety of X-Ray Equipment for Interventional Procedures
44	Particular Requirements for the Safety of X-Ray Equipment for Computed Tomography
45	Particular Requirements for the Safety of Mammographic X-Ray Equipment and Mammographic Stereotactic Devices
46	Particular Requirements for the Safety of Operating Tables
47	Particular Requirements for the Safety, Including Essential Performance, of Ambulatory Electrocardiographic Systems
49	Particular Requirements for the Safety of Multifunction Patient Monitoring Equipment
50	Particular Requirements for the Safety of Infant Phototherapy Equipment
51	Particular Requirements for Safety, Including Essential Performance, of Recording and Analysing Single Channel and Multichannel Electrocardiographs
52	Particular Requirements for Basic Safety and Essential Performance of Medical Beds
53	Particular Requirements for the Safety and Essential Performance of a Standard Communications Protocol for Computer Assisted Electrocardiography
54	Particular Requirements for Basic Safety and Essential Performance of X-Ray Equipment for Radiography and Radioscopy
56	Particular Requirements for Basic Safety and Essential Performance of Screening Thermographs for Human Febrile Temperature Screening
57	Particular Requirements for the Safety and Essential Performance of Intense Light Sources Used on Humans and Animals for Medical and Cosmetic Purposes
58	Particular Requirements for Basic Safety and Essential Performance of Lens Removal and Vitrectomy Devices for Ophthalmic Surgery

In order to design an AED we need to apply the standards included in:

- **EN 60601-1 :** medical electrical equipment - General requirements for basic safety and Essential Performance
- **EN 60601-1-1 :** Safety requirements for medical electrical systems
- **EN 60601-1-2 :** Electromagnetic compatibility - Requirements and tests
- **EN 60601-1-6 :** Usability
- **EN 60601-1-8 :** General requirements, tests and guidance for alarm systems in medical electrical equipment and medical electrical systems
- **EN 60601-2-4 :** Particular requirements for basic safety and Essential Performance of cardiac defibrillators
- **EN ISO 14971 :** Application of risk management to medical devices

In the following section there will be an accurate analysis of the required clauses from the two most important norms -regarding AEDs. The EN 60601-1 and the EN 60601-2-4.

Annex A Before we start though, a special remark should be taken on Annexes A. When reading a EN/IEC standard the first annex -indicated as A on the main document and AA on the collateral/particular norm- is usually called "General guidance and rationale". This annex provides a concise rationale for the important requirements of the norm and is intended for those who are familiar with the subject of the standard but who have not participated in its development. Practically, it provides rationales for specific clauses and sub-clauses which can enormously ease the understanding of norms.

2.3.3 In-depth analysis of EN 60601-1

There are 17 clauses in EN 60601-1, and some of these are just marginally required -or in some cases not required at all- in regards of AEDs. In addition, some clauses are replaced with the particular standards. To ease and lighten the reading of this thesis, only the required parts are taken into account.

1 - Scope, object and related standards

In analogy with the MDD 93/42, in this clause there is a definition for the scope of this norm and a reference on how to correctly integrate it with particular standards.

> **1.1 Scope**
> This International Standard applies to the basic safety and Essential Performance of medical electrical equipment and medical electrical systems, hereafter referred to as ME equipment and medical electrical (ME) systems.
>
> [...]
>
> **1.2 Object**
> The object of this standard is to specify general requirements and to serve as the basis for particular standards. **1.3 Collateral standards**
>
> [...]
>
> If a collateral standard applies to ME equipment for which a particular standard exists, then the particular standard takes priority over the collateral standard.

2 - Normative references

This clause indicates the documents that are indispensable for the norm application. Already listed while presenting the EN 60601 series.

3 - Terminology and definitions

Clause three contain all the terminology and definitions commonly used in the norms. Some are here reported for clarity.

3.2 Accessible Part - part of electrical equipment other than an applied part that can be touched by means of the standard test finger.
3.8 Applied part - part of medical electrical equipment that in normal use necessarily comes into physical contact with the patient for medical electrical equipment or an medical electrical system to perform its function.
3.10 Basic Safety - freedom from unacceptable risk directly caused by physical hazards when medical electrical equipment is used under normal condition and single fault condition.
3.27 Essential Performance - performance necessary to achieve freedom from unacceptable risk.
3.55 Manufacturer - natural or legal person with responsibility for the design, manufacture, packaging, or labelling of medical electrical equipment, assembling an medical electrical system, or adapting medical electrical equipment or an medical electrical system, regardless of whether these operations are performed by that person or on that person's behalf by a third party.
3.63 medical electrical Equipment - electrical equipment having an applied part or transferring energy to or from the patient or detecting such energy transfer to or from the patient and which is:

a) provided with not more than one connection to a particular supply mains; and
b) intended by its manufacturer to be used:
    1) in the diagnosis, treatment, or monitoring of a patient; or
    2) for compensation or alleviation of disease, injury or disability.

3.70 Normal Condition - condition in which all means provided for protection against HAZARDS are intact.
3.116 Single Fault Condition - condition in which a single means for reducing a risk is defective or a single abnormal condition is present.
3.132 Type B Applied Part - applied part complying with the specified requirements of this standard to provide protection against electric shock, particularly regarding allowable patient leakage current and patient auxiliary current.
3.133 Type BF Applied Part - F-Type applied part complying with the specified requirements of this standard to provide a higher degree of protection against electric shock than that provided by type B applied parts.

> **3.134 Type CF Applied Part -** F-Type applied part complying with the specified requirements of this standard to provide a higher degree of protection against electric shock than that provided by type BF applied parts.

4 - General requirements

As the title indicates, this clause define the general requirement for ME equipment. Paragraph 2 requires that a risk management process complying with ISO 14971 shall be performed. In addition to that, paragraph 7 deepens the single fault condition safety -very important for an AED considered the magnitude of energies in play.

> **4.7 Single fault condition for ME equipment**
> ME equipment shall be so designed and manufactured that it remains single fault safe, or the risk remains acceptable as determined through application of 4.2[1]. ME equipment is considered single fault safe if:
>
> a) it employs a single means for reducing a RISK that has a negligible probability of failure, or
> b) a single fault condition occurs, but:
> – the initial fault will be detected during the expected service life of the ME equipment and before a second means for reducing a risk fails; or
> – the probability that the second means of reducing the risk will fail during the expected service life of the ME equipment is negligible.
>
> Where a single fault condition causes another single fault condition, the two failures are considered as one single fault condition.
>
> [...]
>
> Compliance is determined by applying the specific requirements and tests associated with the single fault condition identified in 13.2, and tests for the failures identified from evaluation of the results of the risk analysis. Compliance is confirmed if the introduction of any of the single fault condition described in 13.2, one at a time, does not lead directly to the hazardous situations described in 13.1, or any other outcome that results in an unacceptable risk.

Further safety prescription on single fault condition are provided in clause 13.

[1] 4.2 Risk management process for ME equipment or ME systems

5 - General requirements for testing ME equipment

In this clause there is a general description of test modalities and the necessary equipment (such as the Standard test finger).

6 - Classification of ME equipment and ME systems

Clause six define a ME equipment and applied parts classification based on the protection against electric shock.

> **6.2 Protection against electric shock**
> ME equipment energized from an external electrical power source shall be classified as class I ME equipment or class II ME equipment (see 7.2.6). Other ME equipment shall be classified as internally powered ME equipment.
>
> Internally powered ME equipment having a means of connection to a supply mains shall comply with the requirements for CLASS I ME equipment or CLASS II ME equipment while so connected, and with the requirements for internally ME equipment while not so connected.
> Applied parts shall be classified as type B applied parts, type BF applied parts or type CF applied parts.

In our case, the AED is a internally powered ME equipment.

7 - ME equipment identification, marking and documents

In conjunction with Annexes C and D, this clause defines the marking and documentations required for the ME equipment. This include the regulatory for alarm systems, and user interfaces.

> **7.2.14 High voltage terminal devices**
> High voltage terminal devices on the outside of ME equipment that are accessible without the use of a tool shall be marked with symbol IEC 60417-5036 (Figure 2.2).

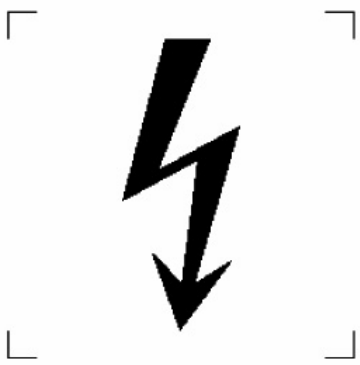


Figure 2.2: IEC 60417-5036 Symbol

[...]

7.8.1 Colours of indicator lights
The colours of indicator lights and their meanings shall comply with Table 2 (here reported as Table 2.6). Note: IEC 60601-1-8 contains specific requirement for the colour, flashing frequency and duty cycle of alarm indicator lights.

[...]

7.9 Accompanying documents 7.9.1 General ME equipment shall be accompanied by documents containing at least the instructions for use and a technical description. The accompanying documents shall be regarded as a part of the ME equipment.

Table 2.6: Colours of indicator lights and their meaning for ME equipment

Colour	**Meaning**
Red	Warning – immediate response by the operator is required
Yellow	Caution – prompt response by the operator is required
Green	Ready for use
Any other colour	Meaning other than that of red, yellow or green

Table 2.6, in reference to what said in paragraph 7.8.1, assert that some colours are strictly bound to specific meanings. This means that red, yellow, and green lights can only be used to communicate the user a specific message.

8 - Protection against electrical hazards from ME equipment

This is one of the most important clauses. It describes procedures, design guideline, limits, and compliance tests to ensure the safety of both patient and operator. This include the single fault condition, a situation when a single component of the device is malfunctioning. Safety is a priority, specially in this condition.

8.1 Fundamental rule of protection against electric shock
The limits specified in 8.4 shall not be exceeded for accessible parts and applied parts in normal conditions or single fault condition. For other hazardous situations in single fault condition, see 13.1.

a) normal condition includes all of the following simultaneously:
   - the presence on any signal input/output part of any voltage or current from other electrical equipment that

is permitted to be connected according to the accompanying documents as specified in 7.9, or, if the accompanying documents place no restrictions on such other electrical equipment, the presence of the maximum mains voltage as specified in 8.5.3;

- transposition of supply connections, for ME equipment intended for connection to a supply mains by means of a mains plug;
- short circuit of any or all insulation that does not comply with the requirements of 8.8;
- short circuit of any or all creepage distances or air clearance that do not comply with the requirements of 8.9;
- open circuit of any or all earth connections that do not comply with the requirements of 8.6, including any functional earth connection.

b) single fault condition includes:

- short circuit of any one insulation that complies with the requirements for one means of protection as specified in 8.8;
- short circuit of any one creepage or air clearance that complies with the requirements for one means of protection as specified in 8.9; – short circuit and open circuit of any component other than a component with high integrity characteristics that is connected in parallel with insulation, with an air clearance or with a creepage distance unless shorting can be shown not to be a failure mode for the component; – open circuit of any one protective earth conductor or internal protective earth connection that complies with the requirements of 8.6: this does not apply to a protective earth conductor of permanently installed ME equipment, which is considered unlikely to become disconnected; – interruption of any one supply conductor, except for the neutral conductor of polyphase ME equipment or permanently installed ME equipment; – interruption of any one power-carrying conductor between ME equipment parts in separate enclosures, if the risk analysis indicates that this condition might cause permitted limits to be exceeded; – unintended movement of a component; but only if the component is not mounted securely enough to ensure that such movement will be very unlikely to occur during the expected service life of the ME equipment, as determined by the risk management process; – accidental detachment of conductors and connectors where breaking free could lead to a

hazardous situation.

[...]

8.4 Limitation of voltage, current or energy
8.4.1 Patient connections intended to deliver current
The limits specified in 8.4.2 do not apply to currents that are intended to flow through the body of the patient to produce a physiological effect during normal use.
8.4.2 accessible parts including applied parts

a) The currents from, to or between patient connections shall not exceed the limits for patient leakage current and patient auxiliary current specified in Table 3 (here reported in Table 2.7) and Table 4 when measured as specified in 8.7.4. Compliance is checked by measurement according to 8.7.4.

A special note about this last paragraph. The defibrillation of the AED -similarly to defibrillators in general- deliver an energy intended to produce a physiological effect, therefore the current flowing *during* defibrillation is excluded from Table 2.7, unlike all the other current flowing when the AED is not performing a defibrillation.

8.4.4 Internal capacitive circuits
Conductive parts of capacitive circuits that become accessible after ME equipment has been de-energized and access covers as present in normal use have been removed immediately thereafter, shall not have a residual voltage exceeding 60 V, or, if this value is exceeded, shall not have a stored charge exceeding 45 μC.
If automatic discharging is not reasonably possible and access covers can be removed only with the aid of a tool, a device that is included and which permits manual discharging is acceptable. The capacitor(s) or the connected circuitry shall then be marked with symbol IEC 60417-5036 (Figure 2.2) and the non-automatic discharging device shall be specified in the technical description.

The latter assert that capacitive circuits -such as the capacitor used to store the defibrillation energy- that cannot be discharged in short time after the device is turned off, shall not be accessible. Practically, the cover of the device should be designed in order to be accessible only with the aid of a tool.

Finally, paragraph 7 of clause 8 contain the specification for ”leakage currents and patient auxiliary currents”. These includes the values of Table 2.7 and some general consideration on how to measure leakage currents

Table 2.7: Allowable values of patient leakage currents and patient auxiliary currents under normal condition and single fault condition

Current	Description
Patient Auxiliary Current	
Patient Leakage Current	From patient connection to earth
	Caused by an external voltage on a SIP/SOP
Total Patient Leakage Current	With the same type of applied part connected together
	Caused by an external voltage on a SIP/SOP
Key SFC = single fault condition NC = normal condition	

	Type B Applied Part		Type BF Applied Part		Type CF Applied Part	
	NC	SFC	NC	SFC	NC	SFC
d.c.	10	50	10	50	10	50
a.c.	100	500	100	500	10	50
d.c.	10	50	10	50	10	50
a.c.	100	500	100	500	10	50
d.c.	10	50	10	50	10	50
a.c.	100	500	100	500	10	50
d.c.	50	100	50	100	50	100
a.c.	500	1000	500	1000	50	100
d.c.	50	100	50	100	50	100
a.c.	500	1000	500	1000	50	100
Current in μA						

9, 10, 11

These clauses are similar to the last, except that they treat about other hazards which are beyond the scope of this thesis. For completeness the scope of these clauses are:

9 – Protection against mechanical hazards of ME equipment and ME systems

10 – Protection against unwanted and excessive radiation hazards

11 – Protection against excessive temperatures and other hazards

12 - Accuracy of controls and instruments and protection against hazardous outputs

The content of this clause describes some specification of the risk management process with reference to ISO 14971. It mainly define the necessity -when compiling the risk management file- to address the risk from usability, accuracy -of both controls and instrument-, alarm systems, and the protection against hazardous output.

13 - Hazardous situations and fault conditions

As mentioned before, this clause deals with the hazardous situations, with special regards to the single fault condition.

> **13.1.1 General**
> When applying the single fault conditions as described in 4.7 and listed in 13.2, one at a time, none of the hazardous situations in 13.1.2 to 13.1.4 (inclusive) shall occur in the ME equipment.

13.1.2 Emissions, deformation of enclosure or exceeding maximum temperature
The following hazardous situations shall not occur:

- emission of flames, molten metal, poisonous or ignitable substance in hazardous quantities;
- deformation of enclosures to such an extent that compliance with 15.3.1 is impaired;
- ...

[...]

13.1.3 Exceeding leakage current or voltage limits
The following hazardous situations shall not occur:

- exceeding the limits for leakage current in single fault condition as indicated in 8.7.3;
- exceeding the voltage limits for the accessible parts including applied parts indicated in 8.4.2.

Which basically means that the single fault conditions shall not endanger the users -patient and operator- and the maximum leakage current shall be less than the values enlisted in Table 2.7.

14 - Programmable electrical medical systems (PEMS)

In this clause there are description of the particular needing related to programmable electrical medical systems. This includes guidelines to:

- apply a risk management plan,
- define the development life-cycle,
- define a strategy for problem resolution,
- specify a safe architecture,
- design and implementation, and
- verification.

The clause is also supported by Annex H.
What follow are some significant extract.

14 programmable electrical medical systems (PEMS)
14.1 General
The requirements of this clause shall apply to PEMS unless:

- the PEMS provides no basic safety or Essential Performance; or
- the application of ISO 14971 demonstrates that the failure of the PEMS does not lead to an unacceptable risk.

14.4 PEMS development life-cycle
A PEMS life-cycle shall be documented.
NOTE 1 Clause H.2 explains PEMS development life-cycle in more detail.

[...]

H.2 PEMS development life-cycle model
Compliance with the PEMS clause of this standard (Clause 14) requires that a PEMS development life-cycle be specified and then followed; it does not require that any particular PEMS development life-cycle is used, but it does require that the PEMS development life-cycle has certain attributes. These requirements can be found in 14.4. The PEMS development life-cycle is a part of the overall product life-cycle. Figure H.2 *(here reported in Figure 2.3)* is a view of the PEMS development life-cycle which shows activities grouped into two main processes. On the left is decomposition process and on the right is the integration process.

[...]

14.5 Problem resolution Where appropriate, a documented system for problem resolution within and between all phases and activities of the PEMS development life-cycle shall be developed and maintained. Depending on the type of product, the problem resolution system may:

- be documented as a part of the PEMS development life-cycle;
- allow the reporting of potential or existing problems affecting basic safety or Essential Performance;
- include an assessment of each problem for associated risks;
- identify the criteria that must be met for the issue to be closed;
- identify the action to be taken to resolve each problem.

15 - Construction of ME equipment

In this clause there are the specific requirements for the physical construction of the device. These are expressed as directives on mechanical strength, arrangement of controls and indicators, environmental influences, temperature and overload control, batteries, transformers, actuation, movement, etc.

16 - ME systems

Clause 16 describes how to manage the risk due to the combination of multiple ME equipment to form a ME system. In the case of AEDs this risk is neglectable since an AED is usually used in critical condition where no other medical equipment is available. Furthermore, even if AEDs can be used in conjunction with other ME equipment -i.e. ECG- the main risk is usually related to the malfunctioning of the latter, caused by the large energy released during defibrillation.

16 ME systems
16.1 General requirements for the ME systems
After installation or subsequent modification, an ME systems shall not result in an unacceptable risk.
Only hazards arising from combining various equipment to constitute an ME systems shall be considered.

17 - Electromagnetic compatibility of ME equipment and ME systems

At last, clause 17 simply states that the electromagnetic phenomena related to the ME equipment shall be taken in account during the risk management process. Any other specification is left to the collateral norm EN 60601-1-2.

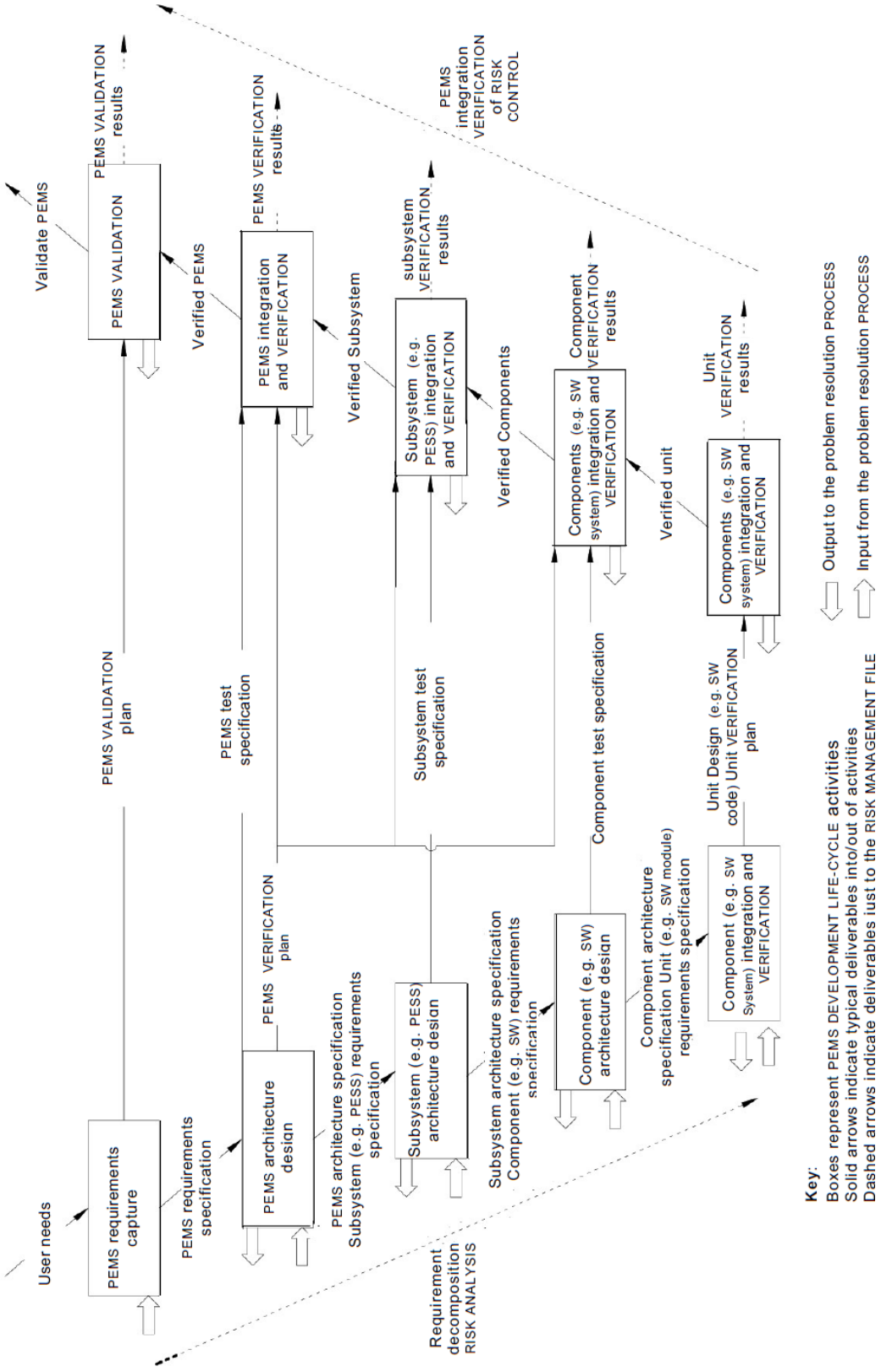


Figure 2.3: PEMS development life-cycle

2.3.4 In-depth analysis of EN 60601-2-4

Particular standards main goal is to *modify, replace or delete* clause from the main standard body, in order to adapt them for specific device needs. Therefore in particular standards, the clauses are indicated with a number that ease the identification of references. The first part -before the dot- is a three digit number, in which the first two digit are just a prefix, whilst the third show the specific norm that clause is referring to. The second part of the number -after the dot- simply indicates the corresponding clause.

For example 201.108 indicate the clause 108 contained in EN 60601-1. Which in this case it does not exist, therefore this denotes a new clause that has to be added at the original text.

201.1 - Scope, object and related standards

Clause 201.1 mainly replace scope and object of the norm in order to adapt it to the specific needs of cardiac defibrillators.

> **201.1.1 Scope**
> This International Standard applies to the basic safety and Essential Performance of cardiac defibrillators, hereafter referred to as ME equipment.
>
> [...]
>
> **201.1.2 Object**
> The object of this particular standard is to establish particular basic safety and Essential Performance requirements for cardiac defibrillators as defined in 201.3.202.

201.2 - Normative references

Similarly to clause 2, this adds nothing to what already said.

201.3 - Terminology and definitions

This just adds some new terms to the list. The most significant for this thesis are:

> **201.3.201 Automated External Defibrillator (AED) -** defibrillator that, once activated by the operator, analyses the ECG obtained from electrodes placed on the patient's skin identifies shockable cardiac rhythms, and automatically operates the defibrillator when a shockable rhythm is detected, hereinafter referred to as an AED.
> **201.3.202 Cardiac Defibrillator -** medical electrical equipment intended to normalize the rhythm of the heart by an electrical pulse via electrodes applied either to the patient's skin with

external electrodes or to the exposed heart with internal electrodes.
201.3.205 Delivered Energy - energy which is delivered through the defibrillator electrodes and dissipated in the patient or in a resistance of specified value.
201.3.210 Frequent Use - term used to describe a defibrillator designed to endure more than 2 500 discharges.
201.3.211 Infrequent Use - term used to describe a defibrillator designed to endure less than 2 500 discharges.
201.3.212 Internal Discharge Circuit - circuit within the defibrillator which discharges the energy storage device without energizing the defibrillator electrodes.
201.3.215 Rhythm Recognition Detector (RRD) - a system that analyzes the ECG and identifies whether a cardiac rhythm is shockable.

201.4 - General requirements

This clause assert that in this particular norm, there are the definition of new Essential Performance requirements. Described in sub-clauses 201.12.1, 201.104 and 201.107.

201.5 - General requirements for testing ME equipment

In this only few minor modifications and additions are made to adapt the general standard.

201.6 - Classification of ME equipment and ME systems

Clause 201.6 allege that defibrillators applied parts cannot be type B applied parts.

201.7 - ME equipment identification, marking and documents

Here the norm define new specification for documentation and alarm systems in order to satisfy the particular requirement of defibrillators and AEDs. Some of the most meaningful requirements are here reported.

201.7.2.101 Concise operating instructions
Instructions for defibrillating, and where relevant, monitoring a patient's ECG, shall be provided by means of either clearly legible markings, or clearly understandable auditory commands.
201.7.9.2.101 Supplementary instructions for use
The instructions for use shall additionally contain the following:

a) warning not to touch the patient during defibrillation;

b) a description of the correct type and method of handling the defibrillator electrodes in use as well as a prominent warning that defibrillator electrodes shall be kept well clear of other electrodes or metal parts in contact with the patient. The operator shall be advised that other ME equipment which has no defibrillation-proof applied parts shall be disconnected from the patient during defibrillation

c) caution for the operator to avoid contact between parts of the patient's body such as exposed skin of head or limbs, conductive fluids such as gel, blood or saline and metal objects such as a bed frame or a stretcher which may provide unwanted pathways for the defibrillating current;

[...]

h) for AEDs, information on the maximum time from the initiation of rhythm analysis with a clear ECG signal to readiness for discharge. The defibrillator will indicate if the ECG signal is not presently analyzable.

i) for AEDs, information on whether or not the defibrillator can automatically abort a charged and ready for shock condition as follows:
   - the rhythm recognition detector has detected a shockable rhythm and the defibrillator is charged and ready to shock;
   - the rhythm recognition detector has continued analyzing ECG after the initial shockable rhythm detection and has then detected a non-shockable rhythm;

[...]

201.7.9.3.101 Essential performance data for defibrillation
The technical description shall additionally provide:

a) graphical plots in terms of time and current or voltage of the waveforms of the delivered pulses when the defibrillator is connected in turn to resistive loads of 25 Ω, 50 Ω, 75 Ω, 100 Ω, 125 Ω 150 Ω and 175 Ω and set to its maximum output, or according to an automatic protocol for the selected energy if applicable;

b) energy accuracy specifications for the DELIVERED ENERGY in a 50 W resistor;

c) if the defibrillator has a mechanism to inhibit its output when the patient impedance is outside certain limits, disclosure of those limits.

201.8 - Protection against electrical hazards from ME equipment

Here are further specifications for means of protection against electrical hazards, adapted to the particular needs of defibrillators.
For instance, the following considerations about the leakage currents.

> **201.8.7 Leakage currents and patient auxiliary currents**
> **201.8.7.1 General requirements**
> For the measurement of patient leakage currents and patient auxiliary currents, the ME equipment shall be operated in turn:
>
> 1) in stand-by;
> 2) while the energy storage device is being charged to maximum energy;
> 3) while the energy storage device is maintained at maximum energy until internal energy discharge is automatically performed, or for 1 min;
> 4) for 1 min, starting 1 s after the commencement of the output pulse into a 50Ω load (the period of discharge being excluded).
>
> [...]
>
> **201.8.7.4.7 Measurement of the patient leakage currents**
>
> [...]
>
> For defibrillator electrodes the patient leakage currents is measured with the defibrillator electrodes connected to a 50 Ω load. The measurement is to be made from either defibrillator electrode to earth, the following parts being connected together and to earth:
>
> 1) conductive accessible parts;
> 2) metal foil on which the ME equipment is positioned and which has an area at least equal to that of the base of the ME equipment;
> 3) any signal input/output parts which may be connected to earth in normal use.

Another important passage is the one that defines creepage distance and air clearance for electrodes, circuits, and cables.

> **201.8.9.1 Values**
> **201.8.9.1.101 Defibrillator electrodes, high-voltage circuits and cables**

a) Between energized parts of defibrillator electrodes and parts of any associated handle and any switches or controls likely to be touched in normal use there shall be a creepage distance of at least 50 mm and an air clearance of at least 25 mm.

b) Except for components where the adequacy of ratings can be demonstrated the creepage distances and air clearances of insulation between the high-voltage circuit and other parts, and between different parts of the high-voltage circuit, shall be at least 3 mm/kV.

This requirement shall also apply to the isolating means between the high-voltage circuit of the defibrillator and other patient circuits.

201.9, 201.10, 201.11

In analogy with the general standards we not consider these clauses for the purpose of this thesis. In any case the first two made no modification at all on the original document, while the third did only less significant changes.

201.12 - Accuracy of controls and instruments and protection against hazardous outputs

This clause specify some important requirements about the energy to deliver. Starting with the accuracy:

201.12.1 Accuracy of controls and instruments
The rated delivered energy (according to ME equipment settings) into loads of 25 Ω, 50 Ω, 75 Ω, 100 Ω, 125 Ω, 150 Ω, and 175 Ω shall be specified. The measured delivered energy into these load resistances shall not vary from the delivered energy for that impedance by more than $\pm 3J$ or $\pm 15\%$, whichever is greater, at any energy level.

Continuing with the hard limit that shall not be exceeded:

201.12.4 Protection against hazardous output
201.12.4.1 Intentional exceeding of safety limits
The control for selected energy shall not allow:

a) the selected energy to exceed 360 J;

b) for internal defibrillator electrodes, the selected energy to exceed 50 J.

Check compliance by inspection and functional test.

201.12.4.101 Output voltage
The output voltage of the defibrillator across a 175 Ω load resistance shall not exceed 5 kV.

And finally, some consideration about the safety:

201.12.4.102 Unintentional energy
The ME equipment shall be so designed that in the event of a power failure (either of the supply mains or of the internal electrical power source) or when the ME equipment is switched off, no unintentional energy shall be available at the defibrillator electrodes.

201.12.4.103 Internal discharge circuit
A defibrillator shall be provided with an internal discharge circuit whereby stored energy that for some reason is not to be delivered through the defibrillator electrodes can be dissipated without energizing the defibrillator electrodes.

In particular, the last paragraph *explicitly require* the defibrillator to have an *internal discharge circuit* where to release the unnecessary energy. The application of this ERs is evident at first sight in Figure 2.4.

201.13, 201.14

No significant changes are made by these two.

201.15 - Construction of ME equipment

This clause add some clarification on batteries and defibrillator cables. Devices employing non-rechargeable batteries need to indicate when the battery require to be changed. In any case, the device must still ensure at least three discharges when the battery indicator requires a replacement.

201.15.4.3.101 Non-rechargeable battery replacement
Means shall be provided to indicate clearly when non-rechargeable batteries require replacement or rechargeable batteries require recharging. These means shall not make the ME equipment inoperative, and the ME equipment shall be capable of delivering three maximum energy discharges after that indication is initially provided. For ME equipment with a pre-programmed energy setting sequence, not changeable by the operator or responsible organization, the AED shall be able to deliver 3 defibrillation discharges at the pre-programmed settings after that indication is initially provided. In case of pre-programmed energy setting sequence being changeable by the operator or responsible organization the AED shall be able to deliver 3 defibrillation discharges at the maximum energy setting sequence selectable.

For rechargeable batteries instead, there are indicated new functional tests:

201.15.4.3.103 Rechargeable battery
Any rechargeable new battery shall enable the ME equipment to pass the following test:

a) Test requirements for manual defibrillator:
After fully charging the battery, the ME equipment is stored while switched off for 168 h (7 days) at a temperature of 20°C ± 5°C and at a relative humidity of 65% ± 10%. The ME equipment is then charged and discharged with the maximum delivered energy of the ME equipment, 14 times into a 50 Ω load at the rate of one charge-discharge per minute. The charging time for the 15th charge is not to exceed 15 s (25 s for infrequent use manual defibrillator).
If the defibrillator can perform a wake-up self-test that is automatically started with pre-selectable intervals when the defibrillator is powered off, the test is to be performed with the wake-up self-test enabled with the shortest possible interval.

b) Test requirements for Automated External Defibrillator:
After fully charging the battery, the ME equipment is stored while switched off for 168 h (7 days) at a temperature of 20°C ± 5°C and at a relative humidity of 65% ± 10%. The ME equipment is then charged and discharged, with the maximum delivered energy of the ME equipment, 14 times into a 50 Ω load at the rate of the pre-programmed defibrillation sequence. For discharge number 15, the time measured from application of the shockable cardiac rhythm to when the defibrillator is ready for discharge is not to exceed
   – 30 s for frequent use AEDs;
   – description 40 s for infrequent use AEDs.

   For ME equipment with a pre-programmed energy setting sequence, not changeable by the operator or responsible organization, the requirement for depletion of batteries with the delivery of maximum energy discharges is relaxed to the number of discharges at the pre-programmed energy setting sequence. In a case of the pre-programmed energy setting sequence being changeable by the operator or responsible organization, the requirement for depletion of batteries with the delivery of maximum energy discharges is relaxed to the number of discharges at the maximum energy setting sequence selectable.
   If the defibrillator can perform a wake-up self-test that is automatically started with preselectable intervals when the defibrillator is powered off, the test is to be performed with

the wake-up self-test enabled with the shortest possible interval.

201.16, 201.17

These two simply states that the clauses from the general standard still applies unchanged.

201.101 - Charging time

This clause define the charging time of the defibrillator in order to perform a defibrillation. These are different depending on the type of defibrillator -manual, automated, frequent-use, etc.

201.101.3 Requirements for frequent use, Automated External Defibrillator

a) The maximum time from activation of the rhythm recognition detector to the defibrillator being ready for discharge at maximum energy, shall not exceed 30 s under the following conditions:
   - description when the AED is operated on 90% of the rated mains voltage;
   - description with batteries depleted by the delivery of 15 discharges at maximum energy.

b) The time from initially switching power on, or from within any operator programming mode, to the defibrillator being ready for discharge at maximum energy shall not exceed 40 s. This requirement shall apply to charging a completely discharged energy storage device to maximum energy under the following conditions:
   - when the AED is operated on 90% of the rated mains voltage;
   - descriptionwith batteries depleted by the delivery of 15 discharges at maximum energy.

201.101.4 Requirements for infrequent use, Automated External Defibrillator

a) The following charge time requirements for infrequent use Automated External Defibrillator apply.
   - The maximum time from activation of the rhythm recognition detector to ready for discharge at maximum energy, shall not exceed 35 s when the AED is operated on 90% of the rated mains voltage.

  - The maximum time from activation of the rhythm recognition detector to ready for discharge at maximum energy, shall not exceed 35 s with batteries depleted by the delivery of 6 discharges at maximum energy.
  - The maximum time from activation of the rhythm recognition detector to ready for discharge at maximum energy, shall not exceed 40 s with batteries depleted by the delivery of 15 discharges at maximum energy.

b) For the time from initially switching power on, or from within any operator programming mode, to ready for discharge at maximum energy the following applies.
  - When the AED is operated on 90% of the rated mains voltage, the time from switching power on, or from within any operator programming mode, to ready for discharge at maximum energy shall not exceed 45 s.
  - With batteries depleted by the delivery of 6 discharges at maximum energy, the time from switching power on, or from within any operator programming mode, to ready for discharge at maximum energy shall not exceed 45 s.
  - With batteries depleted by the delivery of 15 discharges at maximum energy, the time from switching power on, or from within any operator programming mode, to ready for discharge at maximum energy shall not exceed 50 s.

These times can be resumed in Table 2.8.

Table 2.8: AED charge time

	Time required to be ready to defibrillate.		Total time required from power on to defibrillation	
	6 discharges	15 discharges	6 discharges	15 discharges
Frequent use AED		30s		40s
Infrequent use AED	35s	40s	45s	50s

Two different times are taken into account. The time that the defibrillator require to charge when already functioning; and the time that the defibrillator require from when is turned-on to be ready for defibrillation. There are two admissible values for each, depending on how many discharges the device has already delivered -6 or 15.

201.102 - Internal electrical power source

This clause is specifically written to address defibrillator's requirements for batteries. For AEDs the following applies:

201.102.3.1 Frequent use AED
For a frequent use AED, the capacity of a new and fully charged battery shall be such that at 0°C the ME equipment can provide at least 20 defibrillation discharges at the maximum delivered energy of the AED performed using the pre-programmed defibrillation sequence.

[...]

Where a frequent use AED contains the possibility to insert more than one battery at the same time which can be selected randomly by the OPERATOR, the requirement of 20 discharges is the total amount of discharges available when the defibrillator is equipped with the maximum amount of batteries.For any extra batteries that may be available for use in the defibrillator, but are not actually mounted in the defibrillator, the extra batteries shall not be included in the test.

[...]

201.102.3.2 Infrequent use AED
For an infrequent use AED, a new and fully charged battery shall be capable of providing at least 20 maximum energy discharges at the maximum delivered energy of the ME equipment performed using the pre-programmed defibrillation sequence.

Check compliance with 201.102.3.1 and 201.102.3.2 by a functional test at 0°C ± 2°C, the ME equipment having first been prepared as follows:

a) The battery is to be fully charged in accordance with manufacturer's instructions (or until the ME equipment indicates that the battery is fully charged) at an ambient temperature of 0°C ± 2°C, 20°C ± 2°C and 40°C ± 2°C, or according to environmental operating conditions as specified by the manufacturer according to 7.9.3.1 of the general standard, whichever are the most severe conditions.
b) The ME equipment including the battery is cooled to 0°C ± 2°C until it reaches thermal equilibrium.

Where ME equipment contains the possibility to insert more than one battery at the same time which can be selected randomly by the operator, the requirement of 20 discharges is the total amount of discharges available when the defibrillator is equipped with the maximum amount of batteries.
For any extra batteries that may be available for use in the defibrillator, but are not actually mounted in the defibrillator, the extra batteries shall not be included in the test.

That can be resumed in "the AED's fully-charged battery shall be capable of providing at least 20 maximum energy discharges even under the worst operating condition".

201.103 - Endurance

In this clause there is a brief description on a possible stress test, in order to verify how the device respond under intensive use.

201.104 - Synchronizer

Here are described the particular needing of synchronizers. A synchronizer is a device that allow the defibrillator discharge to be synchronized with a specific phase of the cardiac cycle. This is basically useless for an AED since it can only operate during SCA condition, where synchronization loses its meaning.

201.105 - Recovery of the monitor and/or ECG input after defibrillation

This clause indicates time and modalities of ECG circuitry recovering after a correct defibrillation.

201.105.1 ECG signal derived via defibrillator electrodes

[...]

> In addition to the above requirement, the rhythm recognition detector, if present, shall be able to detect a shockable rhythm 20 s after the defibrillation pulse. In this case, the signal applied to the defibrillator electrodes shall be a signal recognizable by the defibrillation as shockable.

201.106 - Disturbance to the monitor from charging or internal discharging

This clause only apply on defibrillator equipped with a monitor, which are beyond the scope of this thesis.

201.107 - Requirements for rhythm recognition detector

In this clause there are the requirement for rhythm recognition. According to the norm, two different ECG database are required: one used for development and the other to establish compliance. It also describes the various rhythm that the defibrillator shall discern from VT and VF.

> The ECG database for validation of rhythm recognition performance shall include, at a minimum, Ventricular Fibrillation (VF) rhythms of varying amplitudes, Ventricular Tachycardia (VT) rhythms of varying rates and QRS width, various sinus rhythms

> including supraventricular tachycardias, atrial fibrillation and atrial flutter, sinus rhythm with PVC (premature ventricular contraction), asystole and pacemaker rhythms. All rhythms shall have been collected using electrode systems and ECG signal processing characteristics similar to the device being tested, and shall be of appropriate length to allow decisions to be made by the detector system.

And finally, it unequivocally define the sensitivity and specificity.

$$Sensitivity = \frac{A}{(A+C)}$$

$$Specificity = \frac{B}{(B+D)}$$

Where A and B correspond to a correct classification of a shockable rhythm and a correct classification of a non-shockable rhythm respectively. While C and D represent to false classification of shockable and non-shockable rhythm respectively. As resumed in Table 2.9 for simplicity.

Table 2.9: Rhythm recognition detector categories

	VF and VT	**All other ECG rhythms**
Shock	A	B
No shock	C	D

According to the norm, sensitivity of the device -in absence of artefacts- shall exceed 90% while recognizing VF and 75% while recognizing VT. Whilst the specificity shall exceed 95%.

201.108 - Defibrillator electrodes

This clause specifies the requirement for electrodes, which are beyond the purpose of this thesis.

201.109 - External Pacing

External pacing may be provided as an optional feature. And AEDs usually do not.

202 - Electromagnetic compatibility - Requirements and tests

Finally, this clause refine EN 60601-1-2 regarding the electromagnetic compatibility. Since the main purpose of an AED is to deliver an electric shock is perfectly reasonable to consider the effect of electrostatic discharges (ESDs).

202.6.2.2.1 Requirements
For open air discharges of $\pm 2kV$ and $\pm 4kV$ and direct contact discharges of $\pm 2kV$, the operator shall not notice any change in ME equipment operation. The ME equipment shall operate within normal limits of its specifications. No degradation of system performance or loss of functionality is allowed. However, ECG spikes, pacemaker pulse detection, display glitches, or momentary light-emitting diode (LED) flashes are acceptable during an ESD discharge.
For open air discharges of $\pm 2kV$, $\pm 4kV$ and $\pm 8kV$ or direct contact discharges of $\pm 2kV$, $\pm 4kV$ and $\pm 6kV$, the ME equipment may exhibit momentary loss of functionality but shall recover within 2 s without operator intervention. There shall be no unintended energy delivery, unsafe failure mode, or loss of stored data.

Which basically means that the AED shall recover correct operation within two seconds after an ESD. Furthermore safety shall be ensured even during this period of time.

The remaining paragraph are about radio-frequency fields compatibility, which are beyond the scope of this thesis.

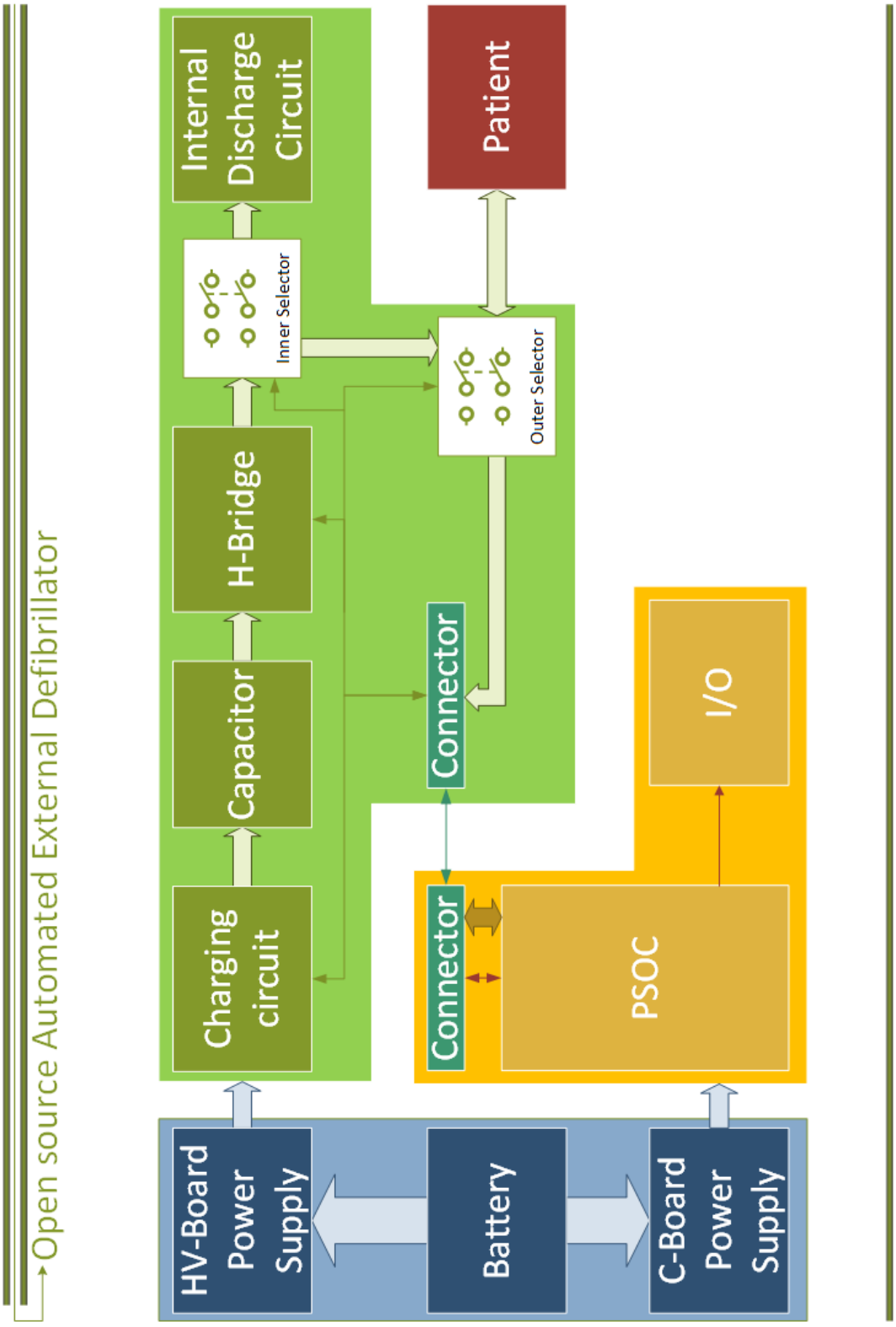


Figure 2.4: OAED block diagram

2.4 Open source Automated External Defibrillator

To summarise, in this chapter we saw the main step of a typical regulatory analysis. We started by identifying and classifying the device, and then we saw in detail the principal norm involved in the design phase. The rules resulting from that represents the Essential Requirements, the foundation from which the actual AED design process will start.

Figure 2.4 show a conceptual block diagram for the proposed solution. From that we can see that there are two main separate boards:

- the **"High-Voltage Board (HV-B)"** - which contain all the necessary circuitry to perform the defibrillation -distinguished by a green background; and
- the **"Control Board (C-B)"** - which contain the micro-controller and act as the "mind" of the device -distinguished by a orange background.

We are going to see more in detail the content of the various blocks in the next two chapters. However, in order to have a better understanding of the matter during the reading, it is helpful to give a preliminary explanation of the blocks, and the motivations beyond this particular conceptual design form.

2.4.1 High-Voltage board

As mentioned in the introduction of this section, the High-Voltage Board (HV-B) contain all the necessary circuitry to perform the defibrillation. This includes:

- a charging circuit,
- a capacitor,
- an H-Bridge circuit,
- an inner selector,
- an internal discharge circuit,
- an outer selector, and
- a connector.

Charging circuit

This circuit charges the capacitor when the AED detect that a defibrillation is needed. As stated in the regulatory analysis, the capacitor's charging speed is a critical aspect and should be maximized when possible, without affecting the overall efficiency and safety. We do not want a device that charges the capacitor in less than a second consuming twice the necessary, that would mean the device could only deliver half of the defibrillation that

is capable of, and power consumption is another critical aspect highlighted during the reading of section 2.3.4.

As charging circuit i have chosen a Ringing Choke Converter (RCC), also known as self-oscillating flyback. A state of the art circuit largely used in this field of application.

Capacitor

The capacitor represent the *storage device* mentioned in EN 60601-2-4 (see section 2.3.4). In fact, a capacitor is the simplest energy storage device known. In this case though, we need a special capacitor capable of resisting high voltages.

H-Bridge

By definition, an H-Bridge is an electronic circuit that enables a voltage to be applied across a load in either direction [9]. In our case, the **load** is represented by the patient, and the voltage to apply in either direction is the defibrillation electric shock. Numerous studies highlighted an higher success rate if the shock is briefly interrupted and re-applied with the opposite direction. This technique is called *biphasic* defibrillation (see Figure 3.1.2).

Inner and outer selector

The inner and outer selector are two double pole, double throw (DPDT) relay, which represent two means of protection for the users (both patient and operator).

The first has the common poles connected to the H-bridge, and the other two connections to the internal discharge circuit and the outer selector. The inner selector is normally closed on the internal discharge circuit, this ensures that failure, or accidental activation of the H-Bridge does not end in an hazardous situation.

The outer selector instead, has the AED's electrodes connected at the common poles, while the other two are connected to the inner selector and the micro-controller. Outer selector is normally closed on the latter, and is needed for two main reason:

1) it provide an insulation for the micro-controller, and
2) it provide another mean of protection for the patient.

In fact, inner and outer selectors are activated in order allow the connection between patient and capacitor, only in the immediate moments before and after the defibrillation.

It should be noted that they can also be seen as a mean of protection against the single fault condition, since they basically insulate the patient from all HV-B circuits connected upstream.

Internal discharge circuit

The internal discharge circuit only consist of a single power resistor. This is needed in order to dissipate the unused energy of the defibrillator -as described by EN 60601-2-4 (see section 2.3.4).

Connector

The connector block is just an header used to connect the High-Voltage Board and the Control Board.

2.4.2 Control board

If the High-Voltage Board can be seen as the ”muscles” of the device, the Control Board is the ”brain”. In fact it contains a Programmable System on Chip (the PSoC 5LP). PSoC 5LP is a device that contains a Micro Controller Unit (MCU), Analog Block Array (ABA), and Digital Block Array (DBA). All embedded in a single chip. Their usage and architecture will be widely illustrated on chapter 4. For now we can limit to say that by combining analog and digital blocks, it is possible to obtain various circuits embedded in the PSoC.

Other than the PSoC, there is the same connector found on the HV-B, and some I/O devices, such as three led diodes used to communicate the status of the device.

The purpose of having two separate boards is that the if they were on the same board, the high-precision electronics inside the PSoC would likely be affected by the high-frequencies, high-magnitude currents flowing in the charging circuit. Thereafter, two separate boards ensure the interactions minimization between the two, hopefully with a negligible effect on the ECG signal.

Bibliography

[1] *Global Medical Device Nomenclature*. https://www.gmdnagency.org/.

[2] *Universal Medical Device Nomenclature System website*. https://www.ecri.org/components/UMDNS/Pages/default.aspx.

[3] *COUNCIL DIRECTIVE 93/42/EEC of 14 June 1993 concerning medical devices*. http://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:31993L0042&from=IT. 1993.

[4] *DIRECTIVE 2007/47/EC OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL of 5 September 2007 amending Council Directive 90/385/EEC on the approximation of the laws of the Member States relating to active implantable medical devices, Council Directive 93/42/EEC concerning medical devices and Directive 98/8/EC concerning the placing of biocidal products on the market*. http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=OJ:L:2007:247:0021:0055:en:PDF. 2007.

[5] World Healt Organization (WHO). *Defibrillator, External, Automated; Semiautomated*. http://www.who.int/medical_devices/innovation/defibrillator_automatic.pdf.

[6] European Committee for Electrotechnical Standardization. *Medical electrical equipment*. EN 60601-1. 2007.

[7] International Electrotechnical Commission. *Fundamental aspects of safety standards for medical electrical equipment*. IEC 60513. 1994.

[8] International Organization for Standardization. *Medical devices - Application of risk management to medical devices*. EN ISO 14971. 2013.

[9] *H-Bridge*. https://en.wikipedia.org/wiki/H_bridge.

Part II

The Hardware

CHAPTER 3

High-Voltage Board

In Chapter 2 we saw a brief analysis of biomedical device's regulations, with special regards on medical electrical equipments and Automated External Defibrillators (AED). This consented us to outline AED's specific Essential Requirements (ERs).

From those we laid the foundations for OAED, starting with a conceptual block design containing two separate boards (see Figure 3.1). In this Chapter we are going to see in detail the first of these: The High-Voltage Board (HV-B).

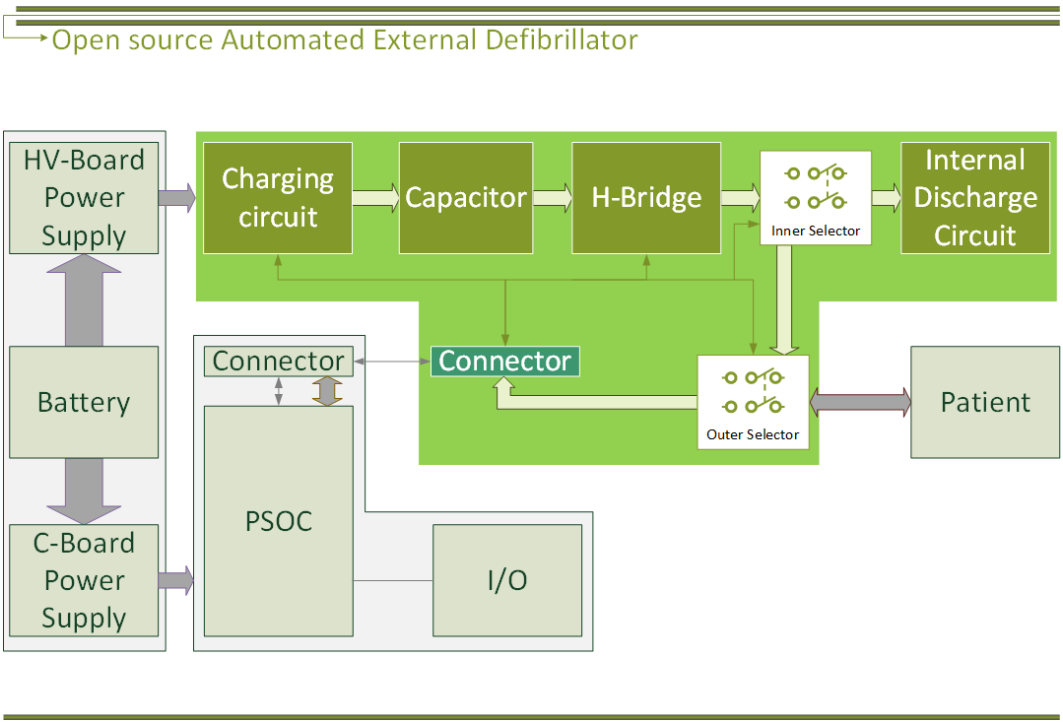


Figure 3.1: OAED High-Voltage Board block diagram

3.1 Overview

With reference to Figure 2.4, the High-Voltage Board is the one highlighted in Figure 3.1. As mentioned before, the HV-B contain all the circuitry

necessary to perform the defibrillation. These include:

- a charging circuit,
- a capacitor,
- an H-Bridge circuit,
- an inner selector,
- an internal discharge circuit,
- an outer selector, and
- a connector.

Since the purpose of circuits in this board is to perform defibrillation, we start by describing it more in detail.

3.1.1 Defibrillation

As already mentioned before, defibrillation consists of delivering a therapeutic dose of electrical current to the heart. This depolarizes a critical mass of the heart muscle, terminates the arrhythmia and allows normal sinus rhythm to be re-established by the heart's natural pacemaker, in the sino-atrial node.

To control the defibrillation, the doctor (or in our case the AED) can vary the energy released to the patient. An old misconception is that a shock is measured by its energy (in Joule [J]). This is due to the fact that the first pioneers of defibrillation thought that the amount of energy released to the body was the most efficient way to measure a defibrillation. While this is not entirely incorrect, we now know that the current flowing through the patient is more important then the energy delivered. In fact, during a defibrillation most of the energy is just dissipated as heat, or lost in the skin and other tissues; and only a small unmeasurable part of it make through the heart. In any case, since also the standards refer to the energy to deliver, rather than the peak current, we also will use energy as a mean of control[1].

From the norm, the maximum energy allowable for a defibrillation is $360J$. In order to consider possible errors or variations during the normal operation of the AED, it is preferable to save a reasonable margin of uncertainty, at least 15%. Therefore, the maximum energy we can consider safe is lowered to around $300J$. Most of the AED currently sold use a range of energy between $150J$ and $250J$. It should be noted that higher energies correspond to higher charging times (considering same voltages), and usually also to increased risk of skin burns.

From an engineering point of view, defibrillation can be simply represented with the discharge of a capacitor on an load -which can be approximated as pure resistive. In other words defibrillation is summarized with a RC circuit, where the patient is represented as a resistor[2], whilst a capaci-

[1] Also the norms refers to energies, rather than currents.

[2] In this model the patient resistor also includes the electrodes impedance, which basically consists in two more resistors. One placed before the patient resistor, and the other after it.

tor represent the defibrillator. At the initial time the capacitor is considered charged at a nominal tension V_0, therefore the discharge is characterized by an inverse exponential trend (see Equation 3.1).

$$v(t) = V_0 \cdot e^{-\frac{t}{Z \cdot C}} \tag{3.1}$$

The first thing to notice in Equation 3.1, is that the time required to perform a defibrillation -considering a fixed nominal voltage V_0- depend on the time constant $Z \cdot C$. Thereafter, *knowing the patient impedance is vital in order to evaluate the time required for a defibrillation*.

A note about the time is that the whole defibrillation should not last neither too much nor too little -compared to a normal heartbeat. If it is too fast we risk to injure the patient by releasing a huge amount of power $P = \frac{U}{t}$. On the other hand, if it is too slow it is likely that:

1. defibrillation might still be in progress when heart (or some region of the heart) would try to start re-polarizing, and/or
2. defibrillation might not be effective at all because the current peak would not be enough to depolarize a critical mass of the heart.

There is not a norm for discharge time, but is reasonable to aim at a range between $5ms$ and $50ms$. Depending on the patient-electrodes series impedance Z and the energy released U.

Choice of defibrillation parameters require the solution of a simple problem, nevertheless the parameters to be considered and adjusted are many, and this make the solution less trivial.

3.1.2 Capacitor

As we have already seen in the last section, defibrillation revolves around two main components: the patient-electrodes series impedance, which we cannot control; and the capacitor. The first component to chose then, is the capacitor, knowing that its value can depends on:

– nominal voltage V_0,
– maximum discharge time T, and
– maximum storable energy U_{max}.

Which are all parameters that we are also going to define in this section. But first we need the relations between C, U, T, and V_0.
Knowing that $i(t) = \frac{v(t)}{Z}$, we can define the energy U as:

$$\begin{aligned} U &= \int_0^T \left(i(t) \cdot v(t)\right) \partial t = \frac{V_0^2}{Z} \cdot \left[-\frac{Z \cdot C}{2} \cdot e^{\frac{-2t}{ZC}}\right]_0^T \\ &= \frac{V_0^2 \cdot C}{2} \cdot \left(1 - e^{\frac{-2T}{ZC}}\right) \end{aligned} \tag{3.2}$$

And from Equation 3.2 we can obtain the discharge time:

$$\begin{aligned} U = \frac{V_0^2 \cdot C}{2} \cdot \left(1 - e^{\frac{-2T}{ZC}}\right) \Rightarrow \frac{2U}{V_0^2 \cdot C} = 1 - e^{\frac{-2T}{ZC}} \Rightarrow \\ \Rightarrow T = -\frac{ZC}{2} \cdot ln\left[1 - \frac{2U}{V_0^2 \cdot C}\right] \end{aligned} \tag{3.3}$$

Patient-electrodes series impedance

Since all these equations strongly depends on patient-electrodes series impedance Z, we need a range of possible values. The norm help us by defining that the normal patient impedance Z_p is to be considered between 25Ω and 180Ω. Therefore we are going to use these values during all the course of this thesis.

The electrodes, on the other hand, are a more complicated matter. According to the 60601-2-4 *the impedance of an electrode pair connected gel-to-gel, in series with a* 50Ω *load and measured at the maximum rated energy of the defibrillator shall not exceed* 3Ω. Given their lower impact on total impedance Z, for simplicity sake I consider their values negligible in this context.

Besides the patient impedance, we also need the desired energy of defibrillation U, and the nominal operative tension V_0. Although there are clinical evidences of defibrillation effectiveness even at lower energies ($150J$) when using a biphasic waveform, I have decided to adapt to the other devices trend, and evaluate a range between $200J$ and $250J$ for U_{max}. For the nominal tension V_0 instead, I have considered a range between $1500V$ and $2000V$ -which are way below the maximum allowed by the norm. Using these we can also calculate the minimum required capacity C_{min} in Equation 3.4.

$$C_{min} = \frac{2U}{V_0^2} \tag{3.4}$$

The result of the combined calculation of Equation 3.3 and Equation 3.4 are summarized in Table 3.1. While in Figure 3.2 there are the trend of the discharge time with respect to the impedance, for different nominal tension V_0 and a fixed capacity of $200\mu F$.

Table 3.1: Maximum defibrillation time and minimum capacitor values in different configurations

				T $[ms]$	
		U[J]	**Cmin** [μF]	**Z =** 25Ω	**Z =** 180Ω
V_0[V]	**1500**	200	178	7.3	52.73
		250	222	9.2	65.9
	1700	200	134	5.7	41.0
		250	173	7.1	51.3
	2000	200	100	4.1	29.6
		250	125	5.1	37.0

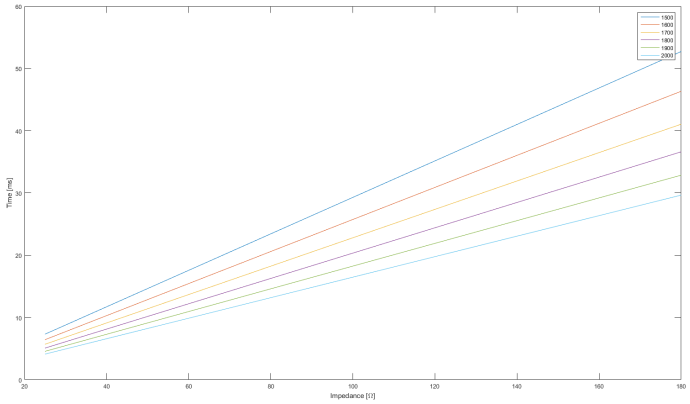


Figure 3.2: Discharge time for different nominal tensions

Discharge times in Table 3.1 are obtained considering a capacity 50% higher than the minimum required. As expected, higher voltages require smaller capacity, and therefore also less time to deliver the required charge. However higher voltages also means higher peak currents, in this case $2000V$ correspond to peak current as high as $110A$, which increase the risk of skin burn and other current related problems -especially for the electrodes. Higher voltages also require bulkier, more expensive components and better insulation. A good compromise then, is to take $V_0 = 1700V$. This correspond to a minimum capacity $C_{min} = 134\mu F$ in the case of $U = 200J$, or $C_{min} = 173\mu F$ in the case of $U = 250J$. Choosing a capacity $C = 200\mu F$ should ensure compatibility for both energy level -even though the latter will have worse discharge time compared to those reported in Table 2.8.

Waveform

The firsts defibrillators used to release the capacitor's charge to the patient "as-is". Namely they used to have a direct link capacitor-patient without any sort of modifications, and that meant the capacitor's discharge followed a exponential decay -truncated in order to stop when enough energy

was released (see Figure 3.3). Following models used to add an inductor in order to obtain a damped sine trend to limit sudden changes of currents.

While we still use and base our knowledge on these waveform, there are many different approaches possible.

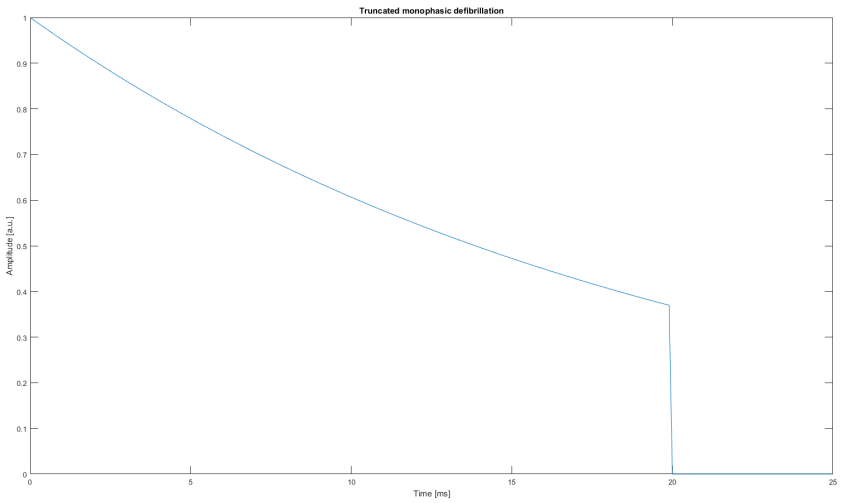


Figure 3.3: Example of a truncated monophasic waveform

Amongst the other waveforms, biphasic (or more in general poly-phasic) waveform are those whom showed the best results. In fact, numerous studies [1, 2, 3] evinced that not only biphasic defibrillation lead to higher chances of resuscitation, but it also require lower energies [4, 5] -when compared to monophasic waveforms such as truncated exponential and dumped sine.

The motivation are not entirely clear, and there are still studies in progress to clarify the precise mechanism of biphasic defibrillation (and defibrillation in general). The primary hypothesis though, is that the first phase leaves a residual charge on the membranes of the unsynchronized cells. These can then still re-initiate fibrillation. The second phase diminishes the residual charge, reducing the potential for re-fibrillation. To suppress this potential re-fibrillation, a monophasic shock must be strong enough to synchronize a critical mass of nearly 100% of the myocytes. Since the biphasic waveform performs this protection function by removing the residual charge, its first phase may be of a lower strength than a monophasic shock of equivalent performance [2].

From an engineer point of view, biphasic defibrillation (see Figure 3.4) can be obtained simply by stopping the capacitor discharge before it release the whole electric dose, then inverting polarities applied on patient, and re-enabling the discharge. In OAED this is obtained with an H-bridge circuit (see section 3.2).

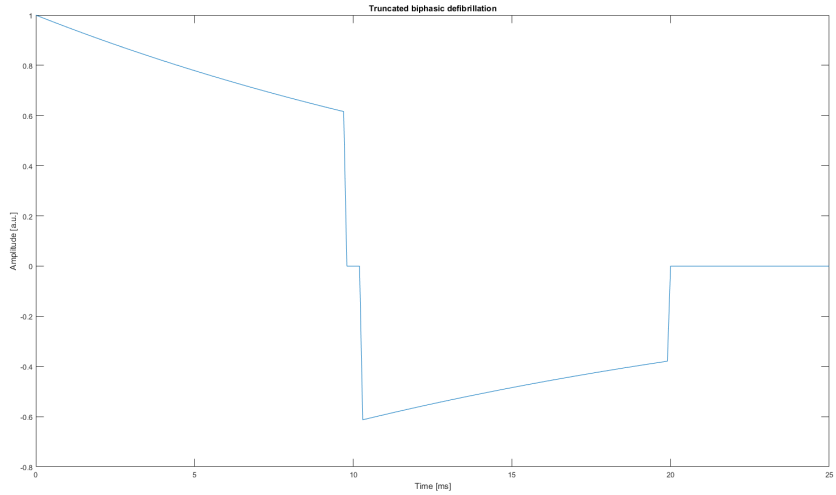


Figure 3.4: Example of a truncated biphasic waveform

3.1.3 Inner and outer selectors

As we mentioned in Chapter 2 (see section 2.4.1), the inner and outer selector represent two means of protection for the both patient and operator.

The first one control whether the discharging circuit shall be connected to the outer selector or not; and it is normally closed on the internal discharge circuit. The second one, instead, control whether the patient shall be connected to the inner selector or not; and it is normally closed on the micro-controller in order to allow the Electrocardiogram signal acquisition. Inner and outer selectors are activated to allow connection amongst patient and capacitor only between the immediate moments before and after the defibrillation.

In OAED, inner and outer selectors are obtained with two double pole, double throw relay. In Figure 3.5 we can see the inner selector on the left, and the outer on the right. In that figure we can also note:

- $DISCH1$ and $DISCH2$, which are the two control connection coming from the PSoC.
- $ECG+$ and $ECG-$, that represent the ECG connection for the PSoC,
- $VIN+$ and $VIN-$, that are connected to the H-bridge,
- $TO\_PATIENT+$ and $TO\_PATIENT-$ which are the defibrillator connection to electrodes,
- resistor R_3, which is the internal discharge circuit, and finally
- GND, that is the common node.

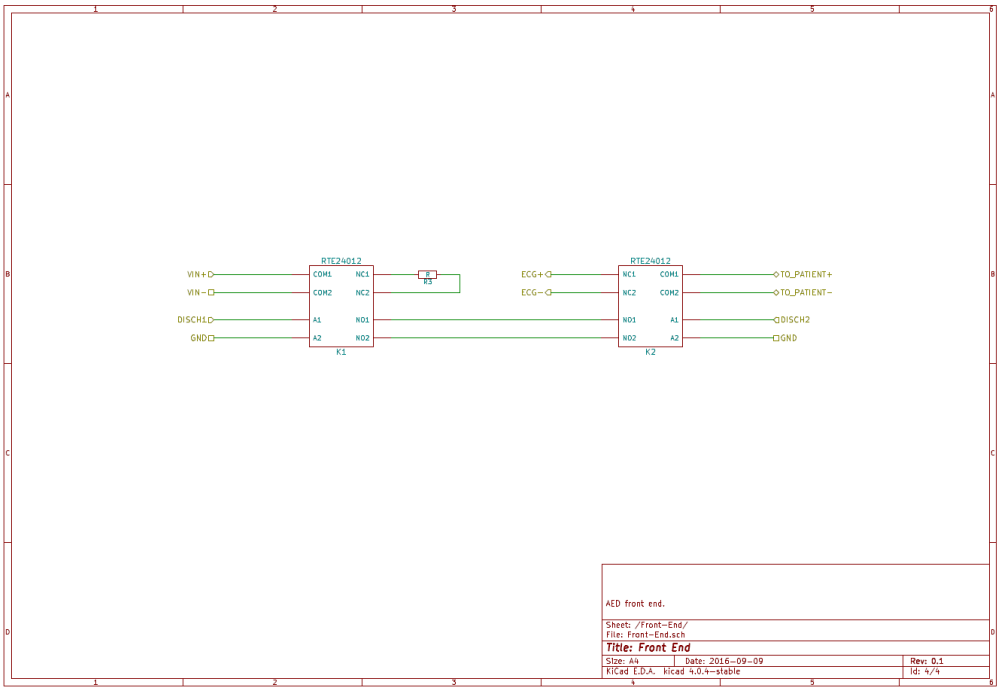


Figure 3.5: Inner and outer selector

3.1.4 Internal discharge circuit

Resistor R_3 in Figure 3.5 represent the internal discharge circuit. This is a 100Ω power resistor capable of dissipating $25W$. A qualitative calculation approximating the discharge time as $T =\simeq 4 \cdot R_3C$ show that the power dissipated are as high as:

$$P = \frac{U}{T} = \frac{\frac{V_0^2C}{2}}{4 \cdot R_3C} = \frac{1700^2}{8 \cdot 100} \simeq 3.6KW$$

Which is ~ 150 times higher than the limit of R_3. However, it is unlikely for a small sized power resistor to dissipate that kind of energy. The solution, in this case, is to pilot the H-bridge to produce a PWM-like discharge, in order to limit the mean current. There will be more detailed information about this in the firmware section.

3.2 H-Bridge

In the last section we introduced the engineer point of view for defibrillation. We also made some consideration about different waveforms, and how a biphasic defibrillation is better than a monophasic one under both the point of view of resuscitation chances and energy requirements.

In order to perform a biphasic defibrillation though, we need a circuit capable of controlling the discharge inverting polarities. The best circuit to perform a biphasic defibrillation then, is the H-Bridge.

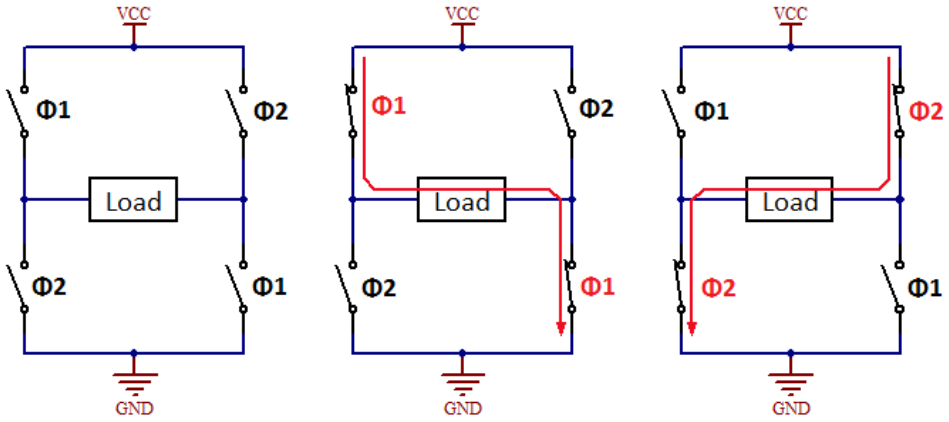


Figure 3.6: Generic H-Bridge

An H-Bridge is a circuit commonly used in robotic and power electronics applications -as an inverter- to control DC motors. In Figure 3.6 there are the schematics that describe the operation of a generic H-Bridge. From those is also evident the name origin, in fact the circuit representation resemble an "H".

An H-Bridge is built with four switches. As we can see in Figure 3.6, by opening and closing them in a controlled manner, it is possible to revert the current flowing through the load. Thus, closing $\Phi 1$ switches while $\Phi 2$ are still open, a positive voltage will be applied across the load. By opening $\Phi 1$ switches and closing the $\Phi 2$ ones instead, a negative voltage will be applied, reverting the current.

As a last observation, it is extremely important to notice that closing both $\Phi 1$ and $\Phi 2$ switches in the same branch will cause a short circuit with devastating effects. To ensure that this would never happen I used redundant controls on $\Phi 1$ and $\Phi 2$ piloting signals, and a short pause between voltage reversions -negligible for the purpose of defibrillation, but vital to ensure a time margin on IGBT switching.

3.2.1 Proposed circuit

In Figure 3.7 there is the schematic of the H-bridge used in OAED. Because of the high voltages and currents in play, each switch of the general design was implemented with two IGBTs. In order to close them, each IGBT require a voltage between Gate and Emitter of about $15V$. Thus, they all require a dedicated insulated power source[3], and an insulated driver. The first

[3]Except for the last two, since their emitters are directly connected to the ground.

is obtained using galvanic-insulated DC/DC converters, whilst the latter require a photocoupler driver. Since IGBTs requires considerable currents when switching, and usually these small insulated DC/DC converter cannot provide it, an additional capacitor was added to each converter in order to store the energy required for commutation.

The two IGBT per switch are required for two main reasons: the first is that IGBT with inverse tension higher than $1500V$ are considerably more expensive and bulky; and the second is that with two IGBT we can obtain an higher de-rating grade. De-rating is the choice of components with much higher rates than those required. This not only ensure a more reliable and robust design, but also significantly extend the device life.

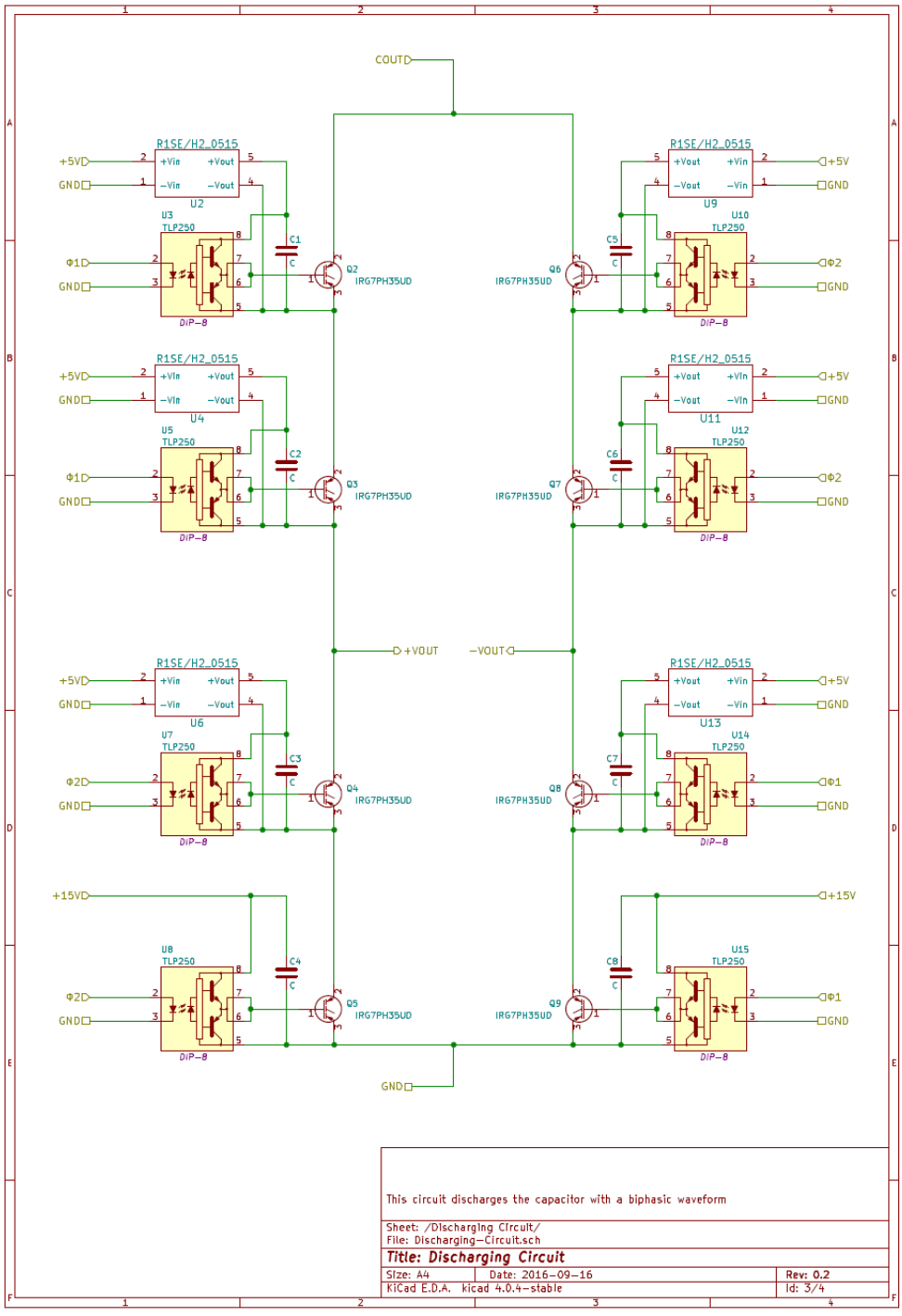


Figure 3.7: Schematics of OAED H-Bridge

3.3 Charging circuit

The last block of Figure 2.4 remaining is the charging circuit, the pulsating heart of the whole defibrillator. The charging circuit is enabled by the PSoC after asserting that the patient need defibrillation. Once enabled, it start charging the capacitor stopping only after it reach the required voltage. At that point the circuit shall ensure that the capacitor remain charged until defibrillation is performed.

The charging circuit makes the difference between good and mediocre defibrillators. Speed and efficiency are two maximum priority in this circuit, and It is in common use for companies to compares their defibrillator mainly with charging time, trying to achieve always better results in a never ending run towards improvement. Albeit the regulations are not so strict about time ($30s$ for frequent use AEDs and $40s$ for infrequent use ones), it is quite clear how faster defibrillators ensure higher chances of resuscitation. During SCAs every second is vital, and in the likely case that multiple defibrillation are required in order to re-establish an effective rhythm, the operator expects a prompt response from the AED.

Summing up, ideally an AED need a charging circuit with the following characteristics:

- ability of producing high-voltage output from low-voltage input,
- high efficiency,
- fast charging speed,
- compact dimensions and low weight,
- relatively cheap,
- compatibility with pure capacitive loads, and
- safety and robustness.

We can then narrow the field to the circuits class of Switching Mode Power Supplies (SMPS). SMPS are electronic power supply that incorporates a switching regulator to convert electrical power efficiently. Conventional power supplies use linear regulators, in which a transistor operate as a variable resistor in order to obtain a stable output voltage. It is quite obvious just from this, that linear regulator's efficiency is limited, and the output voltage cannot be higher than the input. Unlike linear power supplies, SMPS use the transistor as an on/off switch, minimizing thereby energy loss. In fact, an ideal SMPS dissipates no power. Another important characteristic of SMPS, is the considerably higher frequency of operation (up to hundreds of kHz) which allow the use of smaller and more efficient transformers. Especially suitable for portable devices. SMPS are, however, much more complicated to design. Furthermore their switching currents can cause electrical noise problems if not carefully suppressed, simple designs may have a poor power factor, and in some cases circuit analysis and sizing can be very difficult.

A Switching Mode Power Supplies usually consist of a switch, a con-

trol signal, a storage element, and a rectifier. The Pulse-Width Modulation (PWM) control signal open and close the switch with a frequency and a duty cycle depending on the load -even though usually the frequency is fixed and only the duty cycle may vary. The principle of operation is that the storage element is charged when the switch is closed/opened, and it release the energy on the load when the switch is opened/closed. Therefore, by controlling the switching stages it is possible to obtain the desired current on load.

Switching Mode Power Supplies can be classified according to the circuit topology. The most important distinction is between non-isolated converters and isolated ones. The firsts are simplest, and use a single inductor for energy storage. The two basic non-insulated types are the Buck, and the Boost. The insulated SMPS instead, use a transformer as storage element, which allow them to reach a wider range of output voltages.

Amongst the non-insulated SMPS, I selected the flyback topology as the one more suitable for capacitor charging applications.

3.3.1 Flyback

In Figure 3.8 there is represented the generic electric scheme for a flyback converter. Here we can see:

- a BJT transistor Q_1, controlled by
- a PWM signal;
- transformer T_1, which is the storage element;
- a fast recovery diode D_1, needed for the flyback effect; and finally
- the capacitor C, that is the load.

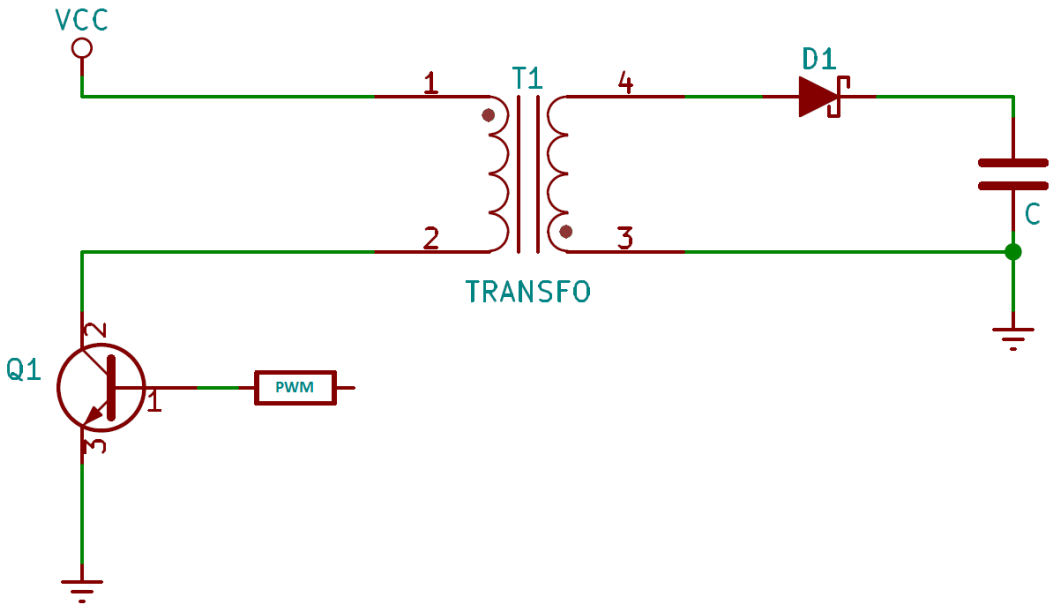


Figure 3.8: Generic flyback converter schematics

The particularity of flyback converters is that the transformer is used as an inductor with multiple windings. When operated under normal conditions, transformers are very simple devices: if we put a voltage across the primary winding, we also get a voltage on the secondary. The voltage ratio follows the turns ratio, irrespective of the output current. However, we have to take note of an important property of transformers: the primary and secondary conduct at the same time. If current flows into the start of the primary winding, it flows out of the start of the secondary winding at the same time.

Figure 3.9 shows the current flowing in a flyback converter. When the transistor Q_1 is on, current flows into the primary winding of T1 (Figure 3.9 (a)), but the secondary diode is not conducting, hence there is no secondary current (Figure 3.9 (b)). When Q_1 turns off, the primary current stops, all winding voltages reverse by flyback effect, and therefore diode D_1 and secondary winding now conduct currents.

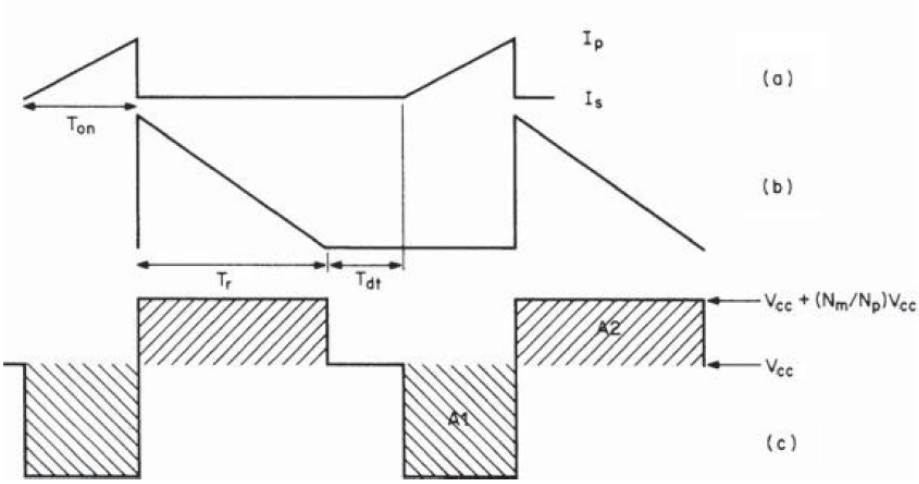


Figure 3.9: Currents flowing in a flyback converter

So the primary and secondary windings in the flyback converter conduct current at different times. This apparently minor difference dramatically changes the rules of operations. As a matter of fact, the so called transformer act as an inductor with multiple winding [6].

In an inductor with multiple windings, there is no correlation between primary and secondary voltages. Instead, the primary to secondary ampere-turns ratios are conserved, as showed in Equation 3.5.

$$N_p \cdot I_p = N_s \cdot I_s \tag{3.5}$$

Hence for instance, if the primary have 100 windings, and there is a current $I_p = 1A$ flowing on it when Q_1 is turned "off", then we have developed 100 ampere-turns in the primary. This must be conserved in the secondary. Therefore with a secondary winding of 10 turns, its current will be $I_s = 10A$.

In this way we can control the current flowing on the secondary, hence the secondary voltage is simply a function of load. Taking the last example, if the load were a 10Ω resistor, then the secondary voltage would have been $V_s = 100V$, while a $1k\Omega$ resistor would produce a $V_s = 10kV$ voltage! This is why flyback are usually used to produce high-voltages[4]. When several secondary windings conducting at the same time, the sum of all the secondary ampere-turns must be conserved. However, we must never forget that voltage transformation is still taking place between primary and secondary windings, even if they are not conducting at the same time. Hence in the last case the voltage $V_s = 10kV$ developed on the secondary winding will reflect back to the primary multiplied by 10. This, added to the supply, will stress transistor Q_1 with more than $100kV$! In our case we are going to develop lower voltages on the secondary; nevertheless if the primary windings are more than the secondary, the voltage reflected would be amplified, and since most of the conventional power transistors cannot stand voltages higher than $200 \sim 300V$, we must avoid that.

To summarize, a flyback converter can force a current on the secondary winding independently from the load. In our case it correspond to charge the capacitor steadily with an high frequency pulsed current. This suits perfectly for our purpose., but we still need more insight on how a flyback works.

Mode of operation

A flyback has two different operating modes: the continuous mode and the discontinuous mode. Waveforms, performance, and transfer functions are different depending on the mode. In Figure 3.10 there are the typical waveforms.

Starting with the discontinuous mode; during the Q_1 "on" time, there is a fixed voltage V_{cc} across N_p, and current in it ramps up linearly (see Figure 3.9 (a)) at a rate of:

$$\frac{\partial I_p^{pk}}{\partial t} = \frac{V_{cc} - V_{on}}{L_p} \tag{3.6}$$

Where L_p is the primary magnetizing inductance. At the end of "on" time, the primary current has ramped up to $I_p = \frac{(V_{cc}-V_{on})\cdot T_{on}}{L_p}$. Which correspond to a stored energy of:

$$E = \frac{L_p \cdot {I_p^{pk}}^2}{2} = \frac{(V_{cc} - V_{on})^2 \cdot T_{on}^2}{2L_p} \tag{3.7}$$

[4]Beware that leaving the secondary open without a load would destroy the semiconductors, but also produce spectacular fireworks.

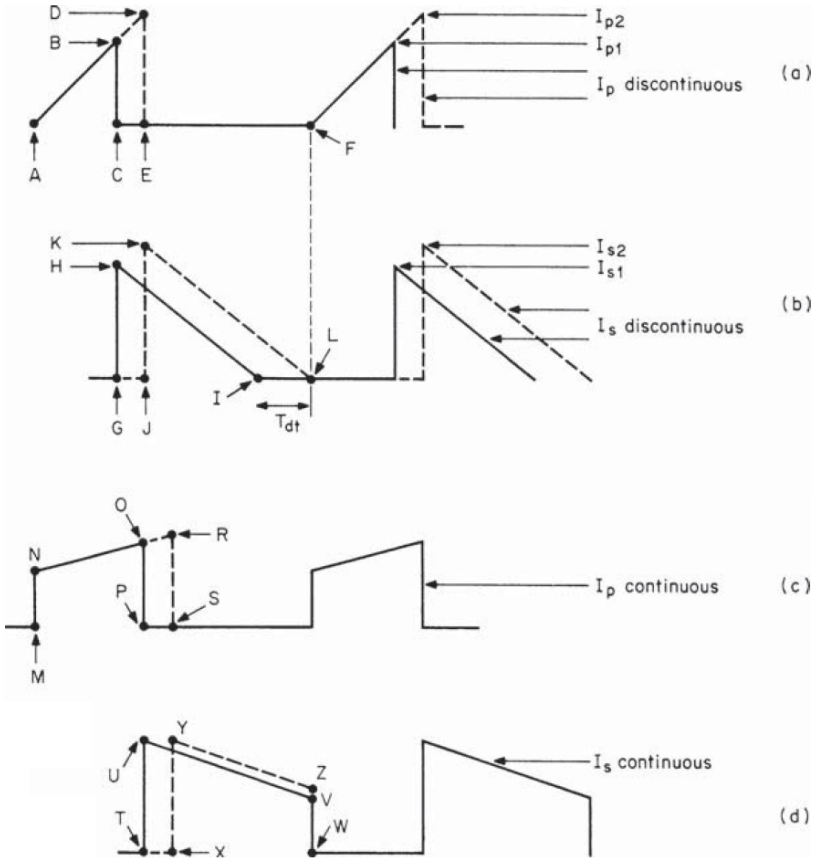


Figure 3.10: Currents flowing in a flyback converter, differences between discontinuous and continuous mode

When Q_1 turns off, the current in the magnetizing inductance forces a reversal of polarities on all windings -this is the flyback effect. The non-dot end of N_s then, is positive with respect to its dot end and current flows out of it, but ramps down linearly (Figure 3.9 (b)) at a rate of:

$$\frac{\partial I_s}{\partial t} = \frac{V_C + V_{\gamma D_1}}{L_s} \tag{3.8}$$

Where L_s is the secondary winding magnetizing inductance[5], and V_C is the voltage across the output capacitor. This current reaches zero at time I (on Figure 3.10 (a)), leaving a dead time T_{dt} before the start of the next turn "on" period at point F. All the energy stored in the primary during the previous "on" period has now been completely delivered to the load before the next period. The average DC output current will be the average of the triangle $\overset{\triangle}{GHI}$, multiplied by its own duty cycle $\frac{T_{off}}{T}$.

So long that there is a dead time T_{dt}, the circuit remain in discontinuous mode. As T_{on} increase, primary current slope remains constant, and its peak

[5]Knowing that the relation between primary and secondary magnetizing inductance is $L_s = \frac{N_s^2}{N_p^2}$.

rises from B to D; secondary current instead, start later in time (point J), and its peak increases from H to K.

Considering also the secondary slope to remain constant[6], the point at which the secondary current falls to zero moves closer to the start of the next turn "on". Reducing therefore dead time T_{dt}, until a point L is reached. In this condition, secondary current has just fallen to zero at the instant of the next turn "on". This marks the end of the discontinuous mode.

Further increase of "on" time T_{on} will delay secondary current at a point later than J, and higher than K in Figure 3.10 (b). Then at the start of the next "on" period (point F), there will be still some current or energy left in the transformer. This will produce a small step in the primary current I_s (see Figure 3.9 (c)), that now have a trapezoid form. This is the continuous mode.

Increasing T_{on} will move the start of secondary current from point T to X. Since the back-end of the secondary current pulse (point W) cannot move further to the right in time, this is traduced in a decreased area. And thereby a lower DC output current.

As a result of this brief analysis of operation then, it is clear how ideally working at the point of transition between discontinuous and continuous mode provide the maximum current to the output capacitor C. To operate in that condition is necessary a feedback loop that modulate T_{on} (or more precisely T_{off}) in order to close the transistor Q_1 as soon as the secondary winding stop conducting, and open it after the desired peak current is reached. Such circuit is called *self-oscillating flyback*, or more commonly **Ringing Choke Converter (RCC)**.

3.3.2 From flyback to self-oscillating flyback

Since Ringing Choke Converter operations are not as intuitive as the case of the simple flyback, I have decided to follow a bottom-up approach to describe it. Namely we add more and more components to the circuit, until we obtain the complete configuration.

The first step towards a Ringing Choke Converter is represented in Figure 3.11. The easiest RCC possible is obtained by adding a tertiary winding (with the same ratio as the primary) to the transformer in order to use it as a *positive feedback*. The operation of this circuit can be resumed in 10 steps.

1) At the beginning Q_1 is open, and no current flow in the transformer T_1.

2) A small current flowing in start-up resistor R_{st} will provide a base current for Q_1 turning it partially "on".

[6]This is not entirely true. Secondary slope depend on output voltage V_C, but since this varies very slowly, compared to the single cycle, we can safely approximate it as constant.

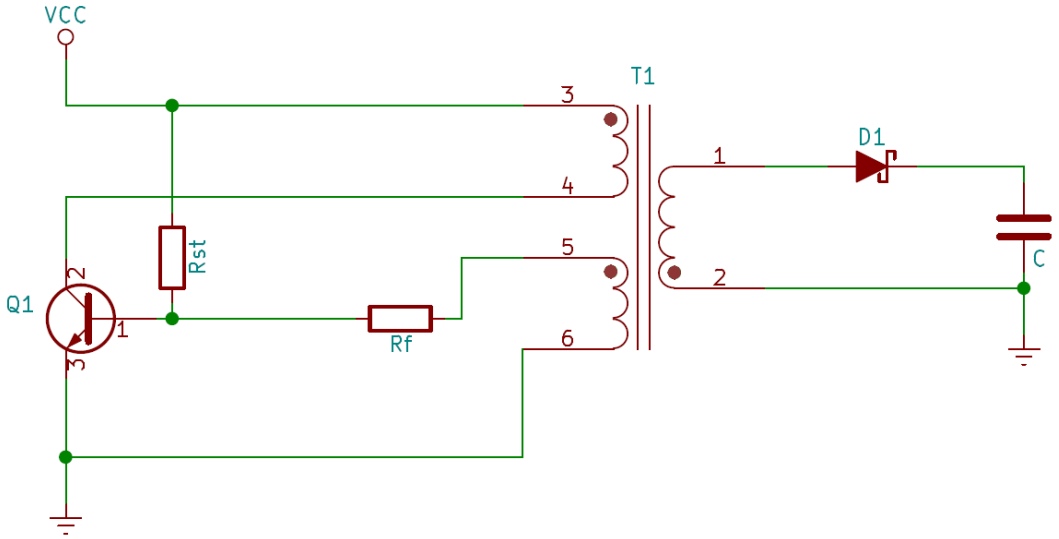


Figure 3.11: Ringing choke converter, first step

3) While Q_1 turn on, a small current will start flowing through the primary winding.

$$\frac{\partial I_p}{\partial t} = \frac{V_{cc} - V_{ce}}{L_p}$$

4) The current I_p flowing in the primary winding start charging the transformer. Meanwhile the primary voltage V_p is reflected on the secondary.

$$V_t = V_p = V_{cc} - V_{ce}$$

5) A current induced by the voltage V_t, and limited by resistor R_f start flowing in the base of Q_1.

6) As the current I_b flowing in Q_1 base increase, the transistor working point moves from forward-active zone towards saturation. By doing so the voltage between collector and emitter V_{ce} decrease. Consequently V_p increase, and hence even V_t and I_b. Thus the secondary winding form a positive feedback that quickly leads Q_1 to saturation.

7) While Q_1 moves towards saturation, I_p increase linearly until it finally meet the critical point $I_p = \beta \cdot I_b$. Where β is the common-emitter gain of Q_1. After the critical point Q_1 moves back into forward-active zone, increasing the voltage V_{ce}.

8) Once again the circuit is driven by the positive feedback of the tertiary winding. This time though, it decrease I_b bringing Q_1 quickly to interdiction.

9) When Q_1 is completely open the flyback effect start, reverting the voltages across all three windings. A current then start flowing on secondary winding to charge C. Meanwhile I_b is still driven by V_t, but since the latter is now negative then also the first is negative. This ensure Q_1 stay open until the secondary is still conducting.

10) When all the energy accumulated on the transformer has been released on capacitor C, there will be no more currents flowing on the circuit and Q_1 is still open. We are in the same condition as 1) and the cycle can restart.

The advantage of this circuit, compared to a conventional flyback, is that just by adding few components it was obtained a PWM generator capable to operate the circuit exactly between continuous and discontinuous mode. The circuit still has two important flaws though: the cycle duration depends on β, which is a notoriously unreliable value; and the output is not regulated, and this means the capacitor will never stop charging.

In order to solve the first problem, we can add a capacitor C_f between transistor's base and R_f, as in Figure 3.12. The last analysis steps are still valid until Q_1 enter in saturation mode (6). After that, the following applies.

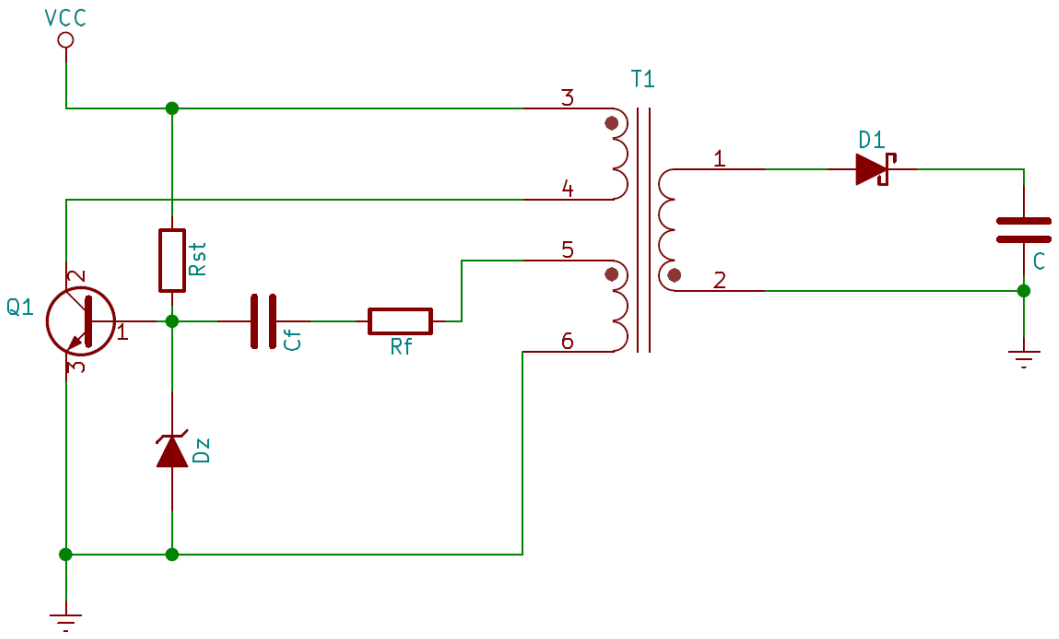


Figure 3.12: Ringing choke converter, second step

7) While I_p increase, C_f will charge. As C_f charges, the voltage across R_f decrease, and therefore even the transistor's base current I_b.

8) With increasing I_p and decreasing I_b the transistor is sent from saturation to forward-active mode, with a consequent increase of V_{ce}.

9) Once again the tertiary winding's positive feedback quickly move the transistor to interdiction.

10) During interdiction the flyback effect take over with the usual consequences. In this case though, C_f is discharged by the negative voltage V_t.

11) Once the secondary have released all the stored energy, the circuit return to the initial condition and the cycle restart.

The above circuit does not rely any more on β to determinate the duration of transistor's saturation, and hence T_{on}. This only depends on C_f, R_f, and R_{st}. Therefore, we can modulate maximum T_{on}, and by consequence the minimum frequency f_{min}, just varying the values of these components.

A special note should be taken on the zener diode D_z, which has two main purposes:

- it protect the transistor Q_1; As the capacitor C get closer to $V_0 = 1700V$, the voltage reflected on primary and tertiary windings can be as high as $100V$ -considering a turn ratio 1:17:1; and
- it prevent the inverse charging of C_f. As the tertiary winding's inverse voltage increase we might arrive at the point when C_f not only is completely discharged, but it is also inversely polarized. This would traduce in increasing T_{on} as the voltage on C grows.

We have then obtained a circuit that operate between continuous and discontinuous mode with a controllable T_{on} -let's not forget that T_{on} also control primary winding's peak current I_p^{pk}. The only thing we still miss now, is the output regulation. In order to achieve it, the next step is represented in Figure 3.13. In this circuit:

- Q_2 aim to limit primary peak current, by sinking current from Q_1's base when I_p surpass a certain threshold;
- R_s and R_b form a voltage divider to pilot Q_2 -note that R_s shall be the smaller possible, in order to limit losses; and
- C_b is used to polarize Q_2.

In other words, the higher will be I_{b2}, the lower will be primary peak current I_p^{pk}. The principle of operation is similar to the previous.

1) Initially there is no current flowing in the circuit, except for the one on R_{st}. This will charge C_f until Q_1 enter in forward-active mode. Hence a small current I_p start flowing on the primary winding accumulating energy in the transformer.

2) The tertiary winding's positive feedback bring Q_1 quickly in saturation. Meanwhile I_p increase linearly.

3) Q_2 stay open until I_p is not high enough to produce a voltage on R_s at least as high as V_γ. Where V_γ is the threshold voltage of Q_2.

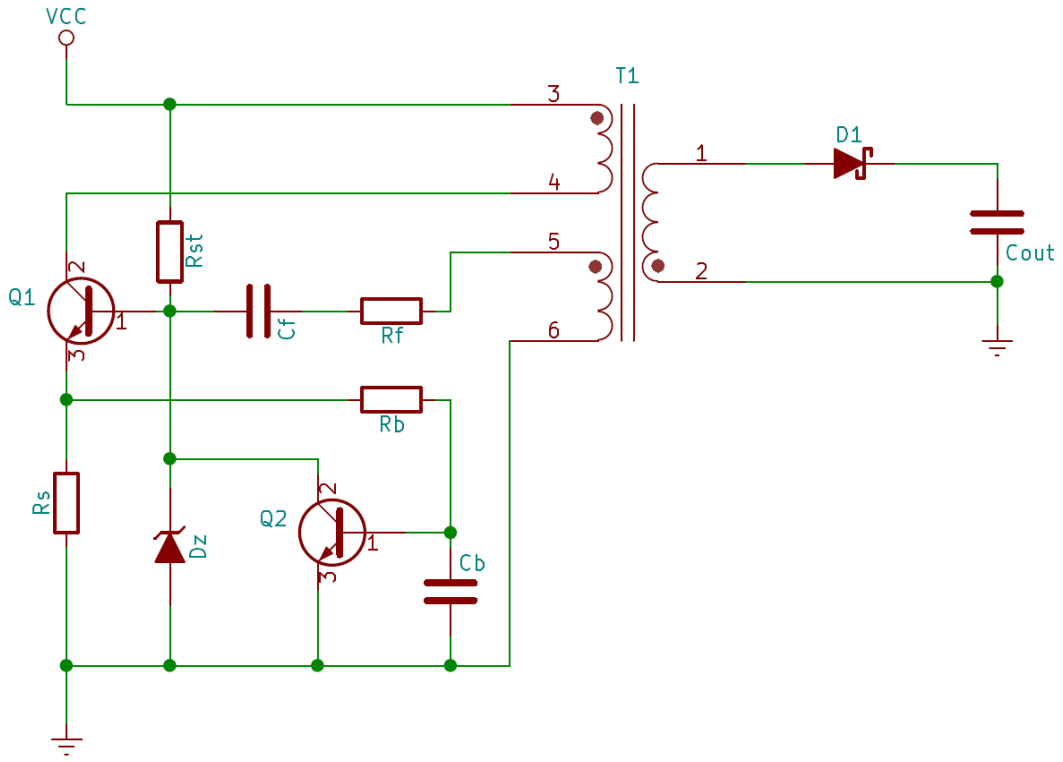


Figure 3.13: Ringing choke converter, third step

4) Once exceeded the threshold voltage, Q_2 move from interdiction to forward-active mode, discharging C_f, and therefore turning off Q_1. As Q_1 turn-off, the voltage across R_s decrease, and also Q_2 turn-off. Once again the tertiary winding's positive feedback ensure the two transistor enter interdiction as the current on primary decreases.

5) When Q_1 turn-off, the flyback effect drive the secondary winding with the same modalities described before. In this phase Q_2 is working in reverse-active mode, ensuring that Q_1 remain interdicted.

6) When all the energy stored in the transformed is released on the secondary winding, the current flowing in the circuit stops (with the exception of the one flowing in R_{st}). Q_2 then return interdicted and the cycle restart.

In this circuit T_{on} is controlled by Q_2 in order to store in the transformer the same amount of energy every cycle.

$$T_{on} = \frac{V_\gamma \cdot L_p}{(V_{cc} - V_\gamma) \cdot (R_s // R_b)} \tag{3.9}$$

Equation 3.9 and Equation 3.7 are the most important relations of the RCC. It is obvious from them, how reducing magnetizing inductance L_p

is the key to obtain more speed by both increasing the energy transferred per cycle, and the number of cycles per seconds. As we know then, higher frequencies also corresponds to better efficiency because of the transformers laws.

At first sight it may seems that Q_2 only purpose is to control T_{on}. Nevertheless, Q_2 will also provide a mean of voltage regulation. In fact, if we have a way to inject a current on Q_2's base when C is charged, then the circuit stops. And this leads us to the final form of OAED's Ringing Choke Converter, shown in Figure 3.19. In this, a voltage divider on the capacitor C will provide the reference voltage. This is then compared with a $2.5V$ shunt regulator that drive a photo-transistor. When the voltage on C pass $1700V$, the shunt regulator start conducting and the photo-transistor inject a current in Q_2's base, which by consequence will stop the charging circuit. When the voltage across C turn back below $1700V$, instead, the shunt regulator stop the current flow on the photo-transistor and the circuit restart operating normally.

3.3.3 Ringing Choke Converter

In Figure 3.19 there is the electronic schematic for OAED's proposed solution. In addition to what already said, this circuit has an op-amp based buffer, which is used to provide the PSoC a measurement of capacitor's charge level. Furthermore, transistor Q_1 has been replaced with the power MOSFET S_1, which offers much better characteristic in terms of: switching frequencies, power loss, maximum reverse voltage, heat, etc.

We have seen in the last section a qualitative analysis of a RCC operation. However, in order to be able of actually designing a RCC, it is necessary a more accurate analysis. We start by redrawing the circuit without the photo-transistor, and the output voltage divider; because for now we are only interested in what happen during the charge, and not after it completes. Then we replace the transistors Q and S with their equivalent model, accordingly to the working point in their input/output characteristic. Finally, we also assume that the system is already oscillating with stable oscillations. What we obtain is an eight stage model whose states are represented in Figure 3.18.

Stage 1

In reference to Figure 3.18 (a), we start when the current flowing on secondary windings for flyback effect is almost over. In this stage, MOSFET S is interdicted, while transistor Q is in active-forward zone. The voltages across L_p and L_t are negative, as a result of the reflection of V_s. Because of that the current flowing on L_t is is charging C_f, and the one flowing in Lp is charging MOSFET's output capacitance C_{oss}. Meantime, the input capacitance C_{iss} is in stale and Q_1's base capacitor C_b is still discharging.

As the secondary current I_s approaches to its end, the polarity of voltages across L_p and L_s gradually decreases until the latter matches the combined voltage across C_f and R_f, and we pass to stage 2.

Stage 2

As we can see in Figure 3.18 (b), this is the precise instant in which the current in L_s branch is stalling. Q enters in inverse cut-off, while C_{iss} is inverse polarized because of the C_{GD} fast discharging. C_{oss} is releasing the charge accumulated during stage 1 and 8, also decreasing voltage V_{DS}. As C_{oss} completely discharges, V_p slowly approaches V_{cc}.

Stage 3

In this stage (see Figure 3.18 (c)) I_s restart flowing in the opposite direction as stage 1, and thus C_f start discharging. Meanwhile C_{oss} is reverted and C_{GD} is still discharging.

Stage 4

When all the energy stored is depleted, I_p inverts once again, and hence C_{oss} restart charging. Now that C_{GD} is not absorbing current any more, C_{iss} can begin its charge. Therefore the increasing value of V_G bring Q working in forward-active mode. As C_{oss} charges, the voltage across L_p, and by transformation also the one across L_s, decreases. This stage is represented in Figure 3.18 (d).

Note that the positive feedback action of L_t here is lost because of the MOSFET. If we still had a BJT, the current in its base would have caused an increase in V_{lp}, and hence a positive feedback with it's own base current, as described in the last section.

Stage 5

When V_{GS} exceeds the MOSFET threshold voltage V_{th}, it turns "on". The current I_p now increases as a ramp, according to Equation 3.6. With increasing I_p, the voltage across R_s increases too. Meanwhile V_t increase again, but not enough to invert the current I_t.

Stage 6

As V_{R_s} increases, it will equalize and eventually exceed the voltage on Q base V_b, starting the charge of C_b. This traduces to higher V_b, and therefore higher I_c, which means C_{iss} restart to discharge, decreasing V_{GS} with him. As V_{GS} decreases, V_{DS} raise and by consequence V_p and V_t reduces too. Once again because of the positive feedback of L_t, I_t will keep decreasing until eventually changes direction.

Stage 7

By consequence of I_t inversion, C_f start recharging. In this stage, represented in Figure 3.18 (g), C_{iss} is dried up quickly from the combined action of I_t and Q collector's current I_{Qc}.

Stage 8

This stage start when the MOSFET is turned "off". Consequently, the flyback effect take action on the secondary windings releasing the energy stored during stages 5, 6, and 7; see Figure 3.18 (h). V_p goes back at the value of the tertiary voltage divided by the windings ratio, and therefore recharges C_{oss}. Since I_p is almost nil, C_b restart its discharge and the transistor Q enter in saturation mode. As the current on the secondary diminishes we turn back at the condition of stage 1 and the cycle can restart.

Baker clamp

Baker clamp is a generic name for a class of electronic circuits that reduce the storage time of a switching bipolar junction transistor by applying a non-linear negative feedback through various kinds of diodes. The diode-based Baker clamps prevent the transistor from saturating and thereby accumulating a lot of stored charge.

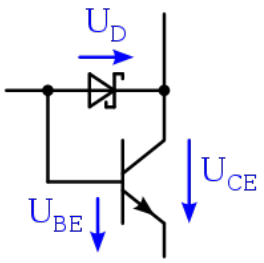


Figure 3.14: Baker clamp

The basic idea is to avoid saturation by decreasing the gain near the saturation point with a non-linear negative feedback (see Figure 3.14). While the transistor is in active mode and it is far away enough from the saturation point, the negative feedback is turned "off" and the gain is maximal; when the transistor approaches the saturation point, the negative feedback gradually turns "on", and the gain quickly drops. To decrease the gain, the transistor acts as a shunt regulator with regard to its own base–emitter junction: it diverts a part of the base current to ground by connecting a voltage-stable element in parallel to the base–emitter junction.

In my RCC, the addition of a baker clamp on transistor Q leave the circuit stability unvaried offering various benefits, amongst which:

- elimination of Q saturation;
- reduction of currents flowing R_{st};
- less ringing effect when MOSFET S turn "off";
- more efficiency,
- faster transitions.

The baker clamp effect on the previous step-to-step explanation is the addition of a current path between the base of transistor Q, and the gate of MOSFET S. This current path avoid the inverse polarization of Q, further

stabilizing S functioning. Lastly, it prevent the complete shut-down of the circuit, which may not restart oscillating under certain sub-optimal conditions; instead it put it in a *energy saving* mode of operation, which is barely sufficient to the capacitor charged at the wanted voltage.

3.3.4 Simulations

All the models described in this section have been validated and tested with LTspice (see the end of this chapter for more information). In Figure 3.15 there is the model I made for the simulations. In Figure 3.16 there are the plots of the most significant parameters, while in Figure 3.17 there is a close-up of a single oscillation. Because LTspice does not natively has the shunt regulator I wanted to use on my RCC, i replaced it with a op-amp and a $2.5V$ reference.

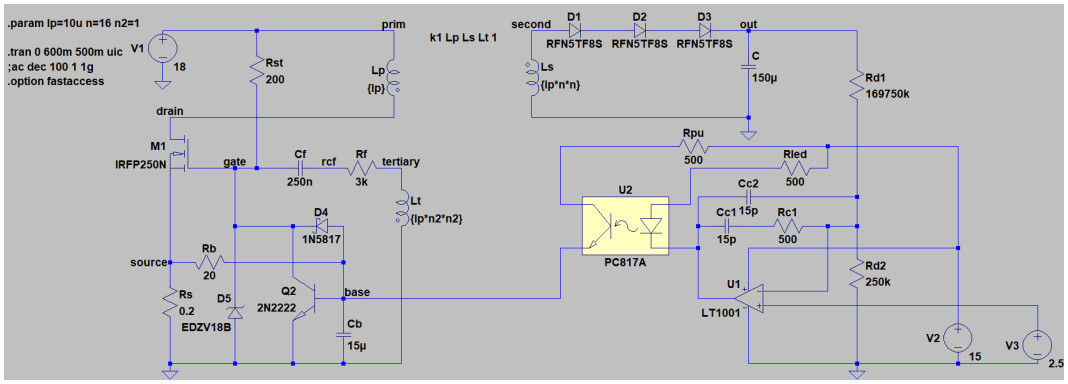


Figure 3.15: RCC circuit on LTspice

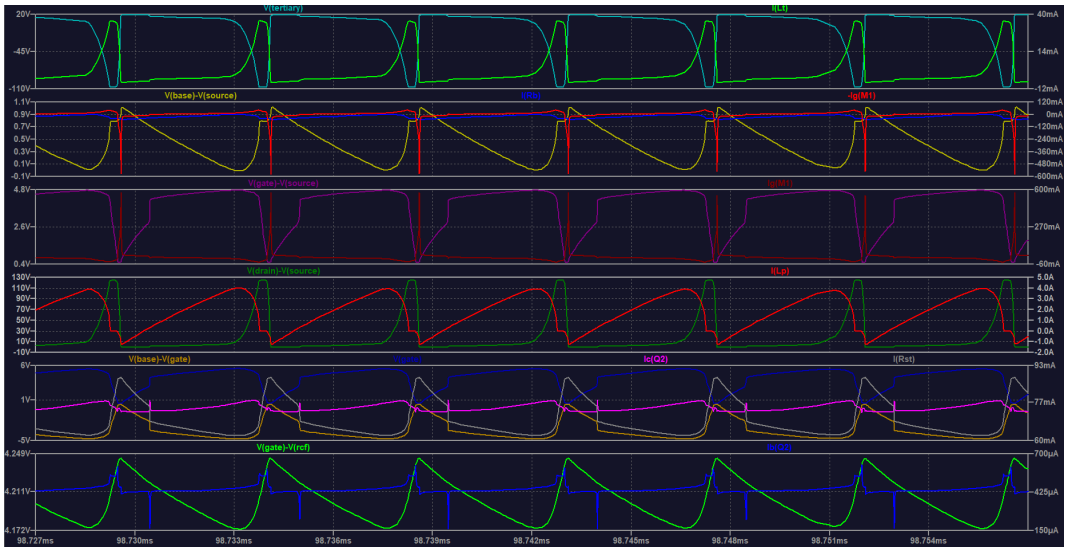
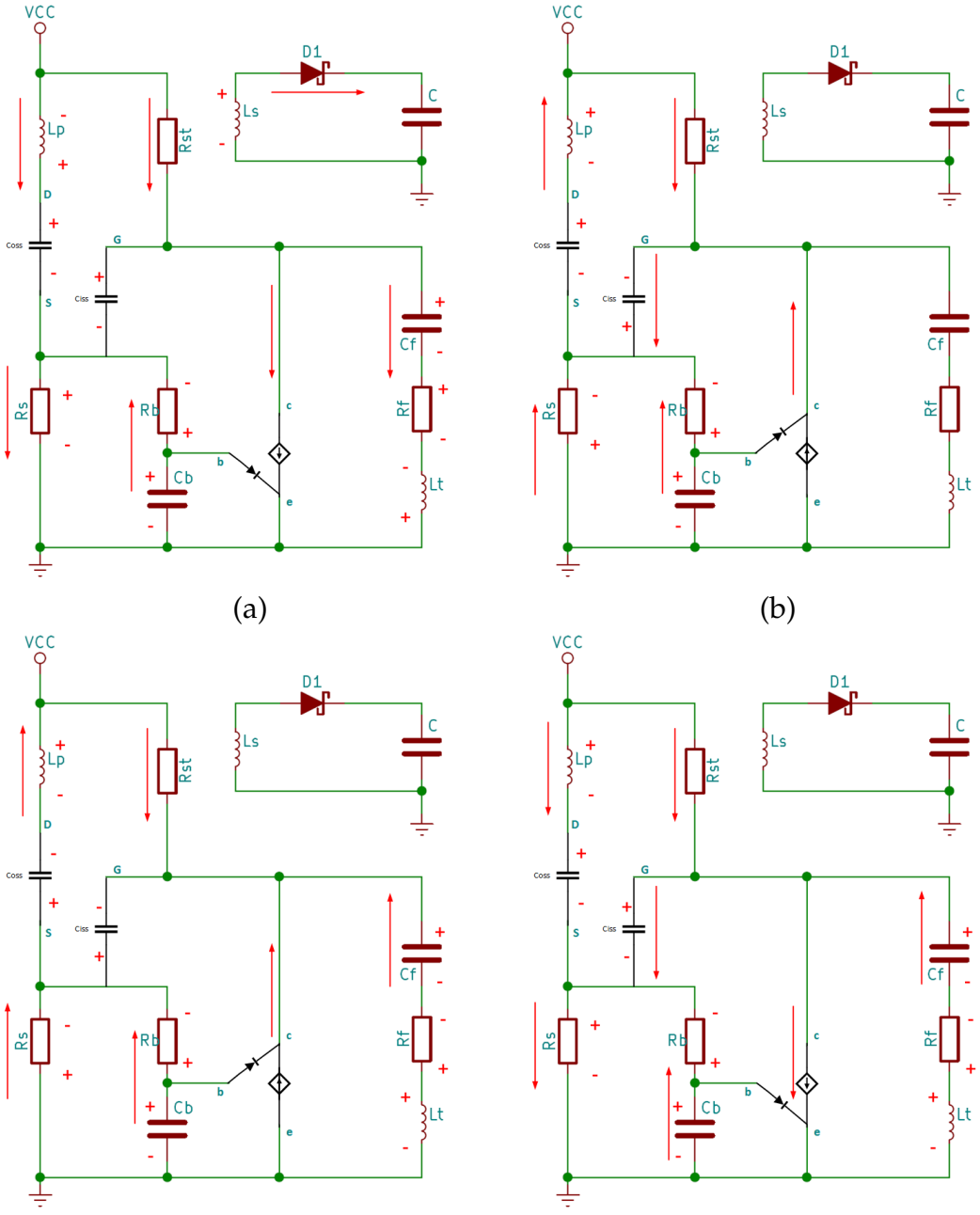


Figure 3.16: RCC simulations.

Figure 3.17: RCC simulations, close-up.

This circuit is capable of charging a $150\mu F$ capacitor at a nominal voltage of $1700V$ in about 6 seconds. Its operative frequency is between $75KHz$ -when the capacitor is fully discharged- and $220KHz$ -when is about to finish its charge.



(a) (b)

(c) (d)

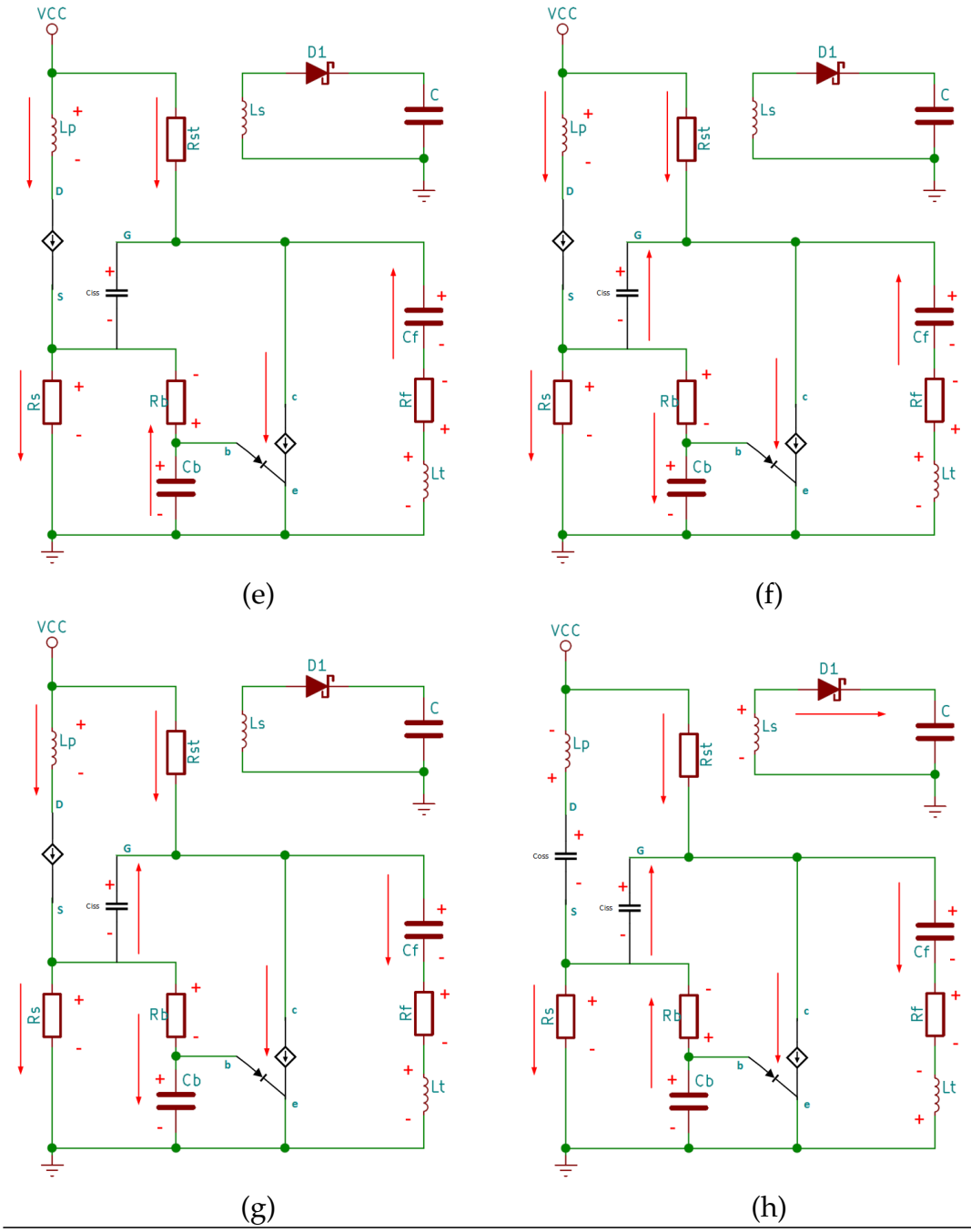


Figure 3.18: Ringing choke converter - the eight different stage of operation

3.4 Software used

As one might expect, during the course of OAED's design I have used several tools and software. At the end of this and the following two chapter, there will be a section dedicated to the tools involved.

During the High-Voltage Board design I have made large use of an electronic simulator software in order to evaluate and validate the concept of operations of the circuits -mainly the Ringing Choke Converter. Trial and error approach is common in self-oscillating circuits, thereby it wouldn't be practical without a good simulator.

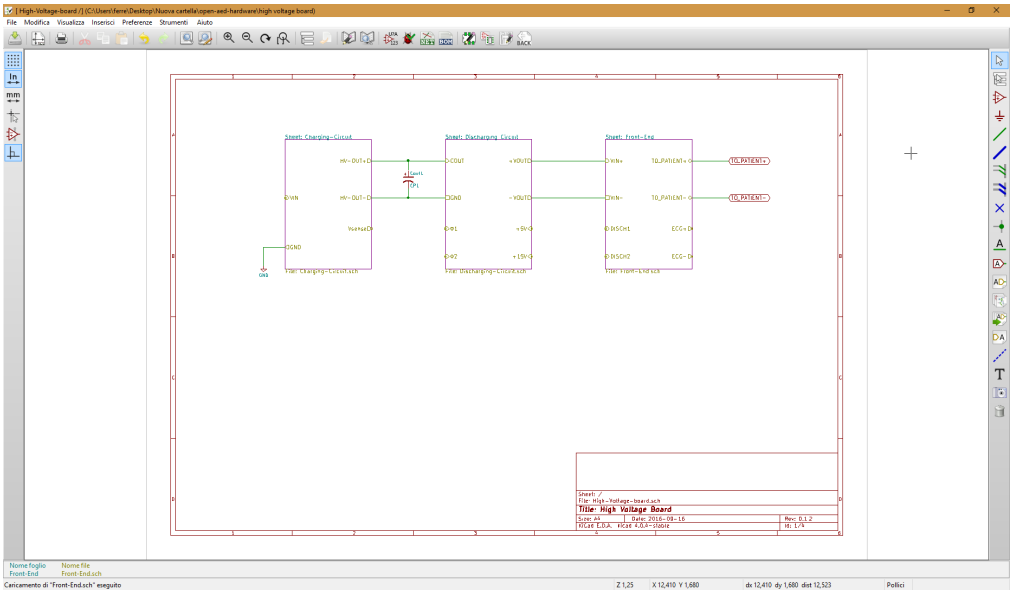
Then in the end, when I finished the design of all circuits, I used an electronic CAD software in order to draw the schematics in an ordered fashion way.

3.4.1 KiCad

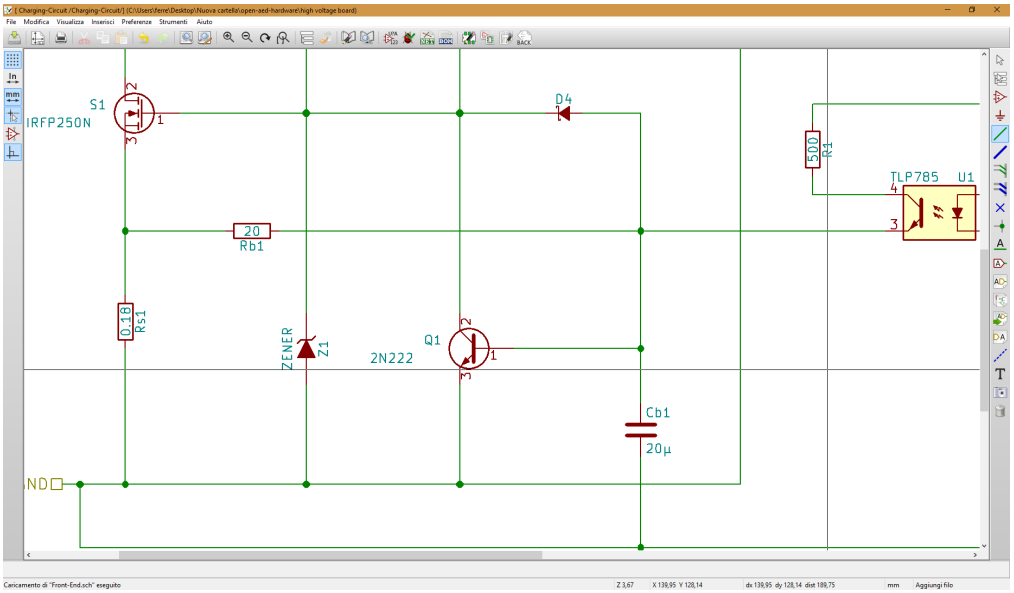
KiCad is an open source and cross platform software suite for electronic design automation (EDA) [7]. It facilitates the design of schematics for electronic circuits and their conversion to PCB designs. It is composed of various software tools:

- KiCad : project manager
- Eeschema : schematic editor and component editor
- CvPcb : footprint selector helper (always run from Eeschema)
- Pcbnew : circuit board layout editor and footprint editor
- GerbView : Gerber viewer

Being open source (GPL licensed), KiCad represents the ideal tool for projects oriented towards the creation of open-source electronic hardware. It also does not present any board-size limitation and it can handle up to 32 copper layers, 14 technical layers and 4 auxiliary layers.



(a)



(b)

Figure 3.20: Examples of KiCad Eschema interface

3.4.2 LTspice

LTspice is a proprietary, free-ware software produced by semiconductor manufacturer Linear Technology (LTC).

LTspice is a high performance SPICE simulator, schematic capture and waveform viewer with enhancements and models for easing the simulation of switching regulators. LT enhancements to SPICE have made this program particularly suitable for simulating switching regulators, allowing the user to view waveforms for most switching regulators in just a few minutes [8].

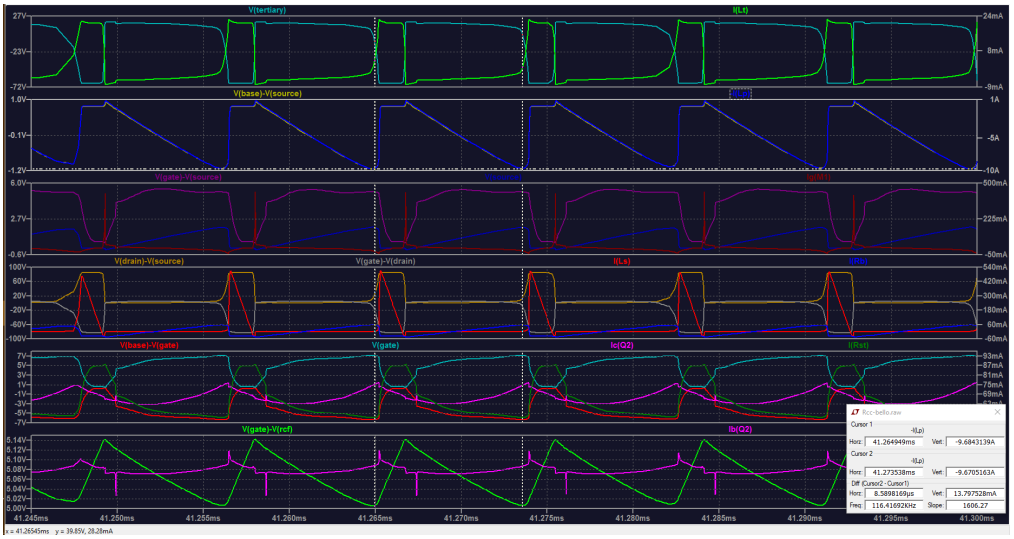
Spice

SPICE (Simulation Program with Integrated Circuit Emphasis) is a general-purpose, open source analog electronic circuit simulator. It is a program used in integrated circuit and board-level design to check the integrity of circuit designs, and to predict circuit behaviour.

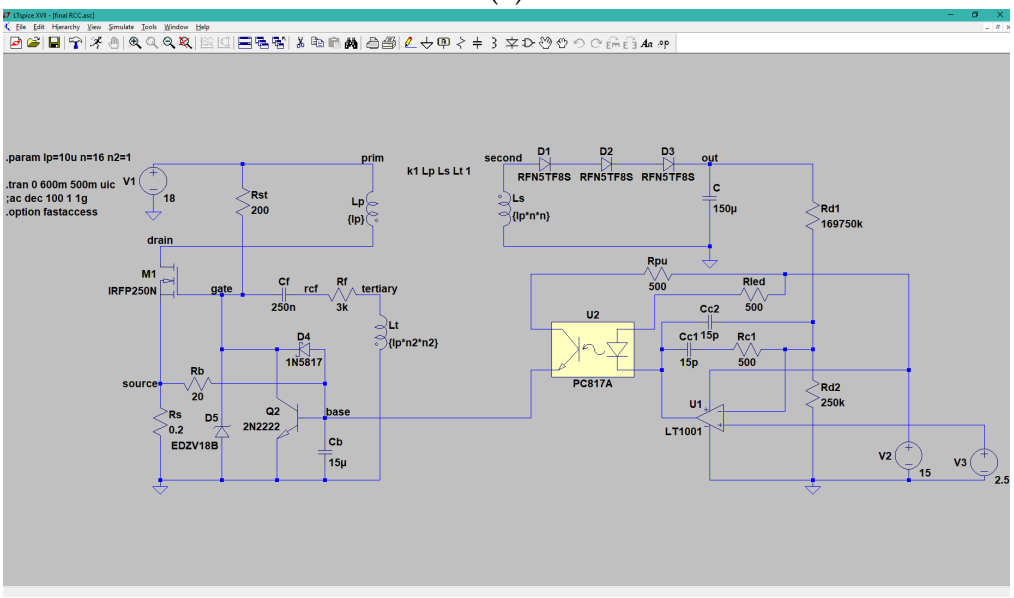
Circuit simulation programs, of which SPICE and derivatives are the most prominent, take a text net-list describing the circuit elements (transistors, resistors, capacitors, etc.) and their connections, and translate this description into equations to be solved. The general equations produced are non-linear differential algebraic equations which are solved using implicit integration methods, Newton's method and sparse matrix techniques.

As an early public domain software program with source code available, SPICE was widely distributed and used. Its ubiquity became such that "to SPICE a circuit" remains synonymous with circuit simulation.

Albeit its third and last release remain untouched from 1993, the BSD license allowed it to be used as a basis for many other circuit simulation programs, amongst which LTspice.



(a)



(b)

Figure 3.21: Examples of LTspice interface

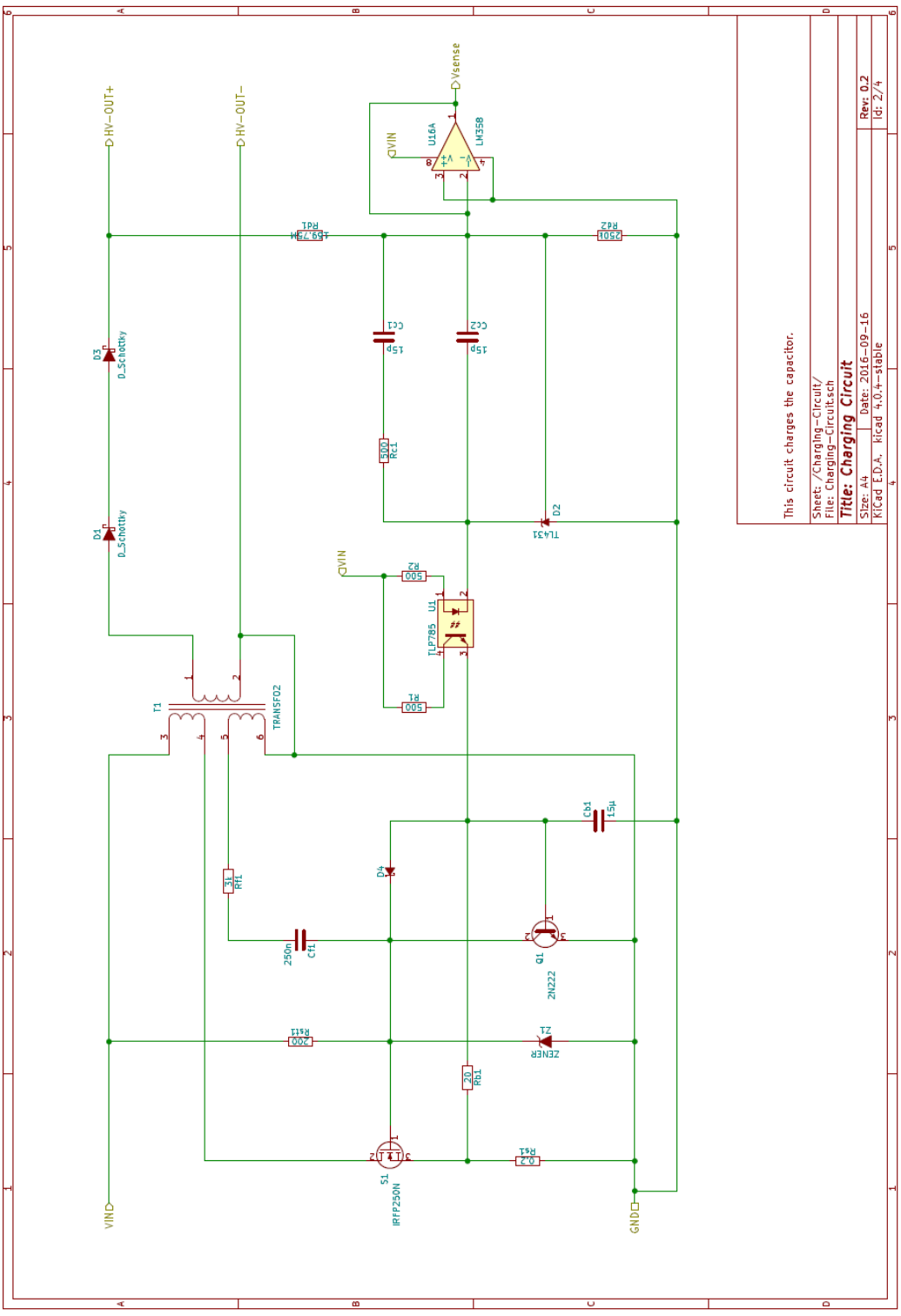


Figure 3.19: OAED's charging circuit

Bibliography

[1] Wanchun Tang et al. "The effects of biphasic and conventional monophasic defibrillation on postresuscitation myocardial function". In: *Journal of the American College of Cardiology* 34.3 (1999), pp. 815–822. ISSN: 0735-1097. DOI: http://dx.doi.org/10.1016/S0735-1097(99)00270-3. URL: //www.sciencedirect.com/science/article/pii/S0735109799002703.

[2] MARK W. KROLL. "A Minimal Model of the Single Capacitor Biphasic Defibrillation Waveform". In: *Pacing and Clinical Electrophysiology* 17.11 (1994), pp. 1782–1792. ISSN: 1540-8159. DOI: 10.1111/j.1540-8159.1994.tb03746.x. URL: http://dx.doi.org/10.1111/j.1540-8159.1994.tb03746.x.

[3] MICHAEL BLOCK and GÜNTER BREITHARDT. "Optimizing Defibrillation Through Improved Waveforms". In: *Pacing and Clinical Electrophysiology* 18.3 (1995), pp. 526–538. ISSN: 1540-8159. DOI: 10.1111/j.1540-8159.1995.tb02563.x. URL: http://dx.doi.org/10.1111/j.1540-8159.1995.tb02563.x.

[4] J. L. Jones et al. "Increasing fibrillation duration enhances relative asymmetrical biphasic versus monophasic defibrillator waveform efficacy". In: *Circ. Res.* 67.2 (Aug. 1990), pp. 376–384.

[5] G. H. Bardy et al. "Multicenter comparison of truncated biphasic shocks and standard damped sine wave monophasic shocks for transthoracic ventricular defibrillation. Transthoracic Investigators". In: *Circulation* 94.10 (Nov. 1996), pp. 2507–2514.

[6] Taylor Morey Abraham I. Pressman Keith Billings. *Switching power supply design*. 2009.

[7] *KiCad EDA - A cross platform and open source electronics design automation suite*. http://kicad-pcb.org/.

[8] *LTspice - Linear Technology spice simulator*. http://www.linear.com/.

CHAPTER 4

Control Board

As we largely mentioned in the last chapters, the Control Board (C-B) represent the mind of the AED. In chapter two we saw the High-Voltage Board (HV-B), which contains all the circuits necessary in order to perform the defibrillation. However, no matter how well is designed, the HV-B alone is not enough. In fact, the main difference between a simple defibrillator and an AED is the ability to acquire the Electrocardiogram signal from the patient, and to recognize if its rhythm is *shockable*. Namely, to recognize if the patient is suffering either Ventricular Tachycardia or Ventricular Fibrillation (see chapter 1 for more information about VT and VF). *Automation* and *control* are the two key words when talking about AEDs. And all this is only possible thanks to the Control Board.

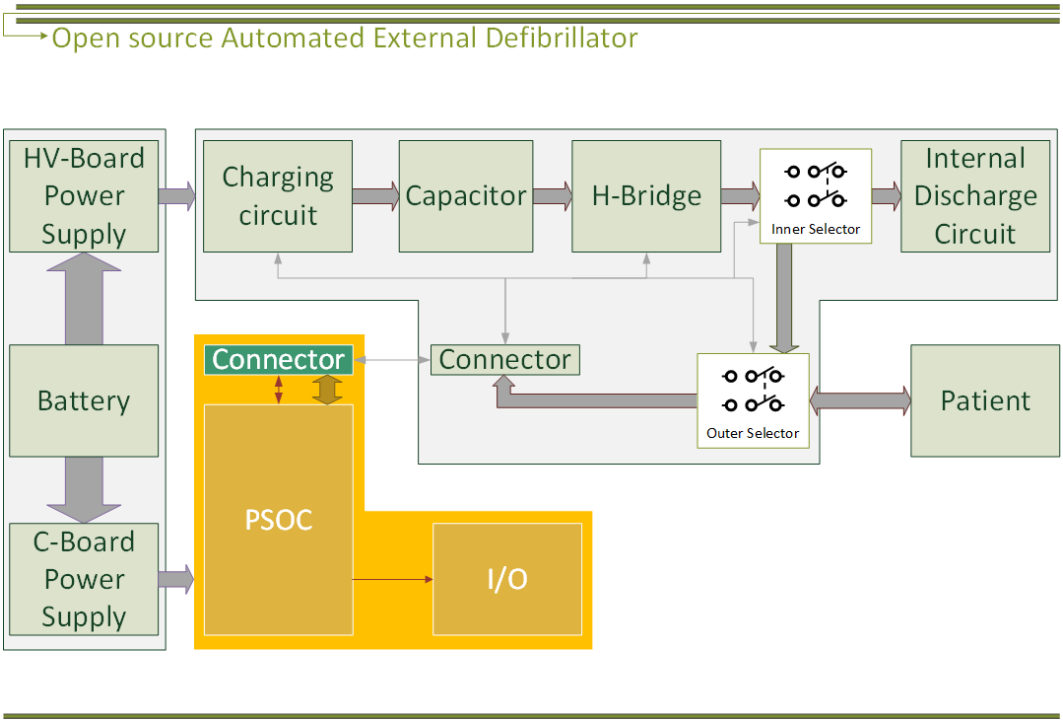


Figure 4.1: OAED Control Board block diagram

4.1 Overview

In reference with Figure 2.4, the Control Board (C-B) is the one highlighted in Figure 4.1. As we can see, it only contains three blocks, and one of those is the connector, which is just needed to remark the physical connection between the two boards.

The first of the other two block is the PSoC block. This owe its name to the Programmable System on Chip that represent the actual *brain* of the AED. The PSoC can be seen as a factotum microchip, as it integrates all the electronic devices the system needs to perform ECG and impedance acquisitions (and more).

The last block is called I/O, and it contains all the components needed to communicate with the users, such as buttons and led diodes.

4.1.1 PSoC 5LP

The PSoC is a one-chip solution, developed by Cypress [1], integrating analog front-end, digital logic and user interface integrated circuits, with an ARM Cortex-M3 CPU. Its exceptional versatility and flexibility allow to reduce the components count of the circuit, while also improving the overall performances due to the monolithic design of the chip. By its nature the PSoC can also be reconfigured even after it get soldered on the PCB (Printed Circuit Board), allowing the analog front-end rearrangement. Impossible with traditional electronic systems.

As already mentioned, the PSoC can be conceptually divided in three main parts. With reference to Figure 4.2, red blocks represent the Micro Controller Unit (MCU), while green highlights the Analog Block Array (ABA), and blue represent the Digital Block Array (DBA) [2].

Micro controller unit

The micro controller unit can be divided in turn in three different parts.

1) The CPU system integrates the Cortex-M3 CPU, the DMA controller, a cache controller and the interrupt controller.
2) In the memory system there are three different type of memory:SRAM, that the CPU use during operations; a FLASH memory, used to store the firmware; and an EEPROM, used to store small amount of data. In addition to those, if the memory is not enough, PSoC also integrate an External Memory InterFace (EMIF).
3) The program and debug block, as also the name suggest, provide an interface for programming the MCU.

The CPU can be programmed either directly in assembly [3], or in C using the PSoC IDE (described at the end of this chapter). Using the latter offer different obvious advantages, including an extensive library of custom-made functions made for the PSoC family.

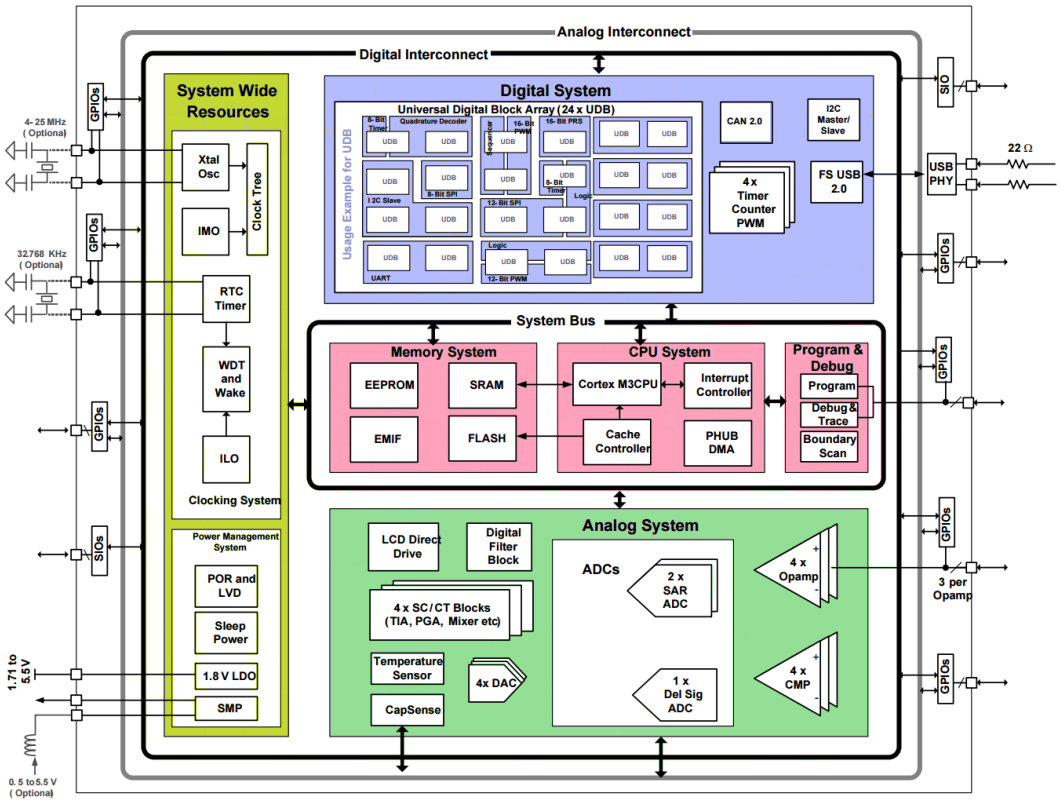


Figure 4.2: PSoC simplified block diagram

The programming interface implements different communication protocols, some of which also allow runtime debug, which can be extremely useful in certain situations. Especially when the system is not intended to communicate with the user in an verbose way, such as OAED.

Digital block array

The digital programmable system creates application specific combinations of both standard and advanced digital peripherals, and custom logic functions. These peripherals and logic are then interconnected to each other and to any pin on the device, providing a high level of design flexibility and intellectual protection security.

Even in this case PSoC Creator ease the design, providing a high level schematic capture graphical interface to automatically place and route resources similar to programmable logic devices (PLD). Therefore there is no need to interact directly with the programmable digital system at the hardware and register level.

The main components of the digital programmable system are:

- **Universal digital blocks (UDB) -** These form the core functionality of the digital programmable system. UDBs are a collection of uncommitted logic (PLD) and structural logic (Datapath) optimized to create all

common embedded peripherals and customized functionality that are application or design specific.

- **Universal digital block array -** UDB blocks are arrayed within a matrix of programmable interconnect. The UDB array structure is homogeneous and allows for flexible mapping of digital functions onto the array. The array supports extensive and flexible routing interconnects between UDBs and the digital system interconnect.

- **Digital system interconnect (DSI) -** Digital signals from UDBs, fixed function peripherals, I/O pins, interrupts, DMA, and other system core signals are attached to the DSI to implement full featured device connectivity. The DSI allows any digital function to any pin or other feature routability when used with the UDB array.

Some example of the digital components obtainable from the combination of various UDB are:

- logic ports such as OR, NOT, XOR;
- timers,
- PWM,
- UART,
- SPI,
- counters,
- etc.

Analog block array

The analog programmable system creates application specific combinations of both standard and advanced analog signal processing blocks. These blocks are then interconnected to each other and also to any pin on the device, providing a high level of design flexibility. In reference with the PSoC analog subsystem block diagram in Figure 4.3, ABA's features are:

- flexible, configurable analog routing architecture provided by analog globals, analog mux bus, and analog local buses;
- high resolution Delta-Sigma ADC;
- two successive approximation (SAR) ADCs;
- four 8-bit DACs that provide either voltage or current output;
- four comparators with optional connection to configurable LUT outputs;
- four configurable switched capacitor/continuos time (SC/CT) blocks for functions that include opamp, unity gain buffer, programmable gain amplifier, transimpedance amplifier, and mixer;
- four opamps for internal use and connection to GPIO that can be used as high current output buffer;
- CapSense subsystem to enable capacitive touch sensing;

- precision reference for generating an accurate analog voltage for internal analog blocks.

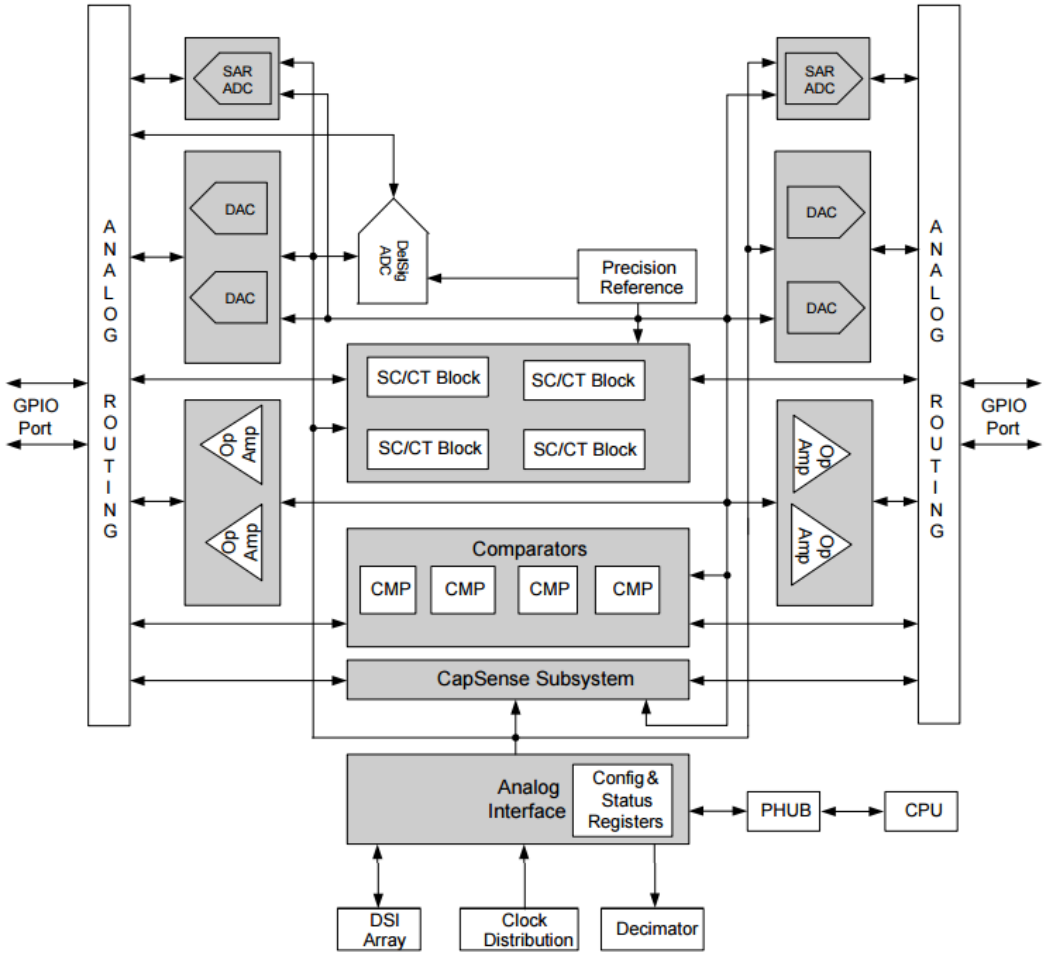


Figure 4.3: PSoC analog subsystem block diagram

The PSoC 5LP family of devices has a flexible analog routing architecture that provides the capability to connect GPIOs and different analog blocks, and also route signals between different analog blocks. One of the strong points of this flexible routing architecture is that it allows dynamic routing of input and output connections to the different analog blocks.

PSoC 5LP comprehensive features list

- Operating characteristics:
  - Voltage range: 1.71 to $5.5V$, up to six power domains;
  - Temperature range(ambient): -40 to $85°C$
    ectended temperature parts : -40 to $105°C$;
  - DC to $80MHz$ operation;
  - Power modes:

    - Active mode $3.1mA$ at $6MHz$, and $15.4mA$ at $48MHz$;
    - $2\mu A$ in sleep mode;
    - $300nA$ hibernate mode with RAM retention.
  - Boost regulator from $0.5V$ input to $5V$ output.
- Performance:
  - **32-bit ARM Cortex-M3 CPU, 32 interrupt inputs**;
  - **24-channel direct memory access (DMA) controller**;
  - **4-bit 64-tap fixed-point digital filter processor (DFB)**;
- Memories:
  - Up to 256 KB program flash, with cache and security features;
  - Up to 32 KB additional flash for error correcting code (ECC);
  - Up to **64 KB RAM**;
  - 2 KB EEPROM;
- Digital peripherals:
  - Four 16-bit timer, counter, and PWM (TCPWM) blocks;
  - I^2C, 1 Mbps bus speed;
  - **USB 2.0** certified Full-Speed (FS) 12 Mbps peripheral interface using internal oscillator;
  - **20 to 24 universal digital blocks (UDB)**, programmable to create any number of functions:
    - 8-, 16-, 24-, and 32-bit timers, counters, and PWMs;
    - I^2C, UART, SPI, I2S, LIN 2.0 interfaces;
    - Cyclic redundancy check (CRC);
    - Pseudo random sequence (PRS) generators;
    - Quadrature decoders;
    - Gate-level logic functions;
- Programmable clocking:
  - $3-$ to $74-MHz$ internal oscillator, 2% accuracy at $3MHz$;
  - $4-$ to $25-MHz$ external crystal oscillator;
  - Internal PLL clock generation up to $80MHz$;
  - Low-power internal oscillator at 1, 33, and $100kHz$;
  - $32.768-kHz$ external watch crystal oscillator;
  - 12 clock dividers routable to any peripheral or I/O;
- Analog peripherals:
  - **Configurable 8- to 20-bit delta-sigma ADC**;
  - Up to two 12-bit SAR ADC;
  - **Four 8-bit DAC**;
  - **Four comparators**;
  - **Four opamps**;
  - **Four programmable analog blocks**, to create:
    - Programmable gain amplifier (PGA);
    - Transimpedance amplifier (TIA);
    - Mixer;
    - Sample and hold circuit;
  - CapSense support, up to 62 sensors;
  - $1.024V \pm 1\%$ **internal voltage reference**;

- Versatile I/O system:
  - 46 to 72 **I/O pins** – up to 62 general-purpose I/Os (GPIOs);
  - Up to eight performance I/O (SIO) pins:
    - $25mA$ current sink;
    - Programmable input threshold and output high voltages;
    - Can act as a general-purpose comparator;
    - Hot swap capability and overvoltage tolerance;
  - **Two USBIO pins** that can be used as GPIOs;
  - Route any digital or analog peripheral to any GPIO;
  - LCD direct drive from any GPIO, up to 46 × 16 segments;
  - CapSense support from any GPIO;
  - $1.2V$ to $5.5V$ interface voltages, up to four power domains;
- Programming, debug, and trace:
  - **JTAG (4-wire)**, serial wire debug (SWD) (2-wire), single wire viewer (SWV), and Traceport (5-wire) interfaces;
  - ARM debug and trace modules embedded in the CPU core;
  - Bootloader programming through I^2C, SPI, UART, USB, and other interfaces;
- Package options:
  - 68-pin QFN,100-pin TQFP, and 99-pin CSP;
- Development support:
  - Schematic and firmware design support;
  - Over 100 PSoC Components integrate multiple ICs and system interfaces into one PSoC. Components are free embedded ICs represented by icons. Drag and drop component icons to design systems in PSoC Creator;
  - Includes free GCC compiler, supports Keil/ARM MDK compiler;
  - Supports device programming and debugging.

4.1.2 I/O block

As mentioned before in this block are included all OAED components of the user interface (UI). Because of the intended use of OAED, ease of usability should be at first place. However the device need a simple, non-distracting mean to communicate the operator whether the patient need defibrillation or not. For this purpose I have thought to a semaphore-like logic, implemented with three led diodes: blue, orange, and green. The colours meaning is reported in Table 4.1.

An additional yellow led is then used to warn the operator in case OAED detect that the electrodes are off.

Besides the led diodes, the I/O block only has a push-button. This is used by the operator to initiate defibrillation.

Table 4.1: OAED indicator lights colours and their meaning

Colour	Meaning
Blue	The device is working, but SCA has not been diagnosed yet for the patient.
Orange	SCA has been diagnosed, and the capacitor is charging; but the device is not ready yet.
Green	SCA is still diagnosed, and the capacitor is ready. Defibrillation has been enabled and the device is waiting for the operator to push the *defibrillation* button.
Yellow	Lead-off.

A quick design note about the colours At first I had decided to use the typical semaphore colour configuration: red, yellow, and green. However, EN 60601-1 asserts that certain colours are strictly bound to specific meanings (see Table 2.6 for more informations). Thereby I had to change them accordingly to the norm.

4.2 Electrocardiogram acquisition

In chapter 1 there was a brief general explanation about ECG. Now it is time to face the engineer point of view.

Since ECG is a signal coming from the patient's chest, it can be seen as the unreferenced difference of voltages taken from two arbitrary points. The conventional solution to read a floating voltage, is to use an instrumentation amplifier (in-amp). The two most common in-amp configurations are the so called three, and two op-amp in-amp. The first is showed in Figure 4.4.

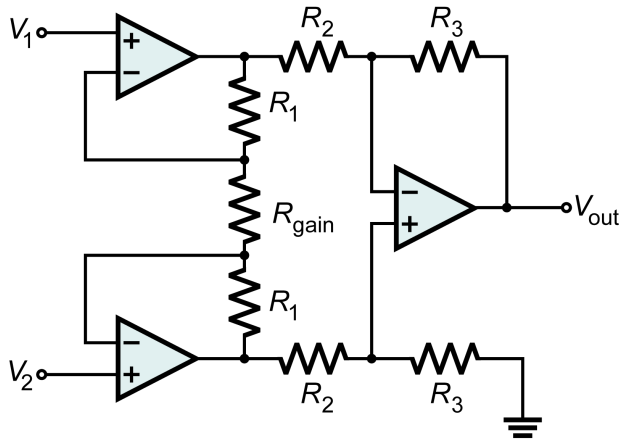


Figure 4.4: Example of a three op-amp instrumentation amplifier

In this circuit, the first stage, represented by the two symmetric op-amp on the left (and their resistors R_1, and R_{gain}), is a buffer with amplification.

Its purpose is to provide an high impedance input for the electrodes (here represented as the voltages V_1 and V_2), and to amplify the voltage difference ($V_d = V_1 - V_2$) while leaving unaltered the common mode ($V_{cm} = \frac{1}{2}(V_1 + V_2)$). The second stage, then, is just a differential amplifier.

The signal coming out of the in-amp, then, will ideally be only constituted by the differential signal V_d, whilst the common mode signal V_{cm} will be rejected. Furthermore it will be refereed to the common reference, and can be treated conventionally. That usually means it is sent to the Analog Digital Converter (ADC), possibly after passing through one or more analogical filter.

For OAED I exploited the particular configuration of the PSoC Delta-Sigma Analog Digital Converter ($\Delta\Sigma$-ADC) to obtain an in-amp-ADC all-in-one solution. In order to provide an high input impedance, in the PSoC $\Delta\Sigma$-ADC is included a buffer, as can be seen in Figure 4.5. This buffer is essentially the same that can be found in a three op-amp instrumentation amplifier, selectable gain included. And since the $\Delta\Sigma$-ADC can work in differential mode, it can act itself as the second stage[1].

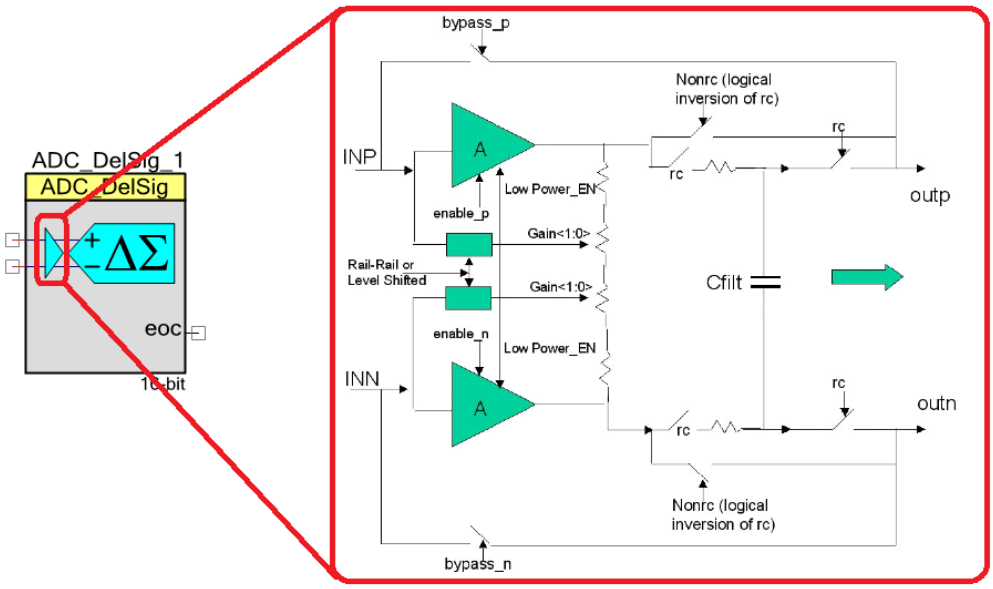


Figure 4.5: Block diagram of PSoC $\Delta\Sigma$-ADC buffer

Electrocardiogram amplitude can sensibly vary from person to person, especially when experiencing a Sudden Cardiac Arrest (or other heart related pathology). Nevertheless it is possible to narrow the values at a magnitude in the order of $\sim mV$.

The $\Delta\Sigma$-ADC input range in differential mode may be set to operate from $\pm 64mV$ ($-input \pm V_{ref}/16$) up to $\pm 6.144V$ ($-input \pm V_{ref} \cdot 6$). Because

[1]It also includes a differential anti-aliasing filter, represented by the capacitor C_{filt}

of ECG signal order of magnitude, I used the smaller values combined with a buffer gain of 2. In this way I could obtain an output digital signal with a reasonable range, and at the same time avoiding possible saturations[2]. In Figure 4.6 there is the configuration window, as it appears on PSoC creator, with all the actual values I used.

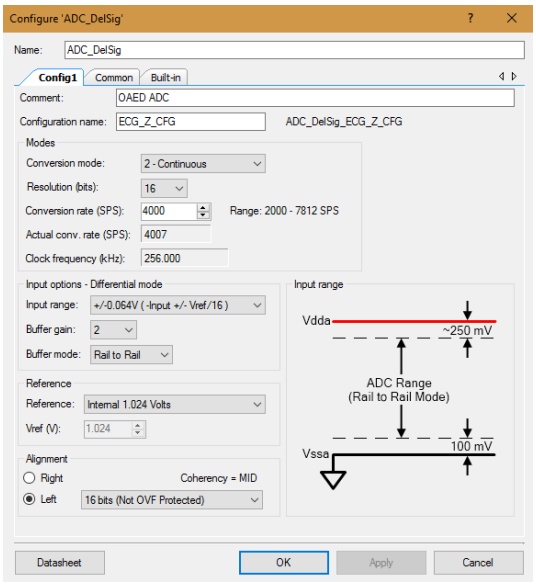


Figure 4.6: PSoC $\Delta\Sigma$-ADC configuration

The signal is then digitalized at a frequency of $4000sps$ by the $\Delta\Sigma$-ADC with a 16 bit resolution, which correspond to $\sim 0.977\mu V$ per bit $(64mV/2^{16})$[3]. Each of the 16 bit integer number in output is then taken at the same time by three different DMA (Direct Memory Access).

- **DMA_DelSig_ECG** is used to send data at the channel A of the Digital Filter Block (DFB).
- **DMA_DelSig_Z** is instead used to send data at channel B of the DFB.
- **DMA_DelSig_RAW** is used to take data directly in the system RAM (SRAM). This is only enabled during debug, in order to make possible the evaluation of $\Delta\Sigma$-ADC and DFB operations during the design phase.

In Figure 4.7 there is the actual configuration used in PSoC Creator in order to obtain what said above. On the left there are the physical pins connected to the electrodes, and on the right there is the Delta-Sigma Analog Digital Converter, with his three DMA. In addition to that, there is a block (on top) related to impedance measurement, which will be described later in this chapter.

[2]Assuming a negligible or nil common mode voltage V_{cm}, the differential input V_d (ECG signal) can be as high as $\pm 32mV$ before saturating the second stage.

[3]Considering also the buffer gain, each bit correspond to $\sim 0.488\mu V$ of the original signal.

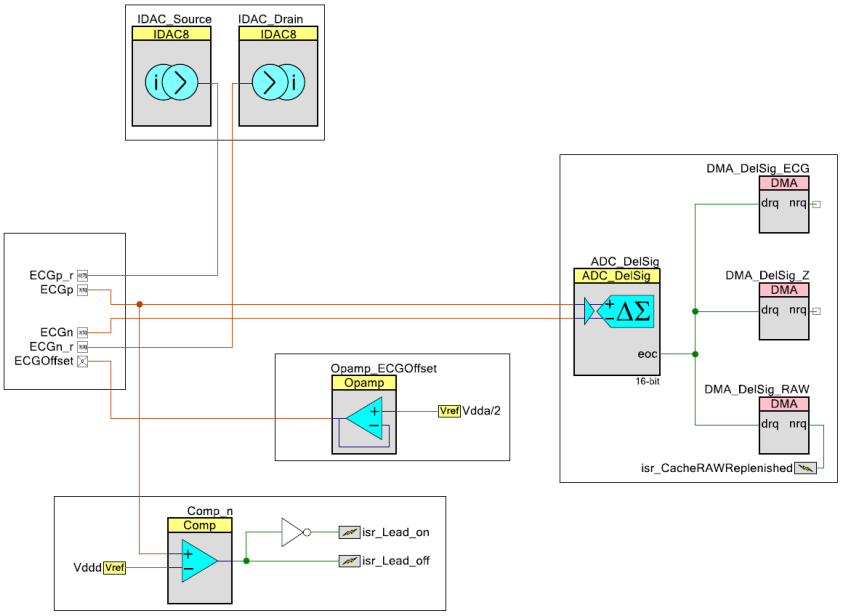


Figure 4.7: OAED's PSoC analog front-end configuration

4.2.1 Filter

The signal obtained from the $\Delta\Sigma$-ADC can be seen as a wide picture representing all electrical effect taking action on the patient chest area between the electrodes, second by second. For example: it includes electromyographic signals, which are similar to ECG in terms of amplitude; or signals due to the movement of electrodes, which appears as a slow variation of the mean value (effect known as baseline wander).

In addition to that, we must also keep in consideration that any conductor line can be seen as an antenna. Even though we can often assume negligible the effect of electromagnetic couplings, the ECG signal low order of magnitude combined with the differential amplification, ease the appearance of a great variety of noise. In fact, one of the worst possible source of noise comes from the electromagnetic coupling with the power mains. This kind of noise is particularly important especially because of its ubiquity. Power lines surrounds us everywhere, they are carried by pylons around the world, in the cities, and they ends inside the walls of every buildings. Fortunately, though, the power mains noise is a perfect sinusoid with a fixed frequency of $50Hz$ (in EU). Therefore we can easily eliminate that noise using an high-attenuation $50Hz$ notch filter.

An example of the $\Delta\Sigma$-ADC raw signal is in Figure 4.8. It appears evident, from this, that the signal obtained from the last section cannot be considered an actual ECG yet. Notice how there is an evident signal with a period of exactly $20ms$ ($50Hz$).

In order to design a filter, we must first know the ECG band of interest. Since the heart signal is a combination of various effects, each with different

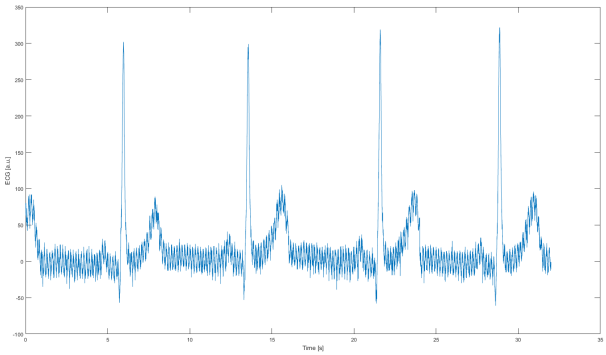


Figure 4.8: Raw ECG signal

frequency, we have to separate and evaluate the impact every contribution. Luckily, the ECG is one of the most extensively studied physiological signal, and the different frequencies are widely available [4]. In Table 4.2 there are some approximations for the more interesting ECG features.

Table 4.2: ECG intervals duration

Feature	Description	Duration	Frequency
P Wave	The p-wave represents depolarization of the atria.	$< 80ms$	$0.67 \sim 5Hz$
PR Interval	This interval reflects the time the electrical impulse takes to travel from the sinus node through the AV node.	120 to $200ms$	$5 \sim 10Hz$
QRS Complex	The QRS complex represents the rapid depolarization of the right and left ventricles.	80 to $100ms$	$10 \sim 25Hz$
ST Segment	ST segment represents the period when ventricles are depolarized.		$\sim 0.05Hz$
T Wave	The T wave represents the repolarization of the ventricles	$160ms$	$\sim 6Hz$

EN 60601-2-47 (Particular requirements for the basic safety and Essential Performance of ambulatory electrocardiographic systems) define some of the Essential Performance that are widely used in ECG equipment[4] [5]. Amongst those standard we can find some indication about the ECG filter bandwidth. These have led to the definitions of two different modes of operation: diagnostic mode, and monitor mode.

In diagnostic mode, the ECG filter is more gentle, so as not to compromise the accuracy of ST segments. On the downside this means a more noisy signal, and an increased baseline wander effect. In monitor mode, instead, the filter is more strict and only let pass a narrow band. This is traduces in a cleaner ECG signal, but also in the gradual disappearance of the ST segment as the band get reduced. As a mean of comparison in diagnostic mode the filter is usually set in such a way to let pass a band between 0.05 and $150Hz$; while in monitor mode the band is usually between 0.67 and $40Hz$.

In our case though, we just need to discriminate heart rhythms containing VT or VF (see chapter 1). The ECG signal in the first case is characterized

[4]Albeit this norm is very useful to get a reference on the Essential Performance of a typical ECG, we don't need to strictly follow it. In fact, there is no indication at all of Essential Requirements or Essential Performances for defibrillators ECG in 60601-2-4.

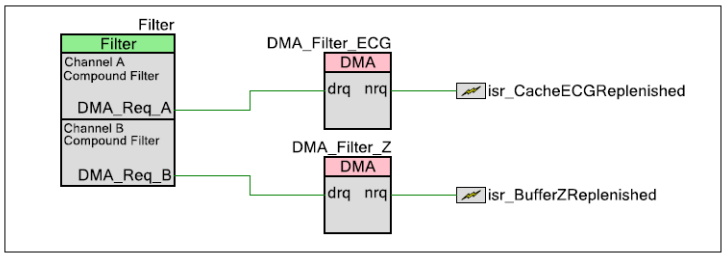
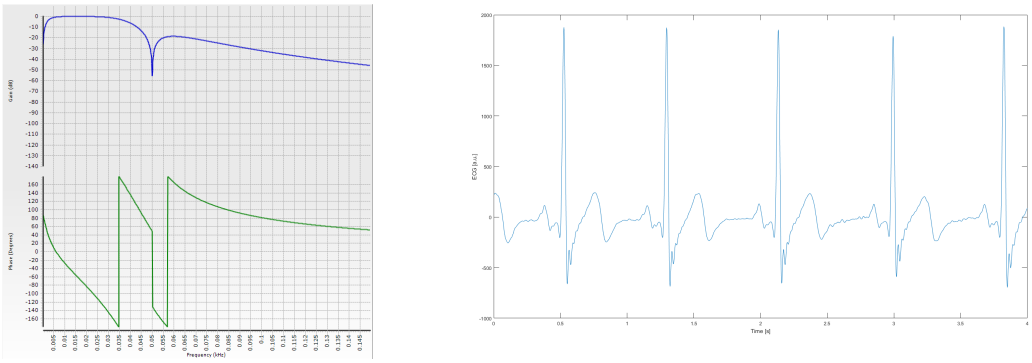


Figure 4.9: OAED's PSoC digital filter configuration

by a massive presence of QRS complexes. In fact, those are usually the only discernible morphological features. On the other hand, during VF there is no recognizable morphological feature at all. Thereby we basically only need to leave unaltered the QRS complexes. In this way, then, we can afford to use a very strict filter, namely with a band in the range between 3 and $40Hz$. The lower limit ensure the almost total absence of baseline wandering in the final signal, which is ideal and greatly ease pattern recognition. The higher limit, instead, is the minimum required to ensure the QRS complexes preservation.

In Figure 4.9 there is the PSoC Creator design used for the OAED's filter. In Figure 4.10 (a), instead, there is the Bode diagram for the biquad IIR filter I used to obtain the ECG signal (DFB Channel A). And finally, in Figure 4.10 (b), there is an example of an ECG signal obtained using OAED's filter.

In the end, DFB output is taken again by a DMA and moved into the SRAM. Here a software-based decimation take act in order to reduce the memory usage by decreasing the sampling rate from $4000sps$ to $500sps$. The signal in any case won't contain any useful information in the frequency range lost for the decimation because of the filtering done before.



(a) (b)

Figure 4.10: (a) ECG filter Bode diagram; (b) Filtered ECG signal

4.3 Impedance measurement

In chapter 3 we highlighted the importance of knowing the patient impedance. In fact, without that information we would have no idea on how much time the defibrillation should lasts in order to deliver the required energy[5].

The easiest way to perform impedance measurements is by injecting a known current in the patient, and evaluate the voltage it produces across the electrodes. There are two philosophies for this, and they differ on whether the injected current is in AC or DC [6]. An AED only has one couple of electrodes, therefore the impedance measurements has to share the same path of the ECG signal. And we must insure the compatibility between the two.

4.3.1 DC impedance measurements

As we mentioned, the idea at the base of impedance measurement is always the same: we inject a known excitation signal and we monitor the changes. In this case the injected excitation signal is a DC current of small amplitude. With reference to Figure 4.11, in order to provide the current a return path the implementation requires two current generators: one acting as a source (positive current), and the other one as a sink (negative current).

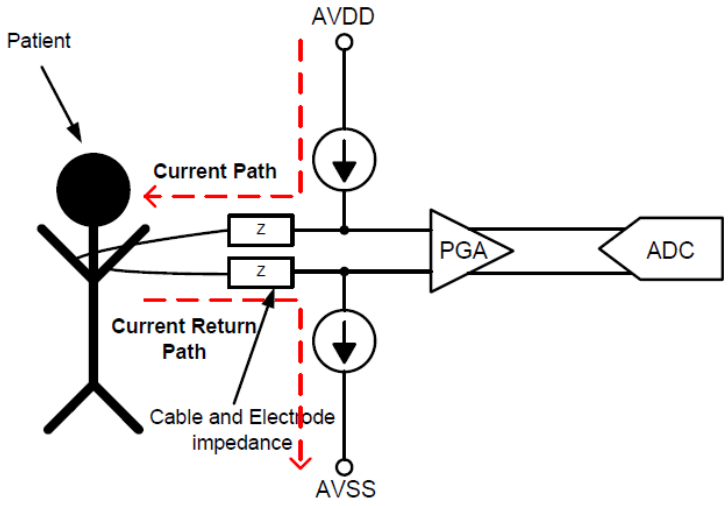


Figure 4.11: Impedance measurements configuration

When the electrodes are correctly placed on the patient chest, the current produces a small voltage which can be read with the $\Delta\Sigma$-ADC. This voltage

[5]Well, for completeness, there is another impractical way. If we had a precise and prompt measurement on the voltage across the capacitor, we would be able to tell how much energy has been released using the relation $U(t) = \frac{V_0 - V(t)^2}{2C}$. But since this involves real time measurement it will be much less safe and accurate.

is in DC, therefore a simple low-pass filter with a very low cut-frequency will provide us a steady signal representing the voltage produced by the injected current. Applying the Ohm's law, then, it is possible to evaluate the impedance. This is the combined impedance of patient, electrodes, and internal paths inside the PSoC. Therefore, knowing the last two[6] we can easily calculate the patient impedance.

Because of the nature of this method, we can configure two PSoC comparators to trigger an interrupt when the input signal exceeds certain values. In this way we can also obtain a software-free lead-off detection. Which is a great advantage in terms of response time, and power consumption. However DC methods involves the introduction of an offset in the $\Delta\Sigma$-ADC, which has to be evaluated in order to avoid possible saturations.

In OAED i used a $125nA$ current. This value should ensure to avoid saturations, while still offering an acceptable resolution. The filter in this case was not an issue. I used a second order low pass, with a cut frequency of $3Hz$; and a second order band-stop filter with a central frequency of $50Hz$, as shown in Figure 4.12. By default the DC method is the one enabled.

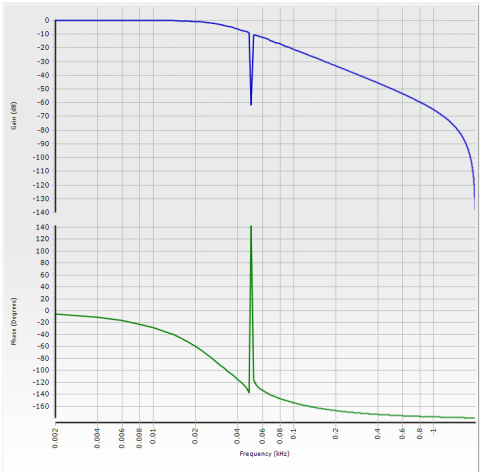


Figure 4.12: DC impedance filter Bode diagram

4.3.2 AC impedance measurements

Even for AC methods Figure 4.11 apply. In this case, though, the two constant current source and sink are substituted with two AC current generators, phase-shifted by $90°$. This will produce an AC voltage across the electrodes having the same exact frequency of the excitation signal. If the frequency is chosen high enough (not less than $200Hz$), then this signal won't interfere with the ECG. The $\Delta\Sigma$-ADC output is then filtered with an high-pass filter with a cut-off frequency high enough to eliminate the ECG and the power mains signals. Finally, by evaluating the resulting signal's

[6]Electrodes impedance are reported in their datasheet, while PSoC internal path's impedance can be determined using PSoC Creator.

peak-to-peak values using Ohm's law, it is possible to obtain the impedance value. Even in this case the results contain also the electrodes and PSoC internal path impedances.

In OAED I used an AC impedance approach, with a $6\mu A$, $250Hz$ square wave, and a narrow biquad bandpass IIR filter (see Figure 4.13). The configurations can be seen in Figure 4.7 and Figure 4.9 (where Z is the abbreviation I used for impedance). In this case, the decimation post filtering was not applied because of the potential loss of information.

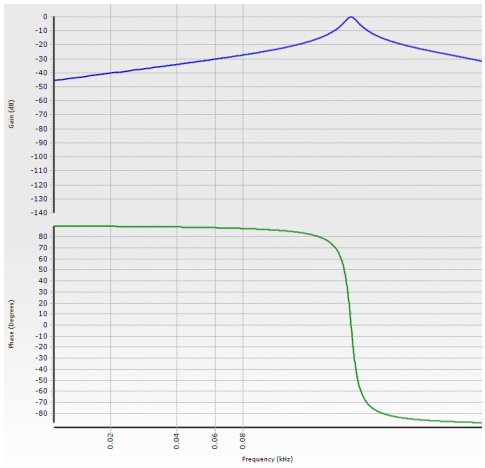


Figure 4.13: AC impedance filter Bode diagram

4.4 OAED PSoC design

The last section of this chapter collects all the remaining blocks and configurations of OAED's PSoC design. These include topics too important to be omitted, but not enough to justify a dedicated section. Starting with the remaining blocks of the design, these are represented in Figure 4.14 and are hereinafter briefly explained.

- The first one is a redundant control for the H-Bridge circuit. The green block on the left is a software-set register, updated every time the firmware wants to control the H-Bridge (see chapter 5 for more information). The output are then evaluated with some logic blocks in order to provide a degree of redundancy -as explained in chapter 3 we must ensure the two phase signal are never high together. And finally the red block represent the connection to physical pins.

- The second block on the left represent the physical pins used to control the inner, and outer selectors (see chapter 3).

- The "Charge_En" is the pin used to enable the charging circuit.

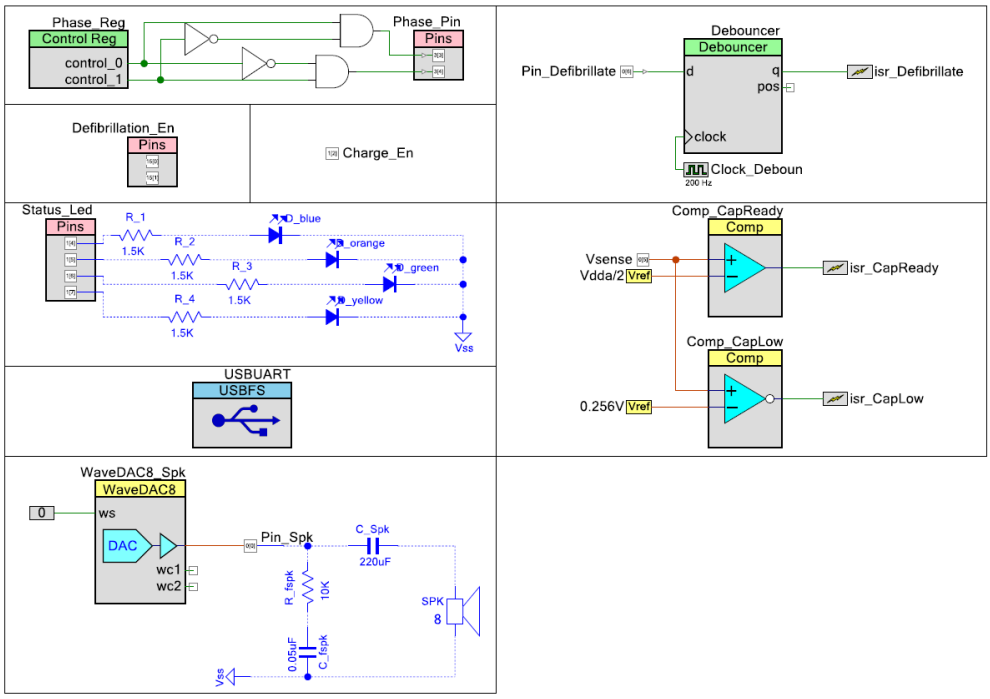


Figure 4.14: OAED's PSoC miscellaneous blocks

- The first block on the right represent the physical pin to be connected at the push-button that the operator should press to release the defibrillation. Due to its criticality I used a de-bouncer (green block) that directly call an interrupt (here represented as a lightening).

- The third block on the left represents the pins connected to the led diodes. In order to provide a more understandable view, I also added the led to the design. These, as the resistors, are not actually inside the PSoC, but on the C-B PCB.

- The penultimate block on the left is required for to implement USB UART protocols.

- The last block on the right is used to evaluate the capacitor voltage. "Vsense" represent the pin attached to the charging circuit buffer output (V_{sense}), seen in chapter 3. This pin is connected to two comparators. When its value arrive to $2.5V$, the first comparator call an interrupt to communicate the MCU that the capacitor is ready; whilst, when its value falls below $256mV$, the second comparator call another interrupt, used to communicate to the MCU that the capacitor's voltage is below 10% of its nominal value V_0.

- Finally, the last block on the left is the DAC which controls the speaker. The speaker is used to warn bystanders that a defibrillation is about to be released.

The purpose and usage of most of these blocks will be discussed extensively on chapter 5, but for consistency I wanted to include them here in this chapter.

Besides the last part of the PSoC internal design, a special mention should be left to the pin-out and resources management.

4.4.1 Pin-out

In Figure 4.15 there is the OAED's PSoC pin-out as it appears in PSoC Creator.

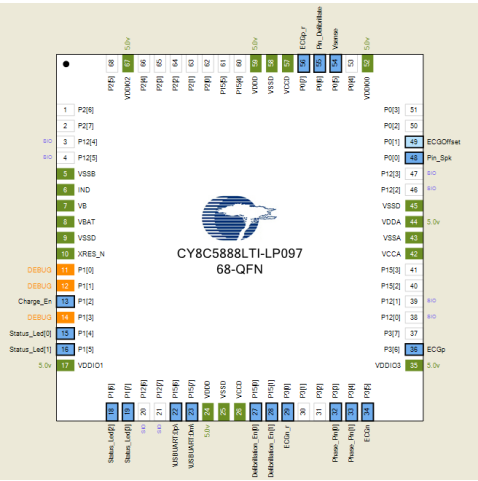


Figure 4.15: OAED's PSoC pin-out

Most of the pins are left free to be automatically assigned by the PSoC Creator auto-routing algorithm. Some of these are also specifically bound to some functionalities -i.e. the USB pins.

Certain pins instead, I had to set them myself in order to obtain a particular design. I.g. the pins in which the excitation current necessary for impedance measurement must be as close as possible to the ECG pins.

4.4.2 Resources

Although the PSoC has great capabilities and excellent versatility, it is still just an embedded system. And this means that in order to remain within the limits of feasibility, intended as physical space inside the chip, energy required, heat dissipated, costs, complexity, etc; the PSoC can only have a certain number or resources.

For OAED I chose to use the PSoC "CY8C5888AXI-LP096". Its specifications can be resumed in the following list.

- Package: 100 pin TQFP
- Maximum frequency: 80 MHz;
- Flash memory: 256 KB;
- SRAM: 64 KB;
- EEPROM: 2 KB;
- Total I/O pins: 72;
- LCD segment drive;
- Digital Filter Block;
- Full speed USB;
- 1x $20b$ Delta-Sigma Analog Digital Converter;
- 2x $12b$ SAR ADC;
- 4x DAC;
- 4x Comparators;
- 4x SC/CT analog block;
- 4x op-amp;
- 24x Universal Digital Blocks;
- 4x $16b$ Timer/PWM;

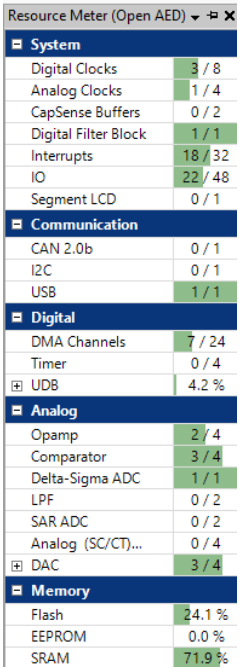


Figure 4.16: OAED's PSoC resources usage

All tests and design operations considered during the thesis though, are done on the "CY8CKIT-059" development kit, which mount a slightly different version of the chip [7]. In Figure 4.16 there is the OAED's PSoC resources usage, as it appears on PSoC Creator.

Battery

Now that we have seen both the HV-B and the C-B, it is possible to make an estimation on the battery. Since the RCC will work with an input voltage of $18V$, it is convenient to chose a battery with a nominal voltage close to that. Thereby, we can for example chose a battery with a nominal voltage of $18V$ and a capacity of $4Ah$ -this kind of batteries are not so uncommon in RC applications. Considering that the RCC consumes an average of $5A$ for 6 seconds every time it needs to charge the capacitor, the battery can sustain around 480 cycles. However, in addition to that we need to subtract the energy required to the PSoC to work -considering negligible the power consumption of the H-bridge, and the selectors. The basic operative current at a frequency of $6MHz$ is $15.4mA$, with a nominal voltage of $5V$. Furthermore, there is a $2.5mA$ current for the $\Delta\Sigma$-ADC, $100mA$ for a Led-diode, and $50mA$ due to other components. For a grand total of $\sim 170mA$. This is significantly lower than the RCC, however unlike the latter, the C-B power consumption continues during the whole operation of OAED. This means that roughly every 635 seconds (10.5 minutes) of OAED operations, the battery lose an equivalent energy of a complete capacitor charge.

4.5 Software used

As we have seen, the Control Board undisputed star is the PSoC. It is obvious then that most of the work done here has been achieved using the PSoC development environment -PSoC Creator. However, PSoC Creator is just a mean to pursuit a existing project. And during the whole signal processing design, the contribution of Matlab was vital.

4.5.1 PSoC Creator

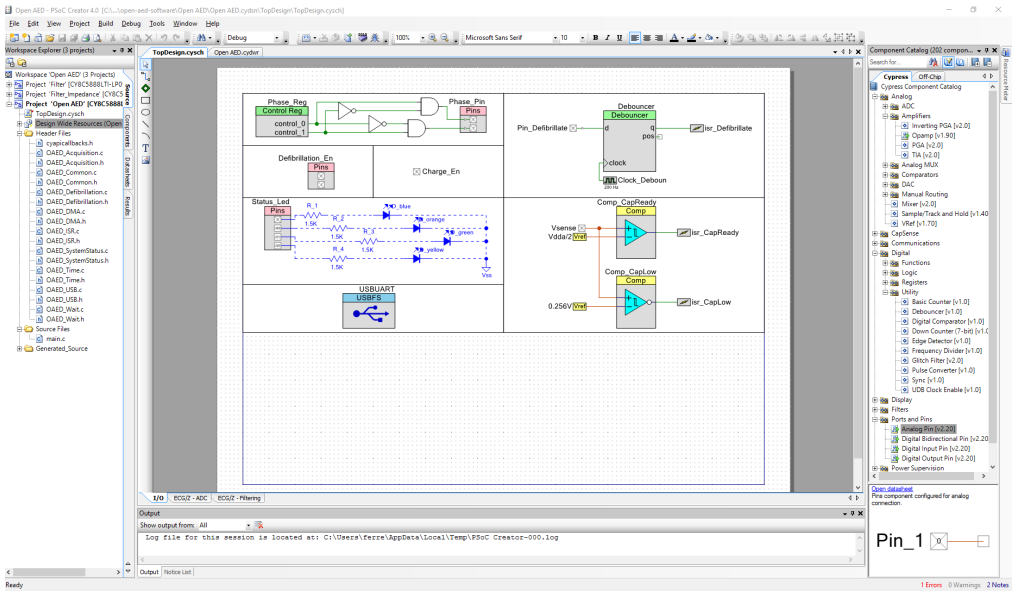
PSoC Creator is an Integrated Design Environment (IDE) that enables concurrent hardware and firmware editing, compiling and debugging of PSoC 3, PSoC 4, PSoC 4 BLE, PSoC BLE, PSoC 5LP and FM0+ systems with no code size limitations. PSoC and FM0+ peripherals are designed using schematic capture and simple graphical user interface (GUI) with over 120 pre-verified, production-ready PSoC and FM0+ Components™ [8].

PSoC and FM0+ Components are analog and digital "virtual chips," represented by an icon that users can drag-and-drop into a design and configure to suit a broad array of application requirements. Each component in the rich mixed-signal Cypress Component Catalog is configured with a Component Customizer and includes a full set of dynamically generated API libraries. Once the PSoC system has been configured, firmware can be written, compiled, and debugged within PSoC Creator or exported to top 3rd party IDEs from IAR, Keil, and Eclipse.

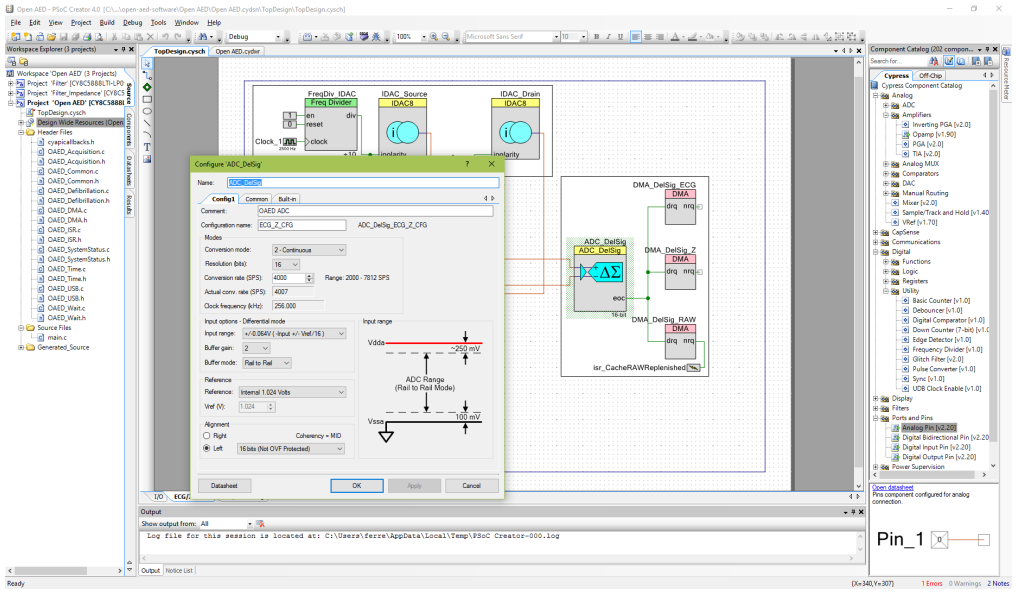
PSoC and FM0+ Systems are energy optimized beyond a typical MCU because PSoC Creator optimizes designs using only the required functionality. Users can create custom PSoC or FM0+ Components using state machine diagrams or Verilog within PSoC Creator to further optimize hardware and energy usage.

PSoC Creator is a free Windows-based IDE that includes:

- Hardware design with complete schematic capture
- Over 120 pre-verified, production-ready Components
- Full communications library including I^2C, USB, UART, SPI, and Bluetooth Low Energy
- Tools to develop custom components in Verilog or via state machine diagram
- Compatibility with the Peripheral Driver Library
- Dynamically generated API libraries
- Integrated C source code compiler and editor
- Built-in debugger



(a)



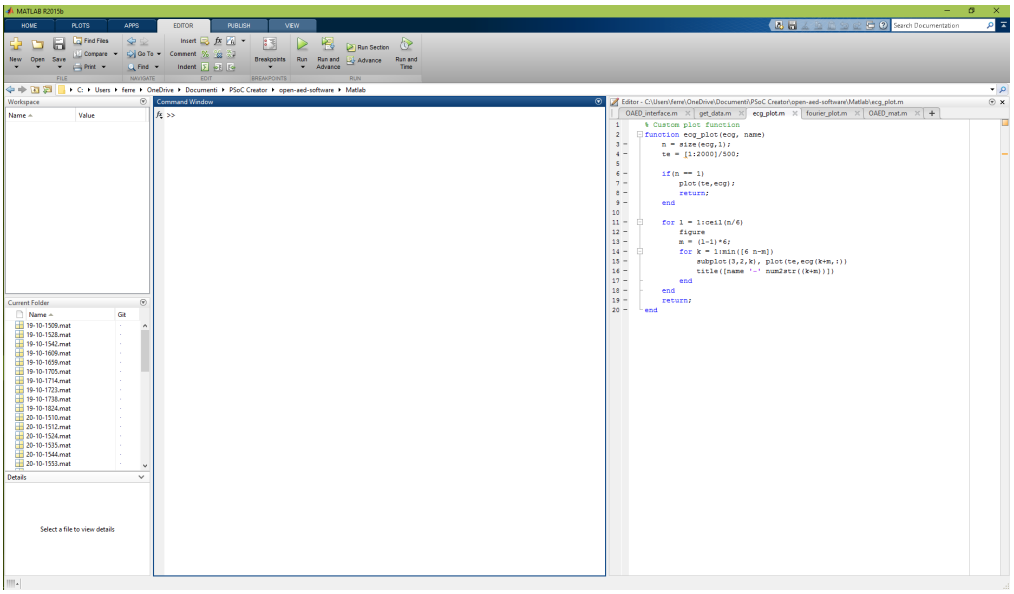
(b)

Figure 4.17: Examples of PSoC Creator interface

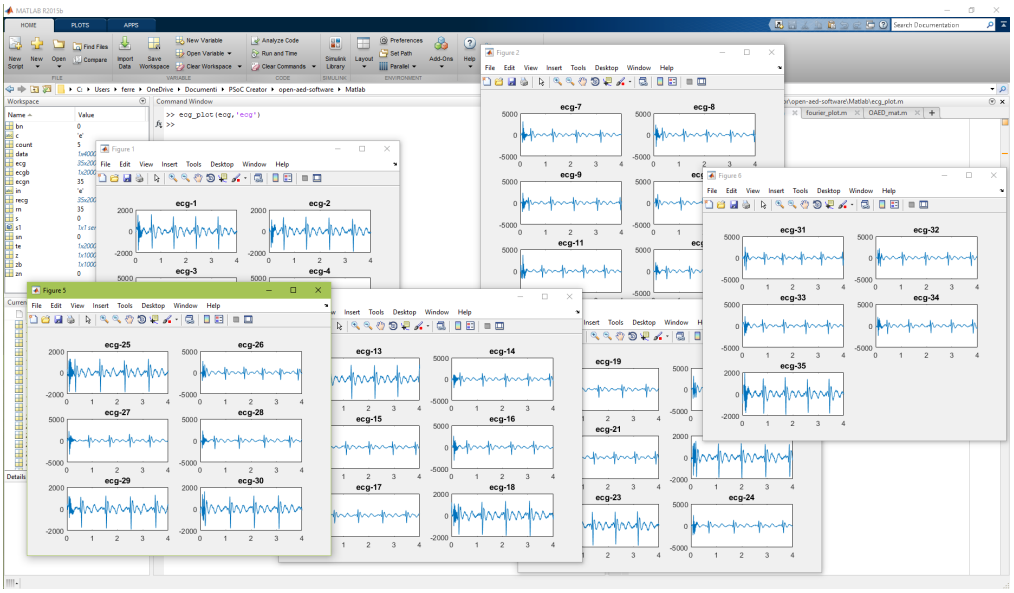
4.5.2 Matlab

MATLAB (MATrix LABoratory) is a multi-platform proprietary software developed by Mathworks [9]. Matlab include an integrated development environment and a proprietary programming language specifically designed to ease matrix manipulations. Because of its features, and it powerful user interface, Matlab is particularly suitable for all applications involving signal and/or image analysis -or more in general, for all the applications in which is possible to represent data as a matrix.

The later versions of Matlab also allow the creation of user interfaces, and interfacing with programmable development boards -such as Arduino-, or programs written in other languages, including C, C++, C#, Java, Fortran and Python.



(a)



(b)

Figure 4.18: Examples of Matlab interface

Bibliography

[1] *Cypress website*. http://www.cypress.com/.

[2] *PSoC® 5LP: CY8C58LP Family Datasheet: Programmable System-on-Chip (PSoC®)*. http://www.cypress.com/documentation/datasheets/psoc-5lp-cy8c58lp-family-datasheet-programmable-system-chip-psoc?source=search&cat=technical_documents.

[3] *Arm - Developers resources*. https://developer.arm.com/.

[4] Wikipedia. *Electrocardiography*. https://en.wikipedia.org/wiki/Electrocardiography.

[5] European Committee for Electrotechnical Standardization. *Medical electrical equipment*. EN 60601-1. 2007.

[6] Texas Instruments. *Understanding Lead-Off Detection in ECG*. http://www.ti.com/lit/an/sbaa196a/sbaa196a.pdf.

[7] Cypress. *CY8CKIT-059 PSoC® 5LP Prototyping Kit With Onboard Programmer and Debugger*. http://www.cypress.com/documentation/development-kitsboards/cy8ckit-059-psoc-5lp-prototyping-kit-onboard-programmer-and.

[8] Cypress. *PSoC® Creator™ Integrated Design Environment (IDE)*. http://www.cypress.com/products/psoc-creator-integrated-design-environment-ide.

[9] Mathworks. *Matlab*. https://uk.mathworks.com/products/matlab.html.

Part III

The Software

CHAPTER 5

Firmware

In the last two chapters we described in detail the hardware configuration of OAED. We metaphorically said that the High-Voltage Board acts as the device *muscle*, while the Control Board acts as the *brain*. If we want to continue with this metaphor, now it is time to fill the brain with a *mind*.

In chapter 3 we saw how the High-Voltage Board stores the energy required for defibrillation, and how and why this is performed as a biphasic wave. Then in chapter 4, we took a closer look to the Control Board, with particular regards on the PSoC and its internal configuration. We described how to obtain a digital signal, from the electrodes, representing the patient ECG; and another signal from which we can extract his/her chest impedance. But all these are just purposeless means, unless we provide them also a control logic.

In this chapter, we are going to see the device final part: the firmware.

5.1 Overview

The firmware is here intended as the C program loaded in the PSoC memory in order to control the device. Unlike computers, where it is possible to run more programs at the same time, the limited computing power of the PSoC -and embedded systems in general- only make possible to run a single program at a time. Therefore, the firmware has to fulfil itself all the functions necessary to control OAED. This means that the firmware is responsible for:

- resources management;
- user interaction;
- patient impedance evaluation;
- ECG data analysis;
- assessment of the necessity for defibrillation; and
- performing defibrillation by operating the HV-B.

The challenges to overcome when designing the firmware for a device so critical such an AED are various. First of all, we must ensure safety for both patient and operator in every situation. Stability and robustness are therefore two maximum priorities, but we also have to take in account what happens if the system stops working properly, and be sure that it will not end up producing any hazardous situations. Second, since we are programming an embedded system, we have to deal with very limited amount of resources. And to make things worse, these have to be shared between all the functionalities necessary to properly operate the device. This means that we have to carefully ration the use of both time and memory (intended as SRAM and flash). Furthermore we have to optimize the code and *break* the procedural paradigm, peculiar of the C language, to insert some foundations of event-driven paradigm.

Finally, all these functionalities have to be performed without the operative system's safety net and convenience functions, normally available when writing a conventional computer program.

A more specific necessity that significantly affected the OAED firmware writing, is the open-source approach. As mentioned in chapter 1, the open-source approach in this thesis is seen as a opportunity to teach about the device, and to allow its improvement. For these reasons it is imperative that the firmware is written in an ordered way, with high modularity.

A note about the nomenclature used

In order to avoid possible confusion, and to ease the firmware reading, I have decided to use a name-based code. Here are reported the basic rules:

- all headers, functions, and macros identifiers shall start with the "OAED_" prefix;
- numeric constants are indicated with capital identifiers;
- numeric constants with composite names use "_" as separators, i.e. "ADC_BUFFER_GAIN";
- flags are indicated with lower-case identifiers, separated by "_", i.e. "capacitor_ready";
- variables are indicated with capitalised identifiers without separations, i.e. "PatientImpedance";
- functions are also indicated with capitalised identifiers without separations, i.e. "void OAED_Init()";
- macros are indicated with capital identifier separated with "_", i.e. "OAED_PIN_CONTROL";
- impedance is indicated with the capital letter Z;
- Electrocardiogram is indicated with its acronym ECG.

There are however few exceptions to the above rules. In order to ease reading, ECG and Z are always capitalized, no matter where they are. In this way, data flow is evident at first sight. Another exception concerns the

PSoC related variables. Most of them are automatically declared from PSoC Creator to resemble their equivalent block name. Thus, for consistency I decided to leave their identifiers as they are defined.

5.1.1 Modularity

The whole idea behind OAED's design, is to provide the basic blocks necessary to build a functional AED. By rearranging these blocks then, it is possible to obtain different behaviours, based on specific necessities. Another possibility is to customize it by adding new blocks, or by replacing some.

It appears evident then, that modularity is a fundamental aspect for OAED's firmware -more than for the other components. A pretty obvious example, is the possibility to change the pattern's recognition algorithms. For instance they could be different based on the target populations, or to improve sensibility rather than specificity.

Modularity in OAED's firmware is achieved using the flexibility of C programming language. The basic blocks of the firmware are represented by the various functions. Each of which perform simple actions, such as: move data from an array to another, or check a flag. By combining simple functions we can then obtain more complex behaviours, such as the *measurement mode*. OAED's functions are grouped on libraries, based on their goals.

Another aspect to cover is the possibility to change some of OAED's hardware, and how this will affect the firmware. For instance, if we want to build an AED with a smaller capacitor -and because of the hardware conformation, this is possible even without changing the PCBs- it is obvious that we also need to change the algorithm that evaluate the discharge timing. Thereby another degree of customization is offered in the form of *parameters*. In OAED's firmware parameters are the numeric constants at the beginning of every header. We can change those to reflect changes in the hardware configuration, or to tune some functionalities of the PSoC.

Libraries

As mentioned before, modularity is achieved with functions grouped in libraries. Each library is in its turn contained in two different files: a ".h" file, which is the header containing variables and function's prototypes definitions, and all the related parameters; and a ".c" file, which contains function's bodies and variables declarations.

Here is reported the current list of the available headers with a brief description.

– **OAED_Acquisition.h**

In this header there are all the functions that control the signal acquisition chain. Namely the impedance and ECG signals.

- **OAED_Algorithms.h**
  This library contains the SCA recognition algorithms, and the SCA decisions rules.

- **OAED_Common.h**
  This is the most important header. It contains all the hardware, $\Delta\Sigma$-ADC, buffers, caches, signal frequencies, and SCA recognitions related parameters. Moreover it contains most of the OAED's global variables, flags, and some common functions definitions.

- **OAED_Defibrillation.h**
  This header contain the defibrillation related functions, including the one that trigger internal discharge.

- **OAED_DMA.h**
  In this header there is the function that initializes all the DMAs. In addition, there are also the functions used to pause/un-pause DMA transfer.

- **OAED_ISR.h**
  This header contain the Interrupt Service Routine (ISR) (see section 5.1.3 for more informations).

- **OAED_SystemStatus.h**
  This is the header that contain the machine-state implementation of OAED. Replacing or hacking this header can lead to completely different behaviours.

- **OAED_Time.h**
  This header contain some functions to add a time reference. Only used for debug purposes, it can be disabled changing the "OAED_TIME" constant (contained in OAED_common.h) to *false*.

- **OAED_USB.h**
  This header is necessary in order to use USB functionalities. These are mandatory in order to communicate with the PC, and are only intended for debug or teaching purposes. In fact the USB connector in the final design shall be covered, and made inaccessible.

- **OAED_Wait.h**
  In this header there are all the idling functions. These are required when the device is waiting for a certain event to trigger. For instance, when waiting for new data to analyse.

Including "OAED_Common.h" in the main is sufficient to include all OAED's headers.

Parameters

OAED's parameters are numeric constant used mainly to represent the device characteristics inside the firmware. Another use is to customize some aspect, and behaviours of OAED. Parameters are declared in firmware using the C directive "#define", and the most important are here reported.

- **ADC_BUFFER_GAIN** represents the $\Delta\Sigma$-ADC buffer gain (see section 4.2 for more information)
- **ECG_CACHE_SIZE** indicate the ECG cache size, set by default at eight.
- **ECG_SIGNAL_LENGTH** define the ECG window length, by default is set at four seconds.
- **ECG_SAMPLING_F** represent the ECG sampling frequency after the decimation. Default value is $500sps$.
- **Z_SIGNAL_LENGTH and Z_SAMPLING_F** are the impedance equivalent of the last two. Default setting is one second for the first, and $4ksps$ for the second.
- **Z_SIGNAL_F** indicate the impedance excitation frequency. Default is $250Hz$.
- **Z_MIN and Z_MAX** represent respectively the minimum and maximum impedance value compatible with the human model (see section 3.1.2).
- **EVENT_NO** represent the number of event to take in consideration when evaluating SCA.
- **POSITIVE_EVENT_NO** is the number of positive event required to confirm SCA.

5.1.2 Direct Memory Access

In the introduction of this section we highlighted the main limits of writing a firmware. One of these is the necessity of sharing the resources between all functionalities. There are however, some optimizations we can implement in order to lighten the CPU load, and one of these is the Direct Memory Access (DMA).

By definition, DMA is a feature that allows certain hardware subsystems to access, and manipulate various memories (RAM, flash, etc), independent of the Central Processing Unit (CPU).

Ideally, we would prefer that OAED's CPU spends the whole time performing important tasks, rather than simple ones. And this is why the DMA is so relevant.

The DMA Controller (DMAC) in PSoC can transfer data between memory and on-chip peripherals including **ADCs**, DACs, **DFB**, USB, UART, and SPI. There are 24 independent DMA channels available, all of them capable of operating with *no necessity of CPU intervention*. This allows the CPU to handle other tasks while the DMA does data transfers, thereby achieving a "multiprocessing" environment.

DMAs are particularly important for OAED because the necessity of contemporaneously handling two different data stream: the ECG, and impedance signals. This means that there is a DMA for each signal that moves data from the $\Delta\Sigma$-ADC output to the DFB. Then, there are two more DMAs taking data from DFB's output to the SRAM.

There is also a special DMA (DMA_DelSig_RAW) that moves $\Delta\Sigma$-ADC output data directly to the SRAM. This is only used during debug, and it can be disabled at a code-level -namely the code required to initiate and enable it, will not be compiled and included in non-debug firmwares.

5.1.3 Interrupts

Interrupts are asynchronous signals, emitted by hardware or software, to communicate the CPU an event that needs immediate attention. As result the CPU immediately interrupts the current code it is executing, save its state, and call a special function called Interrupt Service Routine (ISR). This interruption is only temporary, and, after the Interrupt Service Routine finishes, the CPU resumes its normal activity.

Generally speaking there are two types of interrupts: hardware interrupts and software interrupts. In embedded systems like OAED the latter are more rare, mainly because they require unjustified complications, but nonetheless they are possible.

In other words, interrupts represent the possibility to extend the standard C procedural paradigm, to an event-driven paradigm. We can assign interrupts to certain events in order to alter the CPU work-flow. For instance we can design the PSoC to trigger an interrupt when the voltage V_{sense} reaches $2.5V$, in order to communicate the CPU that the capacitor is ready (see section 4.4).

Interrupts are a very powerful mean, and should be treated such as, with all the due precautions. Since by nature interrupts disrupt the linear work-flow of the CPU, we should take in account all their stability implications on the system. For this reason ISRs should only execute few very simple line of codes, in best case scenario we want them just to change a status flag. Another concern is the risk of bloating a function execution time because the CPU is bombed by interrupts, which prevent it to complete its original work. Hence, we also want to make sure interrupts are relatively rare events, only triggered when actually necessary. To resume the instance of

capacitor ready interrupt, we want its ISR to be executed only once: when the capacitor becomes ready. Therefore, in addition to setting the "capacitor_ready" system flag to $true$, the ISR also disable itself. The ISR will be re-enabled then, by another interrupt triggered when the capacitor voltage is low (see section 4.4).

5.1.4 USB and debug mode

Having no mean of communication, besides the led, OAED's debug can be very tricky, if not impossible at all. For this purpose I added USB connectivity, and a dedicated debug mode which can be enabled setting the C directives "RAW_MODE" and "USB_MODE" (in "OAED_Common.h") to $true$.

When "USB_MODE" is enabled, the device periodically poll the USB connection to see if there is any command pending. In that case it executes the command, and then resume its normal operation. Since USB transfers are usually very slow, if compared with the instruction execution speed of the CPU, I have decided not to use interrupts in this case. For the same reason it is important to enable USB functionalities only when performing debug operations, and *the USB connector shall be inaccessible in the final device*.

The "RAW_MODE" allow the evaluation of $\Delta\Sigma$-ADC raw signal. In chapter 4 we saw the Delta-Sigma Analog Digital Converter, and the acquisition chain design. The signal coming out the $\Delta\Sigma$-ADC is "split" in two, and sent across the DFB channels, and then moved into the SRAM. The raw signal instead, is directly sent into SRAM, allowing the DFB evaluation.
Finally, setting "OAED_TIME" on $true$, enables the module that keep track of elapsed time.

The commands OAED can accept are in the form of a single 8-bit character in ASCII coding. In Table 5.1 there are listed the accepted codes with their effects.

Table 5.1: OAED USB commands

A	Send both the RAW, and ECG data array.
B	Send both the ECG, and Z buffer array.
C	Enable/disable continuous transfer of the RAW cache.
E	Send ECG data array.
I	Send a string containing impedance informations.
K	Send a string containing system status informations.
R	Send RAW data array.
S	Send system status informations
Z	Send impedance (Z) data array.

Because of architectural limits, the messages OAED can send cannot be

longer than 512 byte. If it need to send more data, OAED shall wait until the buffer has been read, in order to send the remaining data[1]. If the results are in form of a string (command **I** and **K**), they are sent character-by-character and can be read right upon arrive on the pc; if instead the results are in data form, each 16-bit number is sent split in two parts using a little-endian format[2]. This means that in this case it will be necessary a data reconstruction on pc side.

During my work I have also made a simple Matlab interface to communicate with OAED. In is called *psoc_talker.m*, and is reported in Annex B.

Besides the high-level debug implementation I have made, there is another possibility for low-level debug offered by PSoC Creator: the Single Wire Debug (SWD). SWD allow to set break-point, and to watch, or alter in real-time the entire memory/register configuration of the device. The downside of SWD is the necessity of using a PSoC programmer, and the MCU programming interface (see MCU architecture in chapter 4).

5.2 Acquisition chain

In chapter 4 we saw the PSoC internal configuration that I used to implement OAED's front-end. In this, the electrodes are directly connected to the $\Delta\Sigma$-ADC, which produce 16-bit data at a sampling frequency of $4000sps$. This data is then taken from different DMAs, and sent through the DFB's channels A, and B; these two data stream represent respectively the ECG, and impedance data. At the DFB outputs then, data is taken again from two other DMAs which will finally send it into the SRAM.

ECG and impedance data have two similar path, but completely different treatment and requirements; for this reason they will be discussed separately.

5.2.1 ECG

As of the ECG signal, we must first remember its original purpose, namely the diagnose of SCA. Albeit real-time analysis of ECG patterns is not entirely impossible, it is surely very impractical either from an algorithm, that from an hardware point of view. The easiest way to perform a pattern analysis, is to split the signal in fixed-length "windows", and apply the algorithms on that "frozen" signal segment. Furthermore, the main idea is that, meanwhile OAED's CPU is analysing the last ECG window, the DMAs are busy stocking the next one in the SRAM.

The ECG signal windows should be wide enough to allow a good recognition of both the normal and pathological heartbeats, but at the same

[1]This is the main reason why USB functionalities shall not be used when OAED is not in debug mode.

[2]The least significant byte is sent first, and the most significant is sent second.

time be narrow enough to limit the diagnose time. Moreover, in order to improve the algorithms efficiency, we only consider SCA diagnosed if it is confirmed at least two times over the last three windows[3]. Since as explained later in this chapter, the capacitor charging circuit is enabled after the first SCA window is confirmed, and it take about eight seconds to charge it, I have decided to set the ECG window length at four seconds. This time is wide enough to insure that even a couple of bradycardic heartbeat can be detected; and narrow enough to limit the diagnose time to OAED's bare minimum response time possible, which would be twelve seconds[4].

All the above is totally possible using the infrastructure described on chapter 4. The only thing required now are: a firmware implementation, and some sort of optimization. The first consideration to do here is that we will need two array to store the ECG windows: an array with the segment ready to be analysed, named "DataECG"; and a buffer array where the DMA can store the next window, named "BufferECG". In this way, the CPU can work on DataECG, while BufferECG is filled. When this is full, an interrupt will change the "ECG_data_pending" flag, in order to communicate it to the CPU.

It could be simple as that, but we have to remember that OAED's resources are limited. From a simple calculation we can obtain that each of those array will need:

$$S = f_s \cdot 2\frac{Byte}{sample} \cdot T_{window} = 4000sps \cdot 2\frac{Byte}{sample} \cdot 4s = 32000KB = 31.25KiB$$

In other words the two array would need $62.5KiB$ of the total $64KiB$ available in the SRAM! We need to find an alternative solution, we can't afford to spend so much memory just to store the ECG.

When we designed the ECG filter in section 4.2, we did it in order to let pass only the frequencies of interest, namely those between the range of $3 \sim 40Hz$. It is reasonable thereby, to consider nil all the signals outside that band. In this way we have a certain margin to reduce the sampling frequency, by decimating the samples coming out the DFB, without breaking the Nyquist criterion. By reducing the sampling frequency we also reduce the array size -and by logical consequence also the analysis time.

The decimation is obtained by adding an additional array layer, called "CacheECG". This array is considerably smaller than the others, and its size determines the decimation ratio -by default the size is 8. The DMA will

[3]These numbers are *parameters*, and can be changed in "OAED_Common.h" (see last section for more informations).

[4]On a side-note, we could enable the charging circuit as soon as the device turn-on. In this way we can lower the response time to eight seconds. However, if the device then can't diagnose SCA, the energy would be wasted, and we want to avoid that.

now move data from DFB's channel A to this cache, instead of the buffer; and when the cache is full, it triggers an interrupt. The ISR assigned to this, will evaluate the mean value of the cache and write it on the buffer, as showed schematically on Figure 5.1. In this way, from every eight sample we obtain only one mean value, and this is equivalent to dividing the sample frequency by eight ($f_{decimated} = 500sps$). After the decimation we can store four seconds of ECG signal in just $\sim 4KiB$, which means $\sim 8KiB$ if we take in account both the buffer, and the data array. $8KiB$ is equal to $1/8$ of total available SRAM, and this is much more reasonable than the last value.

Since we drastically reduced the SRAM usage, it is also possible to save the last ECG window. This can be used by certain SCA recognition algorithms, such as the TCI and TCSC. More on this will be explained in chapter 6.

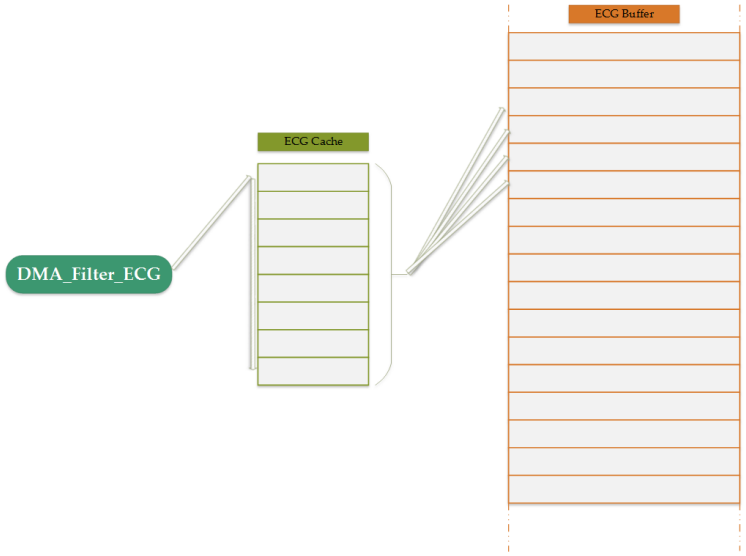


Figure 5.1: ECG acquisition chain - Firmware side

5.2.2 Impedance

As regarding the impedance signal, most of the considerations made for the ECG are still valid. In this case though we have to distinguish between AC and DC impedance acquisitions.

AC impedance

When using AC impedance measurements, the signal has a useful band limited to the $250Hz$ frequency; and this means we can't decimate it without encountering some aliasing artefacts on the final signal. The easiest solution then, is just reducing the window length. In fact, in this case we have less strict timing requirements compared to the ECG; so long that we have at least two peaks in the data array we can evaluate

the impedance. Of course that is just an extreme case, as more data also means more accurate estimations. One second of signal should be a reasonable time in order to keep low the buffer and data SRAM usage; and it is also relatively fast compared to the electrodes physical movements.

As we already largely discussed, the impedance signal OAED obtain is in the form of a digitalised sinusoid. Starting from that, and knowing the excitation signal as well as the electrodes and PSoC impedances, we need to extract the patient impedance value.

We defined the excitation signal in chapter 4 as a current sinusoid with amplitude $I = 6\mu A$, and frequency $f = 250Hz$. This will produce a voltage across the patient, which is read by the $\Delta\Sigma$-ADC. After all data manipulations, OAED will have an array (DataZ) containing one second of the filtered voltage signal, digitalised with a sampling frequency $f_s = 4000sps$.

From DataZ, a simple firmware function[5] will find and extract all the peak-to-peak values, and evaluate their average. Dividing this by the excitation current then, we can calculate the total impedance. As mentioned before, the total impedance is given by the sum of patient, electrodes, and PSoC routing path impedances. Thereby the last passage is subtracting those known values from the total.

If the patient impedance obtained is outside the human limit defined in EN 60601-2-4, the function will return $false$. In this way we have a mean to tell whether the system is actually connected to a patient, or not.

DC impedance

DC impedance measurements are easier to evaluate since the useful band this time is reduced to the DC. Therefore, in addition to reducing its length. we can also decimate the signal. Furthermore, in this case the firmware function used[6] will just evaluate the mean value, and make the same assumption we mentioned during AC impedance evaluation.

5.3 Finite-state machine

My own implementation of OAED firmware can be represented as a finite-state machine, as in Figure 5.2

By definition, a finite-state machine is an abstract automaton that can be in exactly one of a finite number of states at any given time. The state can change in response to an external input. This change is called transition.

In reference with Figure 5.2, a finite-state machine can be defined by the list of its states, represented as round blocks; its initial state, which is the green box; and the conditions for each transition represented by the labels on each arrow. In OAED's firmware I defined five different states:

[5] bool OAED_EvaluateImpedanceAC()
[6] bool OAED_EvaluateImpedanceDC()

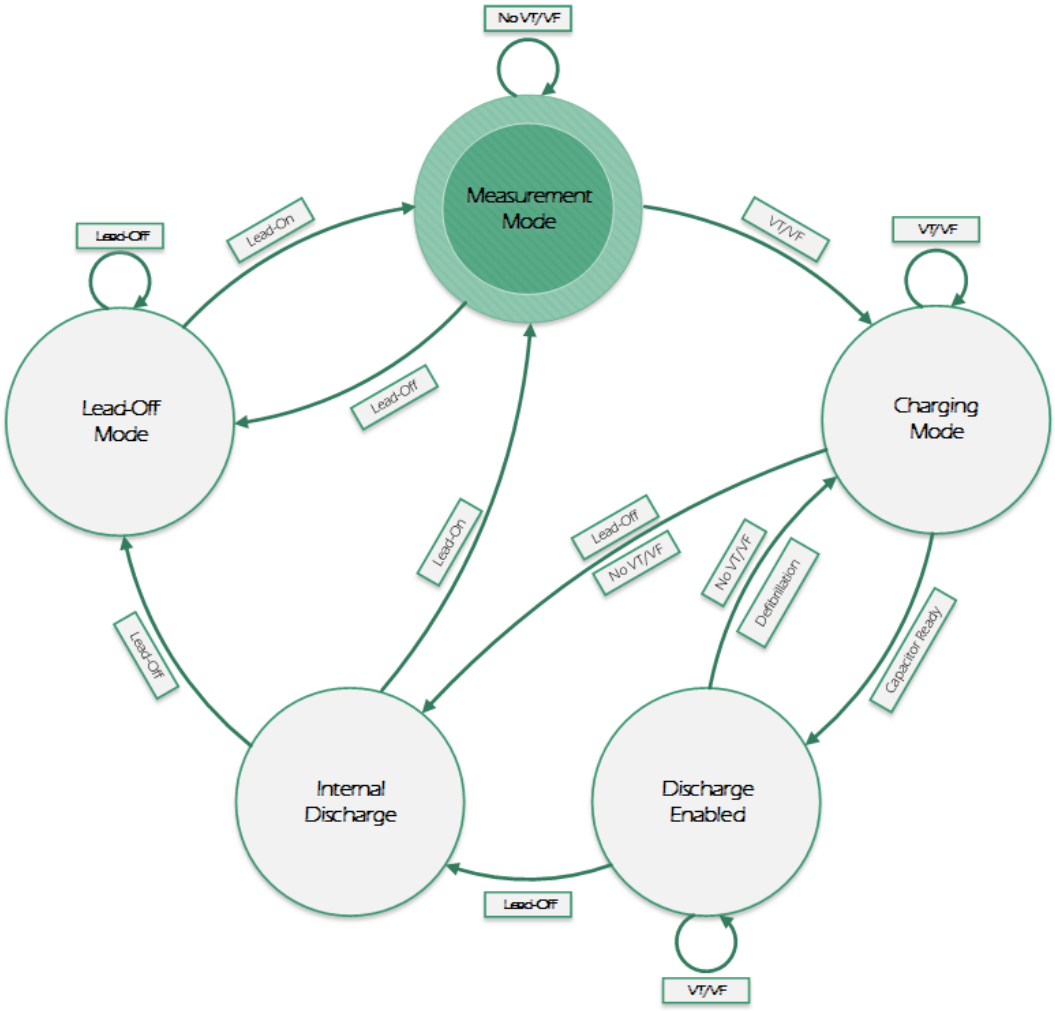


Figure 5.2: OAED states diagram

- measurement mode, which is the initial state;
- charging mode, entered after OAED successfully diagnose SCA;
- discharge enabled mode, which is the state that allow the operator to release a defibrillation;
- internal discharge mode, entered when the charge is no longer needed; and
- lead-off mode, which is the state the device enter until it cannot correctly detect a patient.

It is important to understand that this is just my vision of how OAED should work. Albeit this finite-state machine is the result of a long well-thought design, it is not the only possible solution. As explained in the previous section of this chapter, OAED's firmware is designed to be extremely flexible, and its behaviour can be changed according to specific needs. Re-arranging the basic functions, and/or overwriting the states can lead to completely different solutions.

What follows is an accurate description of each state, united with their specific flow-chart.

5.3.1 Measurement mode

This state is by default the initial state. When entering *measurement mode* the device start ECG acquisitions and continuously check if the electrodes are still on the patient. If it cannot detect them then the "lead_detected" flag is set to $false$, then the status is changed to *lead-off mode*. On the other hand if it is $true$, OAED check if there is new data pending ready to be analysed. In the positive case, it runs the pattern recognition algorithms on the ECG to diagnose SCA. If this is also positive, the system state is changed to *charging mode*. Otherwise it wait for new data to be available. When this happen it stops the acquisition, overwrite the data array with the buffer, and finally it restart the *measurement mode*.

While idling waiting for new data, the device keep checking whether the electrodes are still on the patient. In the negative case, it immediately enters *lead-off mode*.

5.3.2 Charging mode

The *charging mode* is entered as soon as OAED detects the first SCA event, and it represent an "alert" mode. In fact, while in this state the device enable the capacitor charging circuit (see chapter 3), and continue checking the patient status.

The flow-chart in this mode (represented in Figure 5.4) is similar to the last (Figure 5.3). In addition to what said above, after it evaluate the ECG, OAED checks whether the capacitor is charged. If it is ready and the patient is still suffering SCA, then the device pass in *discharge enabled mode*. Otherwise the following may happen:

- whenever the "lead_detected" flag become $false$, the device automatically pass in *internal discharge mode*;
- if OAED cannot assert SCA on the patient any more, the device will remain this state until either: SCA is diagnosed again, or it reaches the maximum limit of false positive event (set to five by default). In the latter case, state is changed to *internal discharge mode*.
- if the capacitor is not ready yet, but there is new data, OAED stops acquisition, overwrites ECG data array with its buffer, and finally it restarts the *charging mode*.

5.3.3 Discharge enabled mode

This state represent the operative mode. OAED can only enter *discharge enabled mode* when SCA is definitely diagnosed and the capacitor is ready to release a defibrillation.

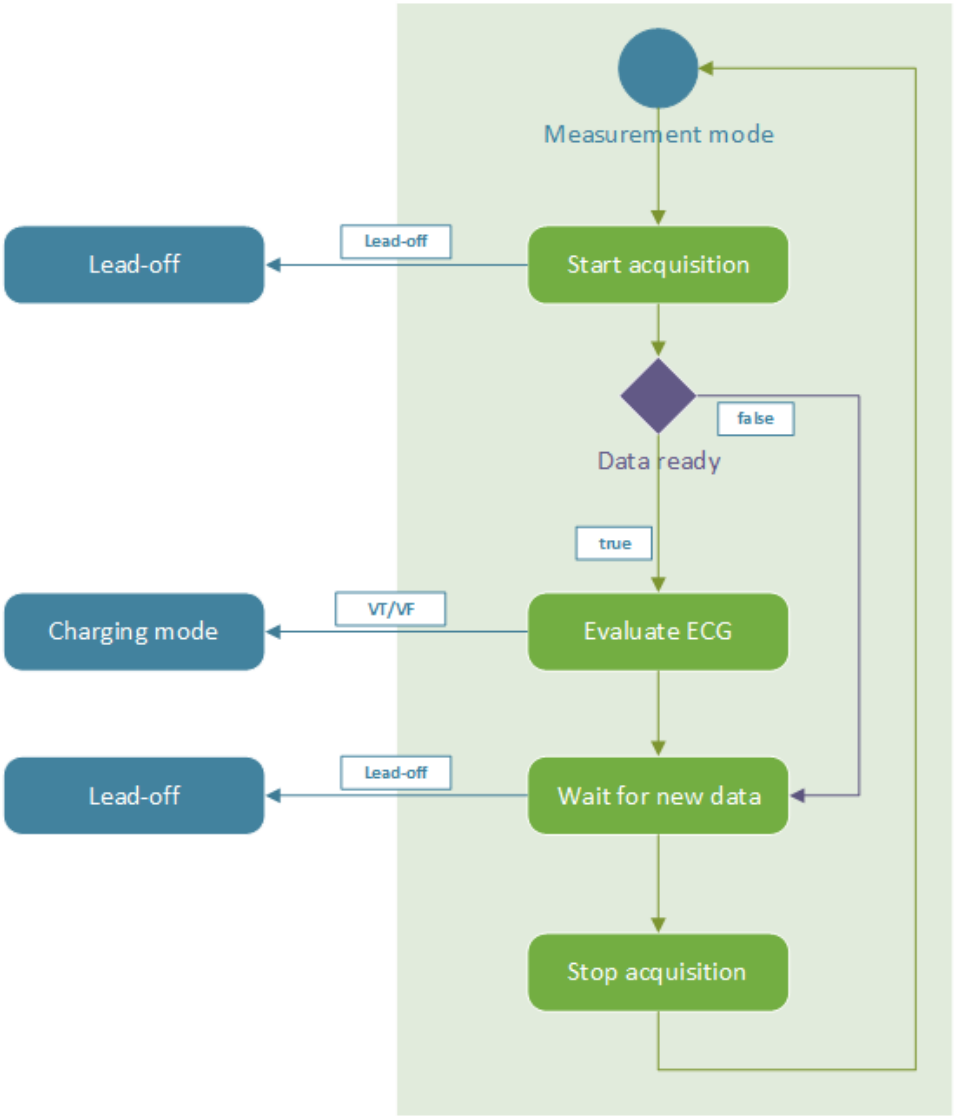


Figure 5.3: Measurement mode flow-chart

While in this mode OAED enable the interrupt triggered by the defibrillation push-button (see chapter 4), allowing the operator to deliver a defibrillation to the patient at any time. When this happen the device will return in *charging mode*.

As we can see from the flow-chart in Figure 5.5, this state is extremely similar to the measurement mode. In fact, if we do not consider the charging circuit and the defibrillation push-button, the system working is analogous. OAED start the acquisition and check if the electrodes are still placed on the patient. Then it wait for new data, analyse it, and stops the acquisition to restart the *discharge enabled mode*.

The main difference are: if OAED detect a lead-off, it pass to *internal discharge mode* instead of *lead-off mode*; and if the ECG pattern recognition fail to diagnose SCA, the device return in *charging mode*.

Defibrillation

In OAED I have implemented various function that perform the defibrillation. By default the firmware uses the biphasic function, but there are also available the monophasic one, and a generic polyphasic function.

All these functions control the H-Bridge circuit dynamically calculating the time required for each phase, depending on the patient impedance.

5.3.4 Lead-off mode

The *lead-off mode* (see Figure 5.6) represent a safety condition for the device. In fact, this state is only entered when OAED cannot detect a patient, and it will not exit until it can.

When OAED enter the *lead-off mode*, it will:

- stop ECG acquisition[7] since is no longer needed;
- reset event counter, buffers, and data arrays. Since we can't know if OAED will be reconnected to the same person, we must restart the diagnose process from the beginning;
- wait until it can detect a patient again.

If a patient is detected, OAED will transit to *measurement mode* and treat him/her like a new patient. We don't have any mean to understand if is connected to the same person, therefore we must restart the diagnose from zero in order to be sure of avoiding to defibrillate a sane person.

5.3.5 Internal discharge mode

The *internal discharge mode* is more a transition-state, than a proper state. This mode represent an emergency stop for the device. In fact, *internal discharge mode* is entered from *charging mode* or *discharge enabled mode* when OAED detects a lead-off, or a false positive case of SCA.

While in this state, OAED release the capacitor charge in the internal discharge circuit (see chapter 3), and then move either in *lead-off mode*, or *measurement mode*, depending on the electrodes status.

An alternative to prioritize energy saving could be to enter this state only when actually needed. E.g. in the case of malfunctioning, or before the device shut-down.

[7] I take the chance to remark that impedance acquisition is never stopped.

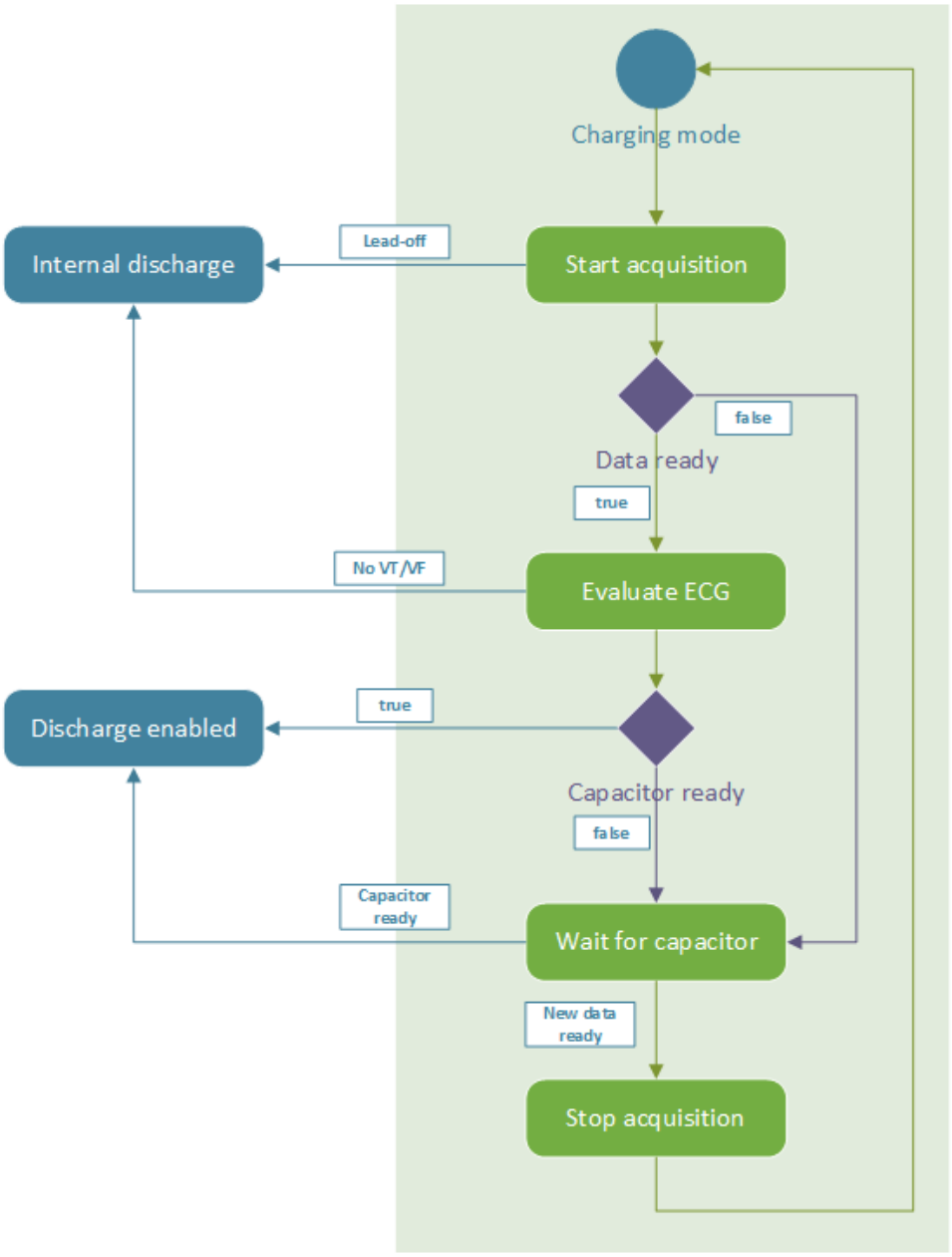


Figure 5.4: Charging mode flow-chart

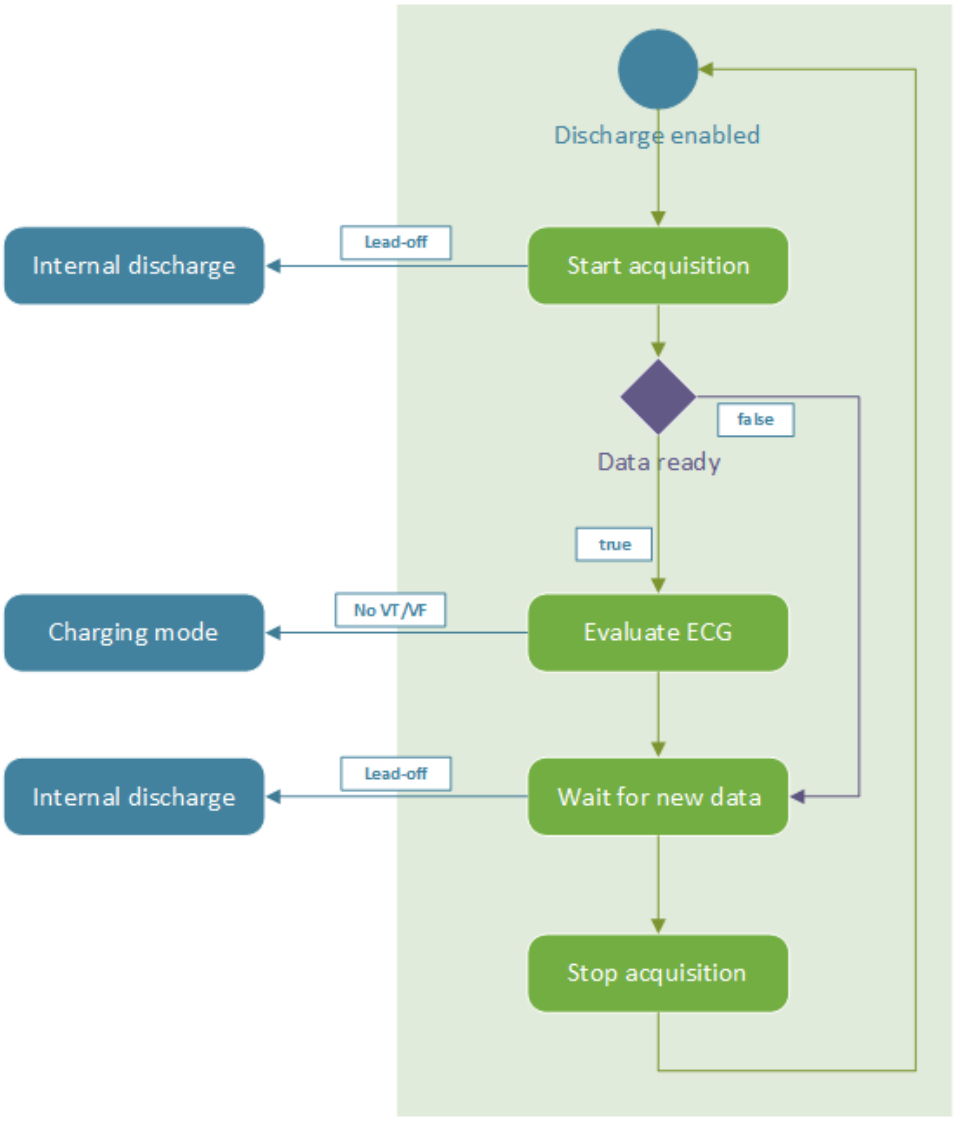


Figure 5.5: Discharge enabled mode flow-chart

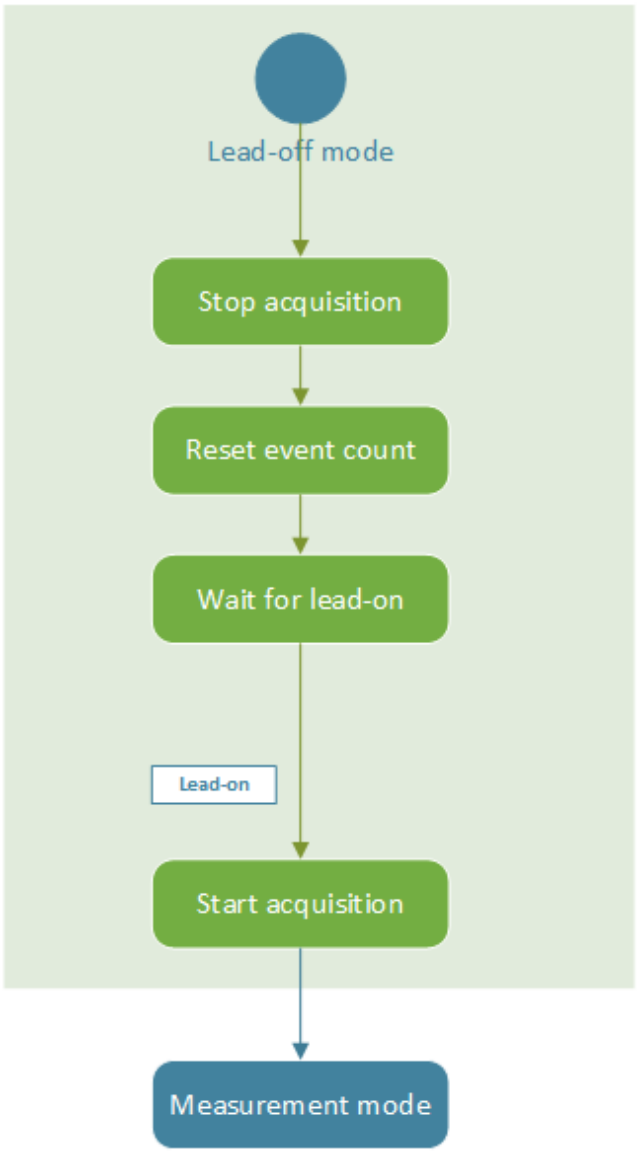


Figure 5.6: Lead-off mode flow-chart

CHAPTER 6

Sudden Cardiac Arrest recognition algorithms

So far we described OAED's hardware (chapter 3 and chapter 4) and firmware (chapter 5). We structured its architecture in order to be able of efficiently performing an effective defibrillation, and to read and store the patient's Electrocardiogram (ECG) and impedance signals. However, the capability of automatically recognize whether the patient need defibrillation or not, is what makes the difference between a smart defibrillator and an automatic one. Thereby, the last piece of the puzzle remaining, is giving to OAED a Sudden Cardiac Arrest (SCA) recognition capability.

In this chapter we will start with an overview on some of the most used SCA recognition algorithms, and some innovative ones. We will then select some of those, and after a more in-depth analysis of their operations, we will implement them in OAED's firmware. Lastly, in order to validate the recognition efficiency, a cross evaluation with their Matlab counterparts will be done using a custom set of ECG signals obtained from PhysioNet's database [1].

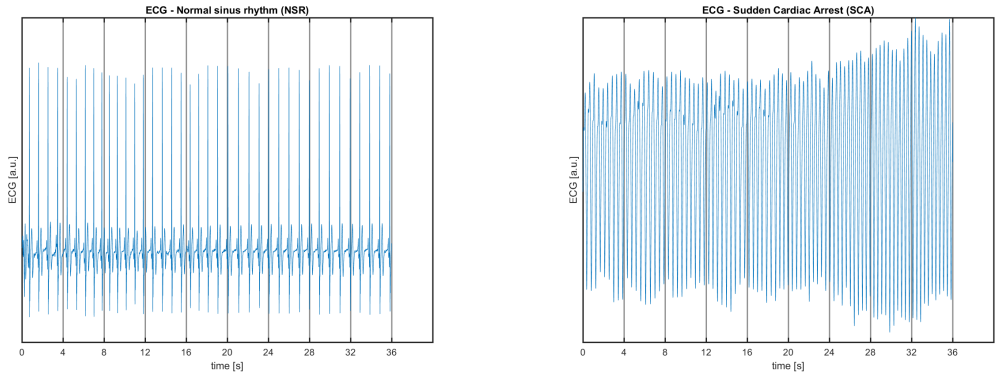
6.1 Electrocardiogram pattern recognition

The Electrocardiogram is one of the most studied physiologic signals, and has been widely discussed over the whole course of this thesis. In chapter 1 we saw an introduction on what is the ECG; in chapter 4 there is a simple explanation on how it is obtained from the patient, and the transformations necessary in order to obtain a noise-free signal; finally, in chapter 5 there is the firmware treatment of ECG data.

Resuming what done until now: the ECG is obtained from the patient and digitalized by the $\Delta\Sigma$-ADC, filtered with a $3 - 40Hz$ band-pass filter in the DFB, and finally moved to the memory, where it is stored as a 4 seconds window at the sampling frequency of $500sps$.

The reason of having this 4 seconds ECG signal is to analyse it and assert whether the patient is suffering from Sudden Cardiac Arrest or not. For this purpose it is necessary to extract some information from the signal, and elaborate them in order to differentiate an SCA ECG from a non-SCA one.

To facilitate the reading, from now on I will refer to an "healthy"[1] ECG (or NSR ECG) if it doesn't show the SCA hallmarks, and to a pathological ECG if it does. As a mean of comparison in Figure 6.1 (a) there is a healthy ECG, and in Figure 6.1 (b) there is a pathological ECG. Over the course of this chapter there will be a lot of reference on those signals. The Normal Sinus Rhythm (NSR) ECG is obtained using OAED on an healthy patient (Me), while the second has been taken from the PhysioNet ECG signal database (see the last section of this chapter for more informations).



(a) (b)

Figure 6.1: Examples of ECG. (a) Healthy ECG; (b) Pathological ECG

There are various challenges to overcome when designing an algorithm that diagnose SCA: the first is the already evidenced lack of micro-controllers computational resources; the second one is the scarcity of pathological data available as reference; the third, related to the second, is that SCA signals have a wide variety of different patterns and shape, especially when the patient is suffering from Ventricular Tachycardia; and lastly, the algorithm should require very short data -in our case not more than 4 seconds- to assert SCA, with an excellent specificity and an acceptable sensibility[2].

At the moment of writing this thesis there is no such thing as the "definitive SCA recognition algorithm" that meet al the above. However, there are many valid algorithms, all of which has its strength and weakness. Therefore the idea is that by combining the results of multiple algorithms, it is possible to achieve better performance, and to satisfy more of the previously said challenges.

[1]Well, in this case healty only means that the patient is not suffering from SCA, but if he's been attached to a AED he's probably not fine.

[2]Specificity and sensibility are discussed on chapter 2.

6.1.1 Four gold standard algorithms

A good practice when looking for a viable algorithm, is to sift through the most used and referenced ones. In the case of SCA pattern recognition there are four recurrent algorithms that form the so called *gold standard*: Threshold Crossing Intervals (TCI) [2]; VF filter (VFf) [3]; Autocorrelation Function (ACF) [4]; and Spectral algorithm (SPEC) [5].

From the moment of their release, these algorithms have been widely discussed [6, 7], analysed [8], and improved [9] in many more publications. Although they are far from being perfect, they havebeen available for more than 20 years, and they are now used as a mean of comparison for every new algorithm developed.

Here follow a brief explanation of how they work.

Threshold Crossing Intervals

TCI is a time-domain algorithm. Its decision are based on the fact that during SCA the apparent heart rate is usually in excess of $250bpm$. Which is much higher than normal values, in the range of $50 \sim 120bpm$.

In TCI the heart rate is measured by comparing the ECG with a threshold, which is set so that the signal crosses it once per beat. The interval between each crossing of a threshold by the signal upstroke was defined as one TCI. Within each sub-segment of the signal, the complete TCIs are used to calculate a mean TCI. This is then used to perform a statistical analysis using the standard distribution characterised using a specific dataset. In particular a value of $TCI > 400ms$ it is distinctive for NSR, while VT has a mean value of $220ms$ and a standard deviation of $16.5ms$, and VF has values of $105ms$ and $6.5ms$ respectively [2].

Autocorrelation Function algorithm

Also ACF is a time-domain algorithm. Its decision are made assuming the pathological signal is relatively aperiodic, whilst the NSR is periodic. Therefore the algorithm is based on peaks analysis in the short-term autocorrelation function.

The peaks in the short-term ACF of a periodic signal with constant amplitude lie exactly on a straight line and decrease monotonically because fewer data are included in the calculation as lag increases. In the case of an aperiodic signal, or a periodic signal with inconsistent amplitude, the peaks will not lie on a straight line. Evaluating the signal autocorrelation function with a regression equation is therefore possible to distinguish an healthy rhythm from a pathological one [4].

VF filter

The VFf is a frequency-domain algorithm, and in contrast with the ACF it relies on the pathological signal approximating a sinusoidal waveform, hence a periodic signal.

At first the mean period of a fixed length data is calculated. The data are then combined with a copy of the data which has been shifted by half a period. This process is equivalent to applying a narrow bandstop filter centred on the mean signal frequency. Therefore if the data approximate a periodic signal, they will cancel each other. On the other hand, if the leakage (namely the residual signal) is higher than a certain threshold, the data does not approximate a periodic signal, hence the rhythm is normal [3].

Spectral algorithm

SPEC is a frequency-domain algorithm. Its based on the fact that most of Normal Sinus Rhythm are broadband signals with major harmonics up to about $25Hz$. However, during SCA, the ECG becomes concentrated in a narrow band of frequencies between $4 \sim 7Hz$. In SPEC, data is transformed using the FFT with an hamming window. The signal amplitude is then approximated by the sum of absolute values of the real and imaginary parts, and finally the energy content in different frequency bands are evaluated in order to make a decision [5].

6.1.2 A more modern approach

The algorithms presented in the last section are usually sufficient to determine whether the patient is suffering from SCA or not. However we should not forget that they have been developed many years ago (VFf is dated 1978), and the continuous progress of technology may have outdated them, or found better alternatives.

At this day, the number of SCA recognitions algorithms are hundreds. Even if we reduce the search radius solely to the algorithms capable of being evaluated in reasonable time using the limited resources of the PSoC[3], there are still dozens available. For the sake of brevity I have here only reported some of the most significant and known ones.

Standard Exponential algorithm

The algorithm operates in time-domain counting the number of crossing points of the ECG signal with an exponential curve decreasing on both sides. This exponential like function start from the absolute maximum value of each evaluated segment. The recognition is then made by counting the number of crossings [6].

Modified Exponential algorithm

This is a modified version of the **Standard Exponential algorithm** obtained by "lifting" the exponential curve at each ECG crossing point onto its relative maximum.

[3]Hence for example, excluding the algorithms based on wavelet transform.

The difference with the previous is, that here the crossing function does not have the exponential trend over the whole investigated signal part, but only in the region from the first relative maximum to the first intersection with the ECG signal. Then, the function coincides with the ECG signal until it reaches a new relative maximum.

Anyhow, the decisions for SCA or not are still made counting the crossings [6].

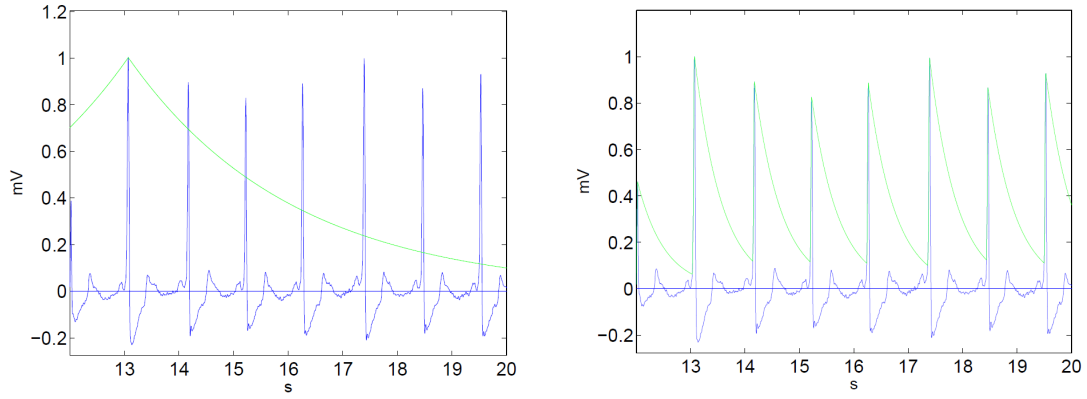


Figure 6.2: SCA recognition algorithms (a) Standard exponential algorithm; (b) Modified exponential algorithm

Pan-Tompkins algorithm

The Pan-Tompkins algorithm is a well known algorithm for QRS complexes recognition which uses slope, amplitude and width informations [10].

The ECG signal at first undergoes to a derivative filter, in order to highlight the QRS complexes. Then is squared to remove the negative values and additionally sharpen the complexes, and finally it is integrated in order to remove/reduce noise. At this point the QRS complexes are recognised with an adaptive threshold.

The QRS are counted in order to calculate the heartbeat rate. The decision then are made by comparing this value with the NSR rate of $50 \sim 120bpm$, and the pathological rate higher than $250bpm$ [6].

Threshold Crossing Sample Count

The Threshold Crossing Sample Count (TCSC) algorithm operate in time-domain, and its decisions are based on the percent of time the ECG signal remains outside a certain threshold.

In TCSC, the ECG segments undergoes a special cosine window that reduce the values on the segment borders. Then the time is calculated by counting the number of normalized ECG samples that cross the predefined threshold V_0, which depends on the absolute maximum of the segment. If

"high" signal percentage is higher than a certain critical value, the patient is suffering from SCA [9].

Phase Space Reconstruction

The Phase Space Reconstruction (PSR) algorithm -also known as time-delay algorithm- is based on a method which is used to reconstruct the so-called phase space. It analyses signals in order to identify a dynamic law or random behaviours. Knowing that the ECG appear as regular in a Normal Sinus Rhythm, and chaotic during SCA, we can then determine whether the patient needs defibrillation or not.

The ECG signal $x(t)$ is plotted in a diagram as follows: on the x-axis we plot $x(t)$, while on the y-axis we plot $x(t+\tau)$, where τ is a proper time constant -such a plot is called a two dimensional phase space diagram. We observe that a typical pathological signal produces a curve in the diagram, that fills the area in an irregular way. The curve is almost uniformly distributed over the entire diagram, and thus it occupies a larger area. On the other hand, a NSR shows a regular structure -resembling a cross- and only small parts of the area are filled. Therefore, based on the area occupied in the phase space plots $(x(t); x(t+\tau))$ we can differentiate SR from VF [11].

Hilbert Transform algorithm

The Hilbert Transform algorithm (HTA) is a frequency-domain algorithm based on the observation of the analytic signal -which is normally used to study non-linear signals.

The analytic signal $Z(t)$ is a complex signal having the form of $z(t) = x(t) + ix_H(t)$, where $x(t)$ is the ECG signal, and $x_H(t)$ denotes the Hilbert transform of $x(t)$. The analytic signal is then used to generate a phase space diagram and the decisions are made exactly the same way as **PSR** [12].

6.2 Sudden Cardiac Arrest recognition in OAED

When taken singularly, the algorithms described in the first part of this chapter, are usually enough to detect Sudden Cardiac Arrest in ideal conditions; but not when ECG signal shows noise, artefacts, asystole, or other variations. Furthermore, the specificity and sensibility are usually not good enough to satisfy the IEC 60601-2-4 standard.

We already mentioned the chance of combining the results of more algorithms to decide whether the patient is suffering from SCA or not. The results combination can be done in various ways, and the final decision can be different according on how it is done. For example, when using few algorithms it is possible to define a *criticality* variable, which depends on the single results -an example of this approach is in [13]; if that variable exceed a predetermined threshold, SCA is confirmed. For the sake of simplicity in OAED I have instead decided to implement a more *democratic* approach: if

the majority of algorithms are positive, SCA is confirmed; contrariwise the heartbeat is assumed as Normal Sinus Rhythm.

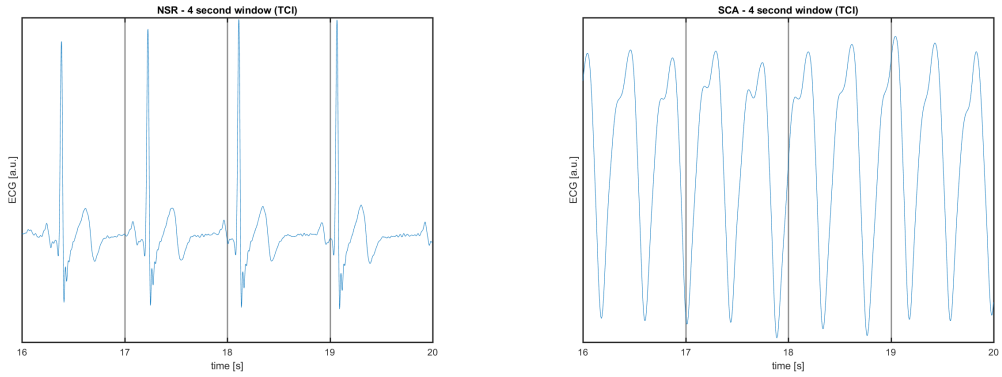
I have implemented in OAED five of the algorithms mentioned in the first part of this chapter. Two traditional ones, and three innovative:

- Threshold Crossing Intervals (TCI);
- VF filter (VFf);
- Threshold Crossing Sample Count (TCSC);
- Phase Space Reconstruction (PSR); and
- Hilbert Transform algorithm (HTA);

Here follow a more in-depth description on how they work and are implemented. All the images are obtained using Matlab and are solely for the sake of a easier explanation of the operations.

6.2.1 TCI

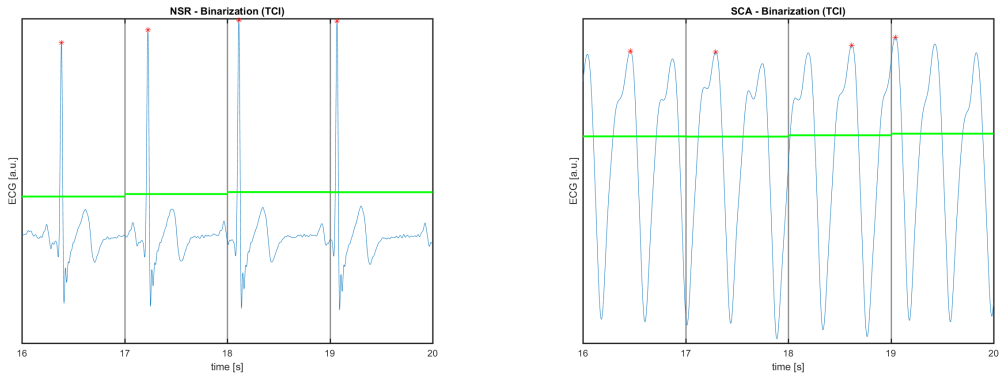
From the literature [2], this algorithm evaluate the ECG of one second segments. Therefore we have to further split our 4 seconds windows in four parts, as showed in Figure 6.3.



(a) (b)

Figure 6.3: TCI: Starting segments. (a) Healthy ECG; (b) Pathological ECG

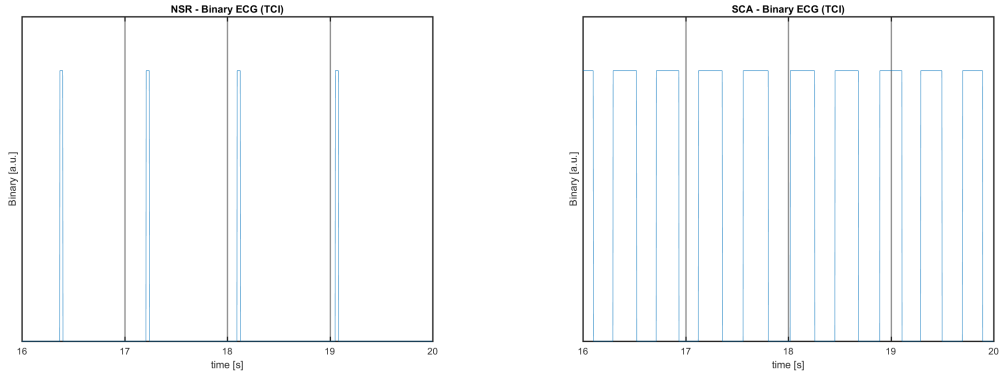
The second step is the ECG binarization. We locate the maximum in each segment and we apply a 20% threshold of that value (see Figure 6.4).



(a) (b)

Figure 6.4: TCI: Binarization. (a) Healthy ECG; (b) Pathological ECG

At this point we obtain the binary signal as in Figure 6.5. From this we can obtain the TCI values.



(a) (b)

Figure 6.5: TCI: Binary ECG. (a) Healthy ECG; (b) Pathological ECG

Equation 6.1 indicates how the TCI values are calculated.

$$TCI = \frac{1000}{N - 1 + \frac{t_2}{t_1+t_2} + \frac{t_3}{t_3+t_4}}[ms] \tag{6.1}$$

As shown in Figure 6.6, N indicates the number of pulses in every segments, while the four time t_x indicates respectively:

- t_1 is the time between the previous segment last pulse, and the start of the analysed segment;
- t_2 is the time between the start of the analysed segment, and the first pulse;
- t_3 is equivalent to the next segment t_1, and indicates the time from the last pulse in this segment and the end;
- t_4 correspond to the next segment t_2, and indicates the time from the start of the next segment and its first pulse.

In signals when pulses are between two segments, times are taken as zeroes -same as the indeterminate form $^0/_0$.

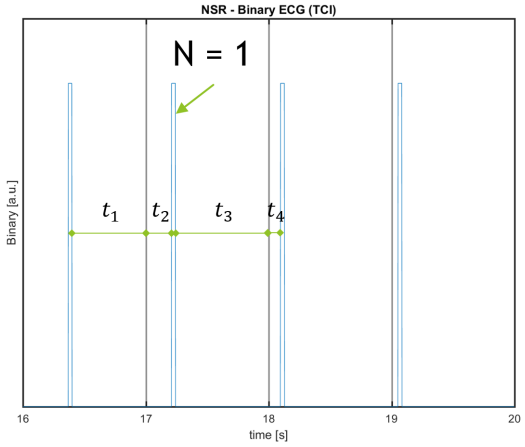


Figure 6.6: TCI: Values.

At this point if the mean value of TCI is over $400ms$, then the heartbeat is considered NSR. If it is smaller then we need a sequential hypothesis statistical test to discriminate VT and VF on the basis of the probability distributions of TCI as indicated in [2].

In this step we have to calculate the parameter F, according to Equation 6.2.

$$F = \frac{1}{{\sigma_{VF}}^2} \sum_{m}^{i=1} (TCI_i - \mu_{VF})^2 - \frac{1}{{\sigma_{VT}}^2} \sum_{m}^{i=1} (TCI_i - \mu_{VT})^2 \tag{6.2}$$

If Equation 6.3 is valid, then the heartbeat is recognised as VT; else if Equation 6.4 is valid, then the heartbeat is recognised as VF; and finally, if neither are valid, the rhythm cannot be correctly identified with the available data.

$$F \geqslant 2 \cdot ln\left(\frac{1-\beta}{\alpha}\right) + 2m \cdot ln\left(\frac{\sigma_{VT}}{\sigma_{VF}}\right) \tag{6.3}$$

$$F \leqslant 2 \cdot ln\left(\frac{\beta}{1-\alpha}\right) + 2m \cdot ln\left(\frac{\sigma_{VT}}{\sigma_{VF}}\right) \tag{6.4}$$

In Equation 6.2, Equation 6.3, and Equation 6.4 α and β indicates the confidence interval; while σ_{VT} and σ_{VF} indicates respectively the standard deviation of VT and VF distributions; and finally μ_{VT} and μ_{VF} are their mean values.

6.2.2 VF filter

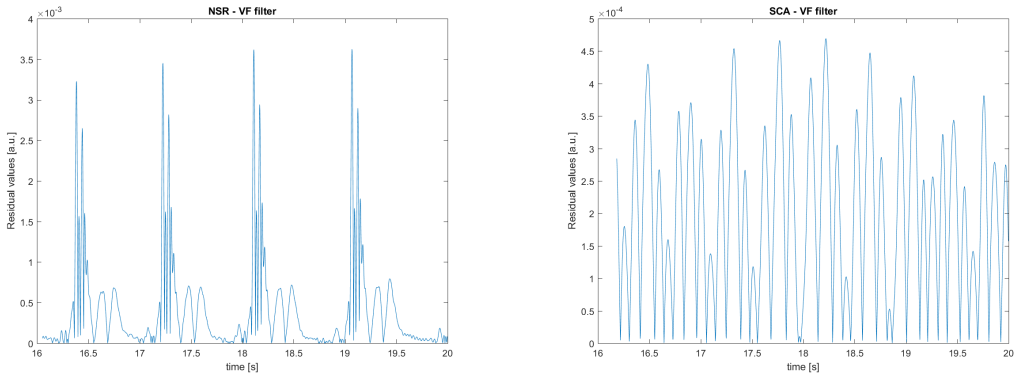
The VF filter algorithm works on any lengths of signal. In this case hence, we can use the whole 4 seconds window.

The first step is finding the mean period T using Equation 6.5.

$$T = \left\lfloor 2\pi \frac{\sum_{i=1}^{m} |V_i|}{\sum_{i=1}^{m} |V_i - V_{i-1}|} + 1 \right\rfloor \tag{6.5}$$

Knowing the mean period of the segment, it is possible to apply the narrowband filter and evaluate the leakage using Equation 6.6. The results are shown in Figure 6.7 and Figure 6.8.

$$leakage = \frac{\sum_{i=1}^{m} \left| V_i + V_{i-T/2} \right|}{\sum_{i=1}^{m} \left(|V_i| + \left| V_{i-T/2} \right| \right)} \tag{6.6}$$



(a) (b)

Figure 6.7: VFf: VF filter leakage. (a) Healthy ECG; (b) Pathological ECG

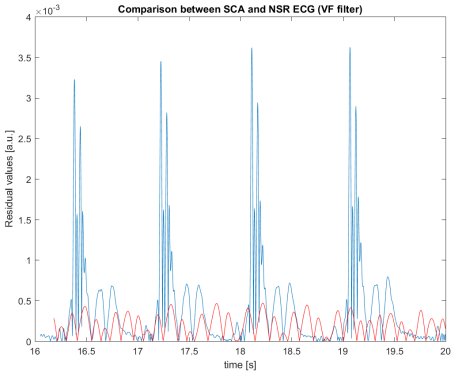


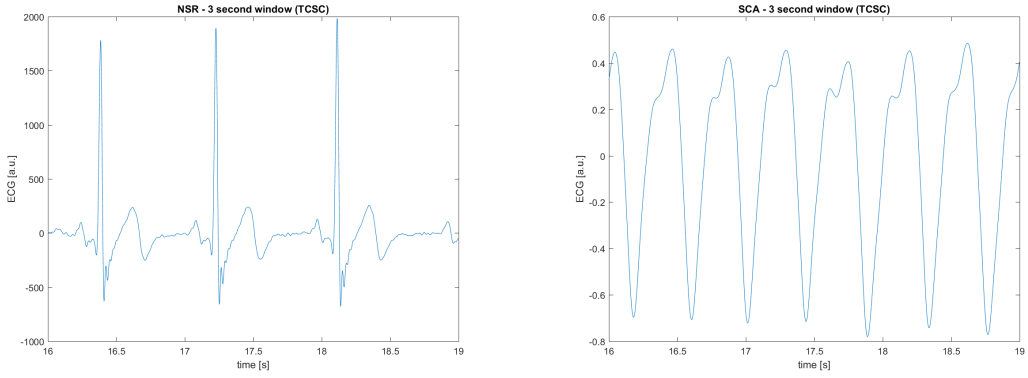
Figure 6.8: VFf: Comparison between healthy and pathological.

6.2.3 TCSC

TCSC works with 3 seconds window, as shown in Figure 6.9. The first passage in this case, is applying a cosine window at the absolute signal with the following form:

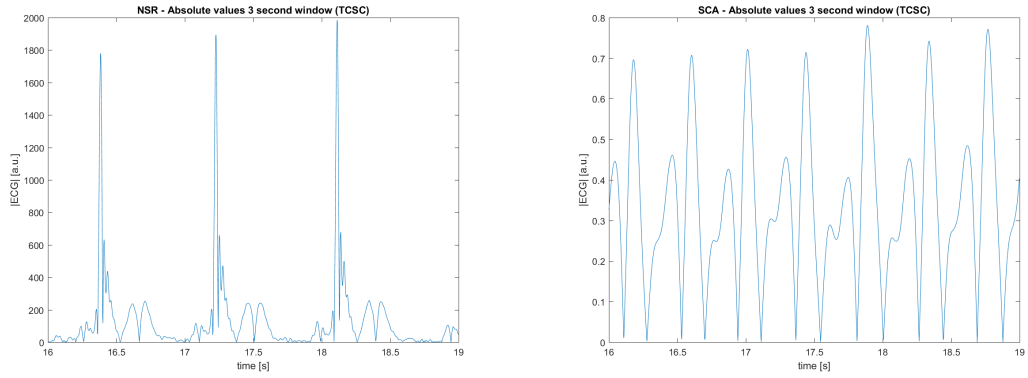
$$W(t) = \begin{cases} \frac{1}{2} \cdot (1 - cos\,(4\pi t)) & 0 \leqslant t \leqslant \frac{1}{4} \\ 1 & \frac{1}{4} \leqslant t \leqslant 3 - \frac{1}{4} \\ \frac{1}{2} \cdot (1 - cos\,(4\pi t)) & 3 - \frac{1}{4} \leqslant x \leqslant 3 \end{cases} \tag{6.7}$$

The results of this operation are shown in Figure 6.10.



(a) (b)

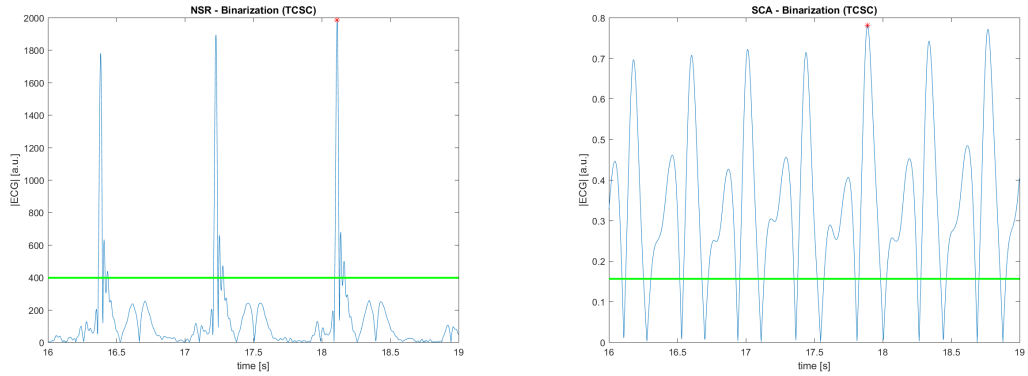
Figure 6.9: TCSC: Starting segments. (a) Healthy ECG; (b) Pathological ECG



(a) (b)

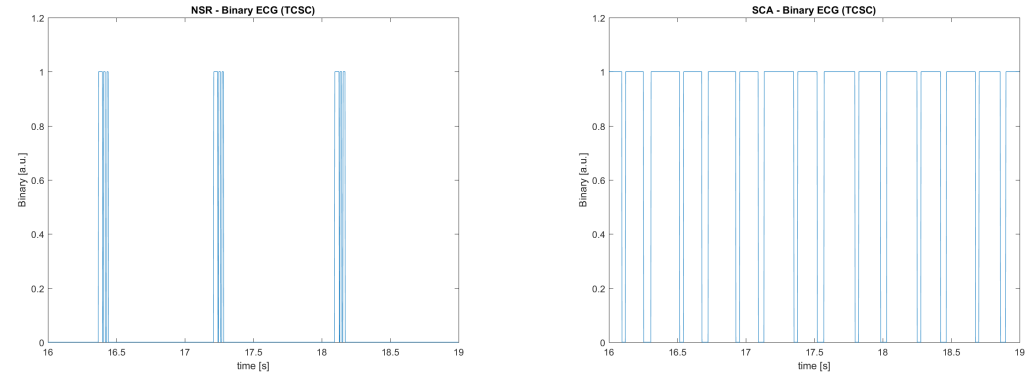
Figure 6.10: TCSC: Absolute values ECG. (a) Healthy ECG; (b) Pathological ECG

The signal then is binarized with a threshold set to 20% of the absolute maximum, as shown in Figure 6.11.



(a) (b)

Figure 6.11: TCSC: Binarization. (a) Healthy ECG; (b) Pathological ECG



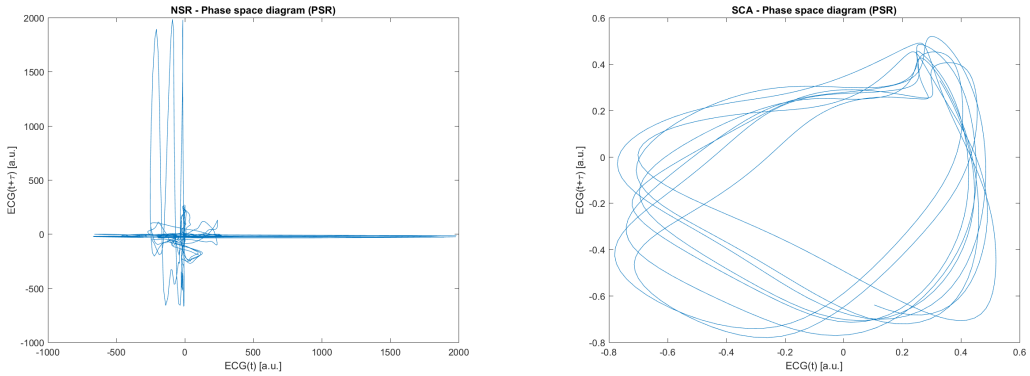
(a) (b)

Figure 6.12: TCSC: Binary ECG. (a) Healthy ECG; (b) Pathological ECG

Finally, using the binarized signal in Figure 6.12, it is possible to calculate the percentage of *high* time on the total. According to [9], the TCSC algorithm is repeated starting 1 second after the beginning of the last 3 seconds window, overlapping 2 seconds with it and with the next one. The decisions are finally made using the results average. If this is higher than a certain threshold the signal is assumed pathological, otherwise it is NSR.

6.2.4 PSR

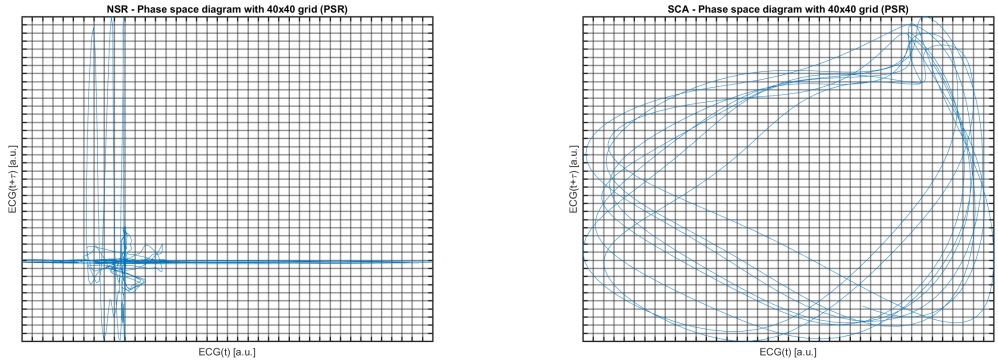
The phase space diagram shown in Figure 6.13 is obtained plotting the ECG signal on the x-axis, and the same signal shifted by 0.5 seconds on the y-axis.



(a) (b)

Figure 6.13: PSR: Phase space diagram. (a) Healthy ECG; (b) Pathological ECG

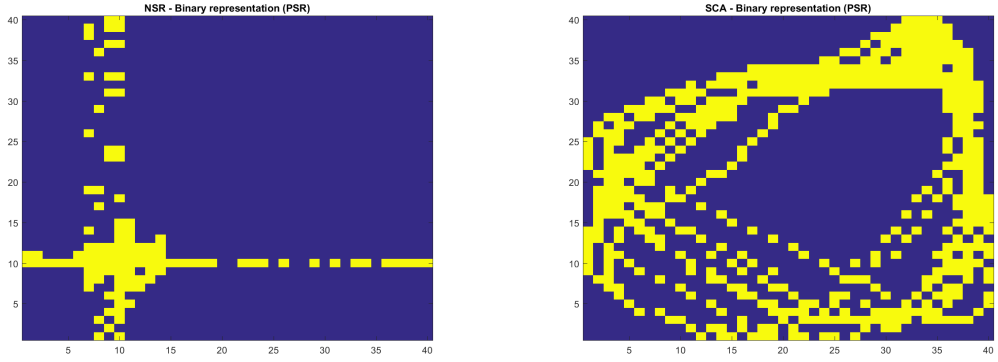
Since the decision are made according to the area occupied by this diagram, the plane is divided in a 40x40 grid, as shown in Figure 6.14. The bounds of this grid depends on the minimum and the maximum values of the ECG.



(a) (b)

Figure 6.14: PSR: Phase space diagram with the 40x40 grid. (a) Healthy ECG; (b) Pathological ECG

Each cell is then highlighted if it contain at least one point of the signal, forming in this way the binary image shown in Figure 6.15.



(a) (b)

Figure 6.15: PSR: Binary representation of the phase space. (a) Healthy ECG; (b) Pathological ECG

The decisions are finally made comparing the number of highlighted cells with a predetermined threshold. If the value is higher the rhythm is assumed pathological, if instead it is lower the rhythm is assumed normal.

6.2.5 HTA

As the name suggest, the Hilbert Transform algorithm require evaluating the Hilbert transform. This is used to calculate the analytic signal, which will be used to draw the phase space diagram and evaluated in analogy with the PSR algorithm.

The Hilbert transform is important in signal processing, where it derives the analytic representation of a signal $u(t)$. This means that the real signal $u(t)$ is extended into the complex plane such that it satisfies the Cauchy–Riemann equations. For example, the Hilbert transform leads to the harmonic conjugate of a given function in Fourier analysis.

The Hilbert transform of $u(t)$ can be thought of as the convolution of $u(t)$ with the function $h(t) = {}^{1}/_{(\pi t)}$. Because $h(t)$ is not integrable, the integrals defining the convolution do not converge. Instead, the Hilbert transform is defined using the Cauchy principal value (denoted here by p.v.). Explicitly, the Hilbert transform of a function (or signal) $u(t)$ is given by Equation 6.8.

$$H(u(t)) = p.v. \int_{-\infty}^{\infty} u(\tau)h(t-\tau)d\tau = \frac{1}{\pi} p.v. \int_{-\infty}^{\infty} \frac{u(\tau)}{t-\tau} d\tau \tag{6.8}$$

An important feature of the Hilbert transform is its relationship with the Fourier transform.

The Hilbert transform is a multiplier operator. The multiplier of H is $\sigma_H(\omega) = -i \cdot sgn(\omega)$ where sgn is the sign function. Therefore:

$$\mathcal{F}(H(u(t))(\omega) = (-i \cdot sgn(\omega)) \cdot \mathcal{F}(u)(\omega) \tag{6.9}$$

Where $\mathcal{F}$ denotes the Fourier transform. Since $sgn(x) = sgn(2\pi x)$, it follows that this result applies to the three common definitions of $\mathcal{F}$.

By Euler's formula:

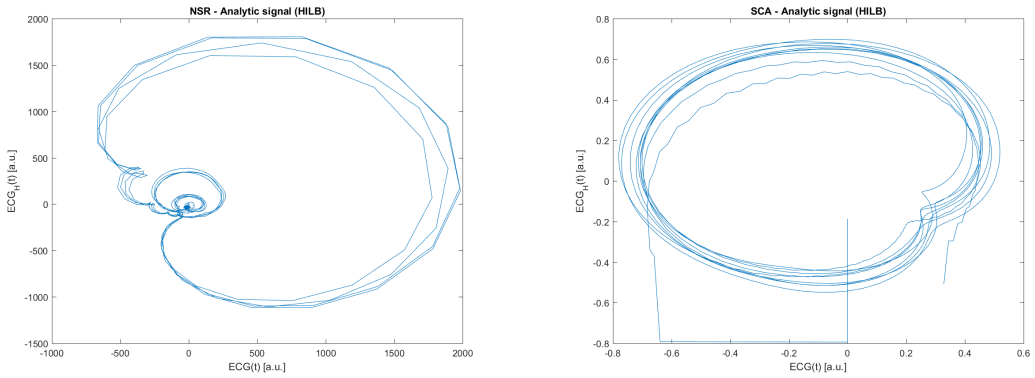
$$\sigma_H(\omega) = \begin{cases} i = e^{+\frac{i\pi}{2}}, & \text{for } \omega < 0 \\ 0, & \text{for } \omega = 0 \\ -i = e^{-\frac{i\pi}{2}}, & \text{for } \omega > 0 \end{cases} \tag{6.10}$$

Therefore, $H(u)(t)$ has the effect of shifting the phase of the negative frequency components of $u(t)$ by $+90$ and the phase of the positive frequency components by -90. And $i \cdot H(u)(t)$ has the effect of restoring the positive frequency components while shifting the negative frequency ones an additional $+90$, resulting in their negation [14]. By consequence, the Hilbert transform can be seen as a time-domain convolution of the signal with the function $\frac{1}{\pi t}$.

According to [15] the analytic signal it is easily obtained by passing in the frequency-domain performing the FFT, multiplying it with a custom function (Equation 6.11), and then returning in time-domain with the $IFFT$.

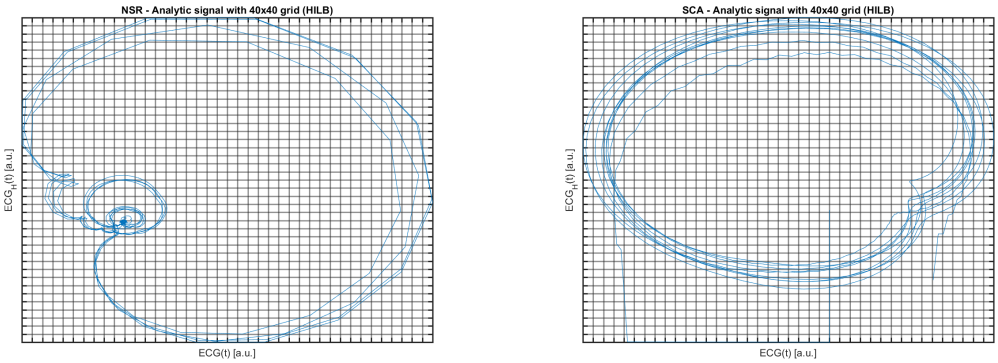
$$Z[m] = \begin{cases} X\,[0] & m = 0 \\ 2X\,[m] & 1 \leqslant m \leqslant \frac{N}{2} - 1 \\ X\left[\frac{N}{2}\right] & m = \frac{N}{2} \\ 0 & \frac{N}{2} + 1 \leqslant m \leqslant N - 1 \end{cases} \tag{6.11}$$

The analytic signal obtained in this way is showed in Figure 6.16. On the x-axis is plotted the real part, whilst the y-axis contain the imaginary one. Then as mentioned before, the plot is evaluated similarly to the PSR. The only difference is the threshold.



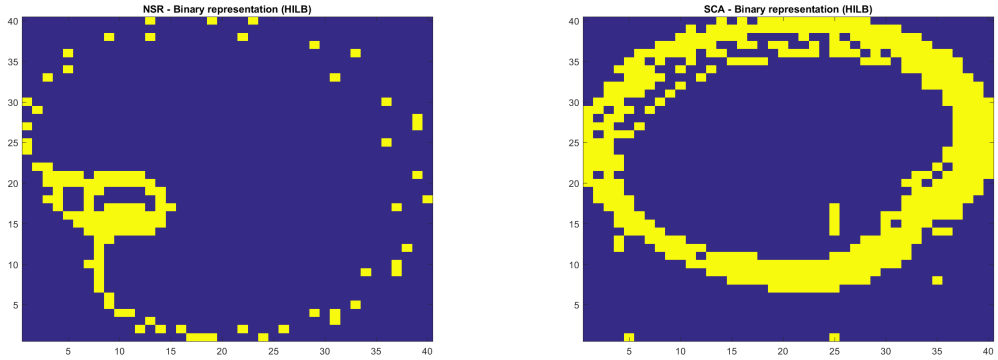
(a) (b)

Figure 6.16: HTA: Phase space diagram. (a) Healthy ECG; (b) Pathological ECG



(a) (b)

Figure 6.17: HTA: Phase space diagram with the 40x40 grid. (a) Healthy ECG; (b) Pathological ECG



(a) (b)

Figure 6.18: HTA: Binary representation of the phase space. (a) Healthy ECG; (b) Pathological ECG

6.2.6 Results

The algorithms were tested in Matlab on a custom dataset of pathological signals obtained from PhysioNet [1]. In addition, also some NSR signals obtained directly using the Control Board were used to additionally validate their specificity. In Figure 6.19 there is the signal 420 of *VFDB*, with the combined results of the algorithms superimposed. In that figure in the y-axis there is a scale from 0 to 5 where: for the individual algorithms a 0 means NSR, and 5 means a pathological segment; whilst for the combined results the scale indicates the total algorithms that detected a pathological rhythm. For this purpose we remark that SCA is only asserted if this value is higher than 2 in at least two ECG segments over the last three.

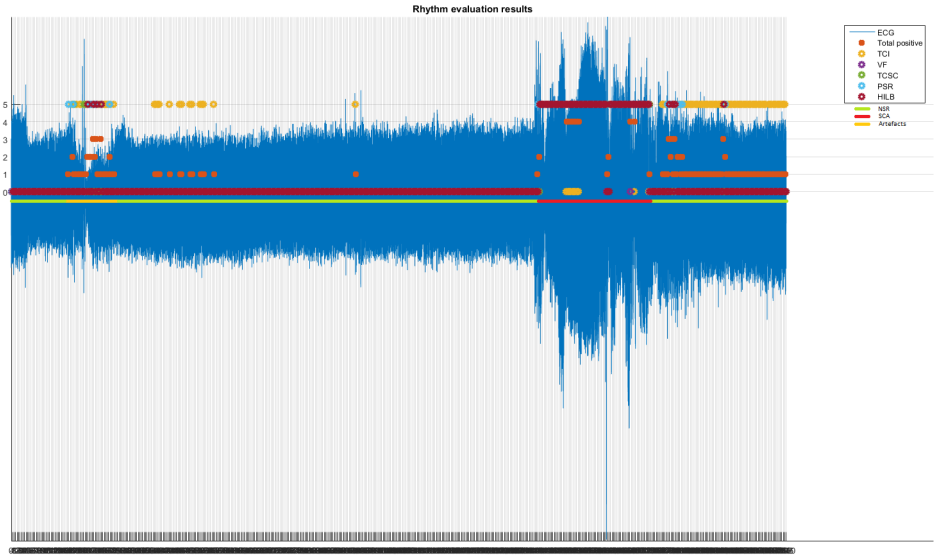


Figure 6.19: Algorithms results for signal 420.

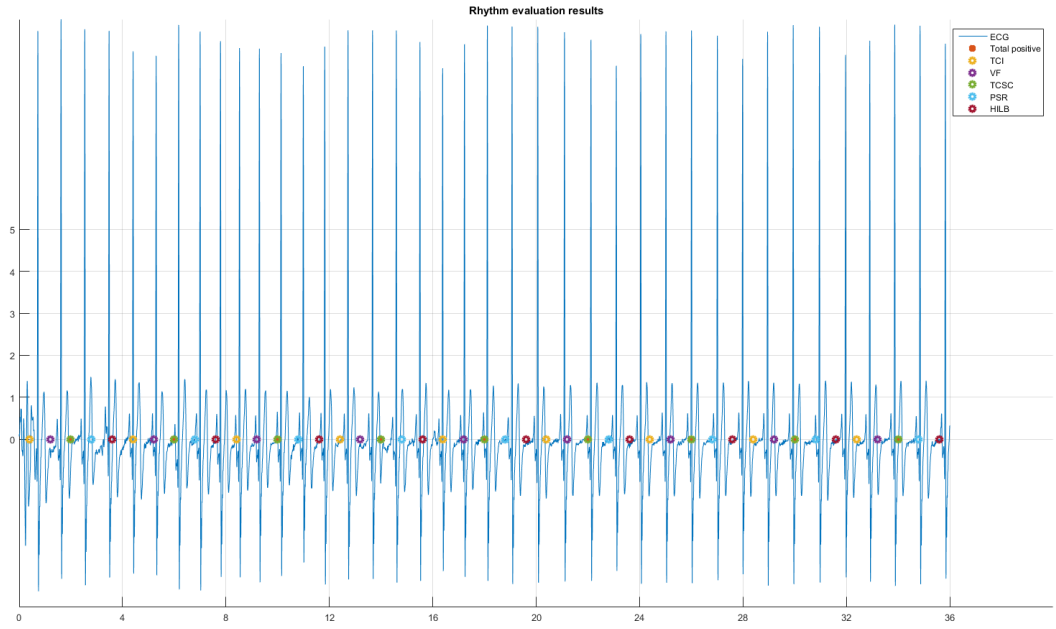


Figure 6.20: Algorithms results for a signal obtained using OAED.

In Figure 6.23 instead, there are the individual results of the five algorithms with their thresholds. From both Figure 6.19 and Figure 6.23 is evident how the results are improved in terms of specificity and sensibility when more algorithm is used to assert SCA. Another important note is the presence of artefacts, probably due to electrodes movement which briefly trigger the algorithms. Albeit the norm 60601-2-4 explicitly says that the algorithms should be evaluated in absence of those artefacts, I personally think that they are an important issue when evaluating a patient ECG.

Therefore the algorithms should be designed in order to recognise the presence of artefacts, or at least minimize their effects to avoid erroneous decisions. As we can see from the plots, the algorithms used in OAED are not entirely capable of this. Furthermore the TCI algorithm lead to the worst results, both in presence and absence of artefacts, hence a future version of OAED may swap this algorithm for a more efficient one.

In Figure 6.20 instead, there are the algorithms results for a signal obtained using OAED on a healthy patient (me). Finally, in Figure 6.24 there are the individual results of the algorithms.

6.2.7 Comparison with the results obtained using OAED

Although very similar, the algorithms implementation on OAED is not exactly the same as its Matlab counterpart. The necessary optimizations and adaptation of the code made to better suit the PSoC, led to subtle differences that altered the algorithms results. In particular, the TCI and TCSC are capable of using more ECG data, accessing also the last 4 second windows. Albeit this was initially seen as an improvement, and it works just as intended for TCSC; the TCI was affected by raising its false positive count. The results of the five algorithms executed by OAED are in Figure 6.21 and Figure 6.22.

These test also evinced that the five algorithms are all executed with an average time of $553.5ms$ and a standard deviation of $2.8ms$. This leave the PSoC more than 3 seconds of spare time. In future it could be possible to use this to add more algorithms, or more sophisticated ones.

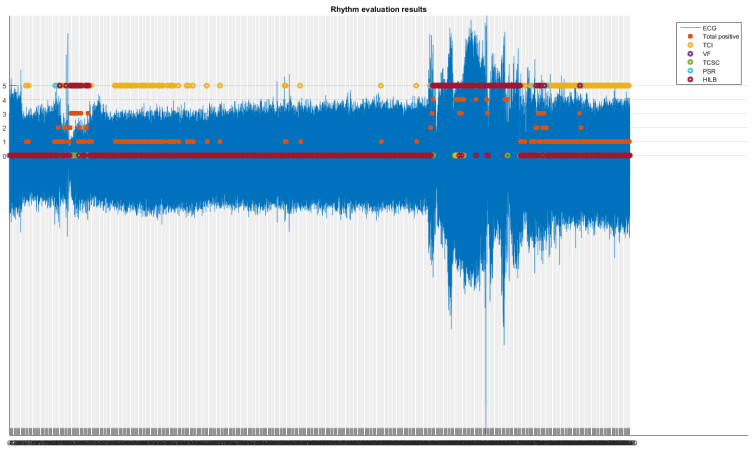


Figure 6.21: OAED results for signal 420.

A note about the datasets used. The norm 60601-2-4 explicitly indicates that the data used to test algorithms during the development phase cannot

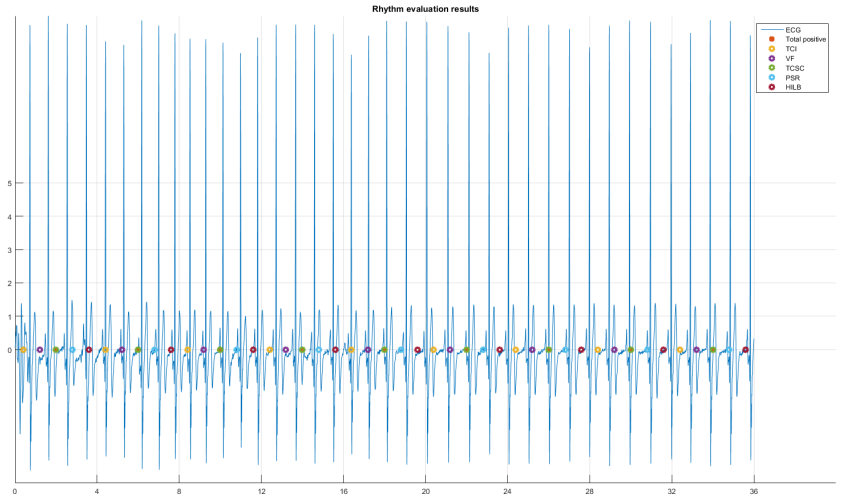


Figure 6.22: OAED results for a signal obtained using OAED.

be used to validate them. Furthermore the dataset should be chosen according to [16] in order to include at least: 250 samples of shockable data, of which 200 of coarse VF data, 50 samples of rapid VT; and 300 samples non-shockable data, composed of: 100 samples of NSR; 30 samples of other arrhythmias; 100 asystole samples; and finally 25 samples of both fine VF and other VT.

In this context a sample is an amount of data required to make a single shock/non-shock decision. Namely, a sample for OAED consist in three 4 seconds ECG segments, for a total of 12 seconds signal. Albeit PhysioNet provides an remarkable amount of data, it is not sufficient to satisfy the massive amount required by the 60601-2-4. In addition, I had to use part of the data during the development phase to validate and correct their operations. Therefore all the results presented in this chapter -and by consequence in the conclusions- are not sufficient to comply the norm. Further tests are required, perhaps even with the use of a signal simulator.

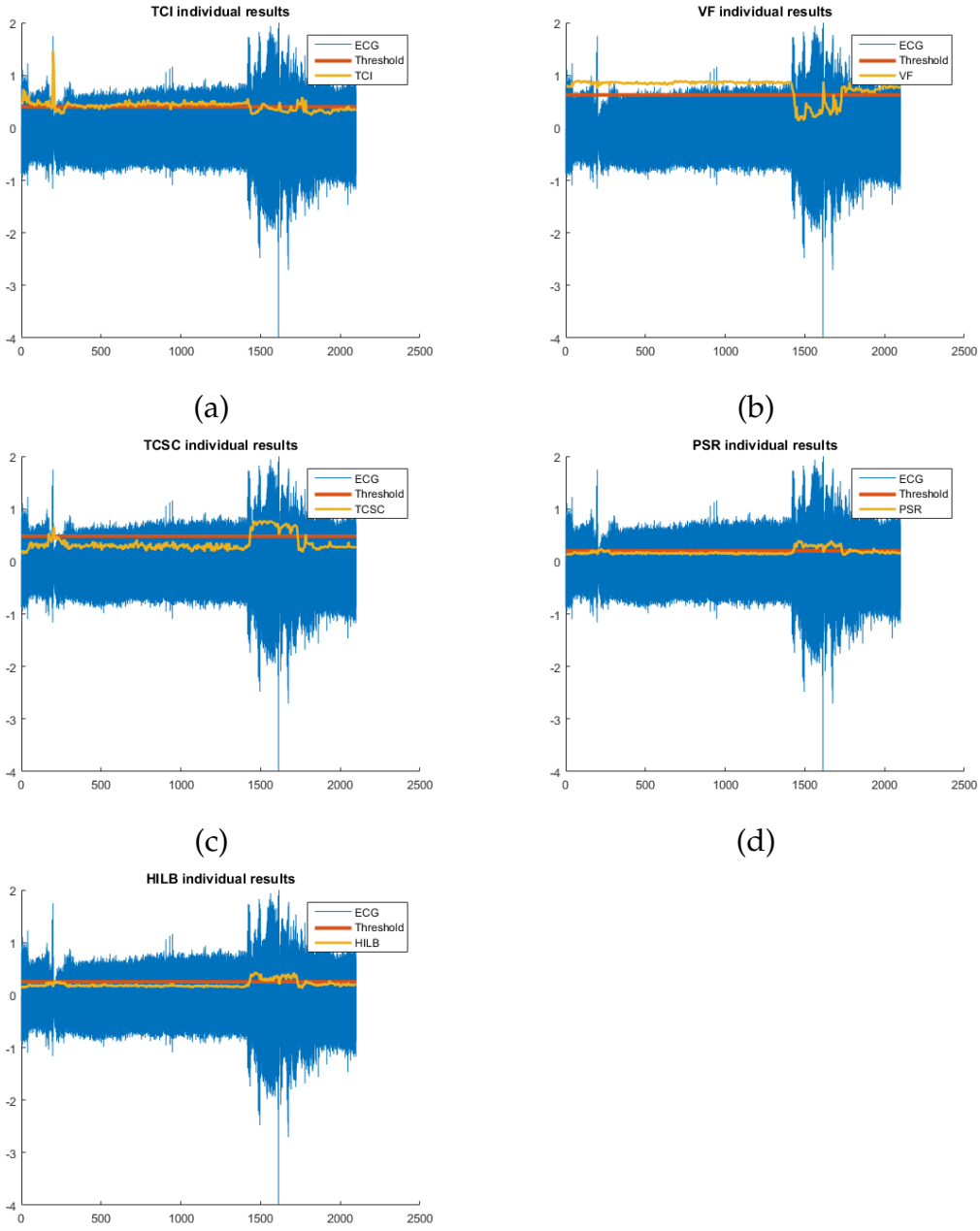


Figure 6.23: Individual algorithms results for signal 420. (a) TCI; (b) VFf; (c) TCSC; (d) PSR; (e) HTA

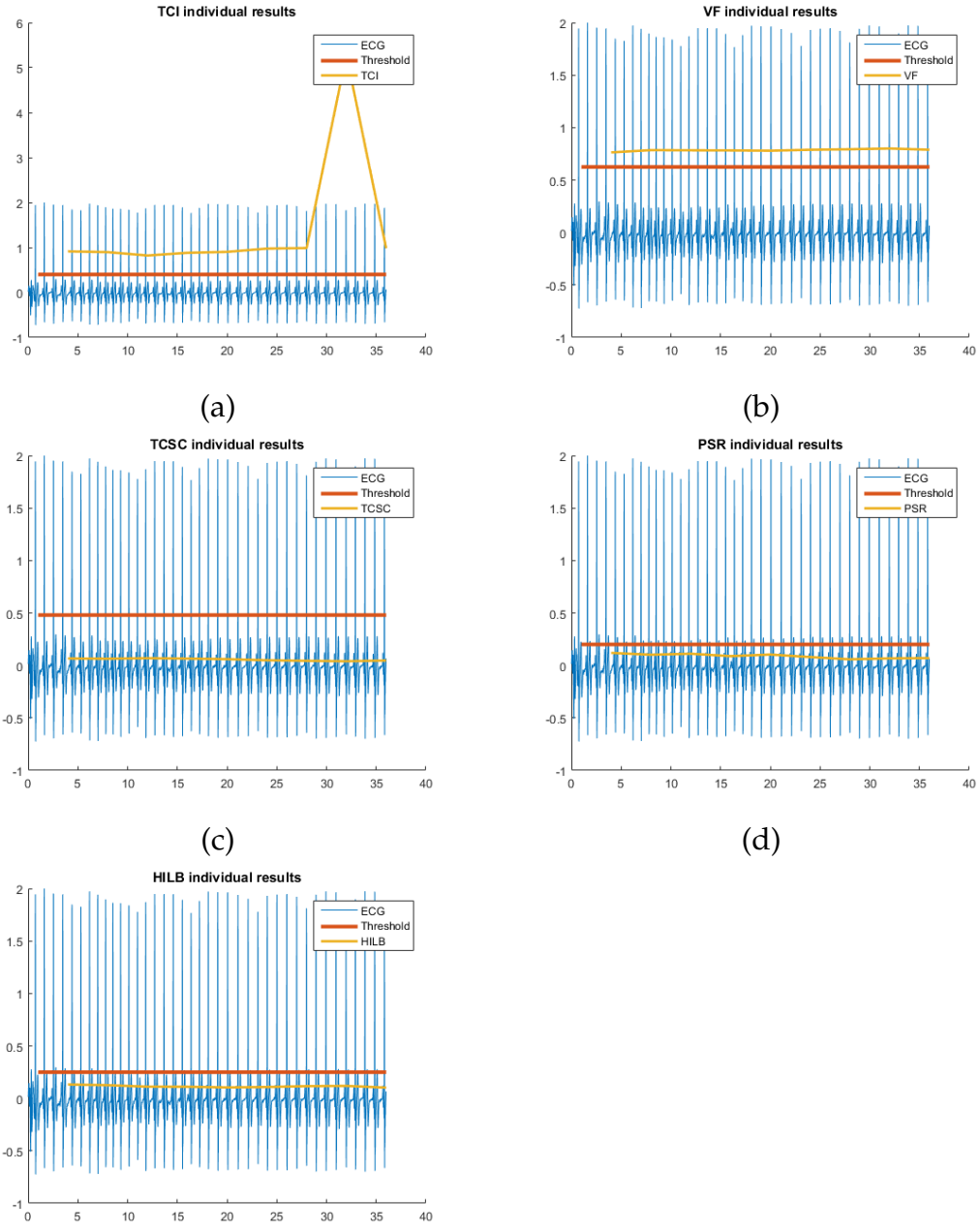


Figure 6.24: Individual algorithms results for a signal obtained using OAED. (a) TCI; (b) VFf; (c) TCSC; (d) PSR; (e) HTA

6.3 Software used

In this chapter the software used is the same I have described in the previous chapters. However, even though it is not a software, a special mention should be done on PhysioNet.

6.3.1 PhysioNet

PhysioNet offers free web access to large collections of recorded physiologic signals (PhysioBank) and related open-source software (PhysioToolkit). PhysioBank databases are made available under the ODC Public Domain Dedication and License v1.0. PhysioBank is a massive archive of physiological data. It contains over 90,000 recordings, or over 4 terabytes of digitized physiologic signals and time series, organized in over 80 databases.

In this context, a database is simply a collection of recordings (records), available as a set of flat files. In contrast to typical relational databases, PhysioBank databases consist of relatively small numbers (tens to thousands) of records that may each be quite large (in some PhysioBank databases, the size of a record can be a gigabyte or more, although typical record sizes are a few Mb) [1].

Each database consists of a set of records, identified by the record name. In most cases, a record consists of at least three files, which are named using the record name followed by distinct suffixes that indicate their contents. For example, the MIT-BIH Arrhythmia Database includes record 100; the three files 100.atr, 100.dat, and 100.hea together comprise record 100. Almost all records include a binary .dat (signal) file, containing digitized samples of one or more signals; these files can be very large. The .hea (header) file is a short text file that describes the signals (including the name or URL of the signal file, storage format, number and type of signals, sampling frequency, calibration data, digitizer characteristics, record duration and starting time). Most records include one or more binary annotation files (in the example, .atr denotes an annotation file). Annotation files contain sets of labels, each of which describes a feature of one or more signals at a specified time in the record; 100.atr, for example, contains an annotation for each QRS complex in the recording, indicating its location and type (normal, ventricular ectopic, etc.), as well as other annotations that indicate changes in the predominant cardiac rhythm and in the signal quality. In other databases, annotations mark other features of the signals.

In addition to the data, PhysioBank integrates a series of useful tools for finding, downloading, and visualizing it.

Bibliography

[1] *Phisionet, Phisiobank database*. https://www.physionet.org/physiobank/database/.

[2] N. V. Thakor, Y. S. Zhu, and K. Y. Pan. "Ventricular tachycardia and fibrillation detection by a sequential hypothesis testing algorithm". In: *IEEE Transactions on Biomedical Engineering* 37.9 (Sept. 1990), pp. 837–843. ISSN: 0018-9294. DOI: 10.1109/10.58594.

[3] S. Kuo and R. Dillman. "Computer detection of ventricular fibrillation". In: *Computers in Cardiology, IEEE Computer Society* (1978), pp. 347–349.

[4] S. Chen, N. V. Thakor, and M. M. Mower. "Ventricular fibrillation detection by a regression test on the autocorrelation function". In: *Medical and Biological Engineering and Computing* 25.3 (1987), pp. 241–249. ISSN: 1741-0444. DOI: 10.1007/BF02447420. URL: http://dx.doi.org/10.1007/BF02447420.

[5] S. Barro et al. "Algorithmic sequential decision-making in the frequency domain for life threatening ventricular arrhythmias and imitative artefacts: a diagnostic system". In: *J Biomed Eng* 11.4 (July 1989), pp. 320–328.

[6] A. Amann, R. Tratnig, and K. Unterkofler. "Reliability of old and new ventricular fibrillation detection algorithms for automated external defibrillators". In: *Biomed Eng Online* 4 (Oct. 2005), p. 60.

[7] Aijun Fan, Peng Han, and Bin Liu. "Shockable Rhythm Detection Algorithms for Electrocardiograph Rhythm in Automated Defibrillators". In: *AASRI Procedia* 1 (2012), pp. 21–26. ISSN: 2212-6716. DOI: http://dx.doi.org/10.1016/j.aasri.2012.06.005. URL: http://www.sciencedirect.com/science/article/pii/S2212671612000066.

[8] R. H. Clayton, A. Murray, and R. W. F. Campbell. "Comparison of four techniques for recognition of ventricular fibrillation from the surface ECG". In: *Medical and Biological Engineering and Computing* 31.2 (1993), pp. 111–117. ISSN: 1741-0444. DOI: 10.1007/BF02446668. URL: http://dx.doi.org/10.1007/BF02446668.

[9] Muhammad Abdullah Arafat, Abdul Wadud Chowdhury, and Md. Kamrul Hasan. "A simple time domain algorithm for the detection of ventricular fibrillation in electrocardiogram". In: *Signal, Image and Video Processing* 5.1 (2011), pp. 1–10. ISSN: 1863-1711. DOI: 10.

1007/s11760-009-0136-1. URL: http://dx.doi.org/10.1007/s11760-009-0136-1.

[10] J. Pan and W. J. Tompkins. "A Real-Time QRS Detection Algorithm". In: *IEEE Transactions on Biomedical Engineering* BME-32.3 (Mar. 1985), pp. 230–236. ISSN: 0018-9294. DOI: 10.1109/TBME.1985.325532.

[11] A. Amann, R. Tratnig, and K. Unterkofler. "Detecting Ventricular Fibrillation by Time-Delay Methods". In: *IEEE Transactions on Biomedical Engineering* 54.1 (Jan. 2007), pp. 174–177. ISSN: 0018-9294. DOI: 10.1109/TBME.2006.880909.

[12] A. Amann, R. Tratnig, and K. Unterkofler. "A new ventricular fibrillation detection algorithm for automated external defibrillators". In: *Computers in Cardiology, 2005*. Sept. 2005, pp. 559–562. DOI: 10.1109/CIC.2005.1588162.

[13] J. Felblinger et al. "Real time detection of ventricular fibrillation and its use in a semi-automatic defibrillator". In: *1992 14th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*. Vol. 6. Oct. 1992, pp. 2442–2443. DOI: 10.1109/IEMBS.1992.5761531.

[14] *Hilbert Transform, wikipedia*. https://en.wikipedia.org/wiki/Hilbert_transform.

[15] S. Lawrence Marple, Jr. "Computing the Discrete-Time "Analytic" Signal via FFT". In: *Signal Processing, IEEE Transactions on* 47.9 (Sept. 1999), pp. 2600–2603. ISSN: 1053-587X. DOI: 10.1109/78.782222.

[16] R. E. Kerber et al. "Automatic external defibrillators for public access defibrillation: recommendations for specifying and reporting arrhythmia analysis algorithm performance, incorporating new waveforms, and enhancing safety. A statement for health professionals from the American Heart Association Task Force on Automatic External Defibrillation, Subcommittee on AED Safety and Efficacy". In: *Circulation* 95.6 (Mar. 1997), pp. 1677–1682.

CHAPTER 7

Conclusions

Summing up what has been done in this thesis work, one of the UBORA project goals is to ease the diffusion of open source, compliant, safe, and affordable devices.

Keeping in mind the philosophy and the principles of the UBORA project then, i have made a research to assess what are the most impending and critical needs, with particular regards on the developing countries perspective. Amongst the various health problems that don't have a well known open source solution, i dedicated my attention to the Sudden Cardiac Arrest (SCA).

The SCA is treated using defibrillators, and more specifically, Automated External Defibrillators (AED). The peculiarity of these devices is that they can be used with the minimum training possible, opening the chance to a capillary diffusion and the possibility of being used by bystanders. AEDs however, are more sophisticated devices when compared to standard defibrillators, due to the necessity of automatically diagnose SCA, united to their relative ease of use.

Once identified the thesis goal, i organized the work in three main steps: problem analysis, regulatory analysis, and design.

During the problem analysis i did a research about the state of the art of the others commercially available AEDs, as well as defibrillation and SCA in general. From this step i have obtained a mean of comparison for the performance of my AED; a model for the defibrillation, namely a standard RC circuit; a model for the biphasic defibrillation; and finally some basic, but very important information regarding the SCA, and namely the Ventricular Tachycardia (VT) and Ventricular Fibrillation (VF).

The problem analysis has been followed by the regulatory analysis. In this phase i identified the class of the device and the required conformity standards. The law of reference is the EEC medical devices directive 93/42 (MDD 93/42). From that i extracted the operations to perform in order to obtain the CE marking, crucial in the case of commercial devices. The MDD 93/42 indicates that AEDs are class IIb devices, and with that they need a notified body to verify them during the design and production phase. In

order to design a compliant device i have then searched the AEDs related standards. These are mainly in the family of IEC 60601, in which the most important are the: 60601-1, which is about the general requirements for basic safety and Essential Performance of medical electrical equipments and systems; and the 60601-2-4, that contain the particular requirements for the safety of cardiac defibrillators and cardiac defibrillators monitors. The rules resulting from the analysis of these norms represents the Essential Requirements (ERs). ERs are the foundation from which the actual AED design process will start.

The third and final step, is the design of the device. I started with a conceptual scheme. In this, there are two separate main boards: the High-Voltage Board (HV-B), and the Control Board (C-B).

The first contain the circuitry necessary to perform defibrillation. This includes: a capacitor, used to store the energy to release; a charging circuit, that rapidly charges the capacitor; an H-Bridge circuit, used to perform biphasic -or polyphasic- defibrillation; an internal discharge circuit, used to dump the unused residual energy; and two selectors, used to isolate the patient from the capacitor, and to route the ECG signal to the C-B.

The second board instead, can be seen as the device *brain*. It contain a Programmable System on Chip (PSoC), which -amongst other functions- integrates the whole analogical front-end used to obtain the Electrocardiogram (ECG) and impedance acquisitions. The PSoC also has inside an ARM CPU which is used to analyse the signals and assert the defibrillation needs.

The PSoC finally, has been programmed with a specific firmware that i wrote. This work as a finite-state machine. Each state depends on the occurrence of specific events, and is used to enable/disable certain circuits. The firmware also integrates five different algorithms for SCA recognition. These have been evaluated and tested using a special dataset of pathological ECG obtained from the PhysioNet database.

The whole designing phase has been done following the ERs defined in the regulatory analysis. Albeit the design is mostly compliant with the standards defined by the 60601, there are two exceptions. The first is absence of tests on the HV-B, because it was not built; and the second is the necessity more data to verify the SCA recognition algorithms, due to the fact that the norm require a huge amount of data that was not readily available.

Therefore the possible future developments for OAED are: a compliance test of the physical HV-B; the verification of SCA recognition algorithms, and the integration of more efficient ones according to the results.

Annex A - Firmware source code

main.c

```
/* ======================================
 *
 * OPEN SOURCE AED
 * Main
 *
 * ======================================
*/

#include <project.h>
#include "OAED_Common.h"

int main(){
   /* Local variables declaration */
   char Status = measurement_mode;

   /* System initialization */
   OAED_Init();

   /* Main Loop */
   for(;;){
      /* Enable or disable specific state functionalities */
      OAED_SetSystemStatus(Status);

      /* Check for pc commands */
      OAED_USBGetCommand();

      switch(Status){
         case lead_off:
            Status = OAED_LeadOffMode();
            continue;
         case measurement_mode:
            Status = OAED_MeasurementMode();
            continue;
         case charging_capacitor:
            Status = OAED_ChargingMode();
            continue;
         case discharge_enabled:
            Status = OAED_DischargeEnabledMode();
            continue;
```

```
40          case internal_discharge:
41             Status = OAED_InternalDischargeMode();
42             continue;
43          default:
44             /* This shouldn't happen. */
45             Status = lead_off;
46             continue;
47       }
48    }
49 }
50
51 /* [] END OF FILE */
```

OAED_Acquisition.h

```
1  /* ========================================
2   *
3   * OPEN SOURCE AED
4   * This header contains all acquisition
5   * functions prototyes and global variable
6   * declarations.
7   *
8   * ========================================
9  */
10
11 #ifndef OAED_ACQUISITION_H
12 #define OAED_ACQUISITION_H
13
14 /* Include */
15 #include <project.h>
16 #include "OAED_Common.h"
17 /* End of include */
18
19 /* Macro */
20 /* Start ECG acquisition DMA. */
21 #define OAED_AcquisitionECGStart() OAED_DMAECGStart()
22 /* Stop ECG acquisition DMA, call this function only in
       lead-off mode. */
23 #define OAED_AcquisitionECGStop() OAED_DMAECGStop()
24 /* End of macro */
25
26 /* Function prototypes */
27 void OAED_AcquisitionInit();
28
29 bool OAED_AcquisitionECGUnpause();
30 inline void OAED_AcquisitionECGPause();
31
```

```
32 void OAED_AcquisitionZ();
33 /* End of function prototypes */
34
35 #endif
36 /* [] END OF FILE */
```

OAED_Common.h

```
1  /* ========================================
2   *
3   * OPEN SOURCE AED
4   * This header contains all common
5   * functions prototyes and global variable
6   * declarations.
7   *
8   * ========================================
9  */
10
11 #ifndef OAED_COMMON_H
12 #define OAED_COMMON_H
13
14 /* Compilation Options */
15 #define OAED_TIME       true      // Enable or disable time
16                                   // functionalities.
17 #define RAW_MODE        true      // Enable or disable raw
      ECG
18                                   // acquisition.
19 /* End of compilation options */
20
21 /* Include */
22 #include <project.h>
23 #include <stdbool.h>
24 #include <math.h>
25 #include "arm_math.h"
26 #include "arm_const_structs.h"
27
28 #include "OAED_Acquisition.h"
29 #include "OAED_Defibrillation.h"
30 #include "OAED_DMA.h"
31 #include "OAED_ECGAlgorithms.h"
32 #include "OAED_ISR.h"
33 #include "OAED_SystemStatus.h"
34 #include "OAED_USB.h"
35 #include "OAED_Wait.h"
36 #if(OAED_TIME)
37    #include "OAED_Time.h"
38 #endif
```

```
39 /* End of include */
40
41 /* Numeric constants */
42 /* Events */
43 #define EVENT_NO        3         // Number of event
      registered
44 #define POSITIVE_EVENT_NO 2       // Number of positive
      event
45                                   // required for
                                        defibrillation
46 /* ADC */
47 #define ADC_BUFFER_GAIN 2         // ADC Delta Sigma buffer
      gain
48 /* Cache */
49 #define ECG_CACHE_SIZE  8         // ECG cache size
50 #define RAW_CACHE_SIZE  8         // RAW cache size
51 /* ECG */
52 #define ECG_SIGNAL_LENGTH 4       // Seconds of signal
      registered [s]
53 #define ECG_SAMPLING_F  500       // Sampling frequency of
      ECG signal
54                                   // [Hz]
55 #define ECG_DATA_SIZE   (ECG_SIGNAL_LENGTH * ECG_SAMPLING_F)
56                                   // Size of ECG
                                        data/buffer vectors
57 /* Impedance */
58 #define Z_SIGNAL_LENGTH 1         // Seconds of signal
      registered [s]
59 #define Z_SAMPLING_F    4000      // Sampling frequency of
      Z signal
60                                   // [Hz]
61 #define Z_SIGNAL_F      250       // Excitation signal
      frequency [Hz]
62 #define Z_PERIOD        (Z_SAMPLING_F / Z_SIGNAL_F)
63                                   // No of samples per
64 #define Z_DATA_SIZE     (Z_SIGNAL_LENGTH * Z_SAMPLING_F)
65                                   // Size of Z data/buffer
                                        vector
66 #define Z_MIN           25        // Minimum impedance [Ohm]
67 #define Z_MAX           180       // Maximum impedance [Ohm]
68 #define IMPEDANCE_DEVIATION 0.50  // Maximum deviation for
      impedance
69                                   // measurement
70 /* RAW */
71 #if(RAW_MODE)
72 #define RAW_DATA_SIZE   4000
73 #endif
74
75 /* Defibrillation signal */
```

```
76 #define ALARM_TIME      1500      // Duration of pre-defib.
      alarm [ms]
77
78 /* Hardware */
79 #define C         (double) 0.00015 // Condensator capacity
      [F]
80 #define V         (double) 1700    // Maximum voltage [V]
81 #define U         (double) 200     // Target energy to
      deliver [J]
82 #define U_MAX     (double) 216     // Maximum storable
      energy [J]
83
84 /* End of numeric constants */
85
86 /* Macro */
87 #define OAED_PINCONTROL(on,pin)((on) ? CyPins_SetPin(pin) :
      CyPins_ClearPin(pin))
88 #define OAED_SWAP(a,b)({ \
89                  __typeof__(a) _a = (a); \
90                  a = b; \
91                  b = _a; \
92                })
93 //#define abs(a) (a<0) ? -a : a
94 /* End of macro */
95
96
97 /* Variable definitions */
98 // Cache
99 extern int16 CacheECG[];          // ECG cache
100 // Buffers
101 extern int16 BufferECG[];         // ECG Buffer
102 extern int16 OldECG[];            // Last ECG
103 extern int16 BufferZ[];           // Z Buffer
104 // Data
105 extern int16 DataECG[];           // ECG Data Vector
106 extern int16 DataZ[];             // Z Data Vector
107 // Impedance
108 extern double PatientImpedance;  // Patient impedance
109 // Event
110 extern uint8 EventCounter;
111 // Raw data
112 #if(RAW_MODE)
113    extern int16 DataRAW[];        // Raw ECG Data
114    extern int16 BufferRAW[];      // Raw ECG Buffer
115    extern int16 CacheRAW[];       // Raw Cache
116 #endif
117 /* End of variable definitions */
118
119 /* Declaration of system flags */
```

```
120 extern bool ECG_buffer_full;      // ECG Buffer Status
121 extern bool Z_buffer_full;        // Z Buffer Status
122 extern bool ECG_data_pending;     // New ECG data available
123 #if(RAW_MODE)
124    extern bool RAW_buffer_full;
125 #endif
126
127 extern bool ECG_enabled;          // ECG acquisition status
128 extern bool Z_enabled;
129
130 extern bool lead_detected;        // Lead Detected
131 extern bool capacitor_ready;      // Capacitor status
132
133 extern bool event_flags[];        // VT/VF Event Flags
134
135 extern bool Continuous_USBECG;    // Currently not working
       properly
136 extern bool Continuous_USBRAW;
137 /* End of system flags */
138
139 /* Function prototypes */
140 void OAED_Init();
141 void OAED_InitFilter();
142
143 void OAED_CopyECGBuffer();
144 void OAED_CopyZBuffer();
145
146 void OAED_Led(bool, bool, bool, bool);
147
148 void OAED_ResetEvent();
149 bool OAED_CheckFlags();
150
151 bool OAED_EvaluateRhythm();
152
153 bool OAED_EvaluateImpedanceAC();
154 bool OAED_EvaluateImpedanceDC();
155 bool OAED_ValidateImpedance(double);
156
157 void OAED_EnableChargingCircuit();
158 void OAED_DisableChargingCircuit();
159 /* End of function prototypes */
160
161 #endif
162 /* [] END OF FILE */
```

OAED_Defibrillation.h

```
/* ========================================
 *
 * OPEN SOURCE AED
 * This header contains all defibrillation
 * related function prototypes and variable
 * declarations.
 *
 * ========================================
*/

#ifndef OAED_DEFIBRILLATION_H
#define OAED_DEFIBRILLATION_H

/* include */
#include <project.h>
#include <stdbool.h>
#include <math.h>
#include "OAED_Common.h"
/* End of include */

/* Numeric constants */
#define OPEN_CIRCUIT (uint8)(0b00000000)
#define PHI_1     (uint8)(0b00000010)
#define PHI_2     (uint8)(0b00000001)

#define PWM_LENGTH 5000                                  // [us]
#define PWM_DUTY  50                                     // [%]
#define PWM_ON    (uint16)( PWM_LENGTH * PWM_DUTY / 100 )
#define PWM_OFF   (uint16)( PWM_LENGTH - PWM_ON )

#define OAED_DISCHARGE_TIME (- Patient_impedance * C * log(
    1 - 2 * U/U_MAX ))
                                                         // [s]
#define OAED_DISCHARGE_TIME_MS (1000 * OAED_DISCHARGE_TIME)
    // [ms]
#define OAED_DISCHARGE_TIME_US (1000 *
    OAED_DISCHARGE_TIMEms) // [us]
/* End of numeric constants */

/* Macro */
#define OAED_HBridgeControl(phase) (Phase_Reg_Write( phase
    ))
/* End of Macro*/

/* Function prototypes */
void OEAD_EnableDefibrillation(); // Need new name
void OAED_DisableDefibrillation(); // Same

inline void OAED_ArmDefibrillator();
```

```
46 inline void OAED_DisarmDefibrillator(bool);
47
48 uint32 OAED_EvaluateDischargeTime();
49
50 void OAED_InternalDischarge();
51 void OAED_MonophasicDefibrillation();
52 void OAED_BiphasicDefibrillation(uint8);
53 void OAED_PolyphasicDefibrillation(uint8);
54 /* End of function prototypes */
55
56 #endif
57
58
59
60 /* [] END OF FILE */
```

OAED_DMA.h

```
1  /* ========================================
2   *
3   * OPEN SOURCE AED
4   * This header contains all DMA related
5   * function prototypes and variable
6   * declarations.
7   *
8   * ========================================
9  */
10
11 #ifndef OAED_DMA_H
12 #define OAED_DMA_H
13
14 /* Include */
15 #include <project.h>
16 #include "OAED_Common.h"
17 /* End of Include*/
18
19 /* Numeric constants */
20 /* Defines for DMA_DelSig */
21 #define DMA_DelSig_ECG_BYTES_PER_BURST 2
22 #define DMA_DelSig_ECG_REQUEST_PER_BURST 1
23 #define DMA_DelSig_ECG_SRC_BASE (CYDEV_PERIPH_BASE)
24 #define DMA_DelSig_ECG_DST_BASE (CYDEV_PERIPH_BASE)
25
26 /* Defines for DMA_DelSig_Z */
27 #define DMA_DelSig_Z_BYTES_PER_BURST 2
28 #define DMA_DelSig_Z_REQUEST_PER_BURST 1
29 #define DMA_DelSig_Z_SRC_BASE (CYDEV_PERIPH_BASE)
```

```
#define DMA_DelSig_Z_DST_BASE (CYDEV_PERIPH_BASE)

#if(RAW_MODE)
/* Defines for DMA_DelSig_RAW */
#define DMA_DelSig_RAW_BYTES_PER_BURST 2
#define DMA_DelSig_RAW_REQUEST_PER_BURST 1
#define DMA_DelSig_RAW_SRC_BASE (CYDEV_PERIPH_BASE)
#define DMA_DelSig_RAW_DST_BASE (CYDEV_SRAM_BASE)
#endif

/* Defines for DMA_Filter_ECG */
#define DMA_Filter_ECG_BYTES_PER_BURST 2
#define DMA_Filter_ECG_REQUEST_PER_BURST 1
#define DMA_Filter_ECG_SRC_BASE (CYDEV_PERIPH_BASE)
#define DMA_Filter_ECG_DST_BASE (CYDEV_SRAM_BASE)

/* Defines for DMA_Filter_Z */
#define DMA_Filter_Z_BYTES_PER_BURST 2
#define DMA_Filter_Z_REQUEST_PER_BURST 1
#define DMA_Filter_Z_SRC_BASE (CYDEV_PERIPH_BASE)
#define DMA_Filter_Z_DST_BASE (CYDEV_SRAM_BASE)
/* End of numeric constants */

/* Function prototypes */
void OAED_DMA_Init();
void OAED_DMAECGStart();
void OAED_DMAECGStop();
void OAED_DMAZStart();
void OAED_DMAZStop();
/* End of Function prototypes */

#endif
/* [] END OF FILE */
```

OAED_ECGAlgorithms.h

```
/* ========================================
 *
 * OPEN SOURCE AED
 * This header contains all ECG pattern
 * recognition algorithms functions
 * prototyes and global variable
 * declarations.
 *
 * ========================================
*/

```

```
12
13 #ifndef OAED_ECGALGORITHMS_H
14 #define OAED_ECGALGORITHMS_H
15
16 /* Include */
17 #include <project.h>
18 #include <math.h>
19 #include <stdlib.h>
20 #include "OAED_Common.h"
21 /* End of include */
22
23 /* Numeric constants */
24 //TCI
25 #define TCI_THRESH      0.2       // TCI binarization
      threshold
26 #define TCI_CRIT        400       // TCI critical value [ms]
27
28 #define TCI_MU_VF       105       // Maan TCI for VF [ms]
29 #define TCI_SIGMA_VF    6.5       // Standard deviation for
      VF [ms]
30 #define TCI_SIGMA2_VF   (TCI_SIGMA_VF * TCI_SIGMA_VF)
31
32 #define TCI_MU_VT       220       // Maan TCI for VT [ms]
33 #define TCI_SIGMA_VT    16.5      // Standard deviation for
      VT [ms]
34 #define TCI_SIGMA2_VT   (TCI_SIGMA_VT * TCI_SIGMA_VT)
35
36 #define TCI_ALFA        0.01      // Alfa
37 #define TCI_BETA        0.01      // Beta
38                                   // The next are just
                                        umeric
39                                   // constants
                                        pre-calculated to ease
40                                   // real-time evaluations.
41 #define TCI_F_VT        (2 * log((1-TCI_BETA) / TCI_ALFA))
42 #define TCI_F_VF        (2 * log(TCI_BETA / (1-TCI_ALFA)))
43 #define TCI_F_VX        (2 * log(TCI_SIGMA_VT /
      TCI_SIGMA_VF))
44
45 // VF filter
46 #define VFf_CRIT        0.625    // Critical leakage
47
48 // TCSC
49 #define TCSC_Ls         3        // Segment width [s]
50 #define TCSC_NS         (ECG_SIGNAL_LENGTH - TCSC_Ls + 1)
51                                  // Number of segments in
                                        one window
52 #define TCSC_NTS        (ECG_SIGNAL_LENGTH + TCSC_NS)
53                                  // Total number of segments
```

```
54 #define TCSC_THRESH     0.2      // TCSC binarization
      threshold
55 #define TCSC_CRIT       0.48     // TCSC N critical value
56                                  // 48% is high specificity
                                       setting
57                                  // for more sensibility it
                                       is
58                                  // possible to use 25% <
                                       CRIT < 35%
59
60 // PSR
61 #define PSR_TAU         0.5      // Tau [s]
62 #define PSR_D0          0.20     // Critical value
63 #define PSR_GRID_N      40       // Grid divisions
64 #define PSR_TAU_N       (0.5 * ECG_SAMPLING_F)
65
66 // HILB
67 #define HILB_D0         0.25     // Critical value
68 #define HILB_GRID_N     40       // Grid divisions
69
70 // FFT
71 #define FFTN            2048
72 #define FORWARD_FFT     false
73 #define INVERSE_FFT     true
74
75 // SCA evaluation
76 #define OAED_POSITIVETHRESH 3    // Number of positive
      algorithms
77                                  // positive to assert SCA
78 /* End of numeric constants */
79
80 /* Macro */
81 #define OAED_TCSC_COSWIN(t) (0.5 * (1 - cos(4*PI*(t)) ))
82 /* End of macro */
83
84 /* Variable definitions */
85 /* End of variable definitions */
86
87 /* Function prototypes */
88 bool OAED_TCI();
89 bool OAED_VFfilter();
90 bool OAED_TCSC();
91 bool OAED_PSR();
92 bool OAED_HILB();
93
94 bool OAED_ECGAnalysis();
95
96 #endif
97
```

```
98 /* [] END OF FILE */
```

OAED_ISR.h

```
1  /* ========================================
2   *
3   * OPEN SOURCE AED
4   * This header contains all ISR related
5   * function prototypes and variable
6   * declarations.
7   *
8   * ========================================
9  */
10
11 #ifndef OAED_ISR_H
12 #define OAED_ISR_H
13
14 /* Include */
15 #include <project.h>
16 #include <stdbool.h>
17 #include "OAED_Defibrillation.h"
18 /* End of Include */
19
20 /* Macro */
21 /* RAW cache interrupt control */
22 #if(RAW_MODE)
23 #define OAED_ISRRAWENABLE() isr_CacheRAWReplenished_Enable()
24 #define OAED_ISRRAWDISABLE()
      isr_CacheRAWReplenished_Disable()
25 #endif
26 /* End of macro */
27
28 /* Custom ISR prototypes */
29 CY_ISR_PROTO(isr_BufferZRe);
30 CY_ISR_PROTO(isr_CacheECGRe);
31 CY_ISR_PROTO(isr_LeadOff);
32 CY_ISR_PROTO(isr_LeadOn);
33 CY_ISR_PROTO(isr_CapReady);
34 CY_ISR_PROTO(isr_CapLow);
35 CY_ISR_PROTO(isr_Defibrillation);
36 #if(RAW_MODE)
37 CY_ISR_PROTO(isr_CacheRAWRe);
38 #endif
39 /* End of ISR prototypes */
40
41 /* Function prototypes */
42 void OAED_ISRInit();
```

```
43 /* ECG cache interrupt control */
44 inline void OAED_ISRECGEnable();
45 inline void OAED_ISRECGDisable();
46 /* Z buffer interrupt control */
47 inline void OAED_ISRZEnable();
48 inline void OAED_ISRZDisable();
49 /* End of Function prototypes */
50
51 #endif
52
53
54 /* [] END OF FILE */
```

OAED_SystemStatus.h

```
1  /* ========================================
2   *
3   * OPEN SOURCE AED
4   * This header contains all system status
5   * functions prototyes and global variable
6   * declarations.
7   *
8   * ========================================
9  */
10
11 #ifndef OAED_SYSTEMSTATUS
12 #define OAED_SYSTEMSTATUS
13
14 /* Include */
15 #include <project.h>
16 #include "OAED_Common.h"
17 /* Include */
18
19 /* System Status */
20 enum system_status {
21    lead_off,                    // Lead-off
22    measurement_mode,            // Lead detected, but no
         VT/VF
23    charging_capacitor,          // VT/VF detected
24    discharge_enabled,           // VT/VF detected,
         capacitor ready
25    internal_discharge           // Safety mode
26 };
27 /* End of System status */
28
29 /* Function prototypes */
30 char OAED_LeadOffMode();
```

```
31 char OAED_MeasurementMode();
32 char OAED_ChargingMode();
33 char OAED_DischargeEnabledMode();
34 char OAED_InternalDischargeMode();
35
36 void OAED_SetSystemStatus(char);
37 /* End of Function prototypes */
38
39 #endif
40
41 /* [] END OF FILE */
```

OAED_Time.h

```
1  /* ========================================
2   *
3   * OPEN SOURCE AED
4   * This header contains all time
5   * functions prototyes and global variable
6   * declarations.
7   *
8   * ========================================
9  */
10
11 #ifndef OAED_TIME_H
12 #define OAED_TIME_H
13
14 /* Include */
15 #include <project.h>
16 #include <stdio.h>
17 #include "OAED_Common.h"
18 /* End of Include */
19
20 /* Global Variables */
21 extern uint16 mscount;
22 extern uint16 seccount;
23 extern uint16 mincount;
24 extern char TimeStamp[];
25 /* End of global variables */
26
27 /* Function prototypes */
28 void OAED_InitTime();
29 void OAED_SysTickISRCallback();
30 void OAED_GetTime();
31 /* End of function prototypes */
32
33 #endif
```

```
34
35 /* [] END OF FILE */
```

OAED_USB.h

```
1  /* ========================================
2   *
3   * OPEN SOURCE AED
4   * This header contains all USB
5   * functions prototyes and global variable
6   * declarations.
7   *
8   * ========================================
9  */
10
11 #ifndef OAED_USB_H
12 #define OAED_USB_H
13
14 /* Include */
15 #include <project.h>
16 #include <stdio.h>
17 #include "OAED_Common.h"
18 #if(OAED_TIME)
19    #include "OAED_Time.h"
20 #endif
21 /* End of include */
22
23 /* Numeric Constants */
24 #define USBFS_DEVICE 0     // Device
25 #define USBFS_BUFFER_SIZE 512 // Buffer Size (max 512)
26 #define OAED_USB_ECHO false // GetData echoes (Default
      false)
27 /* End of numeric constants */
28
29 /* Macro */
30 /* Send data as 8-bit array. LSB first. */
31 #define OAED_USBSendData(message)({\
32                   uint16 _n =
                        sizeof(message)/sizeof(message[0]);\
33                   int8 _tmp[3];\
34                   _tmp[0] = 8*(int8)(sizeof(message[0]));\
35                   _tmp[1] = (int8)(_n&0x00ff);\
36                   _tmp[1] = (int8)(_n>>8);\
37                   OAED_USBSendData8(_tmp, 3);\
38                   OAED_USBSendDataVoid(message, _n);\
39                })
40 #define OAED_USBSendData16(message, n)({\
```

```
41                      int8 _tmp[3];\
42                      _tmp[0] = (16);\
43                      _tmp[1] = n&0x00ff;\
44                      _tmp[2] = n>>8;\
45                      OAED_USBSendData8(_tmp, 3);\
46                      OAED_USBSendData8((int8*) message,
                           (uint16)(2*n));\
47                  })
48  #define OAED_USBSendData32(message, n)({\
49                      int8 _tmp[3];\
50                      _tmp[0] = (16);\
51                      _tmp[1] = n&0x00ff;\
52                      _tmp[2] = n>>8;\
53                      OAED_USBSendData8(_tmp, 3);\
54                      OAED_USBSendData8((int8*) message,
                           (uint16)(4*n));\
55                  })
56  /* End of Macro*/
57
58  /* Global variables */
59  /* End of global variables */
60
61  /* Function prototypes */
62
63     /* WARNING: Any Send/Print function is a blocking
           function. */
64
65  /* Basic functions */
66  void OAED_USBInit();
67  void OAED_USBConfigure();
68  void OAED_USBSendString(char[]);
69  void OAED_USBSendData8(int8[], uint16);
70  void OAED_USBSendDataVoid(void*, size_t);
71  bool OAED_USBGetCommand();
72
73  /* Send data as String. MOSTLY DEPRECATED. */
74  void OAED_USBPrintECG();
75  void OAED_USBPrintECGB();
76  void OAED_USBPrintZ();
77  void OAED_USBPrintSystemImage();
78  void OAED_USBPrintTimeStamp();
79
80  /* Interactive functions */
81  uint16 OAED_USBGetData(uint8[], bool);
82  void OAED_USBReceiveData(int16[], uint16);
83
84  /* Send data */
85  void OAED_USBSendSystemImage();
86  void OAED_USBSendECG();
```

```
87 void OAED_USBSendZ();
88 void OAED_USBSendBuffer();
89 inline int16 OAED_ShiftNAdd(int16,bool);
90
91 #if(RAW_MODE)
92 void OAED_USBSendRAW();
93 #endif
94 /* End of function prototypes */
95
96 #endif
97
98 /* [] END OF FILE */
```

OAED_Wait.h

```
1  /* ========================================
2   *
3   * OPEN SOURCE AED
4   * This header contains all waiting
5   * functions prototyes and global variable
6   * declarations.
7   *
8   * ========================================
9  */
10
11 #ifndef OAED_WAIT_H
12 #define OAED_WAIT_H
13
14 /* Include */
15 #include <project.h>
16 #include "OAED_Common.h"
17 /* End of include */
18
19 /* Function prototypes */
20 void OAED_WaitLeadOn();
21 bool OAED_WaitForData();
22 bool OAED_WaitForCap();
23 bool OAED_WaitForZ();
24 /* End of function prototypes */
25
26 #endif
27 /* [] END OF FILE */
```

OAED_Acquisition.c

```
/* ======================================
 *
 * OPEN SOURCE AED
 * This file contains all acquisition
 * function and global variable
 * definitions.
 *
 * ======================================
*/

#include "OAED_Acquisition.h"

/* Function declarations */
void OAED_AcquisitionInit(){

   /* Never stop DelSig ADC or Z DMA. */
   /* Power up and initialize DelSig ADC */
   ADC_DelSig_Start();
   /* Disable the Delta Sigma ADC ISR as it is not required
      */
   ADC_DelSig_IRQ_Disable();
   /* Start ADC conversions */
   ADC_DelSig_StartConvert();
   /* Start Z DMA */
   OAED_DMAZStart();
   /* Start ECG DMA */
   OAED_AcquisitionECGStart();

   /* Start IDAC source and drain for lead off detection. */
   /* This should also trigger the lead-on/off interrupt. */
   IDAC_Drain_Start();
   IDAC_Source_Start();

   /* Start p and n lead-on/off comparators. */
   Comp_n_Start();

   return;
}

void OAED_AcquisitionZ(){
   /* If already running return. */
   if(Z_enabled)
      return;

   /* Check buffer status */
   if(Z_buffer_full)
      OAED_CopyZBuffer();

   /* Enable Z interrupt */
```

```
49    OAED_ISRZEnable();
50    return;
51 }
52
53 bool OAED_AcquisitionECGUnpause(){
54    /* Unpause ECG acquisition. */
55
56    /* If lead-off detected return false. */
57    if(!lead_detected)
58       return false;
59
60    /* Start Z acquisition */
61    OAED_AcquisitionZ();
62
63    /* Check if ECG is already functioning. */
64    if(ECG_enabled)
65       return true;
66
67    /* Enable ECG cache interrupt */
68    OAED_ISRECGEnable();
69    #if(RAW_MODE)
70    OAED_ISRRAWENABLE();
71    #endif
72
73    return true;
74 }
75
76 inline void OAED_AcquisitionECGPause(){
77    /* Pause ECG acquisition. */
78    /* Check if ECG acquisition isn't running, then proceede
         to stop it. */
79    if(ECG_enabled){
80       OAED_ISRECGDisable();
81       #if(RAW_MODE)
82       OAED_ISRRAWDISABLE();
83       #endif
84    }
85 }
86
87 /* End of function declarations */
88
89 /* [] END OF FILE */
```

OAED_Common.c

```
1 /* ========================================
2  *
```

```
3   * OPEN SOURCE AED
4   * This file contains all common
5   * function and global variable
6   * definitions.
7   *
8   * ======================================
9  */
10
11 #include "OAED_Common.h"
12
13 /* Declaration of global variables */
14 int16 CacheECG[ECG_CACHE_SIZE] = {0};
15 #if(RAW_MODE)
16 int16 CacheRAW[RAW_CACHE_SIZE] = {0};
17 #endif
18
19 int16 BufferECG[ECG_DATA_SIZE] = {0};
20 int16 BufferZ[Z_DATA_SIZE]    = {0};
21
22 int16 DataECG[ECG_DATA_SIZE]  = {0};
23 int16 OldECG[ECG_DATA_SIZE]   = {0};
24 int16 DataZ[Z_DATA_SIZE]      = {0};
25
26 double PatientImpedance       = 0;
27
28 uint8 EventCounter = 0;
29
30 #if(RAW_MODE)
31    int16 DataRAW[RAW_DATA_SIZE] = {0};
32    int16 BufferRAW[RAW_DATA_SIZE] = {0};
33 #endif
34 /* End of global variables declaration */
35
36 /* Declaration of system flags */
37 bool ECG_buffer_full       = false;
38 bool Z_buffer_full         = false;
39 bool ECG_data_pending      = false;
40
41 bool ECG_enabled           = false;
42 bool Z_enabled             = false;
43
44 bool lead_detected         = false;
45 bool capacitor_ready       = false;
46
47 bool event_flags[EVENT_NO] = {false};
48
49 bool Continuous_USBECG     = false;
50 bool Continuous_USBRAW     = false;
51
```

```
52 #if(RAW_MODE)
53 bool RAW_buffer_full       = false;
54 #endif
55 /* End of system flags declaration */
56
57 /* Function declarations */
58 void OAED_Init(){
59
60    OAED_DMA_Init();  /* DMA Initialization. */
61    OAED_AcquisitionInit(); /* Initialize acquisition
         circuitery. */
62    OAED_InitFilter(); /* Initialize filter component. */
63    OAED_ResetEvent(); /* Event data initialization. */
64    OAED_ISRInit();   /* ISR initialization. */
65
66    /* Secure the defibrillation circuitery. */
67    OAED_DisableChargingCircuit();
68    OAED_DisableDefibrillation();
69    OAED_DisarmDefibrillator(true);
70
71    OAED_USBInit();
72    #if(OAED_TIME)
73       OAED_InitTime();
74    #endif
75
76    CyGlobalIntEnable; /* Enable global interrupts. */
77    return;
78 }
79
80 void OAED_InitFilter(){
81    /* Init and start the filter component. */
82    Filter_Start();
83
84    /* Set coherency and dalign for 16bit DMA transfer. */
85    Filter_SetCoherency(Filter_CHANNEL_A, Filter_KEY_MID);
86    Filter_SetCoherency(Filter_CHANNEL_B, Filter_KEY_MID);
87    Filter_SetDalign(Filter_STAGEA_DALIGN, Filter_ENABLED);
88    Filter_SetDalign(Filter_STAGEB_DALIGN, Filter_ENABLED);
89    Filter_SetDalign(Filter_HOLDA_DALIGN, Filter_ENABLED);
90    Filter_SetDalign(Filter_HOLDB_DALIGN, Filter_ENABLED);
91
92    return;
93 }
94
95 void OAED_CopyECGBuffer(){
96    /* The Buffer is copied on the Data vector and
         re-initialized to zeroes. */
97    uint16 i;
98
```

```
99    for(i = 0; i < ECG_DATA_SIZE ; i++){
100      OldECG[i] = DataECG[i];
101      DataECG[i] = BufferECG[i];
102   }
103
104   /* Raw signal */
105   #if(RAW_MODE)
106   if(RAW_buffer_full){
107      for(i = 0; i < RAW_DATA_SIZE ; i++)
108         DataRAW[i] = BufferRAW[i];
109      RAW_buffer_full = false;
110   }
111   #endif
112
113   /* Set system flags */
114   ECG_buffer_full = false;
115   ECG_data_pending = true;
116
117   return;
118 }
119
120 void OAED_CopyZBuffer(){
121   /* The Buffer is copied on the Data vector and
         re-initialized to zeroes. */
122   uint16 i;
123
124   for(i = 0; i < Z_DATA_SIZE ; i++){
125      DataZ[i] = BufferZ[i];
126   }
127
128   /* Set system flag */
129   Z_buffer_full = false;
130
131   return;
132 }
133
134 void OAED_Led(bool blue, bool orange, bool green, bool
      yellow){
135   /* Set or clear pins according to the parameters. */
136   OAED_PINCONTROL(blue, Status_Led_Blue); // Evaluating ECG
137   OAED_PINCONTROL(orange, Status_Led_Orange); // Capacitor
         charging
138   OAED_PINCONTROL(green, Status_Led_Green); // System ready
139   OAED_PINCONTROL(yellow, Status_Led_Yellow); // Lead-off
         only
140   return;
141 }
142
143 void OAED_ResetEvent(){
```

```
144    /* This function reinitialize the event data. */
145
146    uint8 i;
147    for(i = 0 ; i < EVENT_NO ; i++)
148       event_flags[i] = false;
149
150    EventCounter = 0;
151
152    ECG_data_pending = false;
153    return;
154 }
155
156 bool OAED_CheckFlags(){
157    /* Return true in case the positive events registered
          are equal or */
158    /* greater than the required positive events. */
159
160    /* Default settings: */
161    /* POSITIVE_EVENT_NO 2 */
162    /* EVENT_NO       3 */
163
164    uint8 i;
165    uint8 Counter = 0;
166
167    for( i=0 ; i < EVENT_NO ; i++){
168       if(event_flags[i])
169          Counter++;
170       if(Counter == POSITIVE_EVENT_NO)
171          return true;
172    }
173    return false;
174 }
175
176 bool OAED_EvaluateRhythm(){
177    /* Evaluate the Data vector, update the event flags and
          return true */
178    /* if VT/VF is detected. */
179
180    OAED_ECGAnalysis();
181
182    return OAED_CheckFlags();
183 }
184
185 bool OAED_EvaluateImpedanceAC(){
186    /* Evaluate the patient impedance from data and store it
          in */
187    /* patient_impedance variable. */
188    /* This function return false if a lead off is assumed,
          otherwise true.*/
```

```
189     uint16 i;                    // DataZ index
190     uint16 iZmax = 5 * Z_PERIOD; // Maximum index
191     uint16 Count = 0;            // Sum count
192     double NewImpedance = 0;   // Temporary impedance
193
194     /* Find first maximum index */
195     for( i = iZmax + 1 ; (i < iZmax + Z_PERIOD) && (i <
           Z_DATA_SIZE) ; i++)
196        if( DataZ[i] > DataZ[iZmax] )
197           iZmax = i;
198
199     if(i >= Z_DATA_SIZE / 2)
200        return false;
201
202     /* Data Z contains voltages count used to evaluate the
           impedance. */
203     /* First sum up all peak to peak values */
204     for( i = iZmax ; (i + Z_PERIOD / 2) < Z_DATA_SIZE ; i +=
           Z_PERIOD ){
205        /* Correct sampling misalignment */
206        iZmax = i;
207        while( DataZ[i] < DataZ[i+1] ){
208           if(i > iZmax + Z_PERIOD)
209              break;
210           else
211              i++;
212        }
213        while( DataZ[i] < DataZ[i-1] ){
214           if(i < iZmax - Z_PERIOD)
215              break;
216           else
217              i--;
218        }
219        if(fabs(i - iZmax) > Z_PERIOD){
220           /* No peak detected, try again in the next 5th
                 period */
221           i = iZmax + 5 * Z_PERIOD;
222           continue;
223        }
224        /* Peak to peak = max - min */
225        NewImpedance += DataZ[i] - DataZ[i + Z_PERIOD / 2];
226        Count++;
227     }
228     /* If more than 50% periods doesn't have a maximum,
           something is wrong. */
229     if(Count < (Z_DATA_SIZE/Z_PERIOD) * 0.5 )
230        return false;
231
232     /* Get the mean value. */
```

```
233     NewImpedance /= Count;
234     /* Divide peak to peak mean value by 2 */
235     NewImpedance /= 2;
236     /* Remove the ADC buffer gain */
237     NewImpedance /= ADC_BUFFER_GAIN;
238
239     /* Get the voltage value from the ADC counts */
240     //NewImpedance =
           (double)(ADC_DelSig_CountsTo_uVolts((int32)NewImpedance));
241
242     /* Calculate the impedance from the 2 IDAC current
           (expressed in 1/8 uA). */
243     NewImpedance /= ( (double)(IDAC_Drain_Data) * 2 / 8 );
244     NewImpedance *= 10;
245
246     if(!OAED_ValidateImpedance(NewImpedance)){
247        PatientImpedance = 0;
248        lead_detected = false;
249        return false;
250     }
251
252     /* If the impedance calculations are correct, overwrite
           the old impedance */
253     PatientImpedance = NewImpedance;
254     return true;
255 }
256
257 bool OAED_EvaluateImpedanceDC(){
258     uint16 i;                  // For index
259     double NewImpedance = 0;   // Temporary impedance
260
261     /* Find first maximum index */
262     for( i = 0 ; i < Z_DATA_SIZE ; i++)
263        NewImpedance += ((double)(DataZ[i])/Z_DATA_SIZE);
264
265     /* Calculate the impedance from the 2 IDAC current
           (expressed in 1/8 uA). */
266     NewImpedance /= ( (double)(IDAC_Drain_Data) / 8 );
267     NewImpedance /= 32768;      // 16bit = 2^15 max
268     NewImpedance *= 32000;      // ADC input range [uV]
269
270     if(!OAED_ValidateImpedance(NewImpedance)){
271        PatientImpedance = 0;
272        lead_detected = false;
273        return false;
274     }
275
276     /* If the impedance calculations are correct, overwrite
           the old impedance */
```

```
277     PatientImpedance = fabs(NewImpedance);
278     return true;
279 }
280
281 bool OAED_ValidateImpedance(double Impedance){
282     /* If the new impedance is outside the human limits it
           means that the
283       electrodes may not be attached to the patient.
             Therefore we assume
284       a lead-off.
285       */
286     if(Impedance < Z_MIN || Impedance > Z_MAX)
287         return false;
288
289     /* This is optional, but enabled by default. */
290     /* If the last acquisition was drastically different it
           is likely that the
291       electrodes went off.
292       */
293     if(PatientImpedance != 0){
294         double imp_ratio = Impedance / PatientImpedance;
295         if(imp_ratio > 1 + IMPEDANCE_DEVIATION)
296             return false;
297
298         if(imp_ratio < 1 - IMPEDANCE_DEVIATION)
299             return false;
300     }
301     return true;
302 }
303
304 void OAED_EnableChargingCircuit(){
305
306     /* Disarm Defibrillator. */
307     OAED_DisarmDefibrillator(false);
308
309     /* Enable the charging circuit. */
310     OAED_PINCONTROL(true, Charge_En_0);
311
312     /* Enable capacitor sense. */
313     Comp_CapReady_Start();
314     Comp_CapLow_Start();
315     isr_CapReady_Enable();
316
317     return;
318 }
319
320 void OAED_DisableChargingCircuit(){
321     /* Disable Charging circuit. */
322     OAED_PINCONTROL(false, Charge_En_0);
```

```
323
324     /* Disarm Defibrillator. */
325     OAED_DisarmDefibrillator(false);
326
327     /* Disable capacitor sense. */
328     isr_CapReady_Disable();
329     Comp_CapReady_Stop();
330     Comp_CapLow_Stop();
331
332     return;
333 }
334 /* End of function declarations */
335
336 /* [] END OF FILE */
```

OAED_Defibrillation.c

```
1  /* ========================================
2   *
3   * OPEN SOURCE AED
4   * This file contains all defibrillation
5   * related function and global variable
6   * definitions.
7   *
8   * ========================================
9  */
10
11 #include "OAED_Defibrillation.h"
12
13 uint32 OAED_EvaluateDischargeTime(){ // Possibly deprecated
14     uint32 Time;
15     /* Time formula. */
16     Time = (uint32)(- PatientImpedance * C * log( 1 - 2 *
           U/U_MAX ));
17     /* Express Time in ms */
18     Time = Time * 1000;
19     return Time;
20 }
21
22 void OAED_EnableDefibrillation(){
23     /* Pause ECG acquisition.*/
24     if(ECG_enabled)
25         OAED_AcquisitionECGPause();
26
27     /* Switch inner and outer security relay. */
28     CyPins_SetPin(Defibrillation_En_Inner);
29     CyPins_SetPin(Defibrillation_En_Outer);
```

```
30
31    /* Wait for the relay to set. */
32    CyDelay(20);
33
34    return;
35 }
36
37 void OAED_DisableDefibrillation(){
38    /* Switch inner and outer security relay. */
39    CyPins_ClearPin(Defibrillation_En_Inner);
40    CyPins_ClearPin(Defibrillation_En_Outer);
41
42    /* Wait for the relay to set. */
43    CyDelay(20);
44
45    return;
46 }
47
48 inline void OAED_ArmDefibrillator(){
49    /* Enable defibrillator switch interrupt. */
50    isr_Defibrillate_Enable();
51 }
52
53 inline void OAED_DisarmDefibrillator(bool release_charge){
54    /* Disable the protection resistor connection. */
55    OAED_HBridgeControl( OPEN_CIRCUIT );
56
57    /* Disable defibrillator switch interrupt. */
58    isr_Defibrillate_Disable();
59
60    /* If required, activate internal discharge. */
61    if(release_charge)
62       OAED_InternalDischarge();
63    return;
64 }
65
66 void OAED_InternalDischarge(){
67    /* Avoid any possibilities to release the charge to the
         patient.
68       Since a lot of current need to be released, the
            discharge is
69       modulated with a PWM like control.
70
71       Avoid any risk of accidental defibrillation.
72       */
73    OAED_DisableDefibrillation();
74
75    /* The PWM discharge last until the capacitor voltage go
         below the
```

```
76        threshold.
77     */
78     while(capacitor_ready){
79        /* Activate a monophasic PWM discharge. */
80        OAED_HBridgeControl( OPEN_CIRCUIT );
81        CyDelayUs(PWM_OFF);
82
83        OAED_HBridgeControl( PHI_1 );
84        CyDelayUs(PWM_ON);
85     }
86     /* After reaching the threshold it's safe to release the
          remaining charge
87        as a continuous discharge.
88     */
89     return;
90     /* We don't need to wait for the capacitor to fully
          discharge because
91        it's going to need a lot of Time.
92        Also, this way is safer since the capacitor can't
             store energy until
93        the circuit is opened again when the next charge
             command is called.
94     */
95  }
96
97  void OAED_MonophasicDefibrillation(){
98     uint32 Time;
99
100    /* Get discharge Time. */
101    Time = OAED_EvaluateDischargeTime();
102
103    /* Enable patient defibrillation. */
104    OAED_EnableDefibrillation();
105
106    /* Start Defibrillation. */
107    OAED_HBridgeControl( PHI_1 );
108    /* Wait for the charge to be delivered. */
109    CyDelay(Time);
110    /* Stop Defibrillation */
111    OAED_HBridgeControl( OPEN_CIRCUIT );
112
113    /* Disable patient defibrillation. */
114    OAED_DisableDefibrillation();
115
116    /* Discharge residual charge. (optional) */
117    //OAED_InternalDischarge();
118
119    return;
120 }
```

```
121
122 void OAED_PolyphasicDefibrillation(uint8 phase_no){
123    uint32 TotalTime;
124    uint32 Phase1_Time;
125    uint32 PhaseN_Time;
126    uint8 i;
127
128    /* Get discharge Time. */
129    TotalTime = OAED_EvaluateDischargeTime();
130    Phase1_Time = TotalTime / phase_no;
131    PhaseN_Time = TotalTime - phase_no * Phase1_Time;
132
133    /* If phase doesn't last enough execute a biphasic
          instead */
134    if(PhaseN_Time <3){
135       OAED_BiphasicDefibrillation(50);
136       return;
137    }
138
139    /* Enable patient defibrillation. */
140    OAED_EnableDefibrillation();
141
142    /* Start Defibrillation. */
143    OAED_HBridgeControl( PHI_1 );
144    /* Wait for the phase 1 charge to be delivered. */
145    CyDelay(Phase1_Time);
146
147    /* Stop Defibrillation for 1 msec */
148    OAED_HBridgeControl( OPEN_CIRCUIT );
149    CyDelay(1);
150
151    for( i=1; i<phase_no ; i++){
152       /* Invert phase */
153       if(i%2 == 1)
154          OAED_HBridgeControl( PHI_2 );
155       else
156          OAED_HBridgeControl( PHI_1 );
157       /* Wait for the charge to be delivered. */
158       CyDelay(PhaseN_Time);
159       /* Stop Defibrillation. */
160       OAED_HBridgeControl( OPEN_CIRCUIT );
161       CyDelayUs(500);
162
163    }
164    /* Disable patient defibrillation. */
165    OAED_DisableDefibrillation();
166
167    /* Discharge residual charge. (optional) */
168    //OAED_InternalDischarge();
```

```
169
170     return;
171 }
172
173 void OAED_BiphasicDefibrillation(uint8 phase_one_duty){
174     uint32 TotalTime;
175     uint32 Phase1_Time;
176     uint32 phase2_time;
177
178     /* Get discharge Time. */
179     TotalTime = OAED_EvaluateDischargeTime();
180     Phase1_Time = TotalTime * phase_one_duty / 100;
181     phase2_time = TotalTime - Phase1_Time;
182
183     /* Enable patient defibrillation. */
184     OAED_EnableDefibrillation();
185
186     /* Start Defibrillation. */
187     OAED_HBridgeControl( PHI_1 );
188     /* Wait for the phase 1 charge to be delivered. */
189     CyDelay(Phase1_Time);
190     /* Stop Defibrillation for 1 msec */
191     OAED_HBridgeControl( OPEN_CIRCUIT );
192     CyDelay(1);
193     /* Invert phase */
194     OAED_HBridgeControl( PHI_2 );
195     /* Wait for the phase 2 charge to be delivered. */
196     CyDelay(phase2_time);
197     /* Stop Defibrillation. */
198     OAED_HBridgeControl( OPEN_CIRCUIT );
199
200     /* Disable patient defibrillation. */
201     OAED_DisableDefibrillation();
202
203     /* Discharge residual charge. (optional) */
204     //OAED_InternalDischarge();
205
206     return;
207 }
208
209 /* [] END OF FILE */
```

OAED_DMA.c

```
1 /* ========================================
2  *
3  * OPEN SOURCE AED
```

```
4  * This file contains all DMA related
5  * function and global variable
6  * definitions.
7  *
8  * ========================================
9  */
10
11 #include "OAED_DMA.h"
12
13 /* Variable declarations for DMA_DelSig_ECG */
14 uint8 DMA_DelSig_ECG_Chan;
15 uint8 DMA_DelSig_ECG_TD[1];
16
17 /* Variable declarations for DMA_DelSig_Z */
18 uint8 DMA_DelSig_Z_Chan;
19 uint8 DMA_DelSig_Z_TD[1];
20
21 #if(RAW_MODE)
22 /* Variable declarations for DMA_DelSig_RAW */
23 uint8 DMA_DelSig_RAW_Chan;
24 uint8 DMA_DelSig_RAW_TD[1];
25 #endif
26
27 /* Variable declarations for DMA_Filter_ECG */
28 uint8 DMA_Filter_ECG_Chan;
29 uint8 DMA_Filter_ECG_TD[1];
30
31 /* Variable declarations for DMA_Filter_Z */
32 uint8 DMA_Filter_Z_Chan;
33 uint8 DMA_Filter_Z_TD[4];
34
35 void OAED_DMA_Init(){
36
37    /* DMA Configuration for DMA_DelSig */
38    DMA_DelSig_ECG_Chan = DMA_DelSig_ECG_DmaInitialize(
39       DMA_DelSig_ECG_BYTES_PER_BURST,
40       DMA_DelSig_ECG_REQUEST_PER_BURST,
41       HI16(DMA_DelSig_ECG_SRC_BASE),
42       HI16(DMA_DelSig_ECG_DST_BASE)
43    );
44    DMA_DelSig_ECG_TD[0] = CyDmaTdAllocate();
45    CyDmaTdSetConfiguration(
46       DMA_DelSig_ECG_TD[0],
47       DMA_DelSig_ECG_BYTES_PER_BURST,
48       DMA_DelSig_ECG_TD[0],
49       0
50    );
51    CyDmaTdSetAddress(
52       DMA_DelSig_ECG_TD[0],
```

```
53          LO16((uint32)ADC_DelSig_DEC_SAMP_PTR),
54          LO16((uint32)Filter_STAGEA_PTR)
55      );
56      CyDmaChSetInitialTd(
57          DMA_DelSig_ECG_Chan,
58          DMA_DelSig_ECG_TD[0]
59      );
60
61      /* DMA Configuration for DMA_DelSig_Z */
62      DMA_DelSig_Z_Chan = DMA_DelSig_Z_DmaInitialize(
63          DMA_DelSig_Z_BYTES_PER_BURST,
64          DMA_DelSig_Z_REQUEST_PER_BURST,
65          HI16(DMA_DelSig_Z_SRC_BASE),
66          HI16(DMA_DelSig_Z_DST_BASE)
67      );
68      DMA_DelSig_Z_TD[0] = CyDmaTdAllocate();
69      CyDmaTdSetConfiguration(
70          DMA_DelSig_Z_TD[0],
71          DMA_DelSig_Z_BYTES_PER_BURST,
72          DMA_DelSig_Z_TD[0],
73          0
74      );
75      CyDmaTdSetAddress(
76          DMA_DelSig_Z_TD[0],
77          LO16((uint32)ADC_DelSig_DEC_SAMP_PTR),
78          LO16((uint32)Filter_STAGEB_PTR)
79      );
80      CyDmaChSetInitialTd(
81          DMA_DelSig_Z_Chan,
82          DMA_DelSig_Z_TD[0]
83      );
84
85  #if(RAW_MODE)
86      /* DMA Configuration for DMA_DelSig_RAW */
87      DMA_DelSig_RAW_Chan = DMA_DelSig_RAW_DmaInitialize(
88          DMA_DelSig_RAW_BYTES_PER_BURST,
89          DMA_DelSig_RAW_REQUEST_PER_BURST,
90          HI16(DMA_DelSig_RAW_SRC_BASE),
91          HI16(DMA_DelSig_RAW_DST_BASE)
92      );
93      DMA_DelSig_RAW_TD[0] = CyDmaTdAllocate();
94      CyDmaTdSetConfiguration(
95          DMA_DelSig_RAW_TD[0],
96          DMA_DelSig_RAW_BYTES_PER_BURST * RAW_CACHE_SIZE,
97          DMA_DelSig_RAW_TD[0],
98          DMA_DelSig_RAW__TD_TERMOUT_EN | CY_DMA_TD_INC_DST_ADR
99      );
100     CyDmaTdSetAddress(
101         DMA_DelSig_RAW_TD[0],
```

```
102         LO16((uint32)ADC_DelSig_DEC_SAMP_PTR),
103         LO16((uint32)CacheRAW)
104     );
105     CyDmaChSetInitialTd(
106         DMA_DelSig_RAW_Chan,
107         DMA_DelSig_RAW_TD[0]
108     );
109 #endif
110 
111     /* DMA Configuration for DMA_Filter_ECG */
112     DMA_Filter_ECG_Chan = DMA_Filter_ECG_DmaInitialize(
113         DMA_Filter_ECG_BYTES_PER_BURST,
114         DMA_Filter_ECG_REQUEST_PER_BURST,
115         HI16(DMA_Filter_ECG_SRC_BASE),
116         HI16(DMA_Filter_ECG_DST_BASE)
117     );
118     DMA_Filter_ECG_TD[0] = CyDmaTdAllocate();
119     CyDmaTdSetConfiguration(
120         DMA_Filter_ECG_TD[0],
121         DMA_Filter_ECG_BYTES_PER_BURST * ECG_CACHE_SIZE,
122         DMA_Filter_ECG_TD[0],
123         DMA_Filter_ECG__TD_TERMOUT_EN | TD_INC_DST_ADR
124     );
125     CyDmaTdSetAddress(
126         DMA_Filter_ECG_TD[0],
127         LO16((uint32)Filter_HOLDA_PTR),
128         LO16((uint32)CacheECG)
129     );
130     CyDmaChSetInitialTd(
131         DMA_Filter_ECG_Chan,
132         DMA_Filter_ECG_TD[0]
133     );
134 
135     /* DMA Configuration for DMA_Filter_Z */
136     DMA_Filter_Z_Chan = DMA_Filter_Z_DmaInitialize(
137         DMA_Filter_Z_BYTES_PER_BURST,
138         DMA_Filter_Z_REQUEST_PER_BURST,
139         HI16(DMA_Filter_Z_SRC_BASE),
140         HI16(DMA_Filter_Z_DST_BASE)
141     );
142     DMA_Filter_Z_TD[0] = CyDmaTdAllocate();
143     DMA_Filter_Z_TD[1] = CyDmaTdAllocate();
144     DMA_Filter_Z_TD[2] = CyDmaTdAllocate();
145     DMA_Filter_Z_TD[3] = CyDmaTdAllocate();
146     CyDmaTdSetConfiguration(
147         DMA_Filter_Z_TD[0],
148         (DMA_Filter_Z_BYTES_PER_BURST * Z_DATA_SIZE) / 4,
149         DMA_Filter_Z_TD[1],
150         TD_INC_DST_ADR
```

```
151     );
152     CyDmaTdSetConfiguration(
153        DMA_Filter_Z_TD[1],
154        (DMA_Filter_Z_BYTES_PER_BURST * Z_DATA_SIZE) / 4,
155        DMA_Filter_Z_TD[2],
156        TD_INC_DST_ADR
157     );
158     CyDmaTdSetConfiguration(
159        DMA_Filter_Z_TD[2],
160        (DMA_Filter_Z_BYTES_PER_BURST * Z_DATA_SIZE) / 4,
161        DMA_Filter_Z_TD[3],
162        TD_INC_DST_ADR
163     );
164     CyDmaTdSetConfiguration(
165        DMA_Filter_Z_TD[3],
166        (DMA_Filter_Z_BYTES_PER_BURST * Z_DATA_SIZE) / 4,
167        DMA_Filter_Z_TD[0],
168        DMA_Filter_Z__TD_TERMOUT_EN | TD_INC_DST_ADR
169     );
170     CyDmaTdSetAddress(
171        DMA_Filter_Z_TD[0],
172        LO16((uint32)Filter_HOLDB_PTR),
173        LO16((uint32)BufferZ)
174     );
175     CyDmaTdSetAddress(
176        DMA_Filter_Z_TD[1],
177        LO16((uint32)Filter_HOLDB_PTR),
178        LO16((uint32)(BufferZ + Z_DATA_SIZE / 4))
179     );
180     CyDmaTdSetAddress(
181        DMA_Filter_Z_TD[2],
182        LO16((uint32)Filter_HOLDB_PTR),
183        LO16((uint32)(BufferZ + Z_DATA_SIZE / 2))
184     );
185     CyDmaTdSetAddress(
186        DMA_Filter_Z_TD[3],
187        LO16((uint32)Filter_HOLDB_PTR),
188        LO16((uint32)(BufferZ + Z_DATA_SIZE * 3 / 4))
189     );
190     CyDmaChSetInitialTd(
191        DMA_Filter_Z_Chan,
192        DMA_Filter_Z_TD[0]
193     );
194
195     return;
196 }
197
198 void OAED_DMAECGStart(){
199     /* Clear ECG DMA requests */
```

```
200     CyDmaClearPendingDrq(DMA_DelSig_ECG_Chan);
201     CyDmaClearPendingDrq(DMA_Filter_ECG_Chan);
202 
203     /* Enable ECG DMA */
204     CyDmaChEnable(DMA_DelSig_ECG_Chan, 1);
205     CyDmaChEnable(DMA_Filter_ECG_Chan, 1);
206 
207     #if(RAW_MODE)
208     CyDmaChEnable(DMA_DelSig_RAW_Chan, 1);
209     #endif
210 
211     return;
212 
213 }
214 
215 void OAED_DMAECGStop(){
216 
217     OAED_AcquisitionECGPause();
218 
219     /* Disable ECG DMA */
220     CyDmaChDisable(DMA_DelSig_ECG_Chan);
221     CyDmaChDisable(DMA_Filter_ECG_Chan);
222 
223     #if(RAW_MODE)
224     CyDmaChDisable(DMA_DelSig_RAW_Chan);
225     #endif
226 
227     return;
228 }
229 
230 void OAED_DMAZStart(){
231     /* Clear Z DMA requests */
232     CyDmaClearPendingDrq(DMA_DelSig_Z_Chan);
233     CyDmaClearPendingDrq(DMA_Filter_Z_Chan);
234 
235     /* Enable Z DMA */
236     CyDmaChEnable(DMA_DelSig_Z_Chan, 1);
237     CyDmaChEnable(DMA_Filter_Z_Chan, 1);
238 
239     return;
240 }
241 
242 void OAED_DMAZStop(){
243     /* Disable Z DMA */
244     CyDmaChDisable(DMA_DelSig_Z_Chan);
245     CyDmaChDisable(DMA_Filter_Z_Chan);
246 
247     return;
248 }
```

```
249
250 /* [] END OF FILE */
```

OAED_ECGAlgorithms.c

```
1  /* ========================================
2   *
3   * OPEN SOURCE AED
4   * This header contains all ECG pattern
5   * recognition algorithms functions and
6   * Global variable definitions.
7   *
8   * ========================================
9  */
10 #include "OAED_ECGAlgorithms.h"
11
12 /* The algorithms are not described here, for brevity's
       sake.
13   Please refer to the thesis to know more.
14  */
15
16 bool OAED_TCI(){
17     /*
18         In order to achieve better performance (in terms of
              decision making), i
19         have decided to use also the data obtained during the
              last 4 second
20         window. This drastically increase the number of data
              we can use for TCI,
21         but require some sort of optimization in order to
              avoid saving the whole
22         ECG window, which would lead to memory/time waste.
23         Namely we need to save the TCI values calculated, the
              T1-T3, and N values
24         of the last second of the current ECG window.
25         - N is stored in the static variable LastN
26         - T1-T3 in the static array LastT[3]
27         - TCI are saved in the static array called TCI. In
              this data is organized
28           so as the first ECG_SIGNAL_LENGTH-1 values are those
                of the current
29           ECG window; the last ECG_SIGNAL_LENGTH-1 values are
                those of the last
30           ECG window; and the middle value is the last second
                of the previous
31           window, which can only be evaluated having the
                subsequent data.
```

```
32
33      An example with ECG_SIGNAL_LENGTH = 4 is the
           following:
34
35      time:       0-+-+-+-4-+-+-+-8-+-+-+-12-+->
36      first call  0-+-+-+-4
37      TCI values   x'1'2'x
38      second call 0-+-+-+-4-+-+-+-8
39      TCI values  -x'1'2'3-4'5'6'x-
40      third call        4-+-+-+-8-+-+-+-12
41      TCI values        -4'5'6'7-8'9'10'x-
42
43                        TCI array content
44         position | 0 | 1 | 2 : 3 : 4 | 5 | 6 |
45                  ---------------------------------------
46      first call  | \ | 1 | 2 : \ : \ | \ | \ |
47      second call | 4 | 5 | 6 : 3 : 1 | 2 | \ |
48      third call  | 8 | 9 | 10 : 7 : 4 | 5 | 6 |
49      fourth call | 12 | 13 | 14 : 11 : 8 | 9 | 10 |
50      ...         | y-3 | y-2 | y-1 : x : x-3 | x-2 | x-1 |
51                  <-------new-------><mid><-------old------->
52   */
53   /* One array for Tx, each element represent a second in
        the current segment.
54     The last element is an exception because it is swapped
          with the last
55     element of the previous call.
56   */
57   float T1[ECG_SIGNAL_LENGTH],
58         T2[ECG_SIGNAL_LENGTH],
59         T3[ECG_SIGNAL_LENGTH],
60         T4[ECG_SIGNAL_LENGTH];
61   /* The last Ts are made static in order to use it in the
        next analysis */
62   static float   LastT[3];
63   float         Max[ECG_SIGNAL_LENGTH];    // Maximum
        values
64   bool          BinaryECG[ECG_DATA_SIZE]; // Binary ECG
65   int16         N[ECG_SIGNAL_LENGTH];      // Number of
        pulses
66   static int16  LastN;                     // Last N of
        previous
67                                            // analysis
68   static float  TCI[2*ECG_SIGNAL_LENGTH - 1]; // TCI
        values
69   float         TCImean = 0;               // Mean TCI
70   uint8         i = 0;                     // For counter
71   uint16        j;                         // For counter
72
```

```
73    /* The first step is finding the maximum value for each
         second */
74    for( i = 0 ; i < ECG_SIGNAL_LENGTH ; i++ ){
75       Max[i] = -1;
76       for( j = i * ECG_SAMPLING_F; j < (i+1) *
            ECG_SAMPLING_F ; j++ )
77          if(DataECG[j] > Max[i])
78             Max[i] = DataECG[j];
79    }
80
81    /* The second step is to binarize the ECG signal with a
         threshold */
82
83    /* By default the threshold is 20% maximum */
84    for( i = 0 ; i < ECG_SIGNAL_LENGTH ; i++ ){
85       int16 Threshold = Max[i] * TCI_THRESH;
86       for( j = i * ECG_SAMPLING_F; j < (i+1) *
            ECG_SAMPLING_F ; j++ )
87          if(Threshold > 0)
88             BinaryECG[j] = (DataECG[j] > Threshold);
89          else
90             BinaryECG[j] = (DataECG[j] > Max[i] * (2 -
                TCI_THRESH));
91    }
92
93    /* The third step is evaluating the T values */
94
95    /* T2 is defined as the time elapsed until the first
         value not 0, from the
96       beginning of each 1second segment. */
97    for( i = 0 ; i < ECG_SIGNAL_LENGTH ; i++ ){
98       for( j = i * ECG_SAMPLING_F; j < (i+1) *
            ECG_SAMPLING_F ; j++ )
99          if(BinaryECG[j]){
100            T2[i] = (float)(j) / ECG_SAMPLING_F - i;
101            break;
102         }
103   }
104
105   /* T3 is defined as the time elapsed from the last value
         not 0 until
106      the end for each segment. */
107   for( i = 0 ; i < ECG_SIGNAL_LENGTH ; i++ ){
108      for( j = (i+1) * ECG_SAMPLING_F ; j > i *
           ECG_SAMPLING_F ; j-- ){
109         if(BinaryECG[j]){
110            T3[i] = i + 1 - (float)(j) / ECG_SAMPLING_F ;
111            break;
112         }
```

```
113         }
114     }
115
116     /* T1 and T4 are equal to the last T3 and the next T2
           respectively. */
117     for( i = 0 ; i < ECG_SIGNAL_LENGTH ; i++){
118        T1[i] = (i == 0) ? LastT[2] : T3[i-1];
119        T4[i] = (i == ECG_SIGNAL_LENGTH) ? 0 : T2[i+1];
120     }
121
122     /* The fourth step is pulse counting in the binary ECG
           for each segment. */
123     for( i = (EventCounter == 0) ? 1 : 0 ; i <
           ECG_SIGNAL_LENGTH ; i++ ){
124        N[i] = 0;
125        /* Count each transitions */
126        for( j = i * ECG_SAMPLING_F; j < 1 + (i+1) *
              ECG_SAMPLING_F ; j++ )
127           if(BinaryECG[j] != BinaryECG[j+1])
128              N[i]++;
129        //N[i] = ceil( N[i]/2 );
130        N[i] = (N[i]>>1) + N[i]%2;
131        /* If a pulse is between two segments, T2 or T3 are
              zeros. In that case
132           there is a odd number of transitions, hence the
                 ceil fix it adding 1.
133           If both T3 and T2 are zeros the ceil operator adds
                 nothing, therefore
134           we have to add 1 manually.
135         */
136        if(T2[i] == 0 && T3[i] == 0)
137           N[i]++;
138     }
139
140     /* The fifth step is the actual TCI calculation. */
141     /* The last second can't be evaluated at this time, we
           save it for the
142        next time this function is called. We swap its values
              with those
143        supposedly saved on the last call, so we can use them.
              In this way the
144        last element of each Tx array will contain the Tx
              related to the last
145        second of the last call.
146      */
147     OAED_SWAP( T1[ECG_SIGNAL_LENGTH - 1], LastT[0] );
148     OAED_SWAP( T2[ECG_SIGNAL_LENGTH - 1], LastT[1] );
149     OAED_SWAP( T3[ECG_SIGNAL_LENGTH - 1], LastT[2] );
150     OAED_SWAP( N[ECG_SIGNAL_LENGTH - 1], LastN );
```

```
151     T4[ECG_SIGNAL_LENGTH - 1] = T2[0];
152     for( i = (EventCounter == 0) ? 1 : 0 ; i <
           ECG_SIGNAL_LENGTH ; i++ ){
153        /* If this is the first registered event we cannot
              evaluate neither the
154           first, nor the last second (which should correspond
                 to the last
155           second of the last call).
156        */
157        if( EventCounter == 0 )
158           if( i == ECG_SIGNAL_LENGTH-1 ) break;
159        /* If T1/T2 or T3/T4 are zero we must avoid the 0/0
              division */
160        float t12 = 0;
161        float t34 = 0;
162        if(T2[i] != 0)
163           t12 = T2[i]/( T1[i]+T2[i] );
164        if(T3[i] != 0)
165           t34 = T3[i]/( T3[i]+T4[i] );
166        TCI[i] = 1000 / ( N[i]-1 + t12 + t34 );
167     }
168
169     /* The TCI mean is evaluated using also the values from
           the last call, if
170        availables.
171     */
172     uint8 Count = 0;
173     for( i = (EventCounter == 0) ? 1 : 0 ; i < 2 *
           ECG_SIGNAL_LENGTH - 1 ; i++ ){
174        if(EventCounter == 0){
175           if(i == ECG_SIGNAL_LENGTH - 1)
176              break;
177        }
178        if(EventCounter == 1)
179           if(i == 2 * ECG_SIGNAL_LENGTH - 2)
180              break;
181
182        TCImean += TCI[i];
183        Count++;
184     }
185     TCImean /= Count;
186
187     /* Save the TCI values for later */
188     j = ECG_SIGNAL_LENGTH;
189     for( i = (EventCounter == 0) ? 1 : 0 ; i <
           ECG_SIGNAL_LENGTH - 1 ; i++){
190        TCI[j++] = TCI[i];
191     }
192
```

```
193     /* The last step is the comparison of the mean TCI value
           with the pre
194       established critical value and the statistical
             inference.
195     */
196     if(TCImean >= TCI_CRIT) // if the TCI mean is higher
           than the critical value
197        return false;  // the patient rhythm is normal.
198
199     float F = 0;   /* F calculus */
200     if( EventCounter == 0 )
201        Count++;
202     for( i = (EventCounter == 0) ? 1 : 0 ; i < Count ; i++ ){
203        F += (float)(pow((TCI[i] - TCI_MU_VF), 2)) /
              (float)(TCI_SIGMA2_VF);
204        F -= (float)(pow((TCI[i] - TCI_MU_VT), 2)) /
              (float)(TCI_SIGMA2_VT);
205     }
206     /* F evaluation */
207     if( F >= (TCI_F_VF + Count * TCI_F_VX) )
208        return true;   // Ventricular tachicardia
209     if( F <= (TCI_F_VT + Count * TCI_F_VX) )
210        return true;   // Ventricular fibrillation
211
212     return false;     // Rhythm is not normal, but we can't
           confirm
213                       // neither VT, nor VF.
214 }
215
216 bool OAED_VFfilter(){
217     float Den, Num; // Temporary denominator and numerator
218     float L;        // Leakage
219     uint16 T2;      // Half-period
220     uint16 i;       // For counter
221
222     /* First we find the mean period T. */
223     Num = 0;
224     Den = 0;
225     for( i = 1; i < ECG_DATA_SIZE ; i++ ){
226        Num += abs(DataECG[i]);
227        Den += abs(DataECG[i] - DataECG[i-1]);
228     }
229     /* T = 2 * PI * Num / Den + 1 */
230     /* We need the half-period T2 = T/2 */
231     T2 = floor(PI * Num / Den + 0.5);
232     /* Although the variable T may seem to indicate a time,
           it is actually an
233       adimensional number. Hence it indicate a period of
             samples.
```

```
234     */
235
236     /* The second step is finding the mean leakage */
237     Num = 0;
238     Den = 0;
239     for( i = T2; i < ECG_DATA_SIZE ; i++ ){
240        Num += abs(DataECG[i] + DataECG[i-T2]);
241        Den += abs(DataECG[i]) + abs(DataECG[i-T2]);
242     }
243     L = Num / Den;
244
245     /* The final step is the rhythm assertion. */
246     if( L < VFf_CRIT )
247        return true;   // VF confirmed
248     // else
249     return false;     // Normal sinus rhythm
250 }
251
252 bool OAED_TCSC(){
253     static float N[TCSC_NTS];                // N values
254     float Nmean = 0;                         // N mean value
255     float Max[TCSC_NTS];                     // ECG maximum
256     int16 i;                                 // For counter
257
258     /* A note about the choice made during the writing of
           this function.
259       On the first iteration, the function will save and
             consider only the first
260       part of the array N.
261       Starting from the second call, the first part will
             store the new N values,
262       while the second part will the cross evaluated N, and
             finally the third
263       part will contain the old N values.
264
265       Example with default values: Ls = 3 [s],
             ECG_SIGNAL_LENGTH = 4 [s]
266        First call:
267             N0
268           <----->
269           0-+-+-+-4-+-+-+-8->
270             <----->
271               N1
272
273                  new  cross old
274            N = [ N0 N1 | \ \ | \ \ ]
275
276        Second Call:
277             N0                    N2                    N4
```

```
278          <----->                <----->                <----->
279          0-+-+-+-4-+-+-+-8-> 0-+-+-+-4-+-+-+-8->
                0-+-+-+-4-+-+-+-8->
280           <----->                <----->                <----->
281             N1                     N3                     N5
282
283                 new  cross  old
284           N = [ N4 N5 | N2 N3 | N0 N1 ]
285
286       N0 and N1 - are evaluated respectively using the first
             and the last 3s
287                 of the four second window in the first call.
288       N2 and N3 - needs both data from the first and the
             second 4-second window,
289                 therefore we need to save the last 2 seconds
                       (Ls-1) of the
290                 current ECG window for the next call.
291       N4 and N5 - are evaluated with the 4s window of the
             second call.
292
293       Every call after the first work the same way.
294       In this way we use the algorithm with 2 different
             subsequent acquisitions
295       and thus considerably increase the number of Ls window
             we can evaluate.
296    */
297
298    /* Reset N array if this is the first segment */
299    if( EventCounter == 0 ){
300       for( i = 0 ; i < TCSC_NTS ; i++ )
301          N[i] = 0;
302    }
303
304    /* Find the absolute maximum for each segment */
305
306    /* New ECG -> first part of N */
307    for( i = 0 ; i < TCSC_NS ; i++ ){
308       Max[i] = 0;
309       int16 j;
310       for( j = i * ECG_SAMPLING_F ; j < (i+TCSC_Ls) *
             ECG_SAMPLING_F ; j++ ){
311          if( abs( DataECG[j] ) > Max[i] )
312             Max[i] = abs( DataECG[j] );
313       }
314    }
315
316    /* Combined old and new ECG -> second part of N */
317    if( EventCounter != 0 ){
318       for( i = TCSC_NS ; i < ECG_SIGNAL_LENGTH ; i++ ){
```

```
319            Max[i] = 0;
320            int16 j;
321            /* Old ECG */
322            for( j = i * ECG_SAMPLING_F ; j < ECG_DATA_SIZE ;
                  j++ ){
323               if( abs( OldECG[j] ) > Max[i] )
324                  Max[i] = abs( OldECG[j] );
325            }
326            /* New ECG */
327            for( j = 0 ; j < (i+1 - TCSC_NS) * ECG_SAMPLING_F
                  ; j++ ){
328               if( abs( DataECG[j] ) > Max[i] )
329                  Max[i] = abs( DataECG[j] );
330            }
331         }
332      }
333
334      /* Evaluate the N for each segment */
335
336      /* New ECG -> first part of N */
337      for( i = 0 ; i < TCSC_NS ; i++ ){
338         int16 j;
339         for( j = i * ECG_SAMPLING_F ; j < (i+TCSC_Ls) *
               ECG_SAMPLING_F ; j++ ){
340            /* TCSC use absolute values */
341            float temp = fabs( (float)(DataECG[j]) );
342
343            /* Apply the cosine window if we are in the first
                  or last 0.25 sec*/
344            if( (float)(j) / ECG_SAMPLING_F - i < 0.25)
345               temp *= OAED_TCSC_COSWIN( (float)(j) /
                     ECG_SAMPLING_F - i);
346            if( (float)(j+1) / ECG_SAMPLING_F - i >
                  (float)(TCSC_Ls) - 0.25 )
347               temp *= OAED_TCSC_COSWIN( (float)(j+1) /
                     ECG_SAMPLING_F - i);
348
349            /* Check with the threshold */
350            if( temp > (float)(TCSC_THRESH * Max[i]) )
351               N[i]++;
352         }
353
354         /* Ns are weighted on the Ls window */
355         N[i] /= (float)(TCSC_Ls * ECG_SAMPLING_F);
356         /* Add the N to the mean */
357         Nmean += N[i];
358      }
359
360      /* Combined old and new ECG -> second part of N */
```

```
361    if( EventCounter != 0 ){
362       for( i = TCSC_NS ; i < ECG_SIGNAL_LENGTH ; i++ ){
363          int16 j;
364
365          /* Old ECG */
366          for( j = i * ECG_SAMPLING_F ; j < ECG_DATA_SIZE ;
                j++ ){
367             float temp = fabs( (float)(OldECG[j]) );
368
369             /* Apply the cosine window */
370             /* In this case only the left side is possible
                   */
371             if( (float)(j) / ECG_SAMPLING_F - i < 0.25)
372                temp *= OAED_TCSC_COSWIN( (float)(j) /
                      ECG_SAMPLING_F - i);
373
374             /* Check with the threshold */
375             if( temp > (float)(TCSC_THRESH * Max[i]) )
376                N[i]++;
377          }
378
379          /* New ECG */
380          for( j = 0 ; j < (i+1 - TCSC_NS) * ECG_SAMPLING_F
                ; j++ ){
381             float temp = fabs( (float)(DataECG[j]) );
382
383             /* Apply the cosine window */
384             /* In this case only the right side is possible
                   */
385             if( (float)(j+1) / ECG_SAMPLING_F >
386                                    (float)(i+1 - TCSC_NS)
                                          - 0.25 ){
387                temp *= OAED_TCSC_COSWIN(
388                   (float)(j+1) / ECG_SAMPLING_F + (i+1 -
                         TCSC_NS) );
389             }
390
391             /* Check with the threshold */
392             if( temp > (float)(TCSC_THRESH * Max[i]) )
393                N[i]++;
394          }
395
396          /* Ns are weighted on the Ls window */
397          N[i] /= (float)(TCSC_Ls * ECG_SAMPLING_F);
398          /* Add the N to the mean */
399          Nmean += N[i];
400       }
401    }
402
```

```
403     if(EventCounter != 0){
404        /* Add the older values to the mean */
405        for( i = ECG_SIGNAL_LENGTH ; i < TCSC_NTS ; i++ )
406           Nmean += N[i];
407        Nmean /= (float)(TCSC_NTS);
408     }
409     else{
410        Nmean /= (float)(TCSC_NS);
411     }
412
413     /* Overwrite the older N values */
414     for( i = 0; i < TCSC_NS ; i++ ){
415        N[i + ECG_SIGNAL_LENGTH] = N[i];
416     }
417
418     /* Evaluate the rhythm */
419     if( Nmean >= TCSC_CRIT )
420        return true;          // VF confirmed
421     return false;            // Normal sinus rhythm
422  }
423
424  bool OAED_PSR(){
425     bool  Grid[PSR_GRID_N][PSR_GRID_N];    // Grid
426     float d,                               // d value
427           Delta;                           // Cell size
428     uint8 Gx, Gy;                          // Grid indexes
429     int16 ECGmax = -1, ECGmin = 1;         // Maximum and
        minimum
430     int16 i,j;                             // For counter
431
432     /* Init grid */
433     for( i = 0 ; i < PSR_GRID_N ; i++ )
434        for( j = 0 ; j < PSR_GRID_N ; j++ )
435           Grid[i][j] = false;
436
437     /* Find the maximum and the minimum */
438     for( i = 0 ; i < ECG_DATA_SIZE ; i++ ){
439        if(ECGmax < DataECG[i])
440           ECGmax = DataECG[i];
441        else
442           if(ECGmin > DataECG[i])
443              ECGmin = DataECG[i];
444     }
445
446     /* Calculate cell size */
447     Delta = (float)(ECGmax - ECGmin)/PSR_GRID_N;
448     Delta = ceil(Delta);
449
450     /* Light the grid */
```

```
451     for( i = 0 ; i < ECG_DATA_SIZE - PSR_TAU_N ; i = i+2 ){
452        Gx = (uint8) floor( (float)(DataECG[i] - ECGmin) /
              Delta );
453        Gy = (uint8) floor( (float)
454                        (DataECG[(int16)(i + PSR_TAU_N)] -
                           ECGmin) / Delta );
455        Grid[Gx][Gy] = true;
456     }
457
458     /* Calculate d */
459     d = 0;
460     for( i = 0 ; i < PSR_GRID_N ; i++ )
461        for( j = 0 ; j < PSR_GRID_N ; j++ )
462           if(Grid[i][j])
463              d++;
464     d /= (PSR_GRID_N * PSR_GRID_N);
465
466     /* Evaluate rhythm */
467     if( d > PSR_D0 )
468        return true;   // VF confirmed
469     return false;     // VF not confirmed
470  }
471
472  bool OAED_HILB(){
473     bool  Grid[HILB_GRID_N][HILB_GRID_N];  // Grid
474     float d,                               // d value
475           DeltaIm,                         // Cell size
476           DeltaRe;                         // Cell size
477     uint8 Gx, Gy;                          // Grid indexes
478     uint16 i,j;                            // For counter
479
480     int32                MaxRe = -1;       // Real maximum
481     int32                MaxIm = -1;       // Imag maximum
482     int32                MinRe = 1;        // Real minimum
483     int32                MinIm = 1;        // Imag minimum
484     q31_t                EcgQ31[(uint16)FFTN*2]; // q31
           analytic signal
485
486     /* Init grid */
487     for( i = 0 ; i < HILB_GRID_N ; i++ )
488        for( j = 0 ; j < HILB_GRID_N ; j++ )
489           Grid[i][j] = false;
490
491     /* Calculate analytic signal from ECG */
492     /* Find the maximum to upscale ECG values */
493     for( i = 0; i < ECG_DATA_SIZE ; i++ )
494        if( MaxRe < DataECG[i])
495           MaxRe = DataECG[i];
496
```

```
497     /* The FFT in CMSIS-DSP downscale data in order to avoid
           saturation. Because
498       of the fixed point arithmetics this also mean a loss
           of information (up to
499       12 bit). I wanted to limit data loss, hence i used 32
           bit numbers
500       upscaling them to cover all the available values.
501     */
502
503     /* This cycle convert int16 ECG into q31 ECG upscaling
           to 31bit (keep the
504       sign). Also add zero padding. Remember that this is a
           complex signal,
505       hence real values are stored in even elements whilst
           imaginary values (all
506       zeros for now) are stored in odd elements.
507     */
508     for( i = 0 ; i < FFTN ; i++ )
509        EcgQ31[2*i] =
510              (i < ECG_DATA_SIZE) ? (q31_t)(DataECG[i] *
                 pow(2,31)/MaxRe ) : 0;
511     for( i = 0 ; i < FFTN ; i++ )
512        EcgQ31[2*i+1] = 0;
513
514     /* Perfor 32bit FFT on ECG data */
515     arm_cfft_q31(&arm_cfft_sR_q31_len2048, EcgQ31,
           FORWARD_FFT, 1);
516
517     /* The analytic signal is then obtained modifying the
           FFT data (X) with the
518       following transform (Z) :
519              /  X[0]         m = 0
520              |  2X[m]        1 <= m <= N/2 - 1
521       Z[m] = <  X[N/2]       m = N/2
522              |  0            N/2 + 1 <= m <= N - 1
523              \
524       Always remember that arm_xfft functions returns
           complex arrays where even
525       elements contain the real part, and odd ones contain
           the imaginary part.
526     */
527     for( i = 2 ; i < FFTN ; i++ )
528        EcgQ31[i] *= 2;
529     for( i = FFTN + 2 ; i < 2*FFTN ; i++ )
530        EcgQ31[i] = 0;
531
532     /* Return in time domain performing the inverse
           transform */
```

```
533     arm_cfft_q31(&arm_cfft_sR_q31_len2048, EcgQ31,
           INVERSE_FFT, 1);
534     /* ECGQ31 now contains the analytic signal */
535
536     /* Find real and complex maximum and minimum */
537     MaxRe = -1; // Re-init MaxRe
538     for( i = 0; i < 2*FFTN - 1 ; i ++ ){
539        // Real
540        if(MaxRe < EcgQ31[i])
541           MaxRe = EcgQ31[i];
542        else
543           if(MinRe > EcgQ31[i])
544              MinRe = EcgQ31[i];
545        // Imag
546        i++;
547        if(MaxIm < EcgQ31[i])
548           MaxIm = EcgQ31[i];
549        else
550           if(MinIm > EcgQ31[i])
551              MinIm = EcgQ31[i];
552     }
553
554     /* Calculate cell size */
555     DeltaRe = (float)(MaxRe - MinRe)/HILB_GRID_N;
556     DeltaIm = (float)(MaxIm - MinIm)/HILB_GRID_N;
557     DeltaRe = ceil(DeltaRe);
558     DeltaIm = ceil(DeltaIm);
559
560     /* Light the grid */
561     for( i = 0 ; i < 2*FFTN - 1 ; i = i+3 ){
562        Gx = (uint8) floor( (float)(EcgQ31[i] - MinRe) /
              DeltaRe );
563        Gy = (uint8) floor( (float)(EcgQ31[++i] - MinIm) /
              DeltaIm );
564        Grid[Gx][Gy] = true;
565     }
566
567     /* Calculate d */
568     d = 0;
569     for( i = 0 ; i < HILB_GRID_N ; i++ )
570        for( j = 0 ; j < HILB_GRID_N ; j++ )
571           if(Grid[i][j])
572              d++;
573
574     d /= (HILB_GRID_N * HILB_GRID_N);
575
576     /* Evaluate rhythm */
577     if( d > HILB_D0 )
578        return true;   // VF confirmed
```

```
579     return false;      // VF not confirmed
580 
581 }
582 
583 bool OAED_ECGAnalysis(){
584     /* This function call various algorithms and assert SCA
           based on how many
585        return positive.
586     */
587     uint8 i = 0;
588 
589     if( OAED_TCI() )
590         i++;
591     if( OAED_VFfilter() )
592         i++;
593     if( OAED_TCSC() )
594         i++;
595     if( OAED_PSR() )
596         i++;
597     if( OAED_HILB() )
598         i++;
599 
600     /* Check SCA, by default it is 3 positive on 5 */
601     if( i >= OAED_POSITIVETHRESH )
602         return true;
603     return false;
604 }
605 
606 /* [] END OF FILE */
```

OAED_ISR.c

```
1  /* ========================================
2   *
3   * OPEN SOURCE AED
4   * This file contains all ISR related
5   * function and global variable
6   * definitions.
7   *
8   * ========================================
9  */
10 
11 #include "OAED_ISR.h"
12 
13 /* Function Declarations */
14 
15 /* ECG Cache Replenished ISR custom call definition. */
```

```
16 CY_ISR(isr_CacheECGRe){
17    static uint16 n = 0;
18    uint8 i;
19    int32 t = CacheECG[0];
20    for( i = 1 ; i < ECG_CACHE_SIZE ; i++ )
21       t += CacheECG[i];
22 
23    BufferECG[n++] = (int16)(t>>3);
24 
25    if(Continuous_USBECG)
26       OAED_USBSendData16(CacheECG, 8);
27    if(n == ECG_DATA_SIZE){
28       ECG_buffer_full = true;
29       n = 0;
30       OAED_ISRECGDisable();
31    }
32 }
33 
34 #if(RAW_MODE)
35 /* RAW Cache Replenished ISR custom call definition. */
36 CY_ISR(isr_CacheRAWRe){
37    static uint16 n = 0;
38    BufferRAW[n++] = CacheRAW[0];
39    BufferRAW[n++] = CacheRAW[1];
40    BufferRAW[n++] = CacheRAW[2];
41    BufferRAW[n++] = CacheRAW[3];
42    BufferRAW[n++] = CacheRAW[4];
43    BufferRAW[n++] = CacheRAW[5];
44    BufferRAW[n++] = CacheRAW[6];
45    BufferRAW[n++] = CacheRAW[7];
46    if(Continuous_USBRAW)
47       OAED_USBSendData16(CacheRAW, 8);
48    if(n == RAW_DATA_SIZE){
49       RAW_buffer_full = true;
50       n = 0;
51       OAED_ISRRAWDISABLE();
52    }
53 }
54 #endif
55 
56 /* Z Buffer Replenished ISR custom call definition. */
57 CY_ISR(isr_BufferZRe){
58    OAED_ISRZDisable();
59    Z_buffer_full = true;
60 }
61 
62 /* Lead-off ISR custom call definition. */
63 CY_ISR(isr_LeadOff){
64    isr_Lead_off_Disable();
```

```
65    lead_detected = false;
66    isr_Lead_on_Enable();
67
68    //OAED_Led(false,false,false); // DEBUG MODE ONLY //
69 }
70 CY_ISR(isr_LeadOn){
71    isr_Lead_on_Disable();
72    lead_detected = true;
73    isr_Lead_off_Enable();
74
75    //OAED_Led(false,true,false); // DEBUG MODE ONLY //
76 }
77
78 /* CapReady and CapLow ISR custom call definition. */
79 CY_ISR(isr_Cap_Ready){
80    capacitor_ready = true;
81    isr_CapReady_Disable();
82 }
83 CY_ISR(isr_Cap_Low){
84    capacitor_ready = false;
85    isr_CapReady_Enable();
86 }
87
88 /* Defibrillation ISR custom call definition. */
89 CY_ISR(isr_Defibrillation){
90    /* Start speaker signal */
91    WaveDAC8_Spk_Start();
92    CyDelay(ALARM_TIME);
93    /* Pause ECG acquisition. */
94    OAED_AcquisitionECGPause();
95    /* Perform a biphasic defibrillation. */
96    OAED_BiphasicDefibrillation(50);
97    /* Reset Event data. */
98    OAED_ResetEvent(); // Not sure about this
99    /* Stop the speaker */
100   WaveDAC8_Spk_Stop();
101 }
102
103 /* ISR Init */
104 void OAED_ISRInit(){
105
106    /* Enable custom irs call */
107    isr_CacheECGReplenished_StartEx(isr_CacheECGRe);
108                                          /* ECG Buffer
                                                Replenished */
109    isr_BufferZReplenished_StartEx(isr_BufferZRe);
110                                          /* Z Buffer
                                                Replenished */
111    isr_Lead_off_StartEx(isr_LeadOff); /* Lead-off          */
```

```
112     isr_Lead_on_StartEx(isr_LeadOn);   /* Lead-on           */
113     isr_CapReady_StartEx(isr_Cap_Ready); /* Capacitor ready
           */
114     isr_CapLow_StartEx(isr_Cap_Low);   /* Capacitor low V   */
115     isr_Defibrillate_StartEx(isr_Defibrillation); /*
           Defibrillate */
116     #if(RAW_MODE)                      /* Raw cache
           replenished */
117     isr_CacheRAWReplenished_StartEx(isr_CacheRAWRe);
118     #endif
119
120     OAED_ISRECGDisable();
121     OAED_ISRZDisable();
122     #if(RAW_MODE)
123     OAED_ISRRAWDISABLE();
124     #endif
125
126     return;
127 }
128
129 inline void OAED_ISRECGEnable(){
130     ECG_enabled = true;
131     isr_CacheECGReplenished_Enable();
132 }
133
134 inline void OAED_ISRECGDisable(){
135     ECG_enabled = false;
136     isr_CacheECGReplenished_Disable();
137 }
138
139 inline void OAED_ISRZEnable(){
140     Z_enabled = true;
141     isr_BufferZReplenished_Enable();
142 }
143
144 inline void OAED_ISRZDisable(){
145     Z_enabled = false;
146     isr_BufferZReplenished_Disable();
147 }
148 /* End of function declarations */
149
150 /* [] END OF FILE */
```

OAED_SystemStatus.c

```
1 /* ========================================
2  *
```

```
3   * OPEN SOURCE AED
4   * This header contains all system status
5   * function and global variable
6   * definitions.
7   *
8   * ======================================
9  */
10
11 #include "OAED_SystemStatus.h"
12
13 /* Function declarations */
14 char OAED_LeadOffMode(){
15    /* While in lead-off mode the system will wait for lead
16       detection.
17       */
18
19    /* First we need to reset the event data. Lead off mode
          means that the
20       patient may have changed.
21       */
22    OAED_ResetEvent();
23
24    /* Stop ECG acquisition */
25    if(ECG_enabled)
26       OAED_AcquisitionECGStop();
27
28    /* Then the system lock until leads are attached again.
          */
29    OAED_WaitLeadOn();
30
31    /* Another layer of protection is to evaluate the
          patient impedance. */
32
33    if(OAED_WaitForZ())
34       return measurement_mode;
35
36    /* Leads detected */
37    if(lead_detected){
38       OAED_AcquisitionECGStart();
39       return measurement_mode;
40    }
41
42    /* No leads detected */
43    return lead_off;
44 }
45
46 char OAED_MeasurementMode(){
47    /* While in measurement mode the system checks for,
          lead-off, patient
```

```
48       impedance and VT or VF.
49       */
50
51    /* Start ECG acquisition and check for lead-off */
52    if(!OAED_AcquisitionECGUnpause())
53       return lead_off;
54
55    /* Check for new data */
56    if(ECG_data_pending){
57       /* If new data is available the system evaluate the
            rhytm and if */
58       /* VF/VT are found the system start charging the
            capacitor. */
59       if(OAED_EvaluateRhythm())
60          return charging_capacitor;
61    }
62
63    /* Wait for new data */
64    if(!OAED_WaitForData())
65       return lead_off;
66
67    /* Lead are still on and no VT/VF were found */
68    return measurement_mode;
69 }
70
71 char OAED_ChargingMode(){
72    /* In this mode of operation the system works the same
         as in measurement
73       mode, but the capacitor charging circuit is enabled.
74       This mode is only entered if VF or VT are detected for
            2 out of 3 period
75       of time. (DEFAULT SETTINGS)
76       */
77    static uint8 FalsePositiveCount = 0; /* Init the false
         positive counter */
78
79    /* Start ECG acquisition and check for lead-off */
80    if(!OAED_AcquisitionECGUnpause()){
81       FalsePositiveCount = 0;
82       return internal_discharge;
83       /* In case the system can't detect the leads on it
            activate the internal
84          discharge to avoid possible hazardous situations.
85          */
86    }
87
88    /* Check for new data */
89    if(ECG_data_pending){
```

```
90          /* If new data is available the system evaluate the
               rhytm. */
91          if(!OAED_EvaluateRhythm()){
92             /* If no VF/VT are found the false positive
                  counter is increased.
93               Then if in the last 5 time period no VT/VF
                    events are found, it
94               means that it was a false positive.
95               */
96             if(++FalsePositiveCount > 5){
97                FalsePositiveCount = 0;
98                return internal_discharge;
99             }
100         }
101      }
102      /* If VF/VT is detected the system wait for the
            capacitor to charge. */
103      if(OAED_WaitForCap()){
104         /* If the capacitor is charged before new data is
               available the system
105           reset the positive count and switch to discharge
                 enabled mode.
106           */
107         FalsePositiveCount = 0;
108         return discharge_enabled;
109      }
110      /* WaitForCap return false in case data is ready before
            the capacitor is,
111        but also in case a lead-off is detected.
112        */
113      if(!lead_detected){
114         FalsePositiveCount = 0;
115         return internal_discharge;
116      }
117
118      return charging_capacitor;
119   }
120
121   char OAED_DischargeEnabledMode(){
122      /* In this mode the capacitor is charged and the system
            is waiting for the
123        operator to push the discharge button.
124        */
125
126      /* Start ECG acquisition and check for lead-off */
127      if(!OAED_AcquisitionECGUnpause())
128         /* In case the system can't detect the leads on it
               activate the internal
129           discharge to avoid possible hazardous situations.
```

```
130          */
131       return internal_discharge;
132
133    /* Check for new data */
134    if(ECG_data_pending){
135       /* If new data is available the system evaluate the
             rhytm. */
136       if(!OAED_EvaluateRhythm())
137          /* If no VF/VT are found the system go back in
                charging mode.
138             After 5 more period of time without VT/VF its
                   assumed it was a
139             false positive.
140             */
141          return charging_capacitor;
142    }
143
144    /* Wait for new data */
145    if(!OAED_WaitForData())
146       /* In case of Lead-off */
147       return internal_discharge;
148
149    /* Nothing relevant happened, the system remain in
          discharge enabled mode. */
150    return discharge_enabled;
151 }
152
153 char OAED_InternalDischargeMode(){
154    /* This mode is entered when the charge of the capacitor
          is no longer
155       needed, or is potentially dangerous (ie lead off
             detected).
156       */
157
158    /* Stop ECG acquisition before releasing the charge. */
159    OAED_AcquisitionECGPause();
160
161    /* Disarm and release the charge. */
162    OAED_DisarmDefibrillator(true);
163
164    if(!lead_detected)
165       /* If no leads detected return lead off mode */
166       return lead_off;
167
168    return measurement_mode;
169 }
170
171 void OAED_SetSystemStatus(char Status){
172    static char OldStatus = -1;
```

```
173
174     /* If the status has not changed, return */
175     if(OldStatus == Status)
176         return;
177
178     switch(Status){
179         case charging_capacitor:
180             OAED_Led(true, true, false, false); // Blue and
                    orange
181
182             /* Enable charging circuit. */
183             OAED_EnableChargingCircuit();
184             return;
185         case discharge_enabled:
186             OAED_Led(true, true, true, false); // Blue, orange
                    and green
187
188             /* Charging circuit should be already operative. */
189             /* Arm the defibrillator. */
190             OAED_ArmDefibrillator();
191             return;
192         case internal_discharge:
193             OAED_Led(false, false, false, false); // No led
194
195             OAED_DisableChargingCircuit();
196             return;
197         case measurement_mode:
198             OAED_Led(true, false, false, false); // Only blue
199
200             OAED_DisableChargingCircuit();
201             return;
202         default:
203             OAED_Led(false, false, false, false); // No led
204
205             /* Lead-Off. */
206             OAED_DisableChargingCircuit();
207             return;
208     }
209 }
210
211 /* End of function declarations */
212
213 /* [] END OF FILE */
```

OAED_Time.c

```
1 /* ========================================
```

```
2   *
3   * OPEN SOURCE AED
4   * This file contains all time
5   * function and global variable
6   * definitions.
7   *
8   * ========================================
9  */
10
11 #include "OAED_Time.h"
12 uint16 mscount = 0;
13 uint16 seccount = 0;
14 uint16 mincount = 0;
15 char  TimeStamp[13];
16
17 void OAED_InitTime(){
18    uint8 i;
19
20    mscount = 0;
21    seccount = 0;
22    mincount = 0;
23
24    /* Configure the SysTick timer to generate interrupt
         every 1 ms and start
25       its operation.
26       */
27    CySysTickStart();
28
29    /* Find unused callback slot and assign it to the custom
         callback. */
30    for ( i = 0 ; i < CY_SYS_SYST_NUM_OF_CALLBACKS ; i++ ){
31       if (CySysTickGetCallback(i) == NULL){
32          /* Set callback */
33          CySysTickSetCallback(i, OAED_SysTickISRCallback);
34          break;
35       }
36    }
37
38    return;
39 }
40
41 void OAED_SysTickISRCallback(){
42    /* Add 1 msec and update all the counters. */
43    if( ++mscount == 1000 ){
44       mscount = 0;
45       if( ++seccount == 60 ){
46          seccount = 0;
47          mincount++;
48       }
```

```
49      }
50      return;
51  }
52
53  void OAED_GetTime(){
54      sprintf(TimeStamp, "[%3d:%2d:%3d] ", mincount, seccount,
            mscount);
55      return;
56  }
57
58  /* [] END OF FILE */
```

OAED_USB.c

```
1   /* ========================================
2    *
3    * OPEN SOURCE AED
4    * This file contains all USB
5    * function and global variable
6    * definitions.
7    *
8    * ========================================
9   */
10
11  #include <project.h>
12  #include "OAED_USB.h"
13
14  /* Function declarations */
15  void OAED_USBInit(){
16      USBUART_Start(USBFS_DEVICE, USBUART_5V_OPERATION);
17      OAED_USBConfigure();
18      return;
19  }
20
21  void OAED_USBConfigure(){
22      if( USBUART_IsConfigurationChanged() != 0 ){
23          if( USBUART_GetConfiguration() != 0 ){
24              USBUART_CDC_Init();
25          }
26      }
27      return;
28  }
29
30  void OAED_USBSendString(char Message[]){
31      /* Check USBUART configuration. */
32      OAED_USBConfigure();
33
```

```
34    /* Wait CDC to be ready. */
35    while( USBUART_CDCIsReady() == 0 ){}
36
37    /* Send the message. */
38    USBUART_PutString(Message);
39    return;
40 }
41
42 void OAED_USBSendDataVoid(void* data, size_t nsize){
43    /* Used with macro OAED_USBSendData(data) defined on
         header. */
44
45    /* Treat any data type as int8.
46      If data type is wider than 8-bit, it is required a
           reconstruction on PC
47      side. On this purpose LSB is sent first and MSB last.
48      */
49
50    OAED_USBSendData8((int8*)data, nsize );
51    return;
52 }
53
54 void OAED_USBSendData8(int8 Message[], uint16 n){
55    /* Check USBUART configuration. */
56    OAED_USBConfigure();
57
58    uint16 BC;         // Byte count
59    uint16 MI = 0;     // Message index
60
61    do{
62       /* Wait cdc to be ready. */
63       while( USBUART_CDCIsReady() == 0 ){}
64
65       /* USBUART_PutData can only transfer 64 byte per call.
66         Everything else is lost, so we divide the message
              in 64 byte
67         packages.
68         */
69       BC = (n>64) ? 64 : n;
70
71       /* Send message package. */
72       USBUART_PutData( (const uint8*) &Message[MI], BC);
73
74       /* Update the index and message dimension. */
75       MI += BC;
76       n -= BC;
77
78    }while( n > 0 ); /* Repeat until the whole message is
         sent. */
```

```
79
80    return;
81 }
82
83 void OAED_USBPrintECG(){ // DEPRECATED
84    uint16 i;
85    char Message[USBFS_BUFFER_SIZE];
86
87    sprintf(Message,"-- Data ECG\n");
88    OAED_USBSendString(Message);
89
90    for( i = 0 ; i < ECG_DATA_SIZE ; i++ ){
91       sprintf(Message,"%d - %d\n",i,DataECG[i]);
92       OAED_USBSendString(Message);
93    }
94    OAED_USBSendString("-- Data ECG\n");
95
96    return;
97 }
98
99 void OAED_USBPrintECGB(){ // DEPRECATED
100   uint16 i;
101   char Message[USBFS_BUFFER_SIZE];
102
103   sprintf(Message,"-- Buffer ECG\n");
104   OAED_USBSendString(Message);
105
106   for( i = 0 ; i < ECG_DATA_SIZE ; i++ ){
107      if(BufferECG[i] != 0){
108         sprintf(Message,"%d - %d\n",i,BufferECG[i]);
109         OAED_USBSendString(Message);
110      }
111   }
112   OAED_USBSendString("-- Buffer ECG\n");
113
114   return;
115 }
116
117 void OAED_USBPrintZ(){ // DEPRECATED
118   uint16 i;
119   char Message[USBFS_BUFFER_SIZE];
120
121   sprintf(Message,"-- Data Z\n");
122   OAED_USBSendString(Message);
123
124   for( i = 0 ; i < Z_DATA_SIZE ; i++ ){
125      sprintf(Message,"%d - %d\n",i,DataZ[i]);
126      OAED_USBSendString(Message);
127   }
```

```
128     OAED_USBSendString("\n");
129 
130     return;
131 }
132 
133 void OAED_USBPrintTimeStamp(){ // DEPRECATED BUT STILL IN
        USE
134     #if(OAED_TIME)
135     OAED_USBSendString("\n");
136        OAED_GetTime();
137        OAED_USBSendString(TimeStamp);
138     OAED_USBSendString("\n");
139     #endif
140     return;
141 }
142 
143 void OAED_USBPrintSystemImage(){ // DEPRECATED BUT STILL IN
        USE
144     char Message[USBFS_BUFFER_SIZE];
145     bool tmp;
146 
147     OAED_USBPrintTimeStamp();
148     OAED_USBSendString("\n");
149     sprintf(Message,"-- System Status\n");
150     OAED_USBSendString(Message);
151     sprintf(Message,"ECG_buffer_full :
           %1d\n",ECG_buffer_full);
152     OAED_USBSendString(Message);
153     sprintf(Message,"ECG_data_pending :
           %1d\n",ECG_data_pending);
154     OAED_USBSendString(Message);
155     sprintf(Message,"Z_buffer_full : %1d\n",Z_buffer_full);
156     OAED_USBSendString(Message);
157     sprintf(Message,"lead_detected : %1d\n",lead_detected);
158     OAED_USBSendString(Message);
159     sprintf(Message,"capacitor_ready :
           %1d\n",capacitor_ready);
160     OAED_USBSendString(Message);
161     sprintf(Message,"ECG_enabled : %1d\n",ECG_enabled);
162     OAED_USBSendString(Message);
163 
164     sprintf(Message,"Patient Impedance :
           %1d\n",(int32)floor(PatientImpedance));
165     OAED_USBSendString(Message);
166 
167     OAED_USBSendString("\n");
168     sprintf(Message,"-- Pin Status\n");
169     OAED_USBSendString(Message);
170     tmp = CyPins_ReadPin(Status_Led_Blue) != 0;
```

```
171     sprintf(Message,"Blue led   : %1d\n",tmp);
172     OAED_USBSendString(Message);
173     tmp = CyPins_ReadPin(Status_Led_Orange) != 0;
174     sprintf(Message,"orange led : %1d\n",tmp);
175     OAED_USBSendString(Message);
176     tmp = CyPins_ReadPin(Status_Led_Green) != 0;
177     sprintf(Message,"Green led : %1d\n",tmp);
178     OAED_USBSendString(Message);
179     tmp = CyPins_ReadPin(Charge_En_0) != 0;
180     sprintf(Message,"Charge EN : %1d\n",tmp);
181     OAED_USBSendString(Message);
182     tmp = CyPins_ReadPin(Defibrillation_En_Inner) != 0;
183     sprintf(Message,"Inner Defib EN : %1d\n",tmp);
184     OAED_USBSendString(Message);
185     tmp = CyPins_ReadPin(Defibrillation_En_Outer) != 0;
186     sprintf(Message,"Outer Defib EN : %1d\n",tmp);
187     OAED_USBSendString(Message);
188     tmp = CyPins_ReadPin(Phase_Pin_Phi1) != 0;
189     sprintf(Message,"H-Bridge Phase 1 : %1d\n",tmp);
190     OAED_USBSendString(Message);
191     tmp = CyPins_ReadPin(Phase_Pin_Phi2) != 0;
192     sprintf(Message,"H-Bridge Phase 2 : %1d\n",tmp);
193     OAED_USBSendString(Message);
194     /*
195     tmp = CyPins_ReadPin(Comp_Pin_n) != 0;
196     sprintf(Message,"n-Comparator : %1d\n",tmp);
197     OAED_USBSendString(Message);
198     tmp = CyPins_ReadPin(Comp_Pin_p) != 0;
199     sprintf(Message,"p-Comparator : %1d\n",tmp);
200     */
201     OAED_USBSendString(Message);
202     OAED_USBSendString("\n");
203
204
205     return;
206 }
207
208 void OAED_USBSendI(){ // DEPRECATED
209     char Message[USBFS_BUFFER_SIZE];
210     //bool tmp;
211
212     OAED_USBPrintTimeStamp();
213     //OAED_USBSendString("\n");
214
215     sprintf(Message,"Patient Impedance : %ld\n",(int32)(1000
            * PatientImpedance));
216     OAED_USBSendString(Message);
217     OAED_USBSendString("\n");
218     return;
```

```
219 }
220
221 void OAED_USBSendSystemImage(){
222    int16 data = 0;
223
224    data = OAED_ShiftNAdd(data, ECG_buffer_full );
225    data = OAED_ShiftNAdd(data, Z_buffer_full );
226    data = OAED_ShiftNAdd(data, lead_detected );
227    data = OAED_ShiftNAdd(data, ECG_data_pending );
228    data = OAED_ShiftNAdd(data, capacitor_ready );
229    data = OAED_ShiftNAdd(data, ECG_enabled );
230    data = OAED_ShiftNAdd(data, Z_enabled );
231    data = OAED_ShiftNAdd(data, CyPins_ReadPin(Charge_En_0)
          !=0 );
232    data = OAED_ShiftNAdd(data,
          CyPins_ReadPin(Phase_Pin_Phi1) !=0 );
233    data = OAED_ShiftNAdd(data,
          CyPins_ReadPin(Phase_Pin_Phi2) !=0 );
234    //data = OAED_ShiftNAdd(data, CyPins_ReadPin(Comp_Pin_n)
          !=0 );
235    OAED_USBSendData16(&data,1);
236
237    return;
238 }
239
240 inline int16 OAED_ShiftNAdd(int16 data, bool flag){
241    return (data<<1) + flag;
242 }
243
244 void OAED_USBSendECG(){
245    /* OAED_USBSendData need explicit definition of what is
          sending. */
246    extern int16 DataECG[ECG_DATA_SIZE];
247    OAED_USBSendData(DataECG);
248    return;
249 }
250
251 #if(RAW_MODE)
252 void OAED_USBSendRAW(){
253    /* OAED_USBSendData need explicit definition of what is
          sending. */
254    extern int16 DataRAW[RAW_DATA_SIZE];
255    OAED_USBSendData16(DataRAW, 2000);
256    OAED_USBSendData16(DataRAW + 4000, 2000);
257    return;
258 }
259 #endif
260
261 void OAED_USBSendZ(){
```

```
262     /* OAED_USBSendData need explicit definition of what is
           sending. */
263     extern int16 DataZ[Z_DATA_SIZE];
264     //OAED_USBSendData(DataZ);
265     OAED_USBSendData16(DataZ, 2000);
266     OAED_USBSendData16(DataZ + 4000, 2000);
267     return;
268 }
269
270 void OAED_USBSendBuffer(){
271     /* OAED_USBSendData need explicit definition of what is
           sending. */
272     extern int16 BufferECG[ECG_DATA_SIZE];
273     extern int16 BufferZ[Z_DATA_SIZE];
274     OAED_USBSendData(BufferECG);
275     OAED_USBSendData(BufferZ);
276     return;
277 }
278
279 uint16 OAED_USBGetData(uint8 Message[], bool echo){
280     /* Check USBUART configuration. */
281     OAED_USBConfigure();
282
283     /* This is a non-blocking function, so long there's no
           data to get. */
284     if( USBUART_DataIsReady() == 0 ){
285         return 0;
286     }
287
288     uint16 count;
289
290     /* Get data. */
291     count = USBUART_GetAll(Message);
292
293     /* Echo the message received. */
294     if(echo){
295         char command;
296         command = Message[0];
297         OAED_USBSendString("\n");
298         OAED_USBSendString(&command);
299         OAED_USBSendString("\n");
300     }
301     return count;
302 }
303
304 bool OAED_USBGetCommand(){
305     uint16 count;
306     uint8 Message[USBFS_BUFFER_SIZE] = {' '};
307
```

```
308     /* Fetch data. */
309     count = OAED_USBGetData(Message, OAED_USB_ECHO);
310
311     /* This is a non-blocking function, so long there's no
           data to get. */
312     if(count == 0) return false;
313
314     /* The first character is reserved for the command. */
315     switch(Message[0]){
316        case 'S':
317           /* Send System State Flags. */
318           OAED_USBSendSystemImage();
319           return true;
320        #if(RAW_MODE)
321        case 'R':
322           OAED_USBSendRAW();
323           return true;
324        #endif
325        case 'E':
326           /* Send ECG Data array. */
327           OAED_USBSendECG();
328           return true;
329        case 'Z':
330           /* Send Z Data array. */
331           OAED_USBSendZ();
332           return true;
333        case 'B':
334           /* Send Z and ECG buffer array. */
335           OAED_USBSendBuffer();
336           return true;
337        case 'K':
338           /* Print System image as string. */
339           OAED_USBPrintSystemImage();
340           return true;
341        case 'A':
342           /* Send both ECG and RAW data */
343           OAED_USBSendECG();
344           #if(RAW_MODE)
345              OAED_USBSendRAW();
346           #endif
347           return true;
348        case 'C':
349           Continuous_USBRAW = !Continuous_USBRAW;
350           return true;
351        case 'I':
352           OAED_USBSendI();
353           return true;
354        default:
355           return true;
```

```
356     }
357 
358     return true;
359 }
360 
361 
362 void OAED_USBReceiveData(int16 Data[], uint16 n){
363     int16 i = 0;
364     int16 m = 0;
365     int16 Count;
366     int16 Temp;
367     uint8 Message[512];
368 
369     while( m < n ){
370 
371         Count = OAED_USBGetData(Message, false);
372 
373         if(Count == 0)
374             continue;
375 
376         i = 0;
377         while(Count > 0){
378             Temp = Message[i+1];
379             Temp = Temp<<8;
380             Temp = Message[i] + Temp;
381             Data[m++] = Temp;
382 
383             Count -= 2 ;
384             i += 2 ;
385         }
386     }
387     return;
388 }
389 
390 /* [] END OF FILE */
```

OAED_Wait.c

```
1 /* ========================================
2  *
3  * OPEN SOURCE AED
4  * This file contains all common
5  * function and global variable
6  * definitions.
7  *
8  * ========================================
9 */
```

```
10
11 #include "OAED_Wait.h"
12
13 /* Function declarations */
14
15
16 void OAED_WaitLeadOn(){
17    /* ~~~ WARNING: THIS IS A BLOCKING FUNCTION ~~~ */
18    /* ~~~ EXTRA WARNING: THIS ONLY WORK FOR DC LEAD OFF
          DETECTION ~~~ */
19
20    /* This function wait indefinitely until the electrodes
          are attached to */
21    /* the patient. */
22
23    /* The system blinks the blue led while waiting for the
          patient. */
24    bool yellow_led = false;
25    uint8 i;
26
27    while(!lead_detected){
28       /* Blink the blue led with a period of 1sec. */
29       for(i = 0 ; i<5 ; i++){
30          /* Check for lead detection every 100ms. */
31          if(lead_detected){
32             return;
33          }
34          CyDelay(100);
35       }
36       yellow_led = !yellow_led;
37       OAED_Led(false, false, false, yellow_led);
38    }
39 }
40
41 bool OAED_WaitForData(){
42    /* ~~~ WARNING: THIS IS A BLOCKING FUNCTION ~~~ */
43
44    /* Return false if detect lead-off, otherwise return
          true when new data */
45    /* is available. */
46
47    while(!ECG_buffer_full){
48
49       if(!lead_detected)
50          return false;
51       /* Check for z Data */
52       if(Z_buffer_full){
53          OAED_CopyZBuffer();
54          if(!OAED_EvaluateImpedanceDC())
```

```
55              return false;
56           OAED_AcquisitionZ();
57        }
58        /* Wait 10 ms */
59        CyDelay(10);
60     }
61
62     /* Get data */
63     OAED_CopyECGBuffer();
64
65     return true;
66  }
67
68  bool OAED_WaitForZ(){
69     /* ~~~ WARNING: THIS IS A BLOCKING FUNCTION ~~~ */
70
71     /* Return false if detect lead-off, otherwise return
          true. */
72
73     OAED_AcquisitionZ();
74     /* Wait for data */
75     while(!Z_buffer_full){
76        if(!lead_detected)
77           return false;
78     }
79
80     /* Get data */
81     OAED_CopyZBuffer();
82
83     /* Evaluate data and return true if a lead-on is
          assumed. */
84     return OAED_EvaluateImpedanceDC();
85  }
86
87  bool OAED_WaitForCap(){
88     /* ~~~ WARNING: THIS IS A BLOCKING FUNCTION ~~~ */
89     /* Wait until the capacitor is ready. If meanwhile new
          data is available or
90       a lead-off is detected return false.
91       */
92     while(!ECG_buffer_full){
93        if(!lead_detected)
94           return false;
95        /* Check for new Z data */
96        if(Z_buffer_full){
97           OAED_CopyZBuffer();
98           if(!OAED_EvaluateImpedanceDC())
99              return false;
100          OAED_AcquisitionZ();
```

```
101         }
102         /* Check for capacitor ready */
103         if(capacitor_ready)
104            return true;
105         /* Wait 10 ms */
106         CyDelay(10);
107     }
108 
109     /* Get Data */
110     OAED_CopyECGBuffer();
111     return false;
112 }
113 
114 /* End of function declarations */
115 
116 /* [] END OF FILE */
```

Annex B - Matlab scripts

psoc_talker.m

```
classdef psoc_talker
   properties
      s        % Serial port
      message  % Message to send to the psoc
      verbose  % Decide wether the class should give or not
         some feedback
   end % end of properties

   methods

      function obj = psoc_talker(com, m, v)
         try
            obj.s = serial(com, 'Baudrate', 9600);
            fopen(obj.s);
         catch
            obj.s = NaN;
            disp(['Could not init ' com ' port'])
            disp(instrfind)
            return;
         end
         if (nargin < 3)
            obj.verbose = false;
         else
            obj.verbose = v;
            endolaaonfsflnd
         end
         if (nargin < 2)
            obj.message = [];
            return;
         end
         obj.message = m;
         return;
      end % end of constructor

      function delete(obj)
         try
            fclose(obj.s);
         catch
         end
```

```
39          delete(obj.s);
40          return;
41       end % end of delete
42
43       function success = reload(obj)
44          if (strcmp(obj.s.Status,'open'))
45             try
46                fclose(obj.s);
47             catch
48                disp('Could not close')
49                success = false;
50                return;
51             end
52          end
53          try
54             fopen(obj.s);
55          catch
56             disp('Could not re-open')
57             success = false;
58             return;
59          end
60          success = true;
61          return;
62       end % end of reload
63
64       function success = sendsingle(obj)
65          % Outdated function
66          % Send data one by one
67          time = tic;
68          success = false;
69          % Check and open connection
70          if (~strcmp(obj.s.status, 'open'))
71             if (~obj.reload)
72                return;
73             end
74          end
75          n = length(obj.message);
76          for k = 1:n
77             % Send one 16-bit element per iteration
78             try
79                fwrite(obj.s, obj.message(k), 'int16');
80             catch
81                disp([num2str(toc(time)) ' : Could not
                      write']);
82                return;
83             end
84
85             % Wait for the psoc to read it
86             while (obj.s.BytesToOutput ~= 0)
```

```
87              end
88           end
89           success = true;
90           if (obj.verbose)
91              disp([num2str(toc(time)) ' : Finished']);
92           end
93        end % end of send
94
95        function success = send(obj)
96           time = tic;
97           success = false;
98           % Check and open connection
99           if (~strcmp(obj.s.status, 'open'))
100             if (~obj.reload)
101                return;
102             end
103          end
104          t = tic;
105          while (obj.s.BytesAvailable ~= 0 ...
106             || ~strcmp(obj.s.TransferStatus,'idle'))
107             % Timeout 6sec
108             if (toc(t) > 10)
109                message = NaN;
110                disp([num2str(toc(time)) ' : Psoc is busy']);
111                % Received no reply
112                return;
113             end
114          end
115          % n is the length of the message to send
116          n = length(obj.message);
117          % The buffer is not unlimited, m is the number of
                16-bit elements the
118          % buffer can store.
119          m = obj.s.OutputBufferSize / 2;
120          % Message Index
121          MI = 0;
122
123          while ( n > 0 )
124             % Repeat until all the elements of the message
                   are sent
125
126             % Each iteration send m elements, the last
                   iteration send the
127             % remaining n elements.
128             if ( m > n )
129                m = n;
130             end
131
132             % Try to send data
```

```
133             try
134                 fwrite(obj.s, obj.message(MI+1 : MI + m ),
                        'int16');
135             catch
136                 disp([num2str(toc(time)) ' : Could not
                        write']);
137                 return;
138             end
139
140             % Update the message index and the remaining
                    message length
141             MI = MI + m;
142             n = n - m;
143
144             % Wait for the psoc to read it all
145             while (obj.s.BytesToOutput ~= 0)
146             end
147         end
148         % Success
149         success = true;
150         % Visual feedback
151         if (obj.verbose)
152             disp([num2str(toc(time)) ' : Finished']);
153         end
154     end % end of send
155
156     function success = sendchar(obj, cha)
157         % Send a single character to the psoc
158         time = tic;
159         success = false;
160         % Check and open connection
161         if (~strcmp(obj.s.status, 'open'))
162             if (~obj.reload)
163                 return;
164             end
165         end
166         % if the message is longer than a single
                character, return false
167         if (length(cha) > 1)
168             disp('This method should only be used to send
                    one character')
169             return;
170         end
171         try
172             fwrite(obj.s, cha);
173         catch
174             disp([num2str(toc(time)) ' : Could not write']);
175             return;
176         end
```

```
177          success = true;
178
179          if (obj.verbose)
180             disp([num2str(toc(time)) ' : Finished']);
181          end
182       end % end of send
183
184       function [message] = receive(obj)
185          time = tic;
186          % Check and open connection
187          if (~strcmp(obj.s.status, 'open'))
188             if (~obj.reload)
189                return;
190             end
191          end
192          t = tic;
193          while ( obj.s.BytesAvailable == 0 ) %||
                obj.s.BytesToOutput ~= 0
194             % Timeout 6sec
195             if (toc(t) > 10)
196                message = NaN;
197                disp([num2str(toc(time)) ' : Psoc is busy']);
198                % Received no reply
199                return;
200             end
201          end
202          % Read the message head
203          % First byte is data width: 8, 16, or 32 bit
204          [out] = fread(obj.s, 3)';
205          bit = out(1);
206          % Second and third byte are message length
207          n = byte2word([out(2) out(3)]);
208          if (n<0)
209             n = byte2word([out(3) out(2)]);
210          end
211          % Initialize message
212          message = [];
213          % Read the message
214          while (true)
215             % Wait for data to be ready
216             t = tic;
217             while (obj.s.BytesAvailable == 0)
218                % Timeout 6sec
219                if (toc(t) > 6)
220                   message = NaN;
221                   display([num2str(toc(time)) ' :
                         Timeout']);
222                   return;
223                end
```

```
224             end
225
226             % Calculate how many bytes of the buffer still
                   belong to this
227             % message
228             if (obj.s.BytesAvailable + length(message) > n
                   * bit/8)
229                m = n*bit/8 - length(message);
230             else
231                m = obj.s.BytesAvailable;
232             end
233
234             % Try to read the buffer
235             try
236                message = [message fread(obj.s, m)'];
237             catch
238                disp([num2str(toc(time)) ' : Could not
                      read']);
239                return;
240             end
241
242             % Have we done?
243             if (length(message) == n * bit/8)
244                break;
245             end
246          end
247
248          % 16 and 32 bit data need reconstruction.
249          if (bit == 16)
250             message = byte2word(message);
251          elseif (bit == 32)
252             message = byte2long(message);
253          end
254
255          if (obj.verbose)
256             disp([num2str(toc(time)) ' : Finished']);
257          end
258       end % end of receive
259
260    end % end of methods
261 end % end of class
```

OAED_TCI.m

```
1  function [SCA, TCIm, TCI] = OAED_TCI(ecg, fs, echo)
2  %% find n
3
```

```
4     if(nargin == 2)
5        echo = false;
6     end
7     nl = length(ecg);
8     nsec = 1 * fs;
9     n = nl/nsec;
10
11 %% find maximum
12
13    maxECG = zeros(n,1);
14    ecg2 = zeros(n,nsec);
15    for k = 1:n
16       ecg2(k,:) = ecg(1+ nsec*(k-1) : nsec*(k));
17       maxECG(k) = max( ecg2(k,:) );
18    end
19
20 %% binarize the ecg signal
21
22    binaryECG = zeros(n,nsec);
23    for k = 1:n
24       if(maxECG(k) > 0)
25          binaryECG(k,:) = ( ecg(1+ nsec*(k-1) : nsec*(k)) >
               (0.2 * maxECG(k)) );
26       else
27          binaryECG(k,:) = ( ecg(1+ nsec*(k-1) : nsec*(k)) >
               (1.8 * maxECG(k)) );
28       end
29    end
30
31 %  for k=1:4 TC([1:fs]+(k-1)*fs) = binaryECG(k,:); end
32 %  figure, hold on,
   plot([1:1000]/250,2*ecg/max(ecg),'LineWidth', 1),
   plot([1:1000]/250,TC,'LineWidth', 2)
33
34 %% t evaluation
35
36    t = zeros(k,4);
37    for k = 1:n
38       if(binaryECG(k,1) == 1)
39          t(k,2) = 0;
40       else
41          t(k,2) = (find(binaryECG(k,:) == 1, 1, 'first')-1)
               / fs ;
42       end
43       if(binaryECG(k,end) == 1)
44          t(k,3) = 0;
45       else
46          t(k,3) = (nsec - find(binaryECG(k,:) == 1, 1,
               'last')+1) / fs;
```

```
47        end
48
49        if ( k == 1 )
50           continue;
51        end
52        t(k,1) = t(k-1,3);
53        t(k-1,4) = t(k,2);
54     end
55
56  %% N evaluation
57
58     N = zeros(n-2,1);
59     for k = 2:(n-1)
60        for z = 1:nsec-1
61           if( binaryECG(k,z) ~= binaryECG(k,z+1) )
62              N(k-1) = N(k-1) + 1;
63           end
64        end
65        N(k-1) = ceil(N(k-1)/2);
66        if(t(k,2) == 0 && t(k,3) == 0)
67           N(k-1) = N(k-1)+1;
68        end
69     end
70
71  %% TCI evaluation
72
73     TCI = zeros(n-2,1);
74     for k = 2:(n-1)
75        if(t(k,1) ~= 0)
76           t12 = t(k,2)/( t(k,1)+t(k,2) );
77        else
78           t12 = 0;
79        end
80        if(t(k,3) ~= 0)
81           t34 = t(k,3)/( t(k,3)+t(k,4) );
82        else
83           t34 = 0;
84        end
85        TCI(k-1) = 1000 / ( N(k-1)-1 + t12 + t34 );
86     end
87
88  %% Rhythm evaluation
89     TCIm = mean(TCI);
90     TCIcrit = 400; % TCI > 400ms => normal sinus rhythm (NSR)
91     if(TCIm <= TCIcrit)
92        muvf = 105;
93        sigmavf = 6.5;
94
95        muvt = 220;
```

```
96        sigmavt = 16.5;
97
98        alfa = 1 / 100;
99        beta = 1 / 100;
100
101       F = (1/sigmavf^2) * sum((TCI - muvf).^2) -
              (1/sigmavt^2) * sum((TCI - muvt).^2);
102       FVT = 2*log((1-beta)/alfa) +
              2*(k-2)*log(sigmavt/sigmavf);
103       FVF = 2*log(beta/(1-alfa)) +
              2*(k-2)*log(sigmavt/sigmavf);
104
105       if(F >= FVT)
106          if(echo)
107             disp('VT confirmed');
108          end
109          SCA = true;
110       elseif(F <= FVF)
111          if(echo)
112             disp('VF confirmed');
113          end
114          SCA = true;
115       else
116          if(echo)
117             disp('Not enough data');
118          end
119          SCA = false;
120       end
121    else
122       if(echo)
123          disp('Normal sinus rhythm');
124       end
125       SCA = false;
126    end
127 end
```

OAED_VFfilter.m

```
1  function [SCA, l] = OAED_VFfilter (ecg, fs, echo)
2  %% Period evaluation
3
4     if(nargin == 2)
5        echo = false;
6     end
7
8     %T = 2*pi * (sum(abs(ecg))/ sum( abs(ecg - [0
          ecg(1:end-1)]) ) );
```

```
    T = floor(1 + 2*pi * (sum(abs(ecg))/ sum( abs(ecg - [0
       ecg(1:end-1)] ))));

%% Leakage evaluation
   %nt = floor(T * fs/1000);
   nt = floor(T/2);
   l = sum(abs( ecg(nt+1:end) + ecg(1:end-nt) ));
   l = l/ (sum( abs(ecg(nt+1:end)) + abs(ecg(1:end-nt)) ));

%% rhythm evaluation
   l0 = 0.625;
   if(l>l0)
      if(echo)
         disp('Normal rhythm')
      end
      SCA = false;
   else
      if(echo)
         disp('VF confirmed')
      end
      SCA = true;
   end
end
```

OAED_TCSC.m

```
function [SCA, Na, N] = OAED_TCSC(ecg, fs, echo)
%%

   if(nargin == 2)
      echo = false;
   end
   Ls = 3;
   Le = length(ecg)/fs;
   ns = Ls * fs;
   m = Le - Ls + 1;
   %ne = length(ecg);

%% Split signals
   ECGsplitted = zeros(m,ns);
   for k = 0:m-1
      ECGsplitted(1+k,:) = ecg( 1 + k*fs : (k+Ls) * fs );
   end

%% Cosine window
   Lw = 0.25;
   nw = floor(Lw*fs);
```

```
22    W1 = 0.5*(1 - cos(4*pi*(0:1/fs:Lw-1/fs)));
23    W2 = 0.5*(1 - cos(4*pi*(1/fs+Ls-Lw:1/fs:Ls)));
24
25    ECGfiltered = ECGsplitted;
26    for k = 1:m
27       ECGfiltered(k, 1:nw) = ECGsplitted(k, 1:nw).*W1;
28       ECGfiltered(k, end-nw +1:end) = ECGsplitted(k, end-nw
             +1:end).*W2;
29    end
30
31 %% Binarization
32    binaryECG = zeros(m,ns);
33    for k = 1:m
34       binaryECG(k,:) = ( abs(ECGfiltered(k,:)) > 0.2 * max(
             abs(ECGfiltered(k,:)) ) );
35    end
36
37 %% N calculus
38    N = zeros(m,1);
39    for k = 1:m
40       N(k) = sum(binaryECG(k,:))/ns * 100;
41    end
42    Na = sum(N)/(m);
43
44 %% Rhythm evaluation
45    % 25 < Nd < 35 for more sensibility
46    Nd = 48; % High specificity
47    if( Na>=Nd )
48       if(echo)
49          disp('VF confirmed');
50       end
51       SCA = true;
52    else
53       if(echo)
54          disp('Normal sinus rhythm');
55       end
56       SCA = false;
57    end
58
59 end
```

OAED_PSR.m

```
1  function [SCA, d, di] = OAED_PSR (ecg, fs, echo)
2
3     if(nargin == 2)
4        echo = false;
```

```
    end
    tau = floor(fs * 0.5);
    d0 = 0.2;
    n = 40;

    %%
    ecg2 = ecg - min(ecg);
    ecg2 = 2^31*ecg2/max(ecg2);
    delta = ceil(max(ecg2+1)/n);

    %%
    di = zeros(n);
    for k = 1:length(ecg)-tau
       zx = 1 + floor( ecg2(k)/delta );
       zy = 1 + floor( ecg2(k + tau)/delta );
       %di(zx,zy) = di(zx,zy) + 1;
       di(zx,zy) = 1;
    end

    d = sum(sum(di))/(n*n);

    %%
    if(d>d0)
       if(echo)
          disp('VF confirmed');
       end
       SCA = true;
    else
       if(echo)
          disp('Normal sinus rhythm');
       end
       SCA = false;
    end

end
```

OAED_HILB.m

```
function [SCA, d, di] = OAED_HILB(ecg, fs, echo)

    if(nargin == 2)
       echo = false;
    end
    d0 = 0.25;
    n = 40;

    %%
```

```
10    ecg = 2^31*ecg/max(ecg);
11    ecgh = hilbert(ecg, 2^ceil(log2(4*fs)));
12    ecg1 = ecgh - min(real(ecgh)) - min(imag(ecgh))*1i;
13    deltax = ceil(max(real(ecg1)+1)/n);
14    deltay = ceil(max(imag(ecg1)+1)/n);
15
16    %%
17    di = zeros(n);
18    for k = 1:length(ecg1)
19       zx = 1 + floor( real(ecg1(k))/deltax );
20       zy = 1 + floor( imag(ecg1(k))/deltay );
21       %di(zx,zy) = di(zx,zy) + 1;
22       di(zx,zy) = 1;
23    end
24
25    d = sum(sum(di))/(n*n);
26
27    %%
28    if(d>d0)
29       if(echo)
30          disp('VF confirmed');
31       end
32       SCA = true;
33    else
34       if(echo)
35          disp('Normal sinus rhythm');
36       end
37       SCA = false;
38    end
39
40 end
```

OAED_ECGanalysis.m

```
1  function [SCA, results, values, ecgf] =
      OAED_ECGanalysis(dat_list, n, suppress_plot)
2  %% Default parameters
3  if(nargin == 1)
4     n = size(dat_list, 1);
5  end
6  if(nargin < 3)
7     suppress_plot = false;
8  end
9
10 %% Loop
11 for k = 1:n
12    time = tic;
```

```
13    %% Load next signal
14    disp([num2str(toc(time)) ' : Loading '
         dat_list(k).name]);
15    [ecg, fs, te] = rdsamp(dat_list(k).name, 1);
16    ecg = ecg';
17    te = te';
18    disp([num2str(toc(time)) ' : Loaded']);
19
20    %% apply biquad iir filter
21    disp([num2str(toc(time)) ' : Filtering']);
22    [ecgf] = OAED_FiltECG(ecg, fs, 3, 40);
23
24    %% Perform rhythm evaluation
25    disp([num2str(toc(time)) ' : Evaluating rhythm']);
26    t = tic;
27    [SCA, results, values] = OAED_RhythmEvaluation(ecgf,
         fs,...
28                                        ~suppress_plot,
                                             ~suppress_plot);
29    exectime = toc(t);
30    disp([num2str(toc(time)) ' : Done']);
31
32    %% Display numeric result
33    disp([num2str(toc(time)) ' : Results']);
34    disp(['Total execution time : ' num2str(toc(time))]);
35    disp(['TCI           : ' num2str(sum(SCA(1,:)))]);
36    disp(['VF filter     : ' num2str(sum(SCA(2,:)))]);
37    disp(['TCSC          : ' num2str(sum(SCA(3,:)))]);
38    disp(['PSR           : ' num2str(sum(SCA(4,:)))]);
39    disp(['HILB          : ' num2str(sum(SCA(5,:)))]);
40    disp(['SCA evauation'])
41    disp(['At least 2 positive : '
         num2str(length(find(sum(SCA) >= 2)))])
42    disp(['At least 3 positive : '
         num2str(length(find(sum(SCA) >= 3)))])
43    disp(['At least 4 positive : '
         num2str(length(find(sum(SCA) >= 4)))])
44    disp(['5 positive    : ' num2str(length(find(sum(SCA) ==
         5)))])
45
46    if( k ~= n)
47       disp('Press enter to move to the next signal');
48    else
49       disp('Press enter to finish');
50    end
51    pause();
52
53 end
```

OAED_FiltECG.m

```
function [ecgf] = OAED_FiltECG(ecg, fs, hp, lp, notch)
   %% Default parameters
   if(nargin < 5)
      notch = false;
   end
   if(nargin < 4)
      lp = 40;
   end
   if(nargin < 3)
      hp = 3;
   end
   if(nargin < 2)
      return;
   end

   %% apply biquad HP iir filter
   [zhi,phi,khi] = butter(2, hp/(2*fs),'high'); % 2nd order
   soshi = zp2sos(zhi,phi,khi);
   ecgf = sosfilt(soshi, ecg);

   %% apply biquad LP iir filter
   [zhi,phi,khi] = butter(4, lp/(2*fs),'low'); % 4th order
   soshi = zp2sos(zhi,phi,khi);
   ecgf = sosfilt(soshi, ecgf);

   return; % notch still need testing
   %% apply biquad notch iir filter
   if(notch)
      [zhi,phi,khi] = butter(2, [40 60]/(2*fs),'stop'); %
         2nd order
      soshi = zp2sos(zhi,phi,khi);
      ecgf = sosfilt(soshi, ecgf);
   end

end
```

OAED_RecognitionPlot.m

```
function OAED_RecognitionPlot(te, ecg, threshold, c, cmax,
   wl)
   if(nargin == 4)
      cmax = max(c);
   end
   if(nargin < 6)
```

```
6        wl = 4;
7     end
8     figure, hold on;
9     plot( te, 2*ecg/max(ecg));
10    plot( ones(1, ceil(te(end))) * threshold, 'LineWidth',
         3);
11    plot( (1:length(c))*wl, c/cmax, 'LineWidth', 2);
12    hold off;
13    return;
14 end
```

OAED_RhythmEvaluation.m

```
1  function [SCA, result, values ] =
      OAED_RhythmEvaluation(ecg, ...
2                                              fs, ...
3                                              PlotEval, ...
4                                              PlotResults, ...
5                                              wl)
6
7     %% set default arguments
8     switch nargin
9        case 2
10          PlotEval = false;
11          PlotResults = false;
12          wl = 4;
13       case 3
14          PlotResults = false;
15          wl = 4;
16       case 4
17          wl = 4;
18       otherwise
19          disp('Input arguments are:')
20          disp('*Ecg array 1xn;')
21          disp('*fs sampling frequency;')
22          disp('Plot Evaluation results (default false);')
23          disp('Plot individual results (default false);')
24          disp('Window lenght (default 4);')
25          disp('* are mandatory')
26          return;
27    end
28
29    %% init
30    m = floor( length(ecg)/(wl*fs) ); % m = number of wl
         window
31    % result init
32    TCIm = zeros(1, m);
```

```
33    VF = zeros(1, m);
34    TCSC = zeros(1, m);
35    PSR = zeros(1, m);
36    HIL = zeros(1, m);
37    SCA = zeros(5, m);
38    % values init
39    TCI = zeros(m, 2*wl - 2);
40    TCSCN = zeros(m, 2*wl - 2);
41    dPSR = zeros(40, 40, m);
42    dHILB = zeros(40, 40, m);
43    old = [];   % TCI and TCSC also use the last ecg segment
44
45    %% Evaluate each window
46    for k = 1:m
47       % Take the k-th, wl ECG window
48       ecgw = ecg( (k-1) * wl*fs + 1 : k * wl*fs );
49
50       [SCA(1,k), TCIm(k), tmp] = OAED_TCI([old ecgw], fs);
51       if(k == 1)
52          TCI(k,:) = [zeros(1, wl) tmp'];
53       else
54          TCI(k,:) = tmp';
55       end
56       [SCA(2,k), VF(k)] = OAED_VFfilter(ecgw, fs);
57       [SCA(3,k), TCSC(k), tmp] = OAED_TCSC([old ecgw], fs);
58       if(k == 1)
59          TCSCN(k,:) = [zeros(1, wl) tmp'];
60       else
61          TCSCN(k,:) = tmp';
62       end
63       [SCA(4,k), PSR(k), dPSR(:,:,k)] = OAED_PSR(ecgw, fs);
64       [SCA(5,k), HILB(k), dHILB(:,:,k)] = OAED_HILB(ecgw,
            fs);
65       old = ecgw;
66    end
67    %% set result and individual values in a more compact
         list
68    result = {TCIm, VF, TCSC, PSR, HILB};
69    values = {TCI, TCSCN, dPSR, dHILB};
70
71    %% plot evaluation
72    if( PlotEval )
73       te = [ 0 : length(ecg)-1 ]/fs;
74       te2 = [ 0 : length(SCA)-1 ];
75       scale = 10;
76       figure, hold on, grid on,
77       plot(te, scale * ecg/max(ecg) ), % ECG
78       plot( (te2 + 0.5)*wl, sum(SCA,1), '*', 'LineWidth',
            3), % Sum
```

```
79        plot( (te2 + 0.1)*wl, 5*SCA(1,:), 'o', 'LineWidth',
              3), % TCI
80        plot( (te2 + 0.3)*wl, 5*SCA(2,:), 'o', 'LineWidth',
              3), % VF filter
81        plot( (te2 + 0.5)*wl, 5*SCA(3,:), 'o', 'LineWidth',
              3), % TCSC
82        plot( (te2 + 0.7)*wl, 5*SCA(4,:), 'o', 'LineWidth',
              3), % PSR
83        plot( (te2 + 0.9)*wl, 5*SCA(5,:), 'o', 'LineWidth',
              3), % HILB
84        legend('ECG', 'Total positive', 'TCI', 'VF', 'TCSC',
              'PSR', 'HILB'),
85        set(gca, 'xtick', [0: wl : floor(te(end)) + 1] ),
86        set(gca, 'ytick', (0:1:5) ),
87        title('Rhythm evaluation results'),
88        hold off;
89     end
90
91     %% plot individual results
92     if( PlotResults )
93        te = [ 0 : length(ecg)-1 ]/fs;
94        % TCI
95        OAED_RecognitionPlot(te, ecg, 0.4, TCIm, 1000);
96        title('TCI individual results')
97        legend('ECG', 'Threshold', 'TCI');
98        % VF filter
99        OAED_RecognitionPlot(te, ecg, 0.625, VF, 1);
100       title('VF individual results')
101       legend('ECG', 'Threshold', 'VF');
102       % TCSC
103       OAED_RecognitionPlot(te, ecg, 0.48, TCSC, 100);
104       title('TCSC individual results')
105       legend('ECG', 'Threshold', 'TCSC');
106       % PSR
107       OAED_RecognitionPlot(te, ecg, 0.2, PSR, 1);
108       title('PSR individual results')
109       legend('ECG', 'Threshold', 'PSR');
110       % HILB
111       OAED_RecognitionPlot(te, ecg, 0.25, HILB, 1);
112       title('HILB individual results')
113       legend('ECG', 'Threshold', 'HILB');
114    end
115
116    return;
117 end
```

PSOC_Algorithms.m

```
function [SCA, EvalTime ] = PSOC_Algorithms(psoc, ecg, fs)
   %% Params
   if(nargin == 2)
      fs = 500;
   elseif( fs ~=500 )
      nf = floor(fs/500);
      ecg2 = zeros(length(ecg)*2, 1);


      % Optional - Scale up values
      ecg2(1:2:end) = int16( 2^15* ecg/max(ecg) );
      %ecg2(1:2:end) = int16(ecg);
      for k = 2:2:length(ecg2)-2
         ecg2(k) = ecg2(k-1) + ecg2(k+1);
         ecg2(k) = ecg2(k)/2;
      end
      ecg2 = ecg2';
      fs = 500;
   else
      ecg2 = ecg;
   end

   %% Init
   n = length(ecg2);
   m = 4 * 500;

   ns = floor(n/m);

   EvalTime = zeros(1,ns);
   SCA = zeros(5,ns);

   %% Transfer
   psoc.reload;
   for k = 1:ns
      psoc.message = ecg2( 1 + (k-1)*m : k*m );
      psoc.send;
      pause(0.1);
      %TCI1(k,:) = psoc.receive;
      %TCI2(k,:) = psoc.receive;
      SCA(:,k) = psoc.receive;
      EvalTime(k) = psoc.receive;


      % Feedback
      if(k == floor(ns/10))
         disp('10%');
```

```
47        elseif( k == floor(ns*1/4))
48           disp('25%');
49        elseif( k == floor(ns/2))
50           disp('50%');
51        elseif( k == floor(ns*3/4))
52           disp('75%');
53        elseif( k == floor(ns*9/10))
54           disp('90%');
55        end
56
57     end
58
59  end
```

ecg_plot.m

```
1  % Custom plot function
2  function ecg_plot(ecg, name)
3     n = size(ecg,1);
4     te = [1:2000]/500;
5
6     if(n == 1)
7        plot(te,ecg);
8        return;
9     end
10
11    for l = 1:ceil(n/6)
12       figure
13       m = (l-1)*6;
14       for k = 1:min([6 n-m])
15          subplot(3,2,k), plot(te,ecg(k+m,:))
16          title([name '-' num2str((k+m))])
17       end
18    end
19    return;
20 end
```

fourier_plot.m

```
1  function fourier_plot(sig, f, nf)
2
3     if(nargin == 1)
4        f = 500;
5     elseif (nargin < 3)
6        nf = 2^ceil(3+log2(length(sig)));
```

```
7     end
8
9     sig_fshift = fftshift(abs(fft(sig', nf)));
10    nl = length(sig_fshift);
11    df = f / (nl);
12    fk = df * (-nl/2:nl/2-1);
13    % figure, plot(fk,sig_fshift)
14    figure, plot(fk,20*log10(sig_fshift))
15 end
```

signword.m

```
1  % Convert from uint16 to int16
2  function d = signword(data)
3     d = data;
4     for k = 1:length(data)
5           if(bitget(data(k),16) == 1)
6              d(k) = -1 * int16( bitcmp( uint16(data(k)) ) )
                  -1;
7           end
8     end
9  end
```

byte2long.m

```
1  function L = byte2long(B)
2     % Compose 4 Byte in 1 Word
3     L = B(1:4:end);
4     L = L + bitshift(B(2:4:end),8);
5     L = L + bitshift(B(3:4:end),16);
6     L = L + bitshift(B(4:4:end),24);
7
8     % Recover the sign
9     for k = 1:length(L)
10       if(bitget(L(k),32) == 1)
11          L(k) = -1 * int32( bitcmp( uint32(L(k)) ) ) -1;
12       end
13    end
14
15    return;
16 end
```

byte2word.m

```
function W = byte2word(B)
   % Compose 2 Byte in 1 Word
   W = B(1:2:end) + bitshift(B(2:2:end),8);

   % Recover the sign
   for k = 1:length(W)
      if(bitget(W(k),16) == 1)
         W(k) = -1 * int16( bitcmp( uint16(W(k)) ) ) -1;
      end
   end

   return;
end
```

Glossary

$\Delta\Sigma$-ADC Delta-Sigma Analog Digital Converter. vii, 111–113, 116, 117, 121, 130–134, 137, 145

ABA Analog Block Array. 67, 104, 106

ACF Autocorrelation Function. 147

ADC Analog Digital Converter. vii, 111–113, 116, 117, 121, 130–134, 137, 145, 267

AED Automated External Defibrillator. v, vi, 1, 2, 10, 15–18, 21, 31–33, 38, 39, 41, 42, 45, 48, 51, 52, 57–63, 65, 66, 70, 71, 81, 103, 104, 116, 128, 129, 146, 169, 170

BME Biomedical Engineering. 5–7

C-B Control Board. vi, 65, 67, 103, 104, 119, 121, 122, 127, 160, 170

CEN European Committee for Standardization. 29–31

CENELEC European Committee for Electrotechnical Standardization. 29, 30, 68, 125

CPU Central Processing Unit. 104, 108, 109, 131–135, 170

DAC Digital (to) Analog Converter. 119, 121

DBA Digital Block Array. 67, 104

DFB Digital Filter Block. 112, 115, 121, 132–136, 145

DMA Direct Memory Access. 104, 112, 115, 130–132, 134, 135

ECG Electrocardiogram. v, vii, viii, 11, 13, 15, 48, 51–53, 61–63, 67, 76, 103, 104, 110–117, 120, 127, 128, 130–132, 134–136, 139–141, 145–152, 154–163, 170

ENs European Standards. 29–31, 33

EP Essential Performance. 22, 34–36, 38, 39, 47, 48, 51, 52, 114, 170

ERs Essential Requirements. 22–24, 26, 28, 33, 56, 65, 70, 114, 170

ESD electrostatic discharge. 62, 63

ETSI European Telecommunications Standards Institute. 29, 30

EU European Union. 1, 5, 8, 21, 29, 30, 113

FFT Fast Fourier Transform. 148

GMDN Global Medical Device Nomenclature. v, 21–23, 32

HTA Hilbert Transform algorithm. 150, 151, 158

HV-B High-Voltage Board. vi, 65–67, 70, 98, 103, 121, 127, 170

ICT Information and Communication Technologies. 30

IEC International Electrotechnical Commission. 30, 31, 33–35, 68, 150, 170

IGBT Insulated Gate Bipolar Transistor. 78, 79

ISO International Organization for Standardization. 29–31, 68

ISR Interrupt Service Routine. 130, 132, 133, 136

MCU Micro Controller Unit. 67, 104, 119, 134

MDD 93/42 EEC medical devices directive 93/42. ii, v, 21–25, 28, 29, 32, 33, 39, 169

ME medical electrical. v, 21, 33–36, 38–46, 48, 49, 51–57, 60, 63, 70, 170

NSR Normal Sinus Rhythm. 146–151, 153, 156, 160, 163

OAED Open-source Automated External Defibrillator. v–vii, 2, 10, 16–18, 64, 70, 75, 76, 78, 80, 91, 97, 101, 103, 105, 109–111, 113, 115, 117–121, 127–135, 137–141, 145, 146, 150, 151, 162, 163, 170

PCB Printed Circuit Board. 104, 119, 129

PEMS programmable electrical medical systems. vi, 47, 48, 50

PSoC Programmable System on Chip. vi, vii, 67, 76, 81, 91, 104–107, 111–113, 115, 117–123, 127, 129, 132, 134, 137, 148, 162, 170

PSR Phase Space Reconstruction. 150, 151, 158, 159

PWM Pulse-Width Modulation. 77, 82, 88, 121

RCC Ringing Choke Converter. 66, 86, 90, 91, 93, 94, 98, 121

SCA Sudden Cardiac Arrest. vii, 1, 2, 10, 11, 15–18, 32, 61, 81, 110, 111, 130, 131, 134–136, 138–141, 145–151, 160, 161, 169, 170

SMPS Switching Mode Power Supplies. 81, 82

SPEC Spectral algorithm. 147, 148

SRAM System RAM. 104, 112, 115, 121, 128, 132–137

SWD Single Wire Debug. 134

TCI Threshold Crossing Intervals. 136, 147, 151, 153, 162

TCSC Threshold Crossing Sample Count. 136, 149, 151, 155, 156, 162

UBORA "excellence" in Swahili. 1, 2, 4–10, 169

UMDNS Universal Medical Device Nomenclature System. 23

VF Ventricular Fibrillation. 11–15, 61, 62, 103, 114, 115, 147, 153, 163, 169

VFf VF filter. 147, 148, 151

VT Ventricular Tachycardia. 11–14, 61, 62, 103, 114, 146, 147, 153, 163, 169