# Software development of Biosignal Pi

## An affordable opensource platform for monitoring ECG and respiration

JONATAN SNÄLL


KTH Technology
and Health

Master of Science Thesis in Medical Engineering
Stockholm 2014

# Software development of Biosignal Pi
## An affordable open source platform for monitoring ECG and respiration

# Utveckling av mjukvara till Biosignal Pi
## En open-source plattform för övervakning av EKG och andning

JONATAN SNÄLL

Master of Science Thesis in Medical Engineering
Advanced level (second cycle), 30 credits
Supervisor at KTH: Farhad Abtahi
Examiner: Mats Nilsson
School of Technology and Health
TRITA-STH. EX 2014:100

Royal Institute of Technology
KTH STH
SE-141 86 Flemingsberg, Sweden
http://www.kth.se/sth

Technology and Health

**Examensarbete inom medicinsk teknik (HL202X) *30hp***

**Software development of Biosignal Pi**

Jonatan Snäll

| Godkänt | Recensent | Handledare |
|---|---|---|
| 2014-09-30 | Prof. Kaj Lindecrantz | Farhad Abtahi |
| | Uppdragsgivare<br>Dr. Mats Nilsson | |

# *Sammanfattning*

För att hantera den ökande kostnaden för hälso- och sjukvård kommer en större del av övervakning samt vård att ske i patientens hem. Det kommer därför att vara önskvärt att utveckla mindre system som är lättare att hantera än de större traditionella apparaterna för att samla in vanliga biosignaler som exempelvis ett EKG.

Detta projekt är en fortsättning på ett tidigare projekt vars syfte var att framställa en ”sköld” som kan kopplas ihop med en Raspberry Pi via dess GPIO pinnar. Det föregående projektet var lyckat och en sköld innehållande en ADAS1000 som kan samla in bl.a. ett EKG samt andningen framställdes.

Syftet med detta projekt var att utveckla en applikation som kan köras på en Raspberry Pi och på så sätt visa den data som samlas in från skölden på en skärm. Det skulle även vara möjligt att spara insamlad data för senare användning. Projektet resulterade i en applikation som uppfyllde dessa krav.

**Nyckelord:** Raspberry Pi, ADAS1000, Biosignal Pi, EKG


Technology and Health

**Master of Science Thesis in Medical Engineering (HL202X) *30 credits***

**Software development of Biosignal Pi**

Jonatan Snäll

| Approved | Reviewer | Supervisor |
|---|---|---|
| 2014-09-30 | Prof. Kaj Lindecrantz | Farhad Abtahi |
| | Examiner<br>Dr. Mats Nilsson | |


# *Abstract*

In order to handle the increasing costs of healthcare more of the care and monitoring will take place in the patient's home. It is therefore desirable to develop smaller and portable systems that can record important biosignals such as the electrical activity of the heart in the form of an ECG.

This project is a continuation on a previous project that developed a shield that can be connected to the GPIO pins of a Raspberry Pi, a credit-card sized computer. The shield contains an ADAS1000, a low power and compact device that can record the electrical activity of the heart along with respiration.

The aim of this project was to develop an application that can run on the Raspberry Pi in order to display the captured data from the shield on a screen along with storing the data for further processing. The project was successful in the way that the requirements for the software have been fulfilled.


**Keywords:** Raspberry Pi, ADAS1000, Biosignal Pi, ECG

# *ACKNOWLEDGEMENT*

First and foremost I would like to thank my supervisor Farhad Abtahi for the support and guidance throughout the whole project.

I would also like to thank my fellow students Klara Lindskog, Niklas Roos, Lucas Dizon, Martin Johansson and Jing Wang for the constructive discussions during our meetings and for sharing ideas.

I would also like to thank my family for their support and believing in me.

Jonatan Snäll

Stockholm, August 2014

## Abbreviations

| | |
|---|---|
| ADAS | The ADAS 1000 chip |
| ECG | Electrocardiogram |
| EDF | European Data Format |
| GPIO | General Purpose Input/Output |
| GUI | Graphical User Interface |
| RLD | Right Leg Drive |
| RPI | Raspberry Pi |
| SoC | System On Chip |
| SPI | Serial Peripheral Interface |

# TABLE OF CONTENTS

# 1 INTRODUCTION

## 1.1 Background

An ageing population is an obvious phenomenon in most of the world. An increasing percentage of elderly in the population is mainly caused by a longer life expectancy and lowered fertility rates. Due to the increased percentage of the elderly, higher costs for the healthcare follows [1]. In order to tackle the increasing costs more of the care and monitoring will be performed in the caretaker's home, this is an increasing trend which will be more common in the future. [2] According to Stockholm country council (SLL) care in the patient's home is one of their development areas and that it is seen as positive by the patient to receive care outside of the hospital area. [3]

Homecare is usually involving monitoring of the patient and with the current technology a vast number of bio-signals can be measured, such as the hearts electrical activity in the form of an ECG. Due to increasing costs of healthcare and the fact that more of the patient monitoring will take place in the patients home, a low cost and portable system would be desirable. The ADAS1000 (ADAS) is a small and affordable device featuring five-lead ECG digital conversion, lead-off detection and respiration. The ADAS combined with the Raspberry Pi (RPI) which is a small, yet powerful computer, could be an affordable solution.

This project will be a continuation of a previous project that focused on the hardware part, images on the system built in the previous project can be found in appendix C. The conclusions and suggestions for future projects were that the ADAS is a suitable device and that it is possible to connect to the RPI but that the processing power of the RPI was low, thus making it hard to visualize the data in real-time. [4]

## 1.2 Objective

The purpose of this work was to create software running on the RPI that can collect and visualize the biosignals collected from the ADAS.

The goal of this project was:

- Create a way to communicate with existing hardware in order to collect the appropriate bio-signals.
- Visualize collected data in "real-time"
- Store visualized data for further processing later on
- Playback collected data
- Create a suitable graphical user interface (GUI) that can run on the RPI
- Optimum usage of resources considering limited resources of the RPI

### 1.3 Outline

The theoretical framework for the work is presented in chapter 2, followed by the methods used to get the final software in chapter 3. In chapter 4, the results of the work are summarized. Chapter 5 is a discussion where the work will be evaluated. Conclusions from the project are described in chapter 6. Finally, suggestions for future work are presented in chapter 7.

### 2.1 Electrophysiology of the heart

The heart is the primary part of the circulatory system in the human body. The circulatory system consists of three parts, namely, pulmonary circulation, coronary circulation and systemic circulation. The heart acts as a pump, regulating the flow of blood by contracting at different rates. Circulating blood through this system has three main purposes; *transportation* (e.g. oxygen to the cells, carbon dioxide to the lungs, nutrients), *regulation* (e.g. body temperature, blood pressure) and *protection* (e.g. by clotting, carrying antibodies). [5]

The contraction of the heart is triggered by electrical impulses originating from autorythmic fibers, *fibers which generate action potentials*, in the sinoatrial (SA) node. This node is located in the right atrial wall. Action potentials from the SA node travel through the atrial wall down to the atrioventricular (AV) node and both atriums contracts. When the action potential reaches the AV node it is slowed down in order to let blood flow out of the atriums. From the AV node the action potential travels through the wall between the ventricles in the bundle branches until it reaches the bottom part where it is conducted to the remaining part of the ventricle walls causing a contraction of both ventricles. An illustration of the heart and the conduction system can be seen in Figure 1.

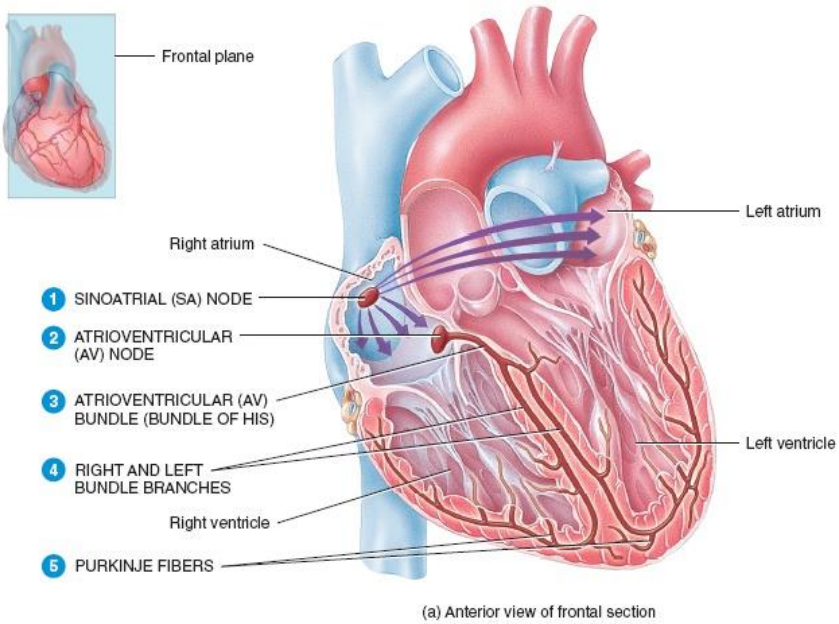



***Figure 1.*** *The heart and its conduction system. [5]*

When the action potential goes through the different parts of the heart, small electrical potentials can be measured through the surface of the body. These potentials can be recorded in the form of an electrocardiogram (ECG). Electrodes used to record the ECG are placed on the legs, arms and chest, an illustration of the electrode placement for a 5-lead ECG can be seen in Figure 2.

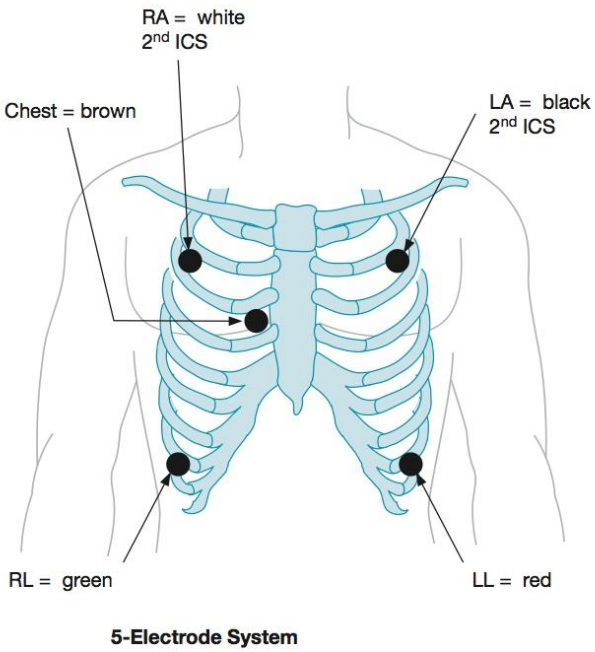



***Figure 2.*** *Electrode placement for a 5-lead ECG. [6]*

In a normal ECG three different waves called P-wave, QRS-complex and T-wave exist. An illustration of a typical ECG signal can be seen in Figure 3. The P-wave is seen first as a small deflection, the second part that can be seen is the QRS complex which starts as a downward deflection, the Q part, followed by a larger spike, the R part, and lastly a small downward deflection which is the S part. The third curve that can be observed is the T wave which is a small upward deflection.

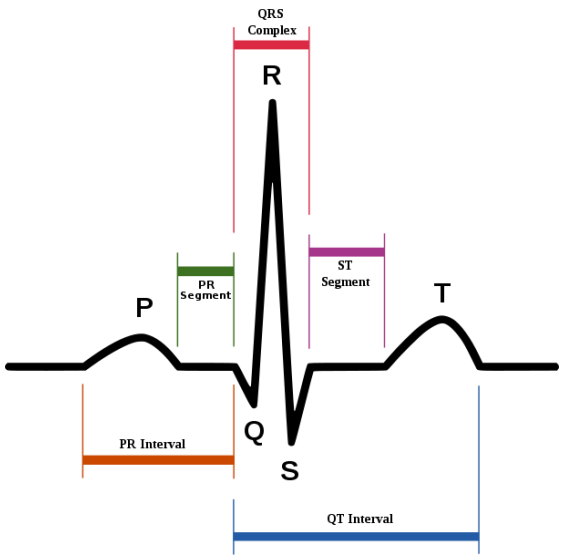



***Figure 3.*** *Illustration of a typical ECG signal. [7]*

The different waves in the ECG correspond to specific events in the contraction of the heart making it a useful diagnostic tool for physicians, e.g. an elevated ST segment can indicate a myocardial infarction. [5]

### 2.2 Raspberry Pi

The Raspberry Pi (RPI) is a small credit-card sized computer and is shown in Figure 4. The development of the RPI began in 2006 when the year-on-year decline in knowledge of the students applying to read computer science at the University of Cambridge. In order to inspire young people to experiment more with computers the Raspberry Pi was born.

The RPI has been released in two different versions called model A and model B. The RPI model B (shown in Figure 4) features the BCM2835 system on chip (SoC) from Broadcom, GPIO (general purpose input/output) pins, two USB 2.0 ports, Ethernet, HDMI, RCA video output and a 3.5mm audio output. A detailed comparison between the RPI model A and model B can be found in appendix A.

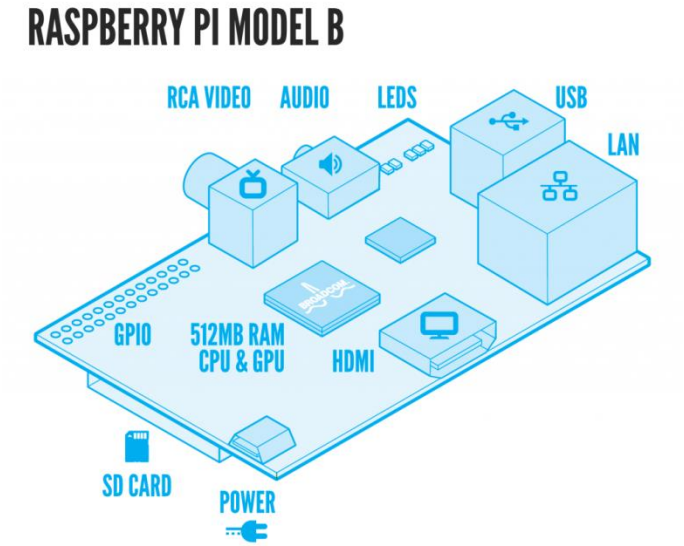



***Figure 4.*** *The Raspberry Pi Model B. [8]*

The BCM2835 SoC from Broadcom is used in the Raspberry. SoC is method used to place multiple components on a single chip. The BCM2835 chip used in model B features a 700MHz single-core ARM CPU, GPU, 512MB RAM along with other necessary components needed to run a computer on a single chip.

The GPIO pin layout on the RPI model B can be seen in Figure 5. The GPIO pins on the RPI make it a useful tool for prototyping.

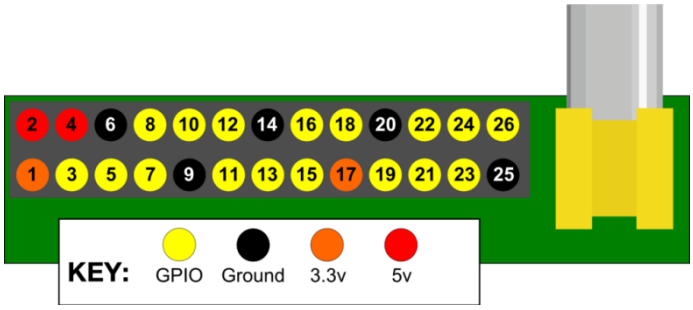



***Figure 5.*** *The RPI Model B GPIO pin layout. [8]*

### 2.3 GPIO and SPI

GPIO stands for general purpose input/output, the RPI has seventeen GPIO pins which can be seen as yellow colored in Figure 5. A GPIO pin can be programmed as either an input or an output by the user. [8]

The GPIO pins can also be configured for other functionalities such as I2C and SPI communication. This report will focus on SPI communication so the reader should consult the web for information regarding I2C communication. The GPIO pins have alternative functionalities such as SPI communication [9]. The pins intended to be used in SPI communication are:

- GPIO pin 8 – *CE0*
- GPIO pin 7 – *CE1*
- GPIO pin 10 – *MOSI*
- GPIO pin 9 – *MISO*
- GPIO pin 11 – *SCLK*

SPI communication is a protocol used to let a master device communicate with one or multiple slave devices. The SPI communication protocol works by letting the SPI master device control the communication between the master and slave devices. The chip-enable (CE) is normally high when no communication between the SPI slave and master occurs. In order to initiate data transfer the CE is set to low, this enables a specific SPI slave device. When the CE is set to low the clock signal (SCLK) is enabled at a specific frequency, the SPI master then sends data at the "master out – slave in" (MOSI) line and samples the signal on the "master in – slave out" (MISO) line. This means that in order to receive information from a SPI slave device, the same amount of bits have to be sent by the SPI master device. An illustration on how the SPI master communicates with one or multiple slave devices can be seen in Figure 6. [10]

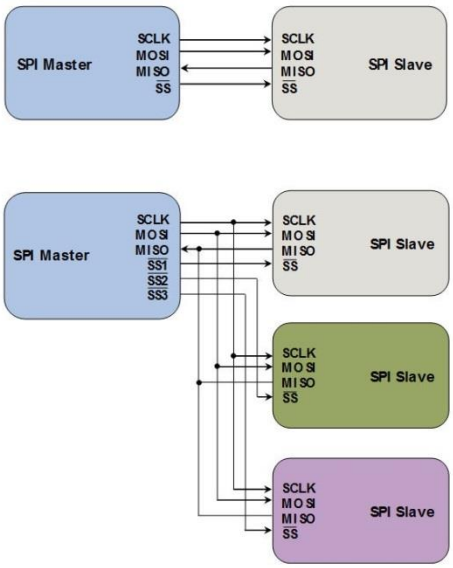



***Figure 6.*** *Illustration on how a SPI master device is connected to one or multiple slave devices. [10]*

SPI communication can be done in four different modes. The difference between the modes is when the MOSI is turned on or off. The mode is set by changing the clock polarity (CPOL) and the clock phase (CPHA). An illustration of the different SPI modes can be seen in Figure 7.

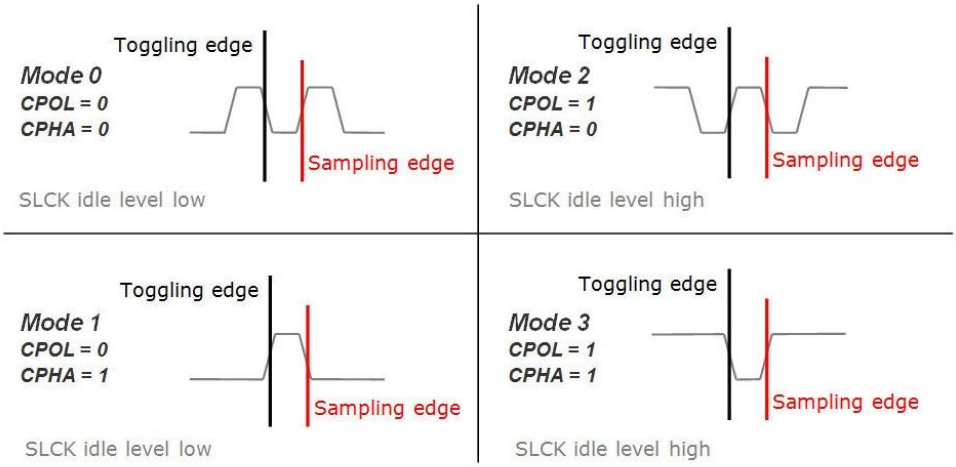



***Figure 7.*** *Illustration of the different SPI modes. [10]*

### 2.4 ADAS1000

The ADAS1000 (ADAS) is an analogue front-end system aimed to simplify the development of monitoring and diagnostic medical devices that need to acquire ECG signals. The ADAS device features a five lead ECG system along with the ability to measure thoracic impedance and detect pacing artefacts. ADAS can also be configured to operate in either low power mode or low noise mode depending on the requirements of the application for the device. The ADAS is available in nine different models with different features, such as different numbers of input channels, right leg drive, shield drive, pace detection, respiration and lead off detection. [11] [12]

The ADAS have 5 input ECG channels[1], three of them are the Lead I, Lead II and Lead III channels and the other two is the V1 and V2 channels. The V1 and V2 channels can be configured to work as either ECG input or perform other measurements. An example of lead compositions and how they are calculated can be seen in Table 1.

***Table 1.*** *Lead composiotions for the ADAS ECG inputs. [12]*

| Lead name | Composition |
|---|---|
| Lead I | RA-LA |
| Lead II | LL-RA |
| Lead III | LL-LA |
| Respiration | RA-LA |

Another important feature that the ADAS has is the ability to detect whether the leads are connected or not. There are two methods to this, the first method is by injecting a

[1] Some models of the ADAS have only 3 input channels.

DC current and the second method injects an AC current into the leads. The two methods can be used separately or work together.

In the DC lead off detection method a connected electrode produces a small voltage shift, if the electrode is disconnected a larger voltage shift will occur which is detected. The DC lead off detection has a delay between disconnection of electrode and detection, this delay is approximately:

$$Delay = Voltage \cdot \frac{Cable\ capacitance}{Programmed\ current}$$

The AC lead off detection works by measuring the change in voltage and compare it to a programmable upper and lower threshold. The frequency of the injected current is above 2 kHz and is therefore removed in the internal filters in order to not interfere with the ECG signal. The delay between disconnection of electrodes and detection for the AC method is below 10 ms.

Some of the ADAS models have the respiration feature which works by injecting a high frequency current through two electrodes and measuring the impedance variation caused by breathing. The frequency of the injected current can be programmed to be between 46.5 kHz to 64 kHz. The respiration can be measured through either the ECG leads or through the external pins dedicated to respiration measurement, respiration can only be measured through one path at a time.

### 2.5 Qt Framework

Qt is a cross-platform application framework using standard C++.In addition, multiple bindings of this framework to other programming languages exist. Development of Qt was started in the 1990's and has since then grown in popularity and is now widely used around the world, thus making it a good platform to build applications on. [13]

Qt is available under two different licenses; a commercial one maintained by Digia and as open-source available under the GNU GPL (GNU General Public License) 3 and GNU LGPL (GNU Lesser General Public License) version 2.1 [14]. The GNU GPL and GNU LGPL allow Qt to be used in both “free software” and commercial solutions. [15] [16]

A basic Qt GUI application consists of a *mainwindow* and a *QApplication* object. The *QApplication* handles the application-wide resources along with the event-loop of the application. After all components of the application are initialized the applications event-loop should be entered, the event loop is a state where the application can idle and wait for events such as a pressed button or a timer event. The *mainwindow* contains all the components of the application, also called widgets (window gadgets). A widget is a visual element e.g. button or label. The widget may also contain other widgets for further flexibility. An example of a nested widget is a widget containing a knob and a label that displays the current value of the knob. [13] Figure 8 illustrates how a widget can contain multiple widgets.

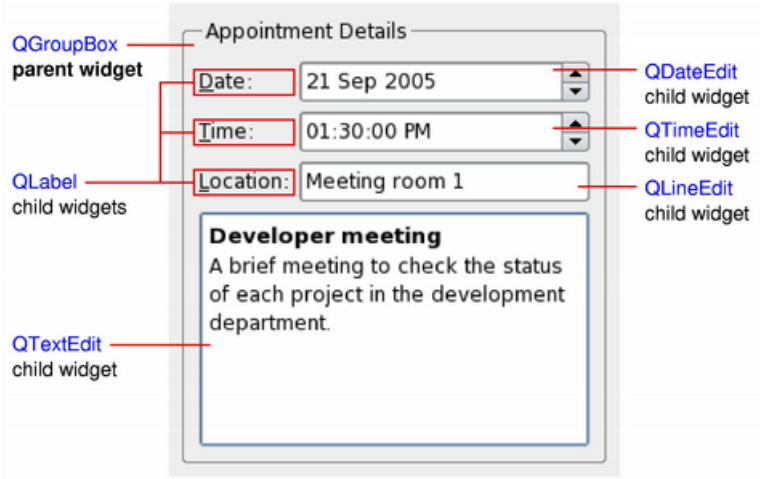



***Figure 8.*** *Example of a Qt widget containing other widgets. [17]*

### 2.5.1 Signals and slots

In a normal application objects want to be able to communicate with each other, an example could be when a push button in the application is pressed the user usually wants to perform some action. The signal & slot mechanism makes this an easy task.

Classes or subclasses that inherit `QObject` can use signals & slots by adding the `Q_OBJECT` macro to the header file. A signal and slot is connected by the following syntax:

```
QObject::connect(sender, signal, receiver, slot)
```

Widgets usually have pre-defined signals and slots but it is common to subclass widgets in order to add custom signals or slots. Subclassed widgets inherit the signals & slots from the base widget.

Signals & slots are loosely coupled which means that an emitted signal does not have to be connected to a slot and a slot does not have to be connected to a signal. As a result it is easy to create independent components.

A signal can be seen as a function call and a slot can be seen as a normal function. A signal can be connected to multiple slots and multiple signals can be connected to a single slot. Signals do not even have to be connected to a slot, a signal can be connected to another signal. An illustration on how signals and slots can work within an application can be seen in Figure 9. [18]

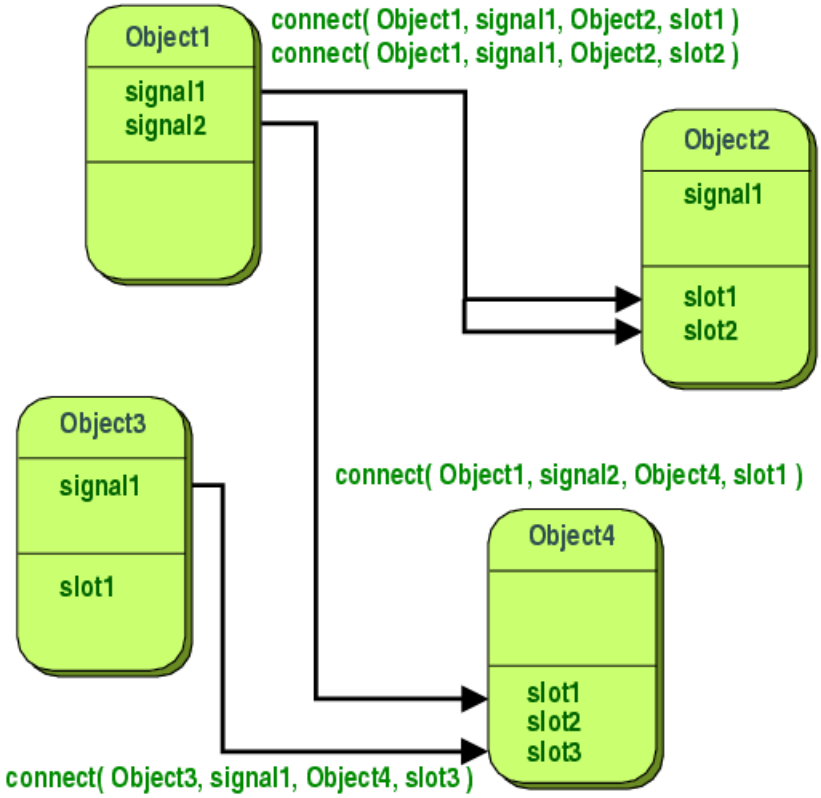



***Figure 9.*** *Example of how objects can communicate with each other through the signal-slot mechanism. [18]*

### 2.5.2 Multithreading in Qt

A thread and a process are two different things, a process can be an application running on the computer, for example a web browser. Multiple processes can be running at the same time, which is called multitasking. Sometimes it is also needed to achieve the same thing within a single process, for example when a large file needs to be saved without locking the GUI of an application. This is also called concurrency – doing multiple tasks simultaneously. [19]

Threads can be used to achieve concurrency within a process, like in the previous example the GUI can have one thread and a worker thread can be created to save the file. The difference between a process and a thread is that different processes do not share the same memory with each other, threads within a process do however share the same memory.

Stating that processes or threads are running at the same time is not always true. True concurrency is only possible in processors with multiple cores. If multiple processes or threads are running on the same core the fact that they are running simultaneously is merely an illusion, the processor is quickly interrupting each process or thread and moves to the next process or thread. The jump from one thread or process to another can occur at any time.

As previously mentioned, threads within a process share the same memory, this can be both good and bad. The fact that they share the same memory makes it fast to jump between the threads and to access shared data. But it is also important that the integrity

of the data is preserved when a reference to the same object exists in two or more threads. An illustration on how threads and processes are stored in the memory can be seen in Figure 10.

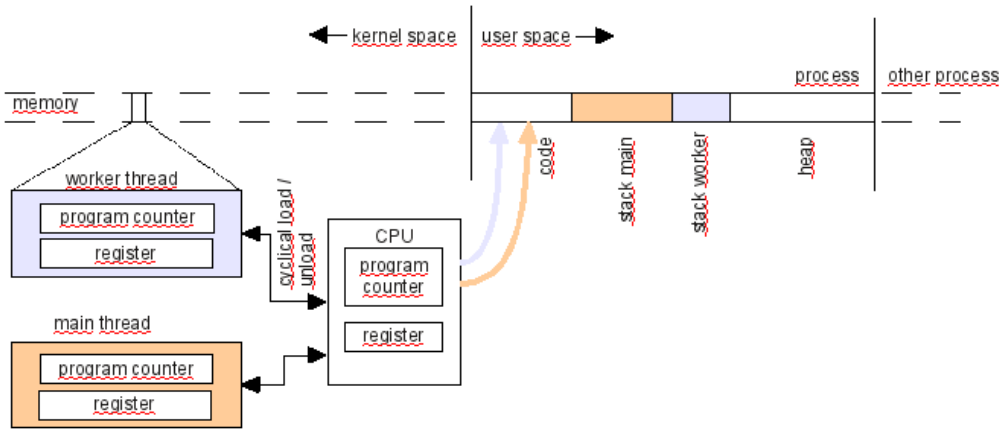



***Figure 10.*** *Illustration on how threads and processes are stored in the memory. [19]*

There are multiple methods that can be used to preserve the integrity of data available in multiple threads, some of them are mentioned below. [20]

- Using a mutex to lock and unlock data
- Using semaphores to protect multiple resources
- Using the Qt event system

In the case where a GUI thread and a sampling thread is used an easy way to exchange data is by using the built in Qt event system by utilizing the signal & slot mechanism. It is important to remember that a queued connection has to be established because that will place the task in the receivers thread event loop and let it finish its current task. [21] [20]

### 2.6 Cross compiling

A cross compiler is a compiler that is able to produce executable code for a different platform than the one the compiler is running on. For instance, compiling an application on the RPI would take much longer than cross compiling the same application on a PC.

A complete guide on how to build Qt 5 in order to cross compile projects for the RPI can be found in appendix B.

### 2.7 Iterative development

One of the goals with iterative development is to break down a bigger project into smaller pieces that are easier to handle. Iterative development can also be combined with other software development methodologies such as the waterfall development process. The waterfall development process also breaks down the project into smaller pieces that follows each other. [22]

The iterative development process is a combination of the Rational Unified Process (RUP) and eXtreme Programming (XP). This means that the resulting software is tailored for the target and at the same time it is tested and easily maintained by following the RUP and XP development methodologies. [23]

Figure 11 illustrates a small piece of the final product which consists of multiple similar pieces. Each piece is first planned in order to identify requirements and then a first implementation is performed. The piece of software is then tested and evaluated in order to see if the initial requirements are fulfilled. If they are not, the process is repeated until the requirements are fulfilled. [22] [23]

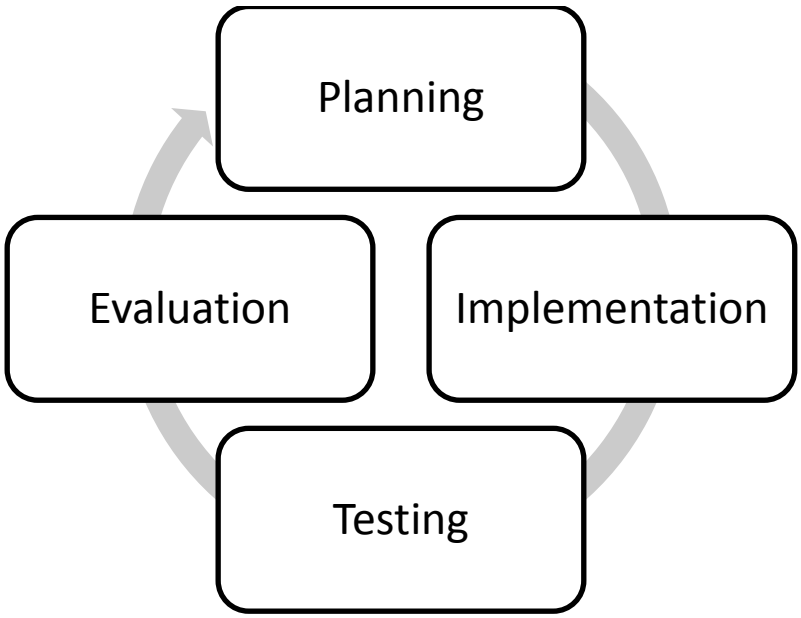



***Figure 11.*** *The iterative process used in this project.*

### 2.8 European Data Format

The European Data Format (EDF) is a file format that can store bio-signals from multiple channels in a single file. It has existed since 1992 and is a common file format for researches that work with EEG or PSG (polysomnography) recordings.

The EDF file format contains a variable header and data record, the header contains 256 bytes for recording information and an additional 256 bytes for each signal, the structure of the header can be seen in Table 2. The rest of the file contains a number of data records, the duration of a record and the number of samples in each record for each signal is specified in the header corresponding to the specific signal. This means that the signals in the file do not need to have the same sampling frequency. An ECG signal could for example be combined with an ICG, EEG or body temperature signal with a different sampling frequency. The structure of a data record can be seen in Table 3. [24]

***Table 2.*** *Data header of an EDF file.*

| | |
|---|---|
| Version | 8 ascii |
| Patient identification | 80 ascii |
| Recording identification | 80 ascii |
| Startdate of recording | 8 ascii |
| Starttime of recording | 8 ascii |
| Number of bytes in header record | 8 ascii |
| Reserved | 44 ascii |
| Number of data records | 8 ascii |
| Duration of data record (s) | 8 ascii |
| Number of signals (ns) | 4 ascii |
| Label | ns*16 ascii |
| Transducer type | ns*80 ascii |
| Physical dimension | ns*8 ascii |
| Physical minimum | ns*8 ascii |
| Physical maximum | ns*8 ascii |
| Digital minimum | ns*8 ascii |
| Digital maximum | ns*8 ascii |
| Prefiltering | ns*80 ascii |
| Nr. of samples in each data record | ns*8 ascii |
| Reserved | ns*32 ascii |

***Table 3.*** *Data record of an EDF file.*

| | |
|---|---|
| First signal | Nr of samples[1]*integer |
| Second signal | Nr of samples[2]*integer |
| … | … |
| Last signal | Nr of samples[ns]*integer |

### 3.1 Requirements and technical goals

During the software requirement analysis it is determined *what* the software has to do, not *how* the software performs a specific task. It is important to identify requirements of the software early on in the project, because adding more requirements later on will increase both time required and cost compared to if the requirement was stated at the start. A list of requirements for the software can be seen in the list below. [25]

- Collect data from the ADAS
- Store collected data
- Visualize sampled data in real time and from file
- Be able to collect data at different rates
- Lead-off detection

In addition to the features listed above the produced software also had to be efficient, testable, readable and modular. The software should also be flexible, reusable and easy to maintain. [25]

The software had to be efficient enough so it could both collect and visualize the data at the same time. Efficiency can be achieved by writing time critical parts of the software in a lower level language such as C or assembler, or even better, use existing and already tested libraries.

The ability to test the software is also an important quality. A way to perform such tests is to use the built in test tone generator in the ADAS and see if the data conversion and data visualization is correct. The ADAS can also tell if frames are missed or being read too slowly, thus efficiency of the program can also be tested. Lastly, the ADAS can also perform tests on the device itself in order to check if it is functioning correctly.

The code written for the software should be readable by writing code that in itself tells the programmer how the application works. Unclear parts of the code should not exist but in case there is, comments should be used to avoid errors. Code is made modular by following the object oriented programming (OOP) paradigm. This way the software could be divided into small modules that can be changed and tested on their own. This means that a module can also be easily changed to another one, for example; the communication with the ADAS is a module, if another device is used it is easy to change only that part without changing the rest of the software.

The software is also flexible and reusable by using the cross-platform Qt framework. This means that the software could easily be recompiled and deployed in another

environment[2]. In order to deploy the software of this project in another environment it is important to take the SPI-communication library into account because it is platform specific for the RPI.

### 3.2 GUI development

A central part of the application is the GUI which is what the user will see and interact with. Therefore it was important to make the interface easy to use but also easy to modify in order to add more features to future versions of the application.

The structure of the GUI was changed multiple times during the development cycle as a part of the iterative process. The final version of the GUI contains a main window which will hold all the graphical components of the application.

In order to create a GUI that could easily be modified a *StackedWidget* was used to hold the different views of the application. A *StackedWidget* works by placing widgets on top of each other and giving them specific ID's. The user can then call the *SetCurrentIndex* method with an ID as an argument in order to view a specific widget.

### 3.3 Interfacing with the ADAS

The ADAS is controlled by SPI, this will enable the program to configure registers and read data frames. SPI communication with the ADAS can be implemented in several ways but the BCM2835 C-library was used in this project for the sake of simplicity.

The BCM2835 C-library provides easy access to the low level peripherals of the RPI, for example the GPIO pins. The library also includes functions for SPI and I2C communication. [26]

Before setting up the SPI the ADAS had to be reset, this was done by setting the $\overline{RESET}$ line to low and wait for the device to reset, the $\overline{RESET}$ line was then set to high.

In order to set up the SPI port the following parameters need to be set:

- Bit order
- Data mode
- Clock divider

The bit order of the ADAS is most significant bit (MSB) first. The different data modes were shown in Figure 7 and the ADAS requires that mode 0 is used, which is CPOL and CPHA is 0. The clock divider is used to set SCLK, the required speed can be calculated by using the equation below.

$$SCLK(min) = frame\_rate \cdot words\_per\_frame \cdot bits\_per\_word$$

The SCLK was set to 488 kHz which is enough for all configurations used in this project.

[2] The fact that the software can be deployed in another environment by just recompiling is not true in general. There may be platform specific libraries used in software that must be taken into account before deploying the application on another platform.

Two different operation modes were implemented, the first was the ECG mode and the other one used the built in test tone feature for testing purposes. In order start framing a number of write commands has to be sent to the ADAS. The write commands for the ECG and test tone mode can be seen in Table 4 and Table 5. The last write command addressing FRAMES tells the ADAS to start framing. A write command consists of 32 bits, the first bit is set to 1 for write or 0 for read, the next 7 bits are the control register address and the remaining 24 bits are the data. An example on what a normal write command to the ADAS looks like is shown in Table 6. It is also possible to read control registers. In order to read a register the first bit has to be set to 0 followed by the control register address and 24 unused bits. The ADAS will then output the data of the register in the next frame. Reading a register will cause the ECG framing to stop so a new framing command has to be sent after reading. A visualisation on how the SPI communication is implemented can be seen in Figure 12.

***Table 4.*** *Register commands to configure ADAS to output ECG frames.*

| Register addressed | Write command |
|---|---|
| CMREFCTL | 0x85E0000A |
| FILTCTL | 0x8B000000 |
| FRMCTL | 0x8A1FCE00 |
| LOFFCTL | 0x82000015 |
| ECGCTL | 0x81F800AE |
| RESPCTL | 0x83002099 |
| FRAMES | 0x40000000 |

***Table 5.*** *Register commands to configure ADAS to output ECG frames using the built in test tone.*

| Register addressed | Write command |
|---|---|
| CMREFCTL | 0x85E0000B |
| TESTTONE | 0x88F8001D |
| FILTCTL | 0x8B00001D |
| FRMCTL | 0x8A1FCE10 |
| ECGCTL | 0x81F800AE |
| FRAMES | 0x40000000 |

***Table 6.*** *Example of a write command to the ADAS.*

| 32 bit write command | R/W | Register address | Data |
|---|---|---|---|
| 0x85E0000A | 1 | 0000101 | 11100000000000000001010 |

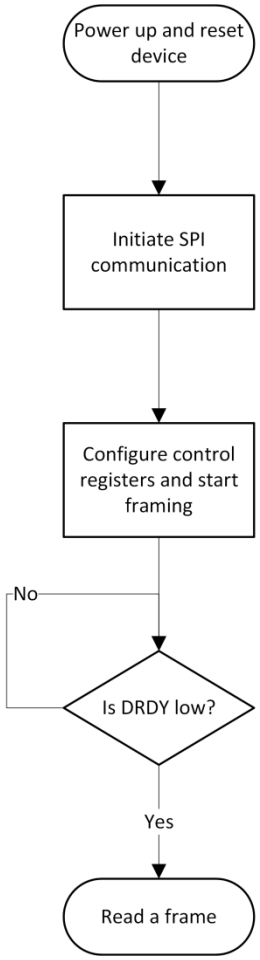



***Figure 12.*** *Flowchart of how the SPI communication is implemented.*

### 3.4 Data capture

The sampling of data is performed in its own thread in order to prevent locking of the GUI. Before the sampling process is started the settings are read and the operating mode is selected. The sampling thread then tells the class communicating with the ADAS which mode is selected and the correct control registers are set.

When the settings are applied, the sampling loop is entered, which will collect samples in a specific time interval, determined by the sampling rate. When a sample is collected it is stored in the memory, the samples are also decimated in order to make the plotting faster. The loop will stop collecting samples when it receives a signal from the GUI thread that tells it to stop.

Samples collected are first stored in a text file but they can also be saved in the EDF file format. More information on how the samples are saved can be found in section 3.7. A visualization of the whole sampling process can be seen in Figure 13.

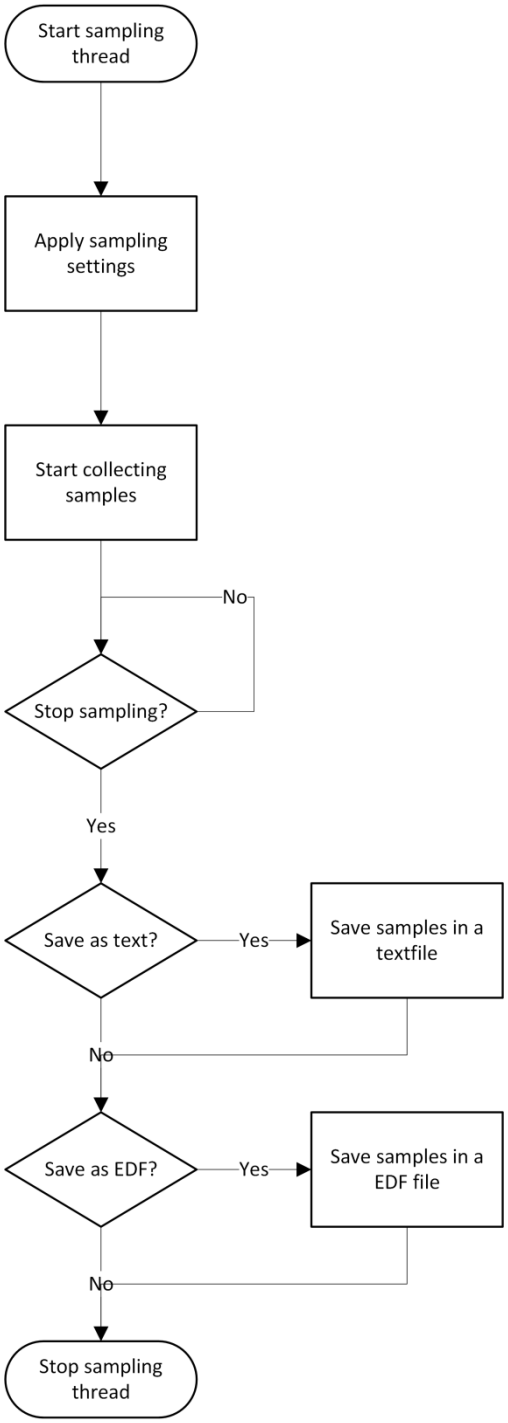



***Figure 13.*** *Illustration of the sampling process.*

### 3.5 Communication between threads

Communication between the sampling thread and the GUI thread is an important part of the application. The important part was that it had to be thread safe, this was achieved by using the signal & slot feature that Qt offers. An illustration on how the GUI thread communicates with the sampling thread during collection of data can be seen in Figure 14.

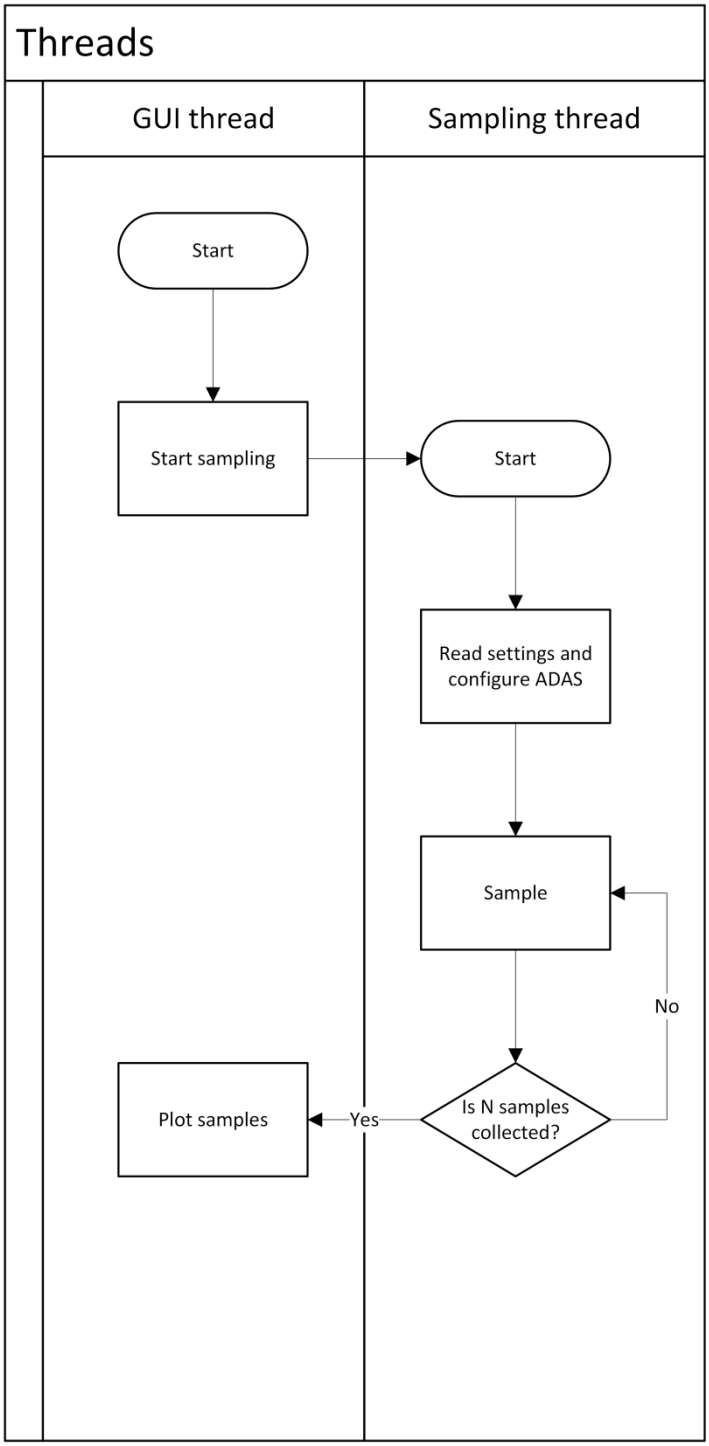



***Figure 14.*** *Flowchart on how the GUI thread communicates with the sampling thread during data collection.*

In order to reduce the number of signals from the sampling thread a number of samples were saved in an array before they were sent to the GUI thread with a signal. The format of the emitted signal was `QVector<QVector<QPointF>>`.

The signal & slot mechanism was made thread safe by using a queued connection which means that the signal is placed in the GUI threads event loop and letting the GUI finish its current task before the slot is invoked. Because a queued connection was used a new type had to be registered. How to register a new type can be seen below.

```
qRegisterMetaType< QVector<QVector<QPointF> > >("QVector<QVector<QPointF> >")
```

### 3.6 Data plotting

An important part of the application was to visualize the recorded biosignals from the ADAS in "real-time", which means that performance is an important aspect. Another important part was that the plotting should also be easy to implement and maintain.

In order to evaluate the performance of the plotting method a text file with pre-recorded samples was provided. From the pre-recorded samples 5000 values were loaded and in order to further simulate how the plotting would work in real-time, a decimation of the samples was performed as well.

Initially the *QwtPlot* class was used which have a method called *SetRawSamples* that stores samples as a vector and the next time the *Replot* method is called the plot canvas will be redrawn with the samples stored. This method proved to be too resource costly and the system could only handle one plot on a small screen (due to a smaller plot area) using all the systems resources. A flowchart of the initial plotting method described above can be seen in Figure 15.

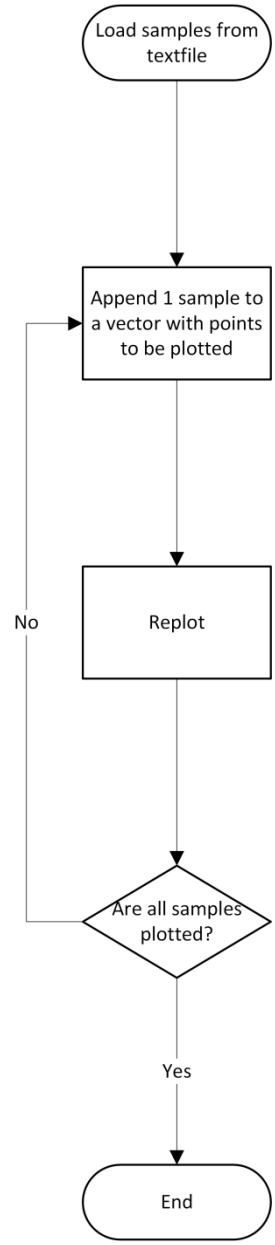



***Figure 15.*** *Initial plotting method*

Instead of using the plotting method mentioned above a new plotting method had to be implemented, capable of plotting the collected samples fast while using a small amount of system resources. The solution was to write a custom plotting class that re-implemented the *QwtPlot* class. The plot starts with an empty canvas and a *DirectPainter* is used to draw the curve segments. This means that the plot canvas only have to be redrawn when the plot has to be cleared which occurs when the plot curve reaches the end of the canvas, how often the plot has to be cleared depends on the scale on the x-axis which was set to 10 s. A flowchart of the improved plotting method can be seen in Figure 16.

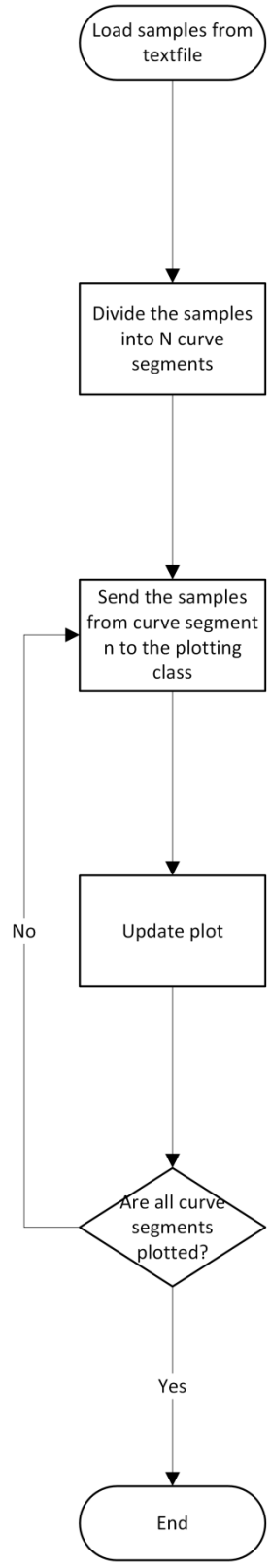



***Figure 16.*** *Final plotting method.*

### 3.7 Data storage

During runtime the samples are stored in the memory of the RPI as a vector and when the user chooses to end the recording the samples are stored in two text files (a header and data file) and optionally an EDF+ file.

The header of the text file contains useful information about the recording such as filename, samples in file, duration of the recording, sample rate, source signal and patient information. The data file holds one sample on each row separated by a tab.

The EDF+ file uses the *edflib, a* C-library, to create a valid EDF file that can be viewed in any EDF compatible software such as the EDF browser.

A Qt-based application running on the RPI has been produced. It can collect and view an ECG along with respiration in real-time and also view an already recorded ECG. Below follows a more thorough description of the software and a guide on how to use the software can be found in appendix D.

### 4.1 Overview of the application

As described in section 3.1 the GUI consists of a stacked widget, a snapshot button, a quit button and navigation bar. The stacked widget contains a widget for the starting screen, a widget for capturing an ECG, a widget for viewing an ECG and finally a settings widget.

The starting screen of the application can be seen in Figure 17 which contains short instructions on how to use the software and an image showing the recommended electrode placement.

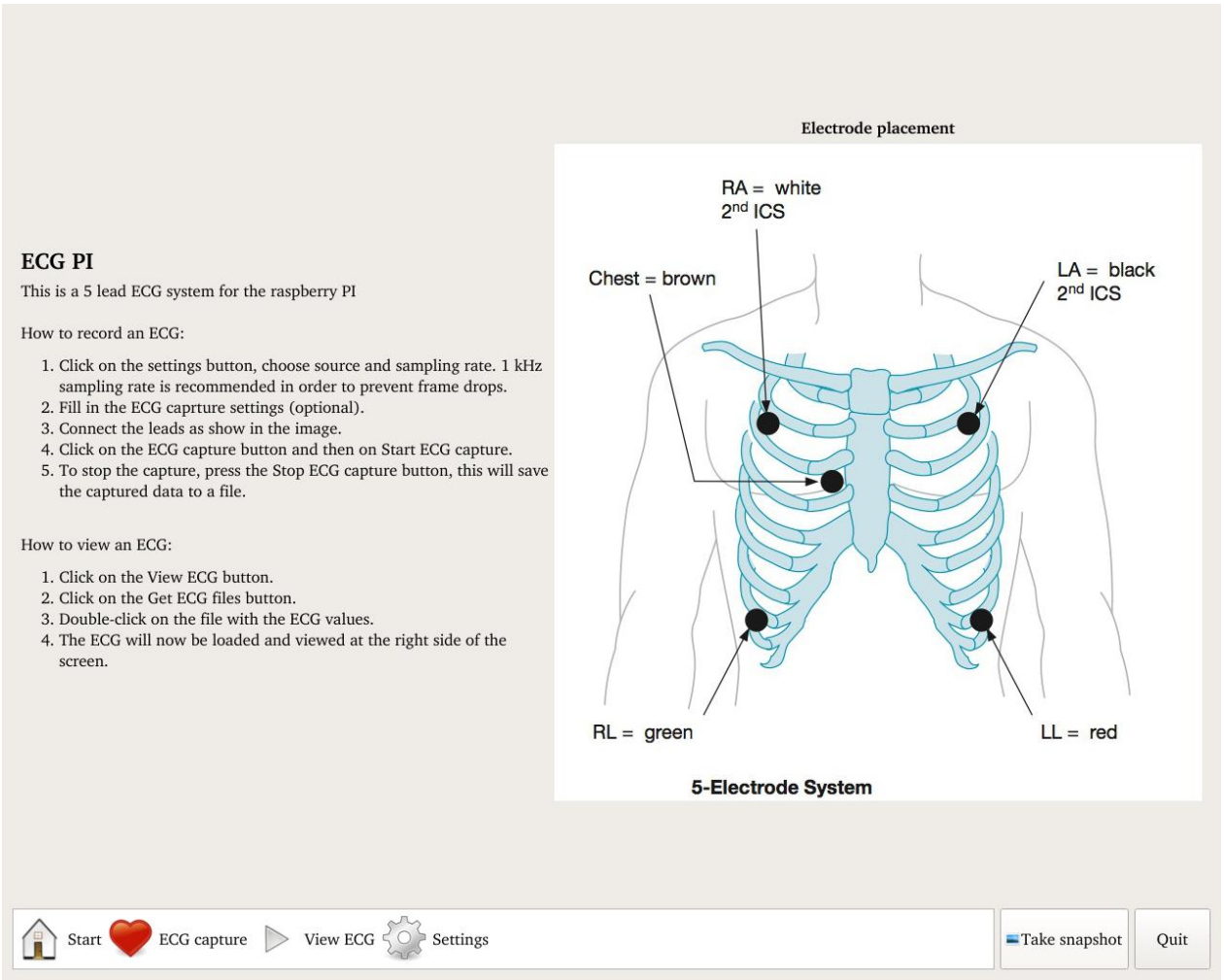



***Figure 17.*** *The starting screen on the BiosignalPI application.*

By using the navigation bar below the stacked widget, the user can navigate between the different widgets, the *ECG capture* and *ECG view* will be explained in detail in section 4.2 and 4.3 respectively.

On the settings screen of the application the user can choose signal source and sample rate along with a number of optional parameters which can be seen in Figure 18.

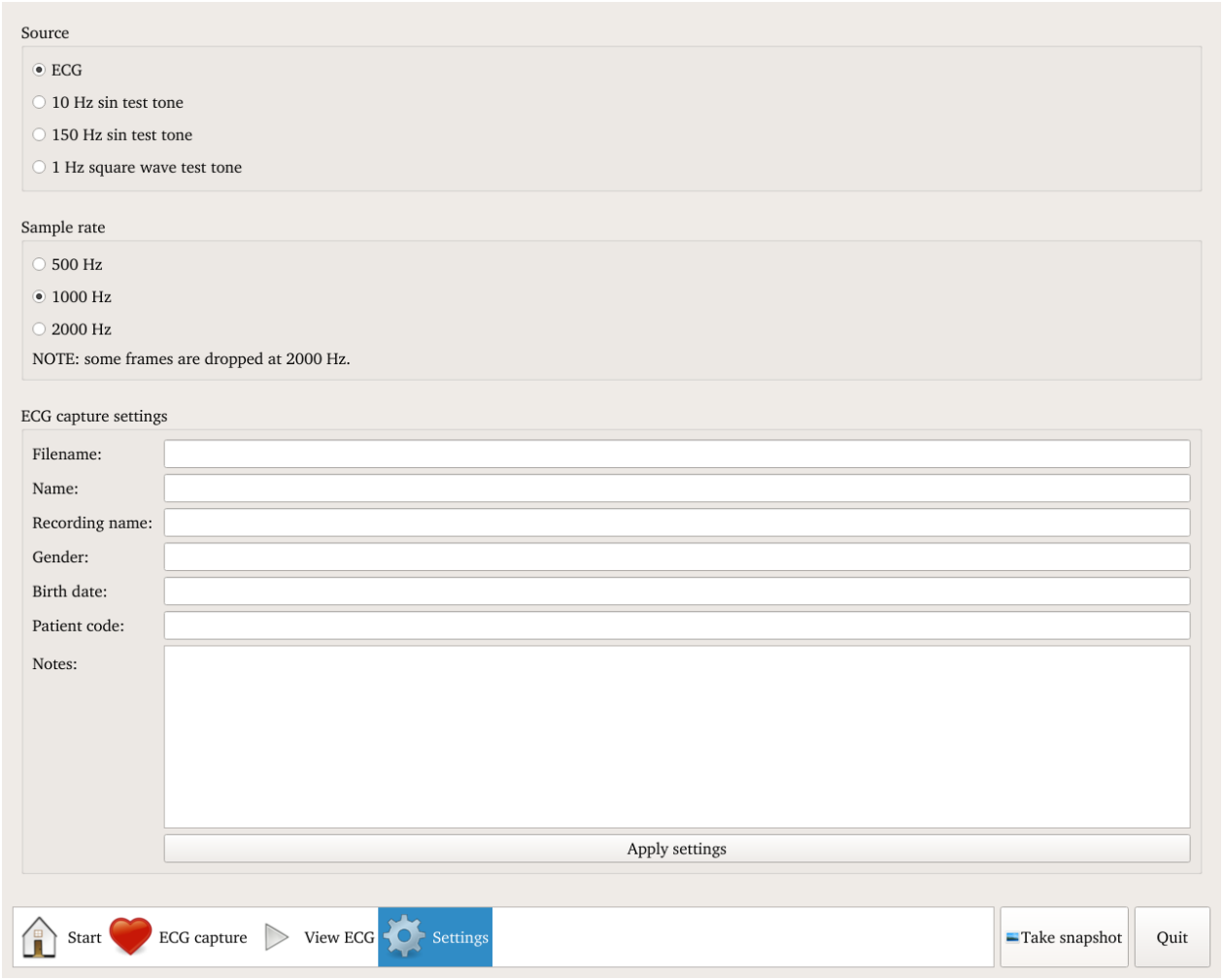



***Figure 18.*** *Settings screen of the application.*

### 4.2 The ECG capture widget

The ECG capture widget which can be seen in Figure 19 consists of a button row which contains buttons to start and stop a recording, a status bar and four plots that displays the collected data in real-time.

The status bar is used to give the user information about what the application is doing, during sampling the status bar notifies the user if the leads is disconnected by using the lead off detection feature on the ADAS.

An example on what the ECG capture widget looks like during sampling can be seen in Figure 20.

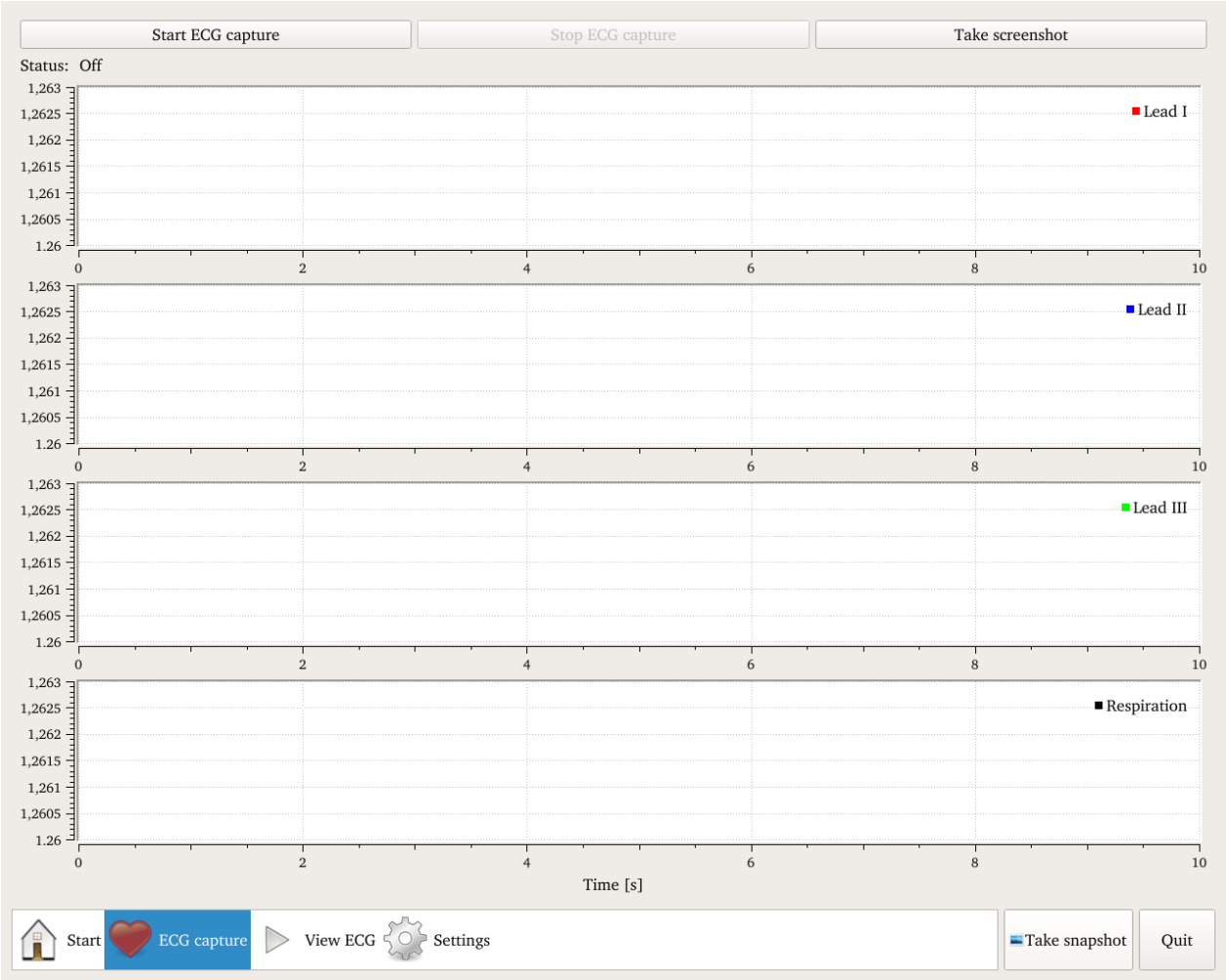



***Figure 19.*** *ECG capture screen of the application.*

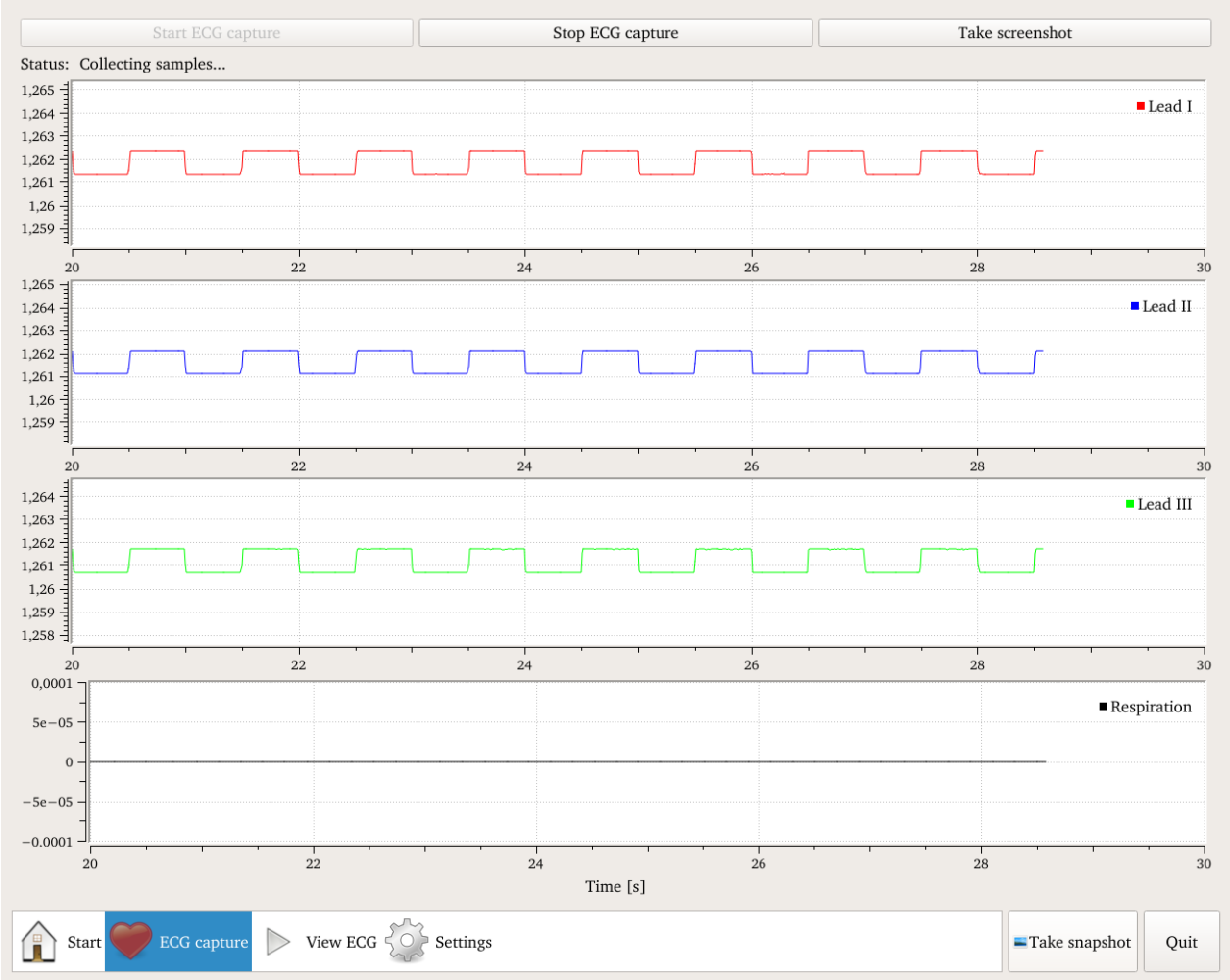



***Figure 20.*** *ECG capture running with the 1 Hz square wave option.*

### 4.3 The ECG view widget

The ECG view widget is divided into two sections, a left section containing buttons and information and a right section containing the plots and navigation controls. Snapshots of the ECG view widget can be seen in Figure 21 and Figure 22.

The user can click on the "Get ECG files" button to show a list of supported files in the box below. In order to view an ECG the user has to double click on the desired data file. When a file is selected the application will show the ECG header in the box below the window size control. An example of what a header contains can be seen in Figure 22. When the header is loaded the application will start to read the data file in a separate thread in order to prevent locking of the GUI. When the data is loaded into the memory it will be plotted on the right hand side of the screen which can be seen in Figure 22. Finally the user can set the window size which controls the scale on the x-axis in the plots.

In order to navigate through the data the user can use the slider below the plots, the scroll wheel on the mouse or by clicking on the arrows on the right and left side of the play and pause buttons. The user also has the option to playback the recording in real-time by pressing the play button, the speed is depending on what window size that is selected.

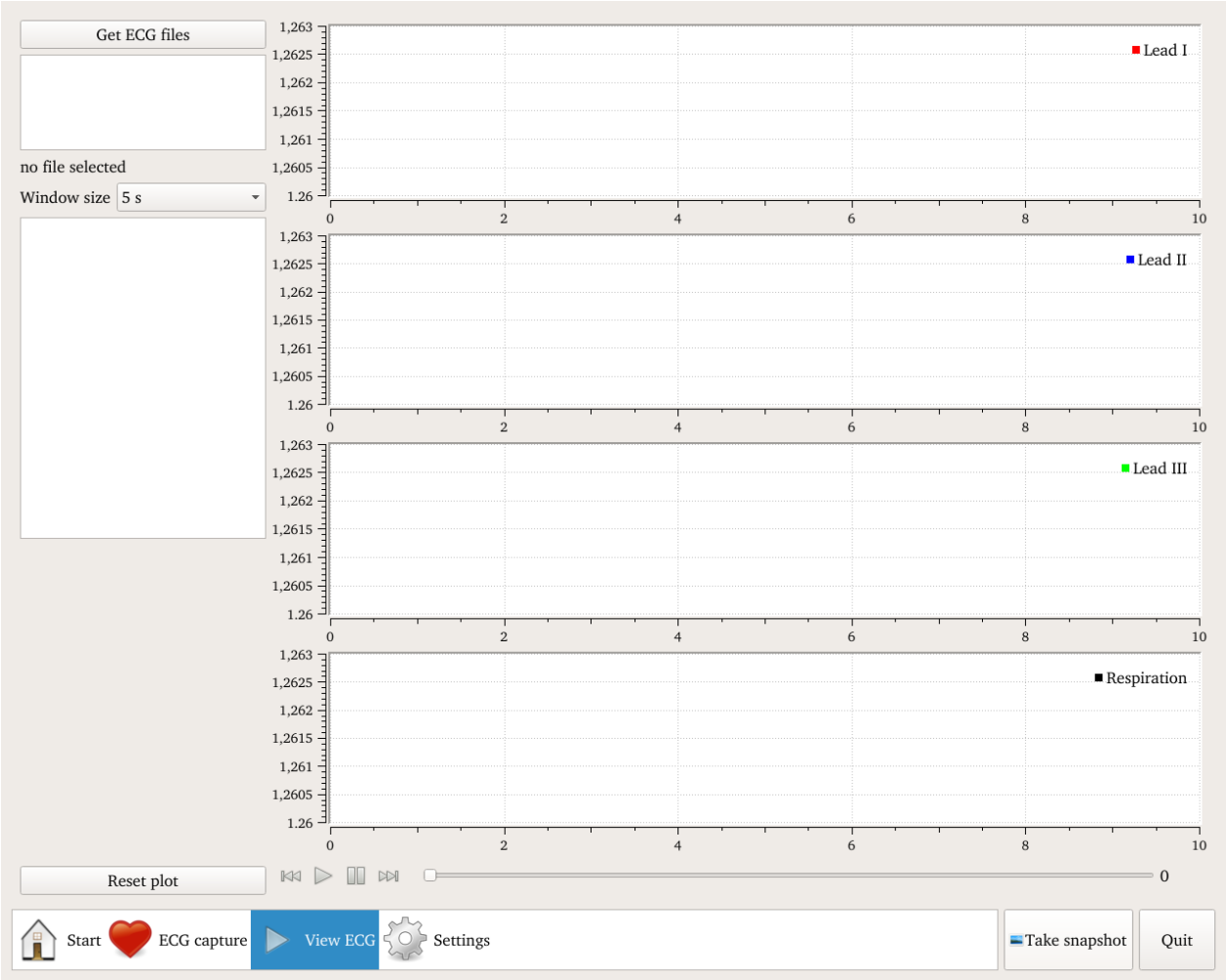



***Figure 21.*** *The view ECG screen of the application.*

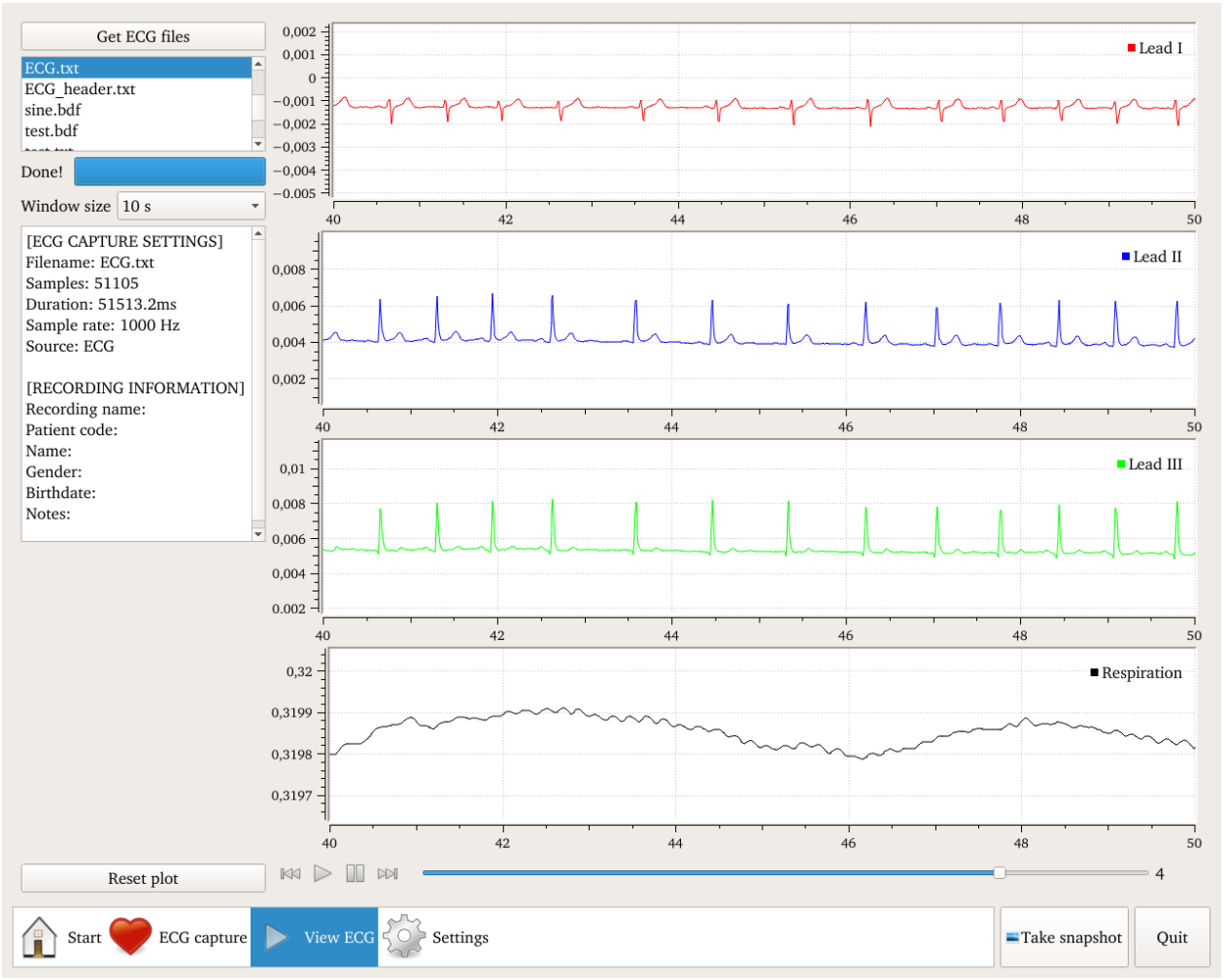



***Figure 22.*** *View of a recorded ECG.*

### 5.1 Discussion of the produced application

As described in section 3.3 (interfacing with the ADAS) and according to the requirements from section 3.1 a way to interface with the existing hardware have been developed. The communication with the hardware uses the BCM2835 C-library which is the only platform specific library used in this project. It is therefore easy to rewrite the class communicating with the hardware if the user wants to change platform due to the modularity of the software which was one of the requirements listed in section 3.1.

Data can also be seen in real-time due to the reimplementation of the *QwtPlot* class utilizing the *QwtPlotDirectPainter* because replotting every time samples were received in the GUI thread would be too expensive.

Collected data can be stored in two different formats through the software, either as a plain text-file or as an EDF+ file. Providing the option to save the file in multiple formats makes the system flexible because tools already exists that can handle EDF+ files for instance. The text-file can easily be imported to MATLAB, a powerful and widely used tool, for further processing.

The fact that there already existed a C-library for storage in the EDF+ file format made it easy to implement. Tests were made and it was visually confirmed that the data was stored correctly by comparing the same data plotted in MATLAB and in the edfbrowser, software that can visualize EDF+ files.

A way to playback collected data was also implemented, the functionality to do this requires the text file header and the data. When the user selects a file by double clicking on it the filename is sent to a class running in a separate thread. The loading of the data file is running in a separate thread because having it in the same thread as the GUI would lock both the GUI and the cursor until the file was completely loaded. A progress bar is also visible to indicate that the file is loading.

The interface of the software is visible in Figure 17, the idea behind the layout was ease of use but also to utilize the screen area in the best way possible. The navigation bar on the bottom of the screen contains both icons and text making it clearer an easier to use. It is also easy to add new features to the software by implementing them as a new widget and add them to the *QStackedWidget* and a navigation bar item to the *QListWidget* as a *QListWidgetItem*.

The biggest challenge of the project was performance but thanks to the header that the ADAS sends each frame it was easy to see if frames were missed or not by reading bit 28 and 29 which indicated how many frames were missed. The largest factor was how many samples that were plotted. In order to solve that problem a decimation of order 10 was implemented, before decimation the samples went through a moving average mean filter to prevent aliasing.

### 5.2 Ethical aspects of using the application

The produced software is not certified as a medical device and should therefore not be used for diagnosis or treatment purposes. Before using the software the user must know that the software is provided as is and that it is intended for scientific and educational purposes.

# 6 CONCLUSIONS

An application that can view an ECG in “real-time” and view an already recorded ECG while being limited by the processing power of the RPI has been developed and a description on how the application is working were shown in chapter 4.

Qt is a framework suitable for development of software where performance is of importance. It has the power of the C++ programming language along with features such as the signal & slot mechanism making multithreading easier to implement.

The RPI combined with the Biosignal Pi shield is small in size but still has the power to show an ECG in real-time which makes good start for future projects.

### 7.1 Additional features

Additional features that could be added in future versions of the software could be HRV-calculations, Bluetooth communication and communication with a web server. Qt has modules for Bluetooth communication for Linux devices with Bluez 4.x installed. If Bluetooth support could be implemented, data could be sent to a smartphone or another computer easily. Qt also supports websockets which could be used to send data to a server.

### 7.2 Future work

It would be interesting to see if the software could run faster on a multicore device where the GUI thread and sampling thread run on different cores, this way true concurrency can be achieved. A multicore device similar to the RPI is the UDOO board which features a more powerful quad-core CPU along with an additional CPU dedicated to GPIO communication. [27]

# 8 EXPERIENCE GAINED

This project has been interesting in many ways and a lot of new experiences have been gained. Before this I had never used a Linux operating system before so setting up the cross compiling environment and installing Linux was something new to me.

Working with Qt and C++ was also something that I had never done before but it ended up working fine. The knowledge was mostly gained from studying sample code on the web and also by asking other developers when I got stuck or was wondering if there was a “best practice” to solve a specific task.

I think that the experience gained in this project will be useful in future projects as well, especially when it comes to software development. I’ve learned how hard it can be to determine how long it will take to implement a seemingly simple task.

[1] P. Zweifel, S. Felder and M. Meiers, "Ageing of population and health care expenditure: a red herring?".

[2] G. B. Devey and W. A. Herman, "FUTURE TRENDS IN MEDICAL DEVICE TECHNOLOGIES: A Ten-Year Forecast," 2011.

[3] Stockholms Läns Landsting, "Framtidsplanen andra steget – konkretisering av det fortsatta arbetet".

[4] F. Abtahi, B. Aslamy, I. Boujabir, F. Seoane and K. Lindecrantz, "An Affordable ECG and Respiration Monitoring System based on Raspberry PI and ADAS1000 : First Step towards Homecare Applications," in *16th Nordic-Baltic Conference on Biomedical Engineering & Medical Physics 2014*, Stockholm, 2014.

[5] G. J. Tortora and B. H. Derrickson, The principles of anatomy and physiology, 13 ed., John Wiley Sons, 2011.

[6] M. Cadogan, "Lif In The Fast Lane," [Online]. Available: http://lifeinthefastlane.com/education/procedures/lead-positioning/. [Accessed 19 7 2014].

[7] P. Pitigalaarachchi, "ECG simulation using MATLAB, a mathematical approach," [Online]. Available: http://thepansilu.blogspot.se/2010/11/ecg-simulation-using-matlab.html. [Accessed 19 7 2014].

[8] Raspberry Pi Foundation, [Online]. Available: http://www.raspberrypi.org/. [Accessed 19 7 2014].

[9] Broadcom, *BCM2835 ARM Peripherals,* 2012.

[10] Byte Paradigm, "Introduction to I²C and SPI protocols," [Online]. Available: http://www.byteparadigm.com/applications/introduction-to-i2c-and-spi-protocols/. [Accessed 23 7 2014].

[11] Analog Devices, "Analog Devices," [Online]. Available: http://www.analog.com/en/analog-to-digital-converters/ad-converters/adas1000/products/product.html. [Accessed 22 7 2014].

[12] Analog Devices, *Low Power, Five Electrode Electrocardiogram (ECG) Analog Front End,* 2012.

[13] J. Blanchette and M. Summerfield, C++ GUI Programming with Qt 4, Prentice Hall, 2006.

[14] Qt Project, "Qt Licensing," [Online]. Available: http://qt-project.org/products/licensing. [Accessed 27 7 2014].

[15] GNU, "GNU General Public License," [Online]. Available: http://www.gnu.org/licenses/gpl.html. [Accessed 27 7 2014].

[16] GNU, "GNU Lesser General Public License, version 2.1," [Online]. Available: http://www.gnu.org/licenses/old-licenses/lgpl-2.1.html. [Accessed 27 7 2014].

[17] Qt Project, "Qt Widgets," [Online]. Available: https://qt-project.org/doc/qt-

5/qtwidgets-index.html. [Accessed 27 7 2014].

[18] Qt Project, “Signals & Slots,” [Online]. Available: http://qt-project.org/doc/qt-5/signalsandslots.html. [Accessed 27 7 2014].

[19] Qt Project, “Threading Basics,” [Online]. Available: http://qt-project.org/doc/qt-5/thread-basics.html. [Accessed 31 7 2014].

[20] Qt Project, “Synchronizing Threads,” [Online]. Available: http://qt-project.org/doc/qt-5/threads-synchronizing.html. [Accessed 31 7 2014].

[21] Qt Project, “Multithreading Technologies in Qt,” [Online]. Available: http://qt-project.org/doc/qt-5/threads-technologies.html. [Accessed 31 7 2014].

[22] M. Rouse, “Iterative development,” TechTarget, [Online]. Available: http://searchsoftwarequality.techtarget.com/definition/iterative-development. [Accessed 9 8 2014].

[23] A. Wiley, “Iterative Software Development Approach,” 11 6 2008. [Online]. Available: https://wiki.nci.nih.gov/x/eKZ8. [Accessed 9 8 2014].

[24] B. Kemp, “European Data Format,” [Online]. Available: http://www.edfplus.info/. [Accessed 11 8 2014].

[25] C. Becchetti and A. Neri, Medical instrument design and development : from market to placements, John Wiley & Sons Ltd., 2013.

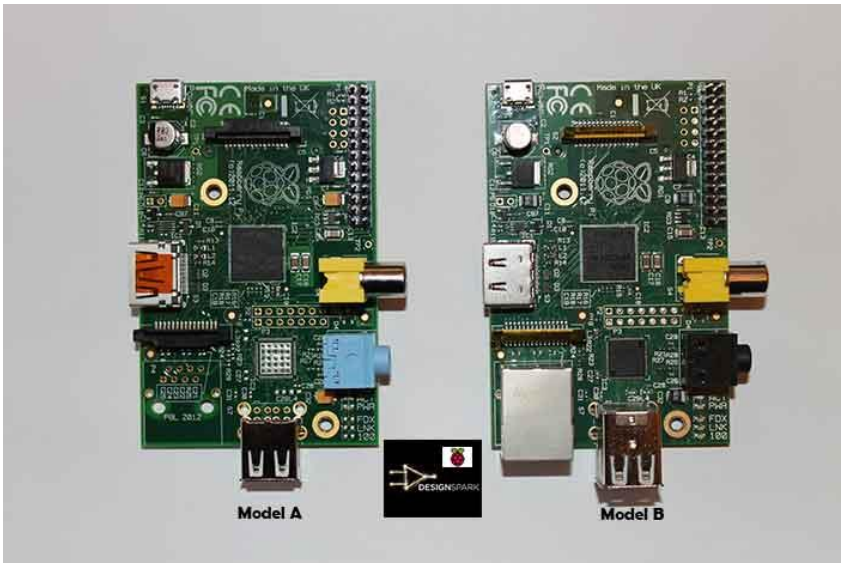
[26] M. McCauley, “C library for Broadcom BCM 2835 as used in Raspberry Pi,” [Online]. Available: http://www.airspayce.com/mikem/bcm2835/. [Accessed 4 8 2014].

[27] udoo.org, “UDOO : Features,” SECO USA Inc, [Online]. Available: http://www.udoo.org/features/. [Accessed 13 8 2014].

[28] eLinux, “Embedded Linux Wiki,” [Online]. Available: http://elinux.org/Main_Page. [Accessed 23 7 2014].

[29] Qt Project, “Cross-Compiling Qt for Embedded Linux Applications,” [Online]. Available: http://qt-project.org/doc/qt-4.8/qt-embedded-crosscompiling.html. [Accessed 27 7 2014].

***APPENDIX A: A comparison between the Raspberry Pi Model A and B***




*The RPI Model A to the left and Model B to the right.*

## Technical specifications

| | | |
|---|---|---|
| **Chip** | Broadcom BCM2835 SoC full HD multimedia applications processor | Broadcom BCM2835 SoC full HD multimedia applications processor |
| **CPU** | 700 MHz Low Power ARM1176JZ-F Applications Processor | 700 MHz Low Power ARM1176JZ-F Applications Processor |
| **GPU** | Dual Core VideoCore IV® Multimedia Co-Processor | Dual Core VideoCore IV® Multimedia Co-Processor |
| **Memory** | 256MB SDRAM | 512MB SDRAM |
| **Ethernet** | None | onboard 10/100 Ethernet RJ45 jack |
| **USB 2.0** | Single USB Connector | Dual USB Connector |
| **Video Output** | HDMI (rev 1.3 & 1.4) Composite RCA (PAL and NTSC) | HDMI (rev 1.3 & 1.4) Composite RCA (PAL and NTSC) |
| **Audio Output** | 3.5mm jack, HDMI | 3.5mm jack, HDMI |
| **Onboard Storage** | SD, MMC, SDIO card slot | SD, MMC, SDIO card slot |
| **Operating System** | Linux | Linux |
| **Dimensions** | 8.6cm x 5.4cm x 1.5cm | 8.6cm x 5.4cm x 1.7cm |

**How to cross-compile and install Qt**

Go to the /home/<user>/ folder and create a new one:

```
/ $ mkdir opt
/ $ cd opt
```

Go to http://www.raspberrypi.org/downloads/ and download the latest image to the

/home/<user>/opt/ folder, unzip it:

```
/ $ unzip <date>-wheezy-raspbian.zip
```

Mount the file system of the raspbian image:

```
/ $ sudo mkdir ./mnt/rasp-pi-rootfs
/ $ sudo mount -o loop,offset=62914560
<date>-wheezy-raspbian.img /mnt/rasp-pi-rootfs
```

Download and extract the cross compile toolchain: (check the link in the beginning if this one doesn't work)

```
/ $ wget http://swap.tsmt.eu/gcc-4.7-linaro-rpi-gnueabihf.tbz
/ $ tar -xf gcc-4.7-linaro-rpi-gnueabihf.tbz
```

Install the ia32libs if a 64 bit system is used:

```
/ $ sudo apt-get install ia32-libs
```

Clone cross compile-tools and Qt5 repository:

```
/ $ git clone
git://gitorious.org/cross-compile-tools/cross-compile-tools.git
/ $ git clone git://gitorious.org/qt/qt5.git
```

Init the repository:

```
/ $ cd qt5
/ $ ./init repository
```

Go back to opt:

```
/ $ cd ..
```

Go to the cross compile-tools directory and fix library paths:

```
/ $ cd cross-compile-tools
/ $ sudo ./fixQualifiedLibraryPaths /mnt/rasp-pi-rootfs/
~/opt/gcc-4.7-linaro-rpi-gnueabihf/bin/arm-linux-gnueabihf-gcc
```

Go to the qtbase folder:

```
/ $ cd ..
/ $ cd qt5/qtbase
```

Then run:

```
/ $ ./configure -opengl es2 -device linux-rasp-pi-g++
-device-option
CROSS_COMPILE=~/opt/gcc-4.7-linaro-rpi-gnueabihf/bin/arm-linux-
gnu
eabihf- -sysroot /mnt/rasp-pi-rootfs -opensource -confirm-
license
-optimized-qmake -reduce-relocations -reduce-exports -release
-make libs -prefix/usr/local/qt5pi
/ $ make -j 4
/ $ sudo make install
```

You now have qmake so you will be able to compile qt-applications. There are also other modules in the qt5 folder that can be installed. To compile and install other modules enter their folder, run your qmake, make and then make install.

Example:

```
/ $ cd <module>
/ $ /usr/local/bin/qmake .
/ $ make -j 4
/ $ sudo make install
```

Recommended modules: qtimageformats, qtsvg, qtjsbackend, qtscript, qtxmlpatterns, qtdeclarative, qtsensors, qt3d, qtgraphicaleffects, qtjsondb, qtlocation, qtdocgallery

When qtbase and additional modules are installed it is time to copy the image to the SD-card.

```
/ $ sync; sudo umount /mnt/rasp-pi-rootfs
/ $ sudo dd bs=1M if=<date>-wheezy-raspbian.img of=/dev/sdX;
sync
```

**How to install qwt**

Qwt version 6.1.0 is required in order to compile the project. The Qwt library can be downloaded here: http://sourceforge.net/projects/qwt/files/qwt/6.1.0/qwt-6.1.0.tar.bz2/download

In order to install the Qwt library a functional qmake is required, make sure qt is working by compiling the example in the link in the beginning.

Make sure that the image is mounted!

```
/ $ cd /opt
/ $ sudo mount -o loop,offset=62914560
<date>-wheezy-raspbian.img /mnt/rasp-pi-rootfs
```

To install qwt, navigate to the folder where the unpacked tarball is located:

```
/ $ cd qwt-6.1.0
/ $ /usr/local/bin/qmake .
/ $ make -j 4
```

```
/ $ sudo make install
```

This should install qwt on your computer and on the mounted image, either overwrite your current image or transfer the installed library to the RPI.

To overwrite, insert the SD-card, check the entry point, then do:

```
/ $ sync; sudo umount /mnt/rasp-pi-rootfs
/ $ sudo dd bs=1M if=<date>-wheezy-raspbian.img of=/dev/sdX;
sync
```

**How to overclock the RPI**

It is recommended to overclock the RPI for some extra performance, boot the pi and log in, then issue the following command:

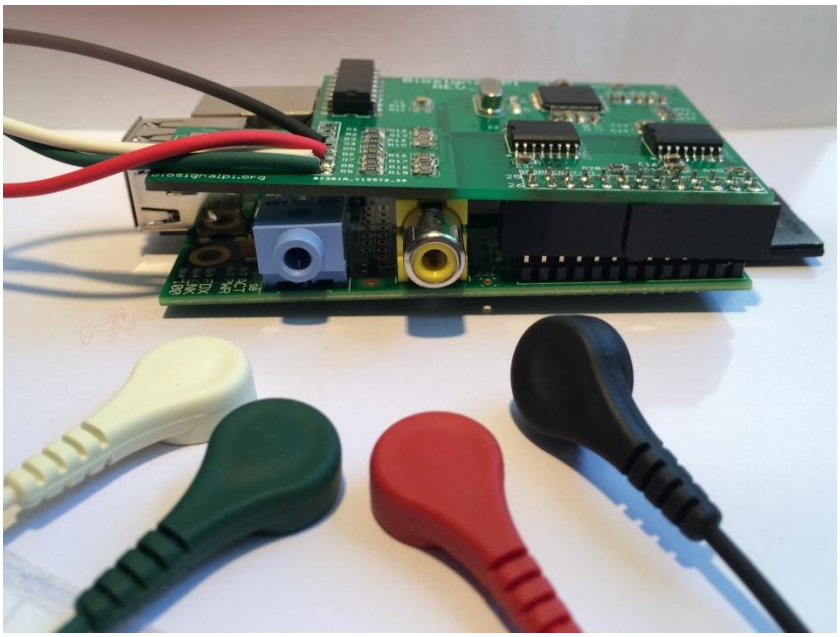
```
/ $ sudo raspi-config
```

Navigate to option #7, overclock

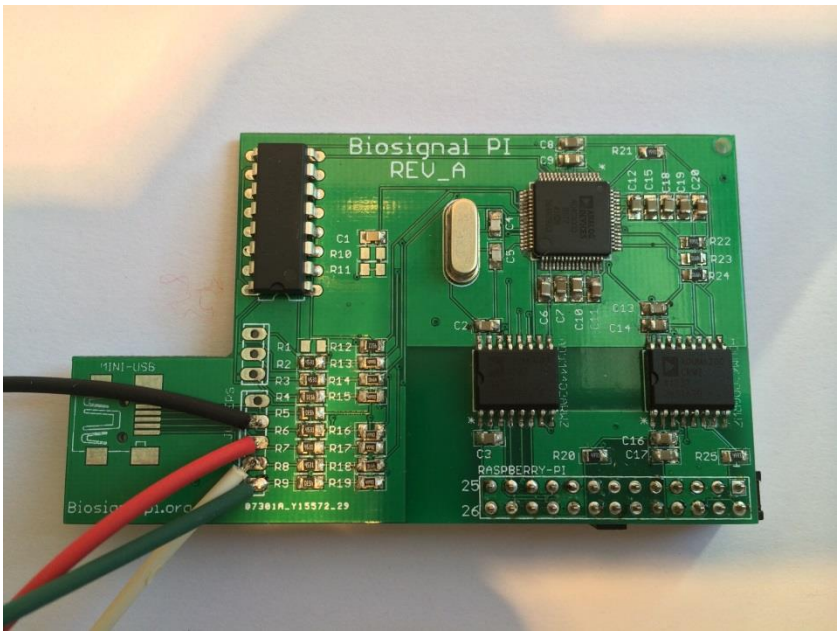
Choose 800 MHz, it should be enough!

Ok → Finish

## *APPENDIX C: Images of the Biosignal Pi*



*The Biosignal Pi shield mounted on top on a Raspberry Pi model B viewed from the side.*




*The Biosignal Pi shield viewed from the top.*

## Starting the application

In order to start the application, navigate to the folder it is placed in and issue the following command:

```
sudo ./BiosignalPI
```

The program should then load and the starting screen will appear. On the bottom of the screen there is a menu that the user can use to navigate through the different views of the program. There is also a button used to take a snapshot of the current screen, this can be useful if the user spots something interesting during ECG capture. There is also a quit button that will exit the application.

**Note:** excessive use of the snapshot button will decrease the system performance and may result in a frame drop during ECG capture.

## How to record an ECG

When the program has started and before and ECG can be recorded the user should go to the settings view by clicking on "settings" in the menu at the bottom of the screen.

The user should first choose a source. If the user wants to record an ECG the "ECG" option should be selected. The user can also choose a number of different test tones in order to verify that the system is working correctly.

When the source is selected the user should choose a sampling rate, the 1 kHz option is selected by default and is the recommended option. The user can also choose the 2 kHz and 500 Hz option.

**Note:** frame drops have been observed when choosing the 2 kHz option.

When source and sample rate have been selected there are a number of additional fields that can be filled in. These are optional and it will still be possible to record an ECG without filling in the fields. If the user choose to leave the filename blank the filename will be "ECG" by default.

**Note:** if a filename has the same name as an already existing file, the old file will be overwritten.

When the settings are applied the user should click on "ECG capture" in the menu on the bottom of the screen.

If the user has chosen one of the test tone options in the settings the user could click on the "Start ECG capture" button without connecting the electrodes, which will start the data collection.

If the user chooses the ECG option in the settings menu it will have to attach the electrodes to the chest by placing one electrode on the upper part of the chest close to the right shoulder and one on the opposite side. One electrode should be placed on the lower right part of the belly and another electrode should be placed on the other side. When the electrodes are connected to the lead the user can press the "Start ECG capture" button. When the user is finished with the data collection the "Stop ECG capture" button should be pressed, this will stop the data collection and the data will be stored to a text-file and also as an EDF-file if the option is selected in the settings view.

**How to view a recorded ECG**

When an ECG has been recorded the user can view the recorded ECG by clicking on the "View ECG" button in the menu on the bottom part of the screen.

To get a list of recorded ECG files the "Get ECG files" button should be pressed, recorded ECGs are stored in two files, one data file and one header file. The user should double click on the file containing the ECG data. This will load the recorded ECG and its header.

Loaded ECG's will by default have the 5s window size selected but the user can choose to the window size to be either 5 s, 10 s, 1 min or the whole recording. To navigate through the ECG the user can use either the mouse scroll button, the scrollbar below the plots or the navigation buttons on the left side of the scroll bar. The user can also choose to playback the ECG by clicking on the play button.