UNIVERSITY OF OSLO
Department of Physics

FPGA Based Development Platform for Biomedical Measurements

ECG Module

Master thesis

Miriam Kirstine Huseby

Spring 2013

FPGA Based Development Platform for Biomedical Measurements

Miriam Kirstine Huseby

Spring 2013

Abstract

This master thesis describes the development of an ECG module made for implementation in an FPGA development platform by Lars Jørgen Aamodt (2013).

The complete ECG system consists of an analogue ECG Front-End, a Data Acquisition Card, and an FPGA Module based on the Pan-Tompkins Algorithm for real-time QRS peak detection.

Measurements principles, instrument design and test results were reviewed. The system works as an independent ECG measuring instrument.

An iPad application was designed as an addition to the Android application designed for the platform. The iPad does not support the Bluetooth protocol used for transmission in the platform, so TCP/IP platform was used. The iPad application was tested and verified with generated data from a TCP/IP server

Acknowledgments

This thesis was carried out in the period from January 2012 to June 2013, under the supervision of professor Ørjan G. Martinsen at UiO Electronics Group, Ph.D Candidate Tore Andrè Bekkeng at UiO Plasma- and Space Physics Group, and Dr. Christian Tronstad at the OUS Rikshospitalet, Department of Medical Technology.

I am very grateful to Ørjan G. Martinsen for giving me the opportunity to work with such an interesting topic. I would like to thank Martinsen for his support and guidance during this work, and for his work with the Bioimpedance group at UiO.

A special thanks to Tore Andrè Bekkeng for the countless hours he patiently helped me when I needed it. Thank you for all the motivation and guidance through the project. A thank you to Christian Tronstad for the insight and help in the medical instrumentation field.

I would also like to thank the guys at the ELAB, especially Halvor Strøm, Stein Nielsen and David Michael Bang, for all the help in the PCB production. I learned a lot from you.

To all my fellow students over the years. Thank you for your support, and for happytimes. A very special thank you to Lars Jørgen Aamodt for his patience, knowledge, and friendship. I would not have made it without you.

To my parents, thank you for all your love and support through everything. To the rest of my family, thank you for being there. A very special thank you to my beautiful sister and her Jonas. This would never have happened without you. Thank you!

Contents

List of Figures

List of Tables

Nomenclature

ADC	Analog-to-Digital Converter
Ag	Silver
$AgCl$	Silver Chloride
API	Application Programming Interface
ATP	Adenosine triphosphate
AV	Atrioventricular
Ca^2+	Calcium ions
Cl^-	Chloride ions
$CMRR$	Common-Mode-Rejection-Ratio
DAC	Digital-to-Analog Converter
DAQ	Data Acquisition Card
DIP	Dual In-line Package
DSP	Digital Signal Processing
ECG	Electrocardiogram
FET	Field-effect Transistor
FIR	Finite Impulse Response
FIR	Finite Impulse Response
$FPGA$	Field-Programmable Gate Array
FPS	Frames per second
GPU	Graphic Processing Unit
HDL	Hardvare Description Language

HRV	Heart Rate Variability
$HSMC$	High-Speed Mezzanine Card
IA	Instrumentation Amplifier
IC	Integrated Circuit
IIR	Infinite Impulse Response
K^+	Potassium ions
LA	Left Arm
LE	Logic Elements
LL	Left Leg
LSB	Least Significant Bit
MSB	Most Significant Bit
mV	Millivolt
Na^+	Sodium ions
PCB	Printed Circuit Board
PLL	Phase-locked Loop
RA	Right Arm
RL	Right Leg
SA	Sinoatrial
SAR	Successive Approximation Register
SMD	Surface-mount Device
TCP/IP	Transmission Control Protocol/Internet Protocol
$UART$	Universal Asynchronous Receiver/Transmitter
$VHDL$	VHSIC Hardware Description Language
$VHSIC$	Very High Speed Integrated Circuits
$XFET$	eXtra implanted junction FET

Chapter 1

Introduction

1.1 Background and Motivation

Measurements of biomedical signals have traditionally been done using PC-based instrumentation and microcontroller technology. Due to the sequential nature of the microcontroller, it offers little flexibility when adding new modules in an existing design.

Field-programmable Gate Arrays (FPGA) are integrated circuits with programmable logic cells and interconnections. The FPGAs are flexible, and functionality can be changed as needed. The technology is concurrent, and new modules can be added without altering the existing design. These properties makes it possible to add new features and perform system maintenance at a low cost and engineering effort. A fully tested and verified FPGA design can easily be transferred to a full-custom Application Specific Integrated Circuit (ASIC) facilitating large scale production. The combination of concurrency, flexibility and low power consumption, makes the FPGA well suited for portable systems.

In the recent years, the use of mobile applications in smartphones and tablets have exploded. This is true for the private market, and it is steadily increasing in the professional market as well. An example of this is the use of tablets, like the Apple Ipad, in public institutions like the parliament and public schools. A mobile application as a monitoring device in a measurement system offers great flexibility for the users. Monitoring can be done in real-time, and with great distance between the observer and the subject measured.

FPGA technology offers a flexible interface that can be used for functions like wireless data exchange. An example of this to use a Bluetooth module for communication and data transfer. Bluetooth is implemented in most computers and mobile devices. This makes it a highly available and low cost alternative for data transfer between an FPGA and a mobile application unit.

1.2 Goals of the Present Work

The goal of this thesis is to design an FPGA based measuring system for ECG measurements. It will be fully functioning as an independent unit, with prospects of implementation on the FPGA development platform prototype as described in Lars Jørgen Aamodt (2013). The complete ECG system will include an analogue ECG circuit, a Data Acquisition card, FPGA based digital signal processing, and an iPad application for monitoring the signal.

Chapter 2

Principles of Cardiac Measurements

Electrocardiophy (or ECG) is the measurement of electrical activity in the cardiac muscular cells. By studying this activity we can analyze the muscular mechanisms of the heart as it pumps blood to the body. The physiological functions of the cardiac cells, gives a predictability as to how the activity should be shown on an ECG. Any abnormalities, or arrhythmia in the heart's muscular activities will therefore be easily picked up when analyzing the output of the ECG. Hence, these measurements are an important tool in medical instrumentation.

2.1 Electrical Conduction in the Heart

The cardiac muscle is comprised of about 300 billion muscle cells, known as myocardiocytes or cardiac myocytes, and cardiac pacemaker cells. As with nervous cells and skeletal cells, the myocardiocytes are classified as excitable cells. The myocardiocytes holds electrical attributes like resting potential, and when appropriately stimulated, an action potential (John G. Webster 2010).

2.1.1 The Resting Potential

All cells in the body have an electric potential difference between its internal and external environments of the membrane. The internal environment, the cytosol, has a large concentration of K^+ ions compared to that of Na^+, Ca^{2+}, and Cl^- ions. The external environment, the extracellular fluid, has a large concentration of Na^+ compared to the concentrations of K^+, Ca^{2+}, and Cl^-. This creates a diffusion gradient directed outwards across the membrane. The cell's membrane is highly permeable to K^+, compared to Na^+. Because of this, there is a higher leakage of K^+ outwards to the extracellular fluid than the Na^+ inwards to the cytosol. With more positive ions leaving the cytosol than entering it, the net value becomes negative and so forth the potential (Olav Sand et al. 2009). The value of the resting potential of a cell is found close to the equilibrium potential of the dominant contributor, measured in volts and calculated from the Nernst equation (2.1)

$$E_K = \frac{RT}{nF} ln \frac{[a]_o}{[a]_i} \tag{2.1}$$

The n is the valence of the ion, $[a]_o$ and $[a]_i$ are the extracellular and intracellular concentrations of the ion in moles pr liter, respectively. R is the universal gas constant (A.1), T is absolute temperature in Kelvin, and F is the Faraday constant (A.1) (John G. Webster 2010). The cell is now polarized. The typical resting potential for myocardiocytes is around -85 to -95 mV, where K^+ is the dominant contributor.

The maintenance of the resting potential for the cell is mainly kept by the $Na^+/K^+ - ATPase$, known as the Sodium-Potassium Pump. With energy from the ATP produced within the cell, the ionic pump actively transports Na^+ out of the cell, and K^+ into the cell at a ratio of $3Na^+$:$2K^+$.

Most cell types maintain a somewhat stable and negative potential. Excitable cells, on the other hand, has the ability to change the potential in a rapid manner to a positive potential. This happens during the action potential.

2.1.2 Cardiac Action Potential

The action potential is the depolarization and repolarization of a cell. It can be explained in five phases as shown in figure 2.1

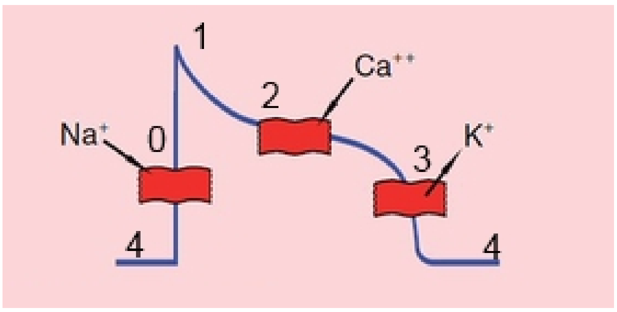


Figure 2.1: **Action Potential in the Cardiac Myocytes.**
(Kreuger 2006) Electrical impulses from nearby cells makes the cell depolarize. When a threshold is met, Na^+ voltage gated channels open and ions start flowing into the cytosol changing the potential. The membrane potential determine the opening and closing of the voltage gated channels for the Na^+, Ca^{2+} and K^+ ions, resulting in a specific potential pattern.

The 5 Phases of the Action Potential:

Phase 4: The resting potential of the membrane. A change from this state will occur when electrical influences from nearby cells open up small gap junctions in the cell membrane. Through the gap junctions, Na^+ and Ca^{2+} ions flow into the cytosol reducing the potential's negativity.

Phase 0: When the depolarization has reached the cell's potential of approximately -70mV, an all-or-none action potential occurs. At this threshold potential, voltage gated channels open and Na^+ starts flowing into the cytosol and rapidly depolarizing the potential towards the Na^+ equilibrium potential of +67mV. Before it reaches this potential, the channels close down, leaving the potential positive.

Phase 1: As the Na^+ channels close down, voltage gated channels open for K^+ leaving the cytosol, and a repolarization occurs.

Phase 2: Before the potential becomes negative, voltage gated channels open for Ca^{2+} to flow into the cytosol. At this stage, the K^+ and Ca^{2+} pull in different directions, and the potential remains positive.

Phase 3: The Ca^{2+} voltage gated channels are closed, and only the voltage gated K^+ channels are open. The K^+-ions flow out of the cytosol, rapidly repolarizing the cell to its resting potential.

(Rishi 2012a)

2.1.3 The Pacemaker Cells

The pacemaker cells are found in the Sinoatrial (SA) node, Atrioventricular (AV) node, Bundle of His, and the Purkinje fibers in the heart. These cells differ from the myocardiocytes in the way that the pacemaker cells has the property of automaticity. When the myocardiocytes require electrical impulses to start their action potential, the pacemaker cells do that themselves. It is this property that keeps the heart beating. The duration of one action potential determines the heart rate of the person (Rishi 2012b). The ion flow through the membrane is shown in figure 2.2

2.1.4 The Cardiac Conduction System

The pacemaker cells in the SA node initiates the rhythmic cardiac impulse. The impulse passes from the SA node through the anterior, middle, and posterior internodal tracts to the AV node, and to the Bachmann's bundle through the interial tract. It does so in an organized manner, first activating the right and then the left atrium. The impulse is delayed at the AV node, before it continues to the Bundle of His, the right bundle branch,

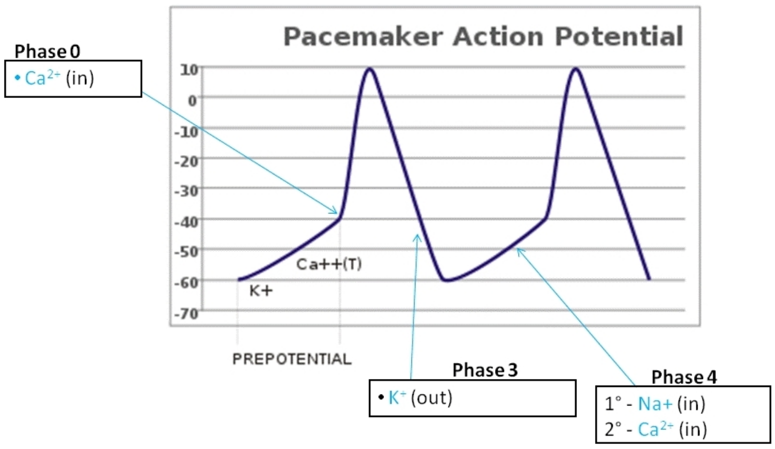


Figure 2.2: **Action Potential of the Pacemaker Cells.**
The pacemaker cells does not have a resting potential. The properties of the cell help maintain a steady flow of ions in and out through voltage gated channels when certain potentials are reached.

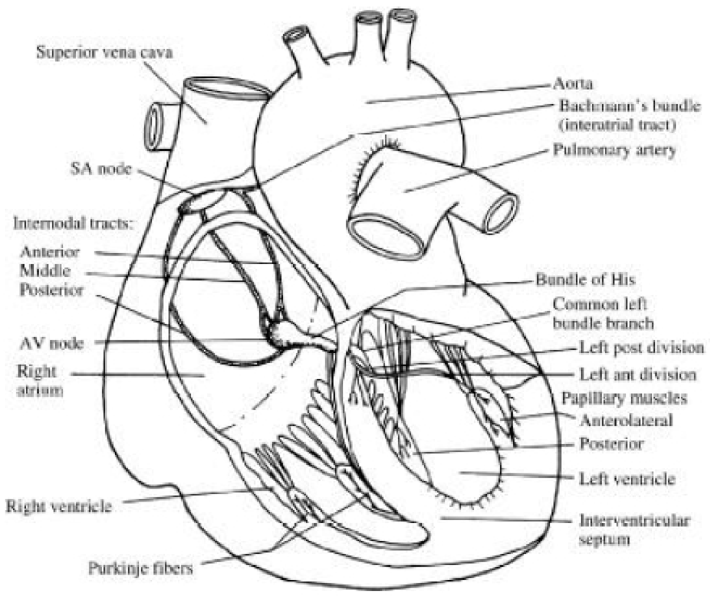


Figure 2.3: **The Cardiac Conduction System**
(©John G. Webster 2010) The electrical impulse originates at the SA node. It passes from the SA node, through specialized conducting tracts, to the AV node and the Bachmann's bundle. After a delay at the AV node, the impulse continues to the Bundle of His, the right bundle branch, the left bundle branch, anterior and posterior divisions of the left bundle, and the Purkinje network.

the common left bundle branch, the anterior and posterior divisions of the left bundle, and the Purkinje network (Figure 2.3) (B.S. Lipman et al. 1972).

ECG Wave during Electrical Conduction

The electrical impulses propagating through the heart is registered by the ECG as a waveform. Each region of the waveform is produced by the various depolarizations and repolarizations of the different muscle tissues as the electrical impulse runs through them. (Figure 2.4)

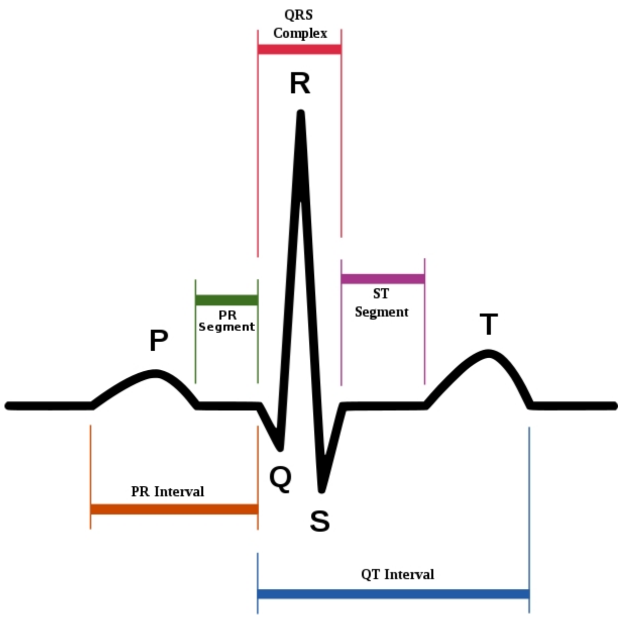


Figure 2.4: **ECG Waveform**
The P-Q-R-S-T waves and spikes on the ECG is a visual representation of the depolarization and repolarization of the muscle cells in the atria and ventricles as the heart beats. (Wikipedia 2004a)

P wave: The P wave represents the depolarization of the atria, and starts just before the atrial contraction. The duration of the P wave is 60 to 110 ms.

PR interval: The PR interval represents the delay in the AV node. Parts of the atrial repolarization is also represented here. The duration of the interval is usually about 120 to 200 ms long.

QRS complex: The QRS complex consists of three spikes closely following each other. As a whole, the spikes represent the depolarization of the ventricles. The ventricular contraction begins during the QRS complex. The muscle mass of the ventricles is larger than that of the atries. This becomes visible in the ECG waveform, as the QRS complex is larger than the P wave. In the ventricles, the left muscle contains more muscle mass and therefore contributes more to the QRS complex. The duration of the complex is normally 60 to 100 ms. In children and during physical activity, it may be shorter.

QT interval: The QT interval represents the depolarization and repolarizarion of the ventricles. The typical duration of this is 80 ms.

ST segment: The ST segment represents the period when the ventricles are depolarized. The segment has a duration of 80 to 120 ms.

T wave: The T wave represents the repolarization of the ventricles. The T wave is smaller than the QRS complex because the ventricular repolarization is slower than the depolarization. The interval from the beginning of the QRS complex to the apex of the T wave is the absolute refractory period and is 250 ms in length. The last half of the T wave is the relative refractory period. The refractory period is the amount of time it takes for an excitable cell to be ready for a second stimulus after it has settled to its resting state following excitation.

U Wave: The U wave is normally not seen in the ECG. It follows the T wave, and is thought to represents the repolarization of the papillary muscles or Purkinje fibers. An example of the U wave is shown in figure 2.5.

(Olav Sand et al. 2009) (Wikipedia 2004b)

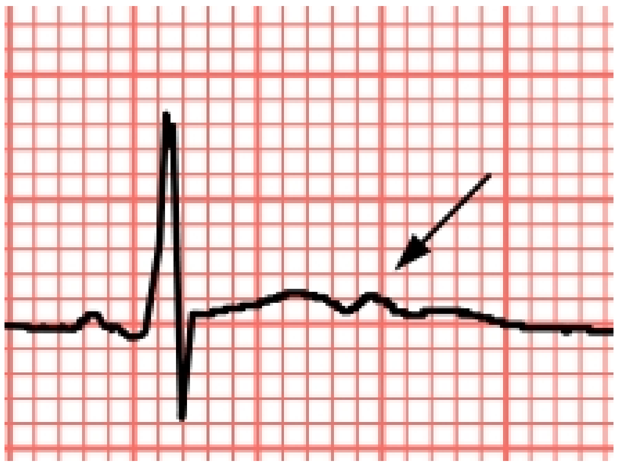


Figure 2.5: **U Wave seen on an ECG**
(Wikipedia 2007) The U wave is thought to represent the repolarization of the papillary muscles or the Purkinje fibers. It is rarely visible on the ECG

2.2 ECG Electrodes and Leads

To be able to measure the electrical impulses in the heart, a connection of electrodes and leads is set up from the human body to the ECG.

2.2.1 The Electrode-Electrolyte Interface

To be able to measure the currents in the body, there needs to be an interface between the body and the electronic measuring apparatus. The body's current is carried by ions, while the electrodes carries electrons. An electrode-electrolyte interface, illustrated in figure 2.6 is created by adding an electrolytic gel on the metal electrode.

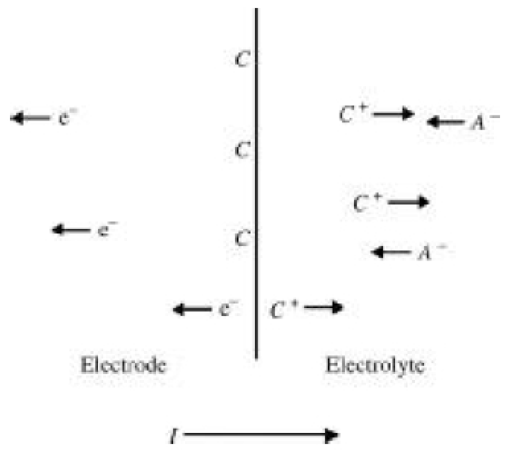


Figure 2.6: **Electrode-electrolyte interface**
(John G. Webster 2010) The electrode consists of metallic atoms C. The electrolyte is an aqueous solution containing cations of the electrode metal C^{+}, and anions A^{-}. The current crosses from left to right.

At the interface, the chemical reaction represented by the reactions (2.2) and (2.3), occurs.

$$C \rightleftharpoons C^{n+} + ne^{-} \tag{2.2}$$

$$A^{m-} \rightleftharpoons A + me^{-} \tag{2.3}$$

The electrode metal is made up of atoms of the same material as cations in the gel. The cation is discharged into the electrolyte, and the electron remains as a charge carrier in the electrode. The anion at the interface is oxidized to a neutral atom, giving off free electrons to the electrode. This results in a net current crossing the interface, passing from the electrode to the electrolyte (John G. Webster 2010).

A commonly used ECG electrode is made up of a disk of Ag with a layer of AgCl on its surface. The disk is placed on the test subject's chest. It stays in place, either by surgical tape or a plastic foam disc with adhesive tack on one surface (John G. Webster 2010).

When a metal comes in contact with a solution containing ions of that metal, the chemical reaction represented in equation (2.2) begins immediately. Depending on the concentration of the cations and the reactions' equilibrium conditions, the reaction goes predominantly either to the left or the right. At the interface, the local concentration of the cations change, which also affects the anion concentration. At this point on the interface, the neutrality of charge is not maintained and a difference in electrical potential from the rest of the solution is created. This potential difference is known as the *half-cell potential*, and is 0.223V for Ag/AgCl electrodes (John G. Webster 2010).

The outer layer of the skin, the stratum corneum, is dielectric and impairs the transference of ions in the tissue to electrons in the electrode as an impedance of the skin. This, in addition to the resistance of the electrolytic gel, the double layer at the electrode-electrolyte interface, and the half-cell potentials at both electrolyte interfaces, can be described as the electrical circuit in figure 2.7

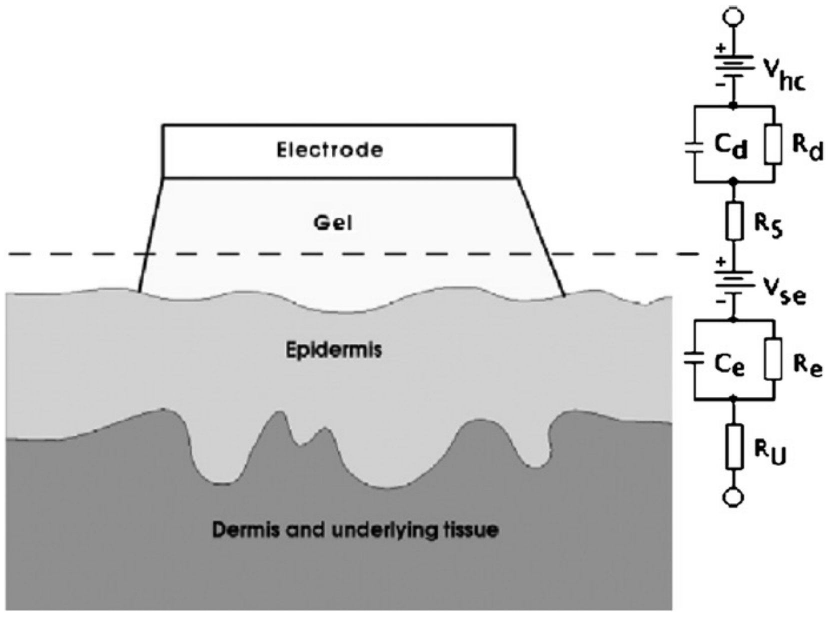


Figure 2.7: **Electrical equivalent circuit when body-surfaced electrodes is placed on the skin**
The circuit elements on the right represent the physical process at which they are aligned with on the left. (John G. Webster 2010.)

The Ag/AgCl show a low ohmic impedance in the whole frequency specter. Its main purpose is to reduce the impact of the skin impedance by making the stratum corneum ion conductive (Anna Gruetzmann et al. 2007). The half-cell potential at the electrodes introduce an offset to the signal. The Ag/AgCl electrodes have a maximum offset of +/-300mV.

2.2.2 Leads

To better understand the electrical activity of the heart, electrocardiographers have developed a model where "the heart consists of an electric dipole located in the partially conducting medium of the thorax" (John G. Webster 2010). The dipole, and the field produced by it, changes magnitude and orientation according to the electrical activity of the heart at that given time. Instead of drawing a field plot to analyze the dipole field of the heart, it is represented by its dipole moment. The dipole moment is a vector directed from the negative to the positive charge. The magnitude of the vector is proportional to the amount of charge multiplied by the separation of the two charges. The vector is known as the cardiac vector, illustrated in figure 2.8 (John G. Webster 2010).

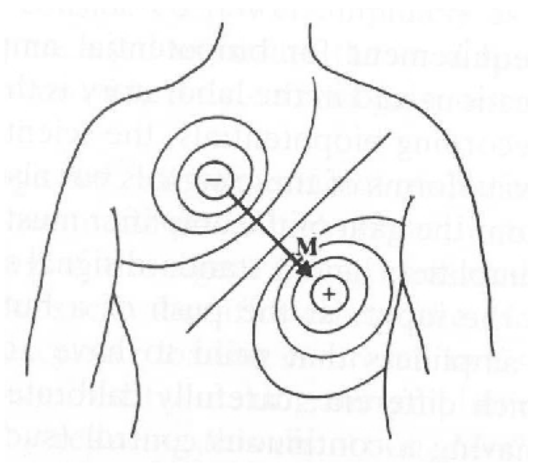


Figure 2.8: **Dipole field of the heart when the R wave is maximal**
The dipole moment vector **M** is made up of points of equal positive and negative charge separated from one another. (John G. Webster 2010)

During one cardiac cycle, 3 main changes are registered in the cardiac vector. These are the atrial depolarization, the ventricular depolarization, and the ventricular repolarization. They all differ in magnitude, direction and duration, resulting in the ECG waveform.

A number of electrodes are connected and placed at different places on the body surface. A pairing of two electrodes is known as a lead. The space between them is defined as a lead vector. The lead vector defines the direction the cardiac vector must have to generate maximum voltage in a lead (John G. Webster 2010).

The three basic leads of the ECG setup makes up the frontal-plane ECG. In this setup, electrodes are placed on the right arm (RA), left arm (LA) and left leg (LL). If ground is required, or an addition of a separate special circuit like the driven-right-leg (Figure 2.13), an electrode is placed on the right leg. We then have the three leads:

I:	LA to RA
II:	LL to RA
III:	LL to LA

This forms an equilateral triangle, known as Einthoven's triangle. (Shown in figure 2.9) These three leads are all bipolar, meaning they have one positive and one negative pole (John G. Webster 2010).

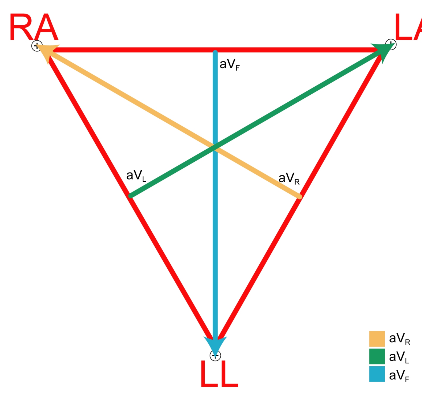


Figure 2.9: **Einthoven's triangle**
Vector diagram of the three limb leads I, II, III, and the augmented leads aV_R, aV_L, aV_F. The limb leads and augmented leads are derived from the same electrodes, but view the heart activity from different angles.

A clinical ECG is made up of the three limb leads, three augmented leads in the frontal plane, and six leads in the transverse plane. The three augmented leads in the frontal plane, aV_R, aV_L and aV_F, are derived from the same electrodes as the limb leads. (Shown in figure 2.9). The augmented and transverse leads differ from the the limb leads in the sense that they are based on signals obtained from more than one pair of electrodes. They are called unipolar leads, because potential appearing on one electrode is taken with respect to an equivalent reference electrode. The reference electrode is the average of the signals seen at two or more electrodes. This setup is known as the Wilson Central Terminal, shown in figure 2.10 (John G. Webster 2010).

A setup with the leads in both the frontal and the transverse plane is called a 12 lead ECG. The placement of the electrodes is shown in figure 2.11, and the ECG signal at each lead is shown in figure 2.12

ECG electrodes have dedicated labels and colours. The standards are different in the USA and in Europe. A list of these standards is found in table 2.1.

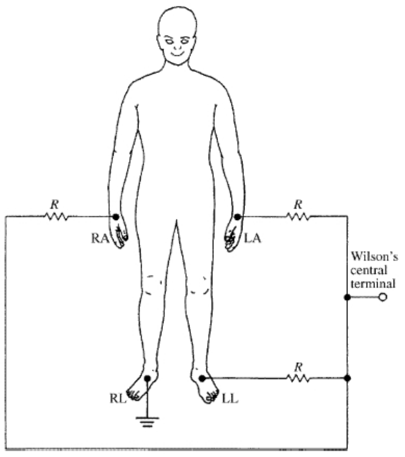


Figure 2.10: **Electrodes connected to the body to obtain the Wilson Central Terminal**
The three limb electrodes are connected through equal-valued resistors to a common node, the Wilson Central Terminal. The voltage here is the average of the voltages at each electrode. (©John G. Webster 2010)

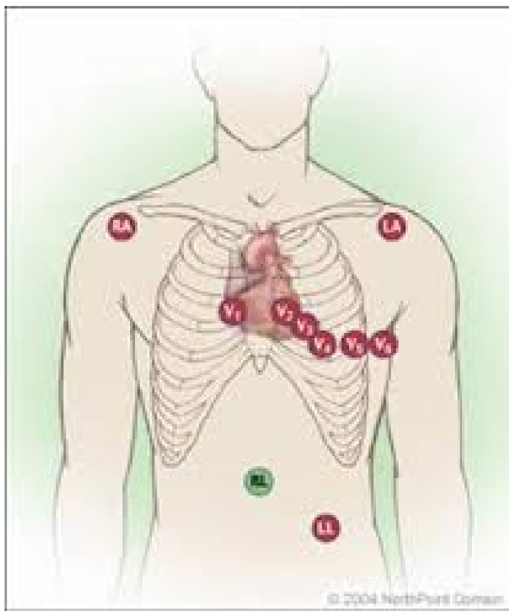


Figure 2.11: **Electrode placement in a 12 lead ECG**
The three limb leads, LA, RA, LL, form Einthoven's triangle. V1, V2, V3, V4, V5 and V6 are the leads seeing the potentials in the transverse plane. RL is connected to ground the subject.

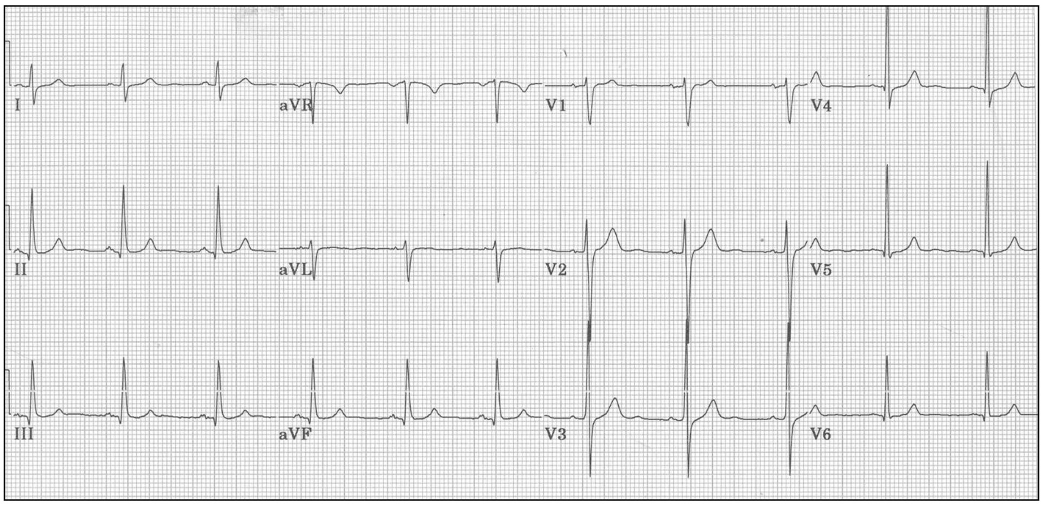


Figure 2.12: **Signals registered on a 12 lead ECG**
ECG signal seen from the limb leads; I, II, III, the augmented leads; aV_R, aV_L, aV_F, and the leads in the transverse plane; V1, V2, V3, V4, V5, V6 .

Table 2.1: Colour System of ECG leads

	AHA (American Heart Association)		IEC (International Electrotechnical Commission)	
Location	**Inscription**	**Colour**	**Inscription**	**Colour**
Right Arm	**RA**	White	**R**	Red
Left Arm	**LA**	Black	**L**	Yellow
Right Leg	**RL**	Green	**N**	Black
Left Leg	**LL**	Red	**F**	Green
Chest	**V1**	Brown/Red	**C1**	White/Red
Chest	**V2**	Brown/Yellow	**C2**	White/Yellow
Chest	**V3**	Brown/Green	**C3**	White/Green
Chest	**V4**	Brown/Blue	**C4**	White/Brown
Chest	**V5**	Brown/Orange	**C5**	White/Black
Chest	**V6**	Brown/Purple	**C6**	White/Violet

ECG leads in USA and Europe are labeled after different standards

2.3 Clinical ECG

A typical clinical ECG consists of the following blocks:

2.3.1 Amplifiers

The purpose of the amplifier is to take the weak electric biopotential and increase its amplitude for further processing and analysis. The biopotential signal is obtained by bipolar electrodes, so a differential amplifier is used. The amplifier should have a high input impedance so the signal measured is provided with minimal load. It must have a high common-mode-rejection-ratio (CMRR) to minimize interference from a common-mode signal. Due to the small amplitudes of biopotential signals, the amplifier must have a high gain. It must also be able to operate within the frequency spectrum were the signals are found (John G. Webster 2010).

2.3.2 Noise Reduction

The ECG signal experience interference from several sources of noise. From the test subject itself there can be some muscle noise. From the surroundings, the system will experience interference from electrical devices and power lines. The leads connected to the subject can also add to the noise as a form of antenna when not properly connected.

To reduce the noise in the system, filtering is needed. A notch filter can reduce the 50 Hz noise from the power lines. Since the ECG frequencies are in the lower area, one can also limit interference with a lowpass filter. Shielding the leads and grounding them, will help reducing interference. Low impedance in the electrodes is also helpful for this purpose.

Driven-Right-Leg Circuit

The Driven-right-leg circuit reduces interference seen from the ECG amplifier. Through a circuit as shown in figure 2.13 , the common mode voltage is driven low in the negative feedback. It also grounds the patient. If a high voltage appears between the subject and ground, the auxiliary op.amp will saturate and consequently unground the subject.

2.3.3 Isolation Circuit

An isolation circuit contains a barrier from the test subject from the power line. Such a barrier will prevent dangerous currents from flowing either to or from the test subject. Examples of an isolation barrier can be an optocoupler or an isolation amplifier.

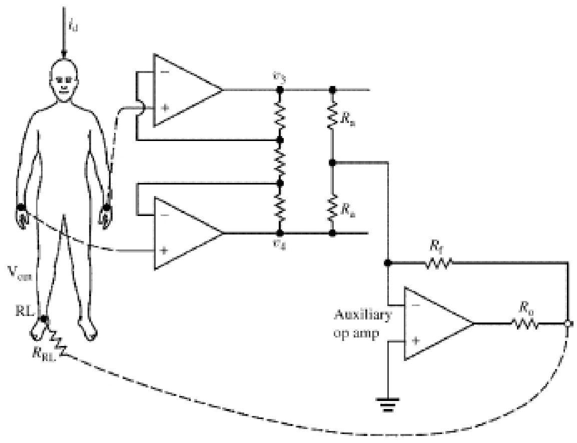


Figure 2.13: **Driven-right-leg circuit**
The circuit minimizes common-mode interference from a pair of averaging resistors connected to the instrumentation amplifier. The right leg is connected to the output of the auxiliary op.amp. (John G. Webster 2010)

2.3.4 Analog-to-Digital Converter

The Analog-to-Digital Converter (or ADC) digitizes the analog ECG signal for processing in a microcomputer or other digital environments.

2.3.5 Microcomputer

The microcomputer perform analysis on the ECG signal. With the help of digital processing, the user can implement algorithms computing the wanted information from the original signal. A processed signal can, among others, determine heart rate and recognize arrhythmia in the heart.

2.3.6 Monitor/Printer

A printer provides a hard copy of the ECG signal. The output is written on specialized paper helping the physician to easily read the signal. In digital systems, memory can be implemented and the information can be stored and later analyzed.

2.4 The Pan-Tompkins QRS Detection Algorithm

The following section describes the Pan-Tompkins real-time QRS detection algorithm. Full reference can be found in J.Pan (1985). This whole section is based on the findings in that publication.

The algorithm consists of a series of digital filters and signal processing implemented in a digital processing unit. The signal processed is the output from an ADC, at a sampling rate of 200 Hz. It passes through a lowpass filter and a highpass filter to eliminate noise, before it continues through a filter that approximates a derivative. Next, there is an amplitude squaring process before the signal passes through a moving-window integrator. The final step is an adaptive threshold method to determine the location of the signal peaks.

Bandpass Filter

The bandpass filter reduces the influence of muscle noise, 50 Hz interference, baseline wander and T-wave interference. The desired passband to maximize the QRS energy is approximately $5 - 15$ Hz. The filters all have poles and zeros on the unit circle of the z plane. The bandpass filter is built from cascaded lowpass and highpass filters to achieve the desired bandpass

Lowpass Filter

The lowpass filter is of the second order, and given by the transfer function (2.4).

$$H(z) = \frac{(1 - z^{-6})^2}{(1 - z^{-1})^2} \tag{2.4}$$

With T as the sampling period, the difference equation of the filter is given by (2.5).

$$y(nT) = 2y(nT - T) - y(nT - 2T) + x(nT) - 2x(nT - 6T) + x(nT - 12T) \tag{2.5}$$

This gives a cutoff frequency of about 11 Hz.

Highpass Filter

The highpass filter is given by the transfer function in (2.6).

$$H(z) = \frac{(-1 + 32z^{-16} + z^{-32})}{(1 + z^{-1})} \tag{2.6}$$

The difference equation is given by (2.7).

$$y(nT) = 32x(nT - 16T) - [y(nT - T) + x(nT) - x(nT - 32T)] \tag{2.7}$$

The low cutoff frequency is about 5 Hz.

Derivative

The signal is differentiated to provide the QRS complex slope information. A five-point derivative is used, with the transfer function (2.8)

$$H(z) = (1/8T)(-z^{-2} - 2x^{-1} + 2z^{1} + z^{2}) \tag{2.8}$$

This give the difference equation in (2.9).

$$y(nT) = (2x(nT) + x(nT - T) - x(nT - 3T) - 2x(nT - 4T))/8 \tag{2.9}$$

The fraction 1/8 is an approximation of the gain factor of 0.1 to permit fast power-of-two calculation. The derivative approximates an ideal derivative between dc and 30 Hz.

Squaring Function

The signal is now squared point by point with the equation (2.10).

$$y(nT) = [x(nT)]^2 \tag{2.10}$$

All data points are now made positive, and the squaring also does a nonlinear amplification of the derivative emphasizing the higher frequencies.

Moving-Window Integration

The moving-window integration obtain waveform information and the slope of the R wave. It is given by the function (2.11).

$$y(nT) = (1/N)[x(nT - (N-1)T + x(nT - (N-2)T) + ... + x(nT)] \tag{2.11}$$

where N is the number of samples in the width of the integration window. The width of the window should be approximately the same as the widest possible QRS complex. A window too narrow will produce several peaks in the integration waveform, while a window too wide will merge the QRS and T complexes together. For a sample rate of 200 Hz, a window of 30 samples has proven to be a good size.

Threshold adjustment

The time duration of the rising edge of the integration waveform corresponds to the duration of the QRS complex. Thresholds are automatically adjusted to float over the noise. Due to the good signal-to-noise ratio improved by the bandpass filter, low thresholds are possible.

Two thresholds are used to detect a QRS complex. The higher of the two, Threshold1, identifies the peaks of the signal. If no peaks have been identified within a certain time interval, the lower Threshold2 is used to find a peak using a search back method. The thresholds are adjusted as the integrated signal moves forward, and computed from

$$SPKI = 0.125PEAKI + 0.875SPKI \tag{2.12}$$

$$NPKI = 0.125PEAKI + 0.875NPKI \tag{2.13}$$

$$THRESHOLDI1 = NPKI + 0.25(SPKI - NPKI) \tag{2.14}$$

$$THRESHOLDI2 = 0.5THRESHOLDI1 \tag{2.15}$$

with the variables given as

PEAKI:	The overall peak
SPKI:	The running estimate of the signal peak
NPKI:	The running estimate of the noise peak
THRESHOLDI1:	The first threshold applied
THRESHOLDI2:	The second threshold applied

To be identified as a QRS complex, a peak must be recognized as such a complex both in the integration and the bandpass-filtered waveforms.

RR Intervals

Two RR intervals are maintained. One is the average of the last eight beats, regardless of their values. The second is the average of the eight last beats that fall within certain limits. The reason for this is to be able to adapt to quickly changing or irregular heart beats. They are computed by the following functions

$$RRAVERAGE1 = 0.125(RR_{n-7} + RR_{n-6} + ... + RR_n) \tag{2.16}$$

where RR_n is the most recent RR interval.

$$RRAVERAGE2 = 0.125(RR'_{n-7} + RR'_{n-6} + ... + RR'_n) \tag{2.17}$$

where RR'_n is the most recent RR interval falling between the acceptable low and high limits given as

$$RRLOWLIMIT = 92\% RRAVERAGE2 \tag{2.18}$$

$$RRHIGHLIMIT = 116\% RRAVERAGE2 \tag{2.19}$$

$$RRMISSEDLIMIT = 166\% RRAVERAGE2 \tag{2.20}$$

If a QRS complex is not found within the RR MISSED LIMIT, the maximal peak between the two established thresholds are considered to be a QRS candidate.

If each of the intervals calculated from RR AVERAGE1 is between RR LOW LIMIT and RR HIGH LIMIT, the heart rate is considered as normal.

T-Wave Identification

If the RR interval is less than 360 ms, the peak detected might be a peak from the T wave. If the maximal slope from the waveform is less than half of the preceding one, the peak is identified as a T wave. Otherwise it is determined as a QRS complex.

Heart Rate Variability

Heart rate variability is the variation in the time between heartbeats. The RR interval calculations output the heartbeat. The difference between each of these outputs, will give the heart rate variability measured over time.

Waveforms in the Pan-Tompkins Algorithm

The processing of the signal from the output of the ADC til the peak detection give the waveforms as shown in figure 2.14

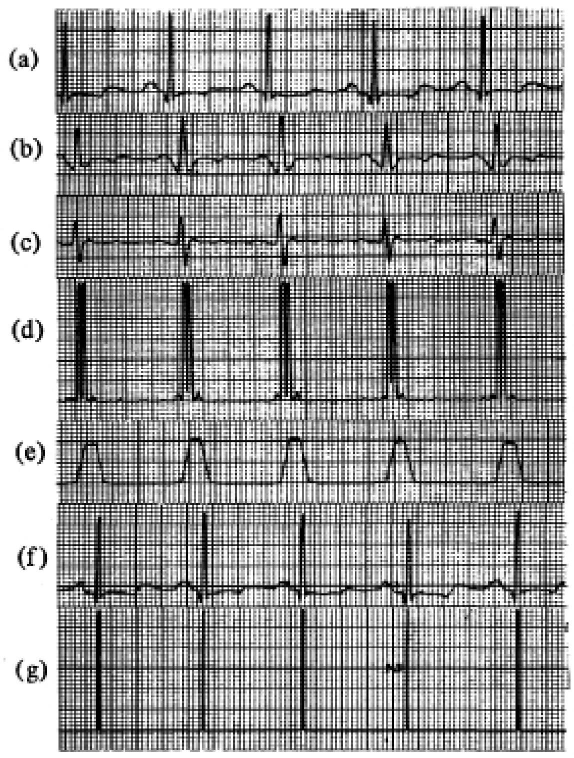


Figure 2.14: **Waveforms of the Steps through the Pan-Tompkins QRS Detection Algorithm**
(a) Original signal. (b) Output of bandpass filter. (c) Output of derivative filter. (d) Output of squaring process. (e) Output of Moving-Window Integrator. (f) Original signal delayed by processing time. (g) Output of pulse stream (©J.Pan 1985)

Chapter 3

Analog Front-Ends

This chapter describes the design and development of the two front-end cards, the ECG circuit and the Data Acquisition Card (DAQ).

3.1 ECG

The ECG circuit board take the weak electric biopotential and increase its amplitude for further processing and analysis. The heart rate is the number of peaks in the QRS complexes pr minute. As seen in figure 2.12 in section 2.2.2, a 1-lead ECG is sufficient for that purpose. The outline of the circuit was inspired by an article published by Shawn Carlson in the Scientific American (2000), and a paper by Bobbie and Arif at the Southern Polytechnic State University in Georgia, USA (2004). The filters at the output of the instrumentation amplifier were designed to meet the requirements of anti-aliasing and data processing in the ADC.

3.1.1 Circuit Design

The circuit was first designed and tested on a breadboard, before being realized on a Printed Circuit Board (PCB). The frequency response of the filters were simulated in Texas Instruments' TI Spice software. Ag/AgCl electrodes and a 5-lead ECG cable were provided by OUS Rikshospitalet. The output of the circuit was verified on an oscilloscope. For practical purposes, the test subject during the development of the circuit was the author of this thesis.

Electrodes and Lead

Ag/AgCl electrodes were placed on LA, RA and RL. The LA was connected to the negative input of the instrumentation amplifier, the RA to the positive input, and RL was connected to a $10M\Omega$ resistor connected to ground. This setup measures lead I in an ECG and outputs a signal as seen in figure 2.12 in section 2.2.2 on page 12.

For practical purposes, the lead cables used during circuit verification on the breadboard were made of lead wires intertwined and connected to the electrodes with a banana connector. For the final PCB prototype, a 5-lead ECG cable was used for this purpose. The ECG cable offers greater protection from interference, as each lead is shielded and the shield is grounded at the ECG PCB.

Instrumentation Amplifier

The instrumentation amplifier (IA) used was the AD624AZ from Analog Devices. It holds many of the desired properties wanted from an IA in an ECG; high input impedance, high CMRR, high gain accuracy, high linearity, and low noise. The AD624AZ is designed to provide noise performance near the theoretical noise floor. Input noise spectral density is shown in figure 3.1

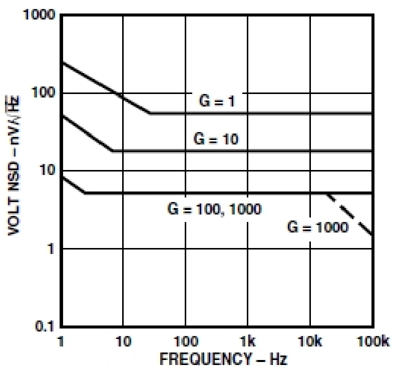


Figure 3.1: **Input Noise Spectral Density vs Gain for AD624AZ**
With a gain of 1000, the input noise spectral density is less than 10 $\frac{nV}{\sqrt{Hz}}$

The AD624AZ includes high accuracy pre-trimmed internal gain resistors, allowing gains up to 1000. At this gain setting the AD624AZ gives a CMRR of approximately 130 dB for the input frequency. (Figure 3.2)

A Driven-Right-Leg circuit was considered for the purpose of minimization of common-mode interference. However, the specifications of the AD624AZ guarantees a very high CMRR at the gain of 1000 so the driven-right-leg circuit was left out in the final design.

Noise Reducing Filters

The frequency range of an ECG signal for monitoring purposes, is 0.05 - 40 Hz. To reduce noise and DC offset on the signal from the AD624AZ

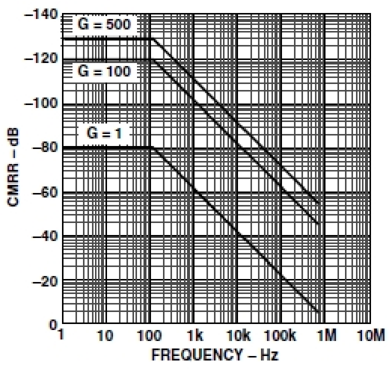


Figure 3.2: **CMRR vs Input Frequency**

output, three filters were implemented. At the output of the AD624AZ, cascaded lowpass and highpass filters were implemented to remove any high and low frequency noise. The highpass filter also works as an AC coupling removing the DC offset introduced by the half-cell potential in the electrodes. A notch filter was made to reduce the 50 Hz interference from the surroundings. An operation amplifier (LMC6464) was used as a voltage follower between the highpass filter and the notch filter, to prevent any loading effects from the latter. The LMC6464, by Texas Instruments, is a quad micropower, rail-to-rail input output, CMOS operational amplifier with very low power consumption. Simulations of the frequency response of the filters were done with Texas Instruments' TI Spice software.

Lowpass Filter :
The lowpass filter was a second order lowpass filter with the cutoff frequency calculated from (3.1)

$$f_c = \frac{1}{2\pi RC} \tag{3.1}$$

$f_c \approx 48$ Hz for this filter. The second order attenuates the signal with 40 dB pr decade.

Highpass Filter :
The highpass filter was a first order filter with $f_c \approx 0.048$ Hz, calculated from (3.1). The highpass filter attenuates the signal with 20 dB pr decade. The cascaded lowpass and highpass filters form a bandpass filter. A schematic of the complete bandpass filter is shown in figure 3.3

The simulated frequency response of the bandpass filter is shown in figure 3.4

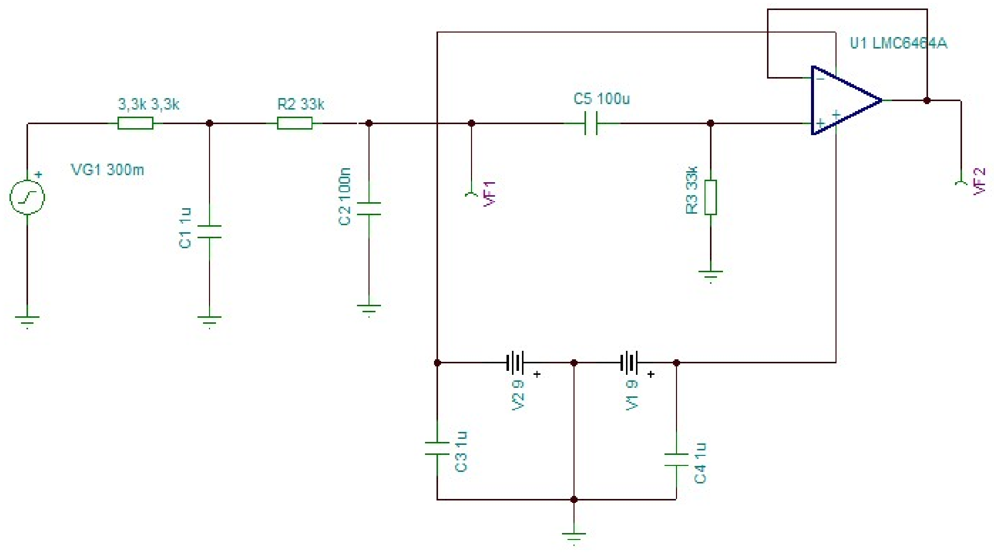


Figure 3.3: **Bandpass Filter**
Bandpass filter placed at the output of the AD624AZ

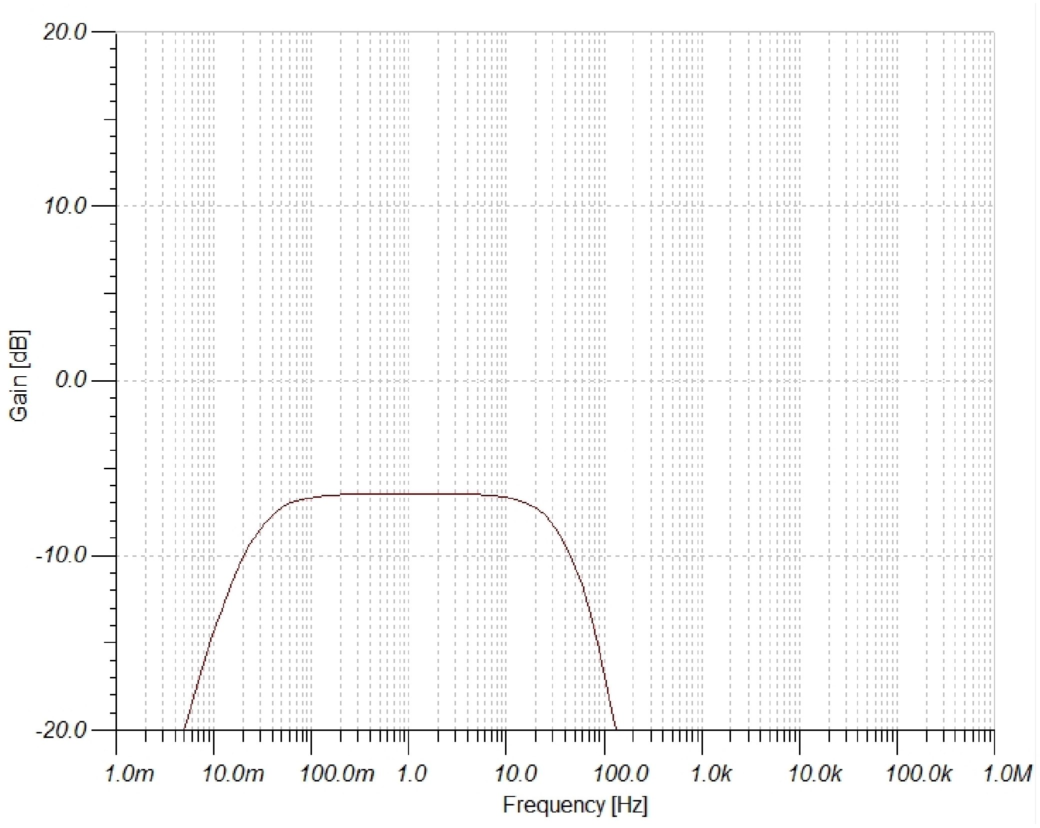


Figure 3.4: **Simulated frequency response of Bandpass Filter**

Notch Filter

After a voltage follower, an op amp notch filter was implemented. A diagram of the ciruit is shown in figure 3.5

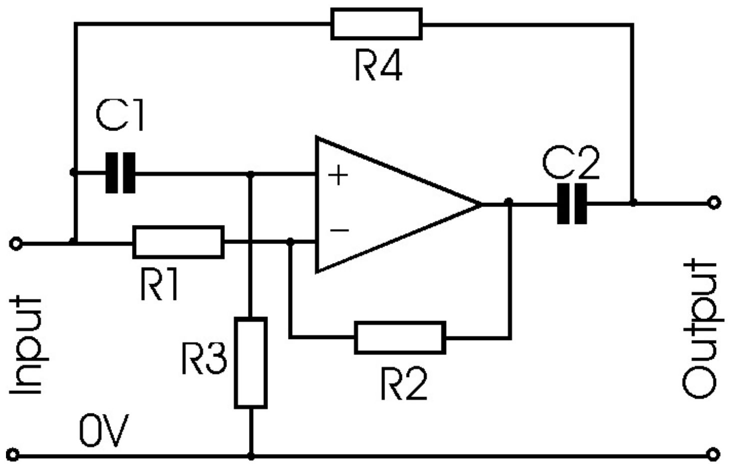


Figure 3.5: **Circuit of an Operation Amplifier Notch Filter to eliminate 50 Hz interference.**
The f_{notch} is calculated from equation (3.2), where the component values are set as R = R3 = R4 and C = C1 = C2. (Poole 2013)

The circuit employs both positive and negative feedback around the op.amp, providing a high degree of performance (Poole 2013). The center frequency of the notch filter is calculated from equation 3.2

$$f_{notch} = \frac{1}{2\pi RC} \tag{3.2}$$

The values of the resistors and capacitors are given as

R = R3 = R4

C = C1 = C2

Values for the 50 Hz notch filter are

C1 = C2 = 47nF

R1 = R2 = $10k\Omega$

R3 = R4 = $68k\Omega$

The frequency response of the filter, simulated in TI Spice, is shown in figure 3.6

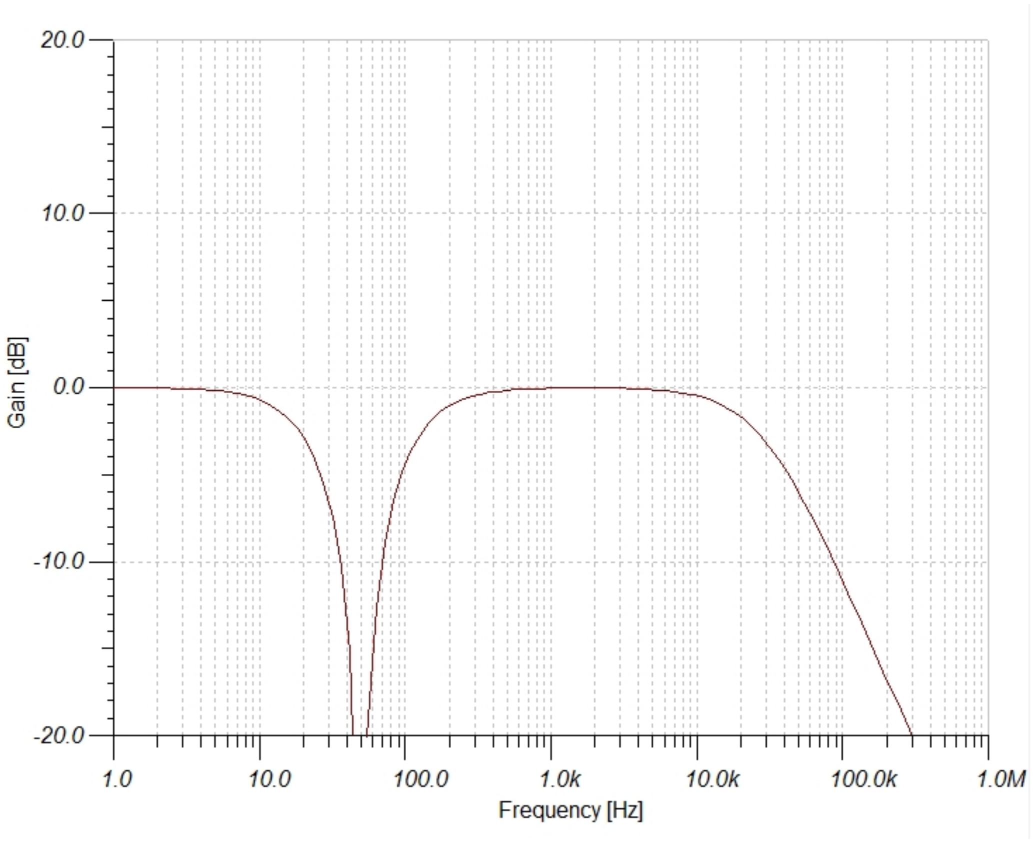


Figure 3.6: **Simulated frequency response of the notch filter circuit**

Gain

For a better control over the amplitude of the ECG signal on the output of the PCB, an adjustable gain was placed at the end of the circuit. The gain help increase the amplitude of the signal, so the full range of the following ADC can be taken advantage of. The gain has a maximum of 5. The simulated frequency response of the filters, with gain, is shown in figure 3.7

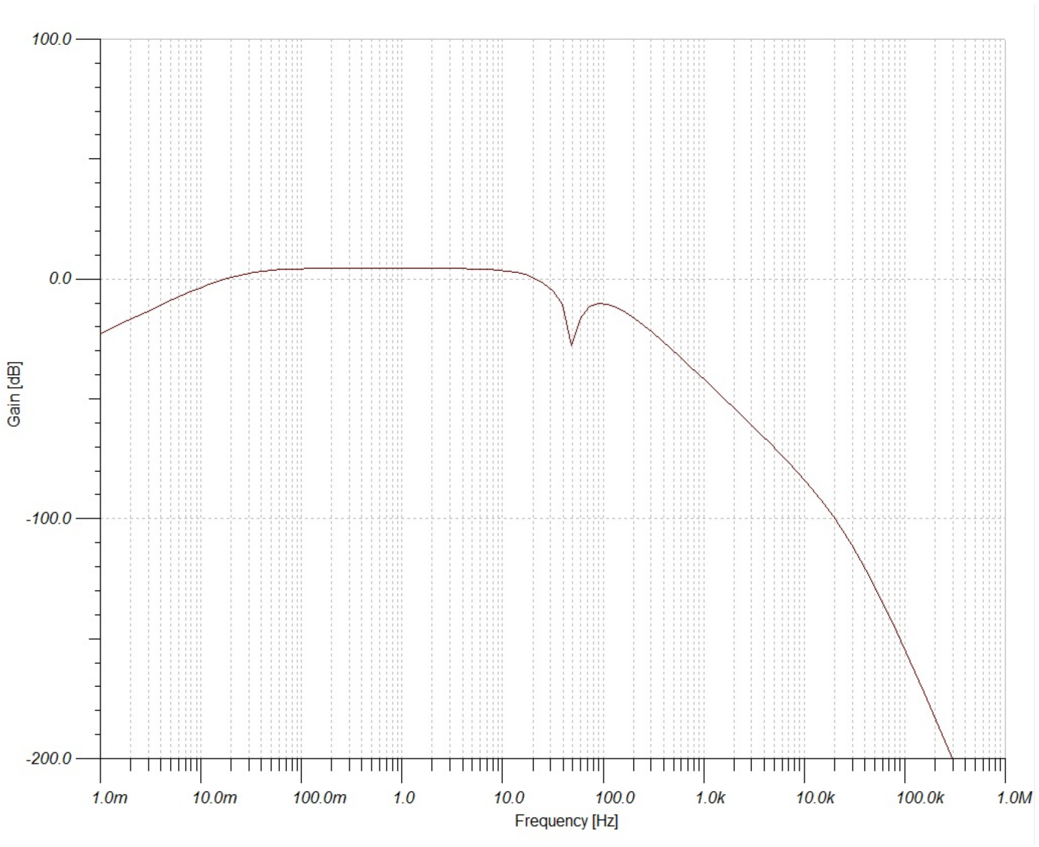


Figure 3.7: **Simulated frequency response of the ECG filter circuit**

The signal then goes through a voltage follower, to eliminate any loading effects from the DAQ. Schematic of the final ECG circuit can be seen in Appendix B.1

Isolation Circuit

The ECG circuit runs on 9V batteries. If currents from these runs to the patient, this will not be lethal. An isolation barrier was therefore left out of the design. During the work of this thesis, the ECG was connected to the FPGA development board through the ADC. The development board was connected to a power socket of 220V. This voltage will be dangerous for the patient. It did, however, not seem necessary to include the isolation barrier for testing purposes only.

3.1.2 PCB realization

The PCB for the ECG circuit is a two-layer card produced at the ELAB at the University of Oslo. It is produced as a prototype issue. The schematics and routing is done in the Cadstar PCB design tool. Where available, surface mounted components (SMD) were used. The AD624AZ was only available as a dual in-line package (DIP). To keep the signal lines in the top layer of the PCB, the AD624AZ was mounted from the bottom layer leaving its legs available for connection in the top layer. The print for the realized PCB of the ECG Front-End is found in Appendix **??**

The finished PCB is seen in figures 3.8 and 3.9

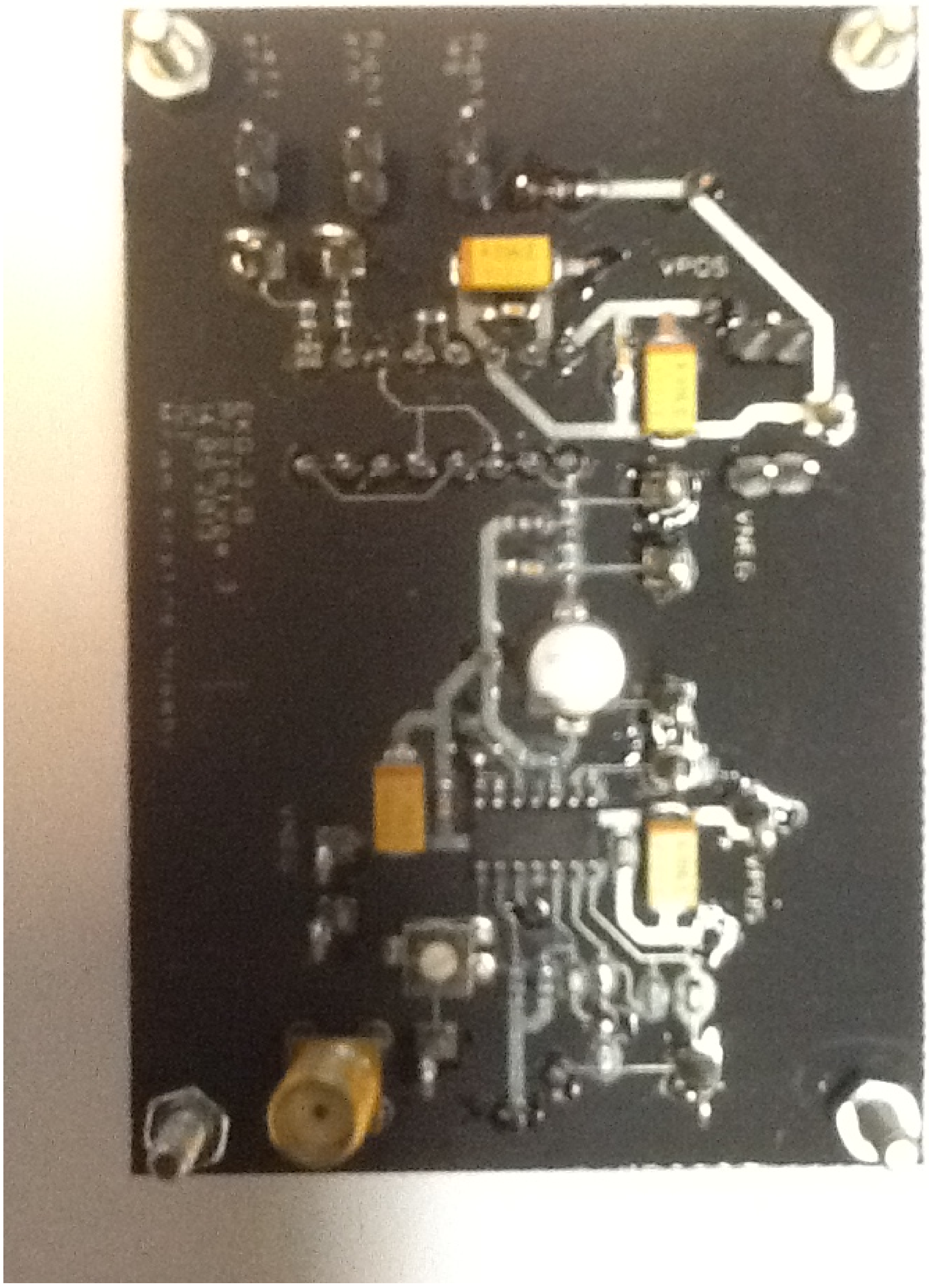


Figure 3.8: **Top layer of ECG PCB**

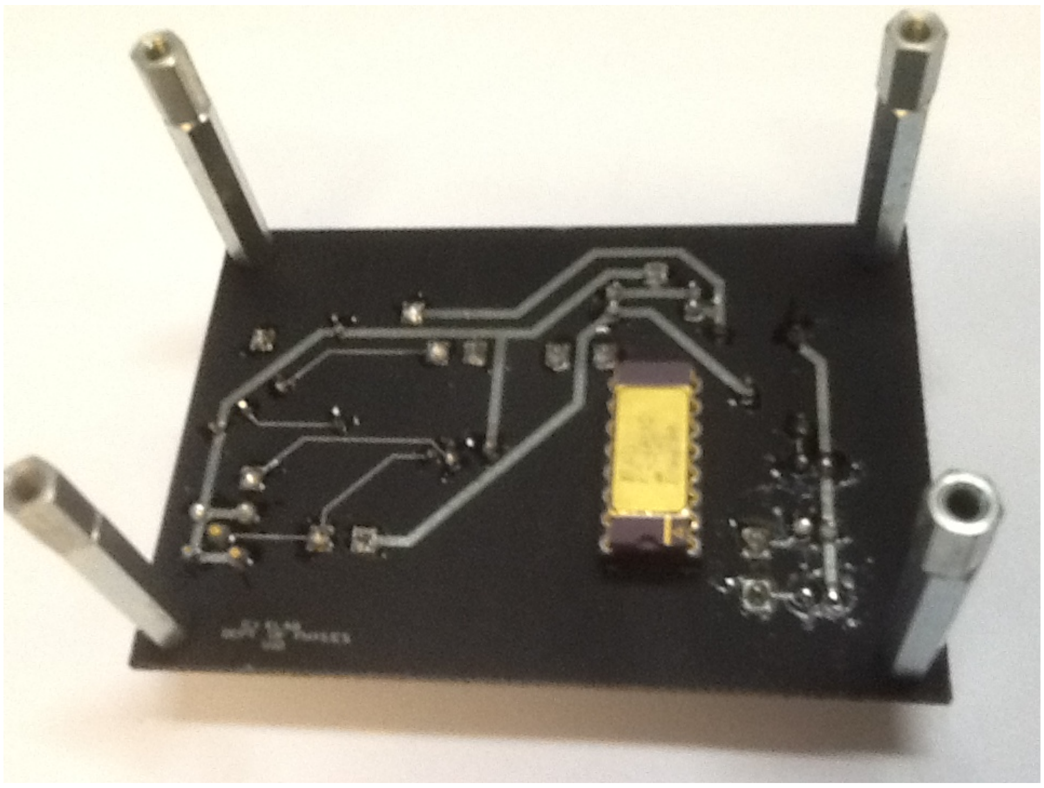


Figure 3.9: **Bottom layer of ECG PCB**

Power Distribution

The circuit's power supply was made up of two 9V batteries. They are connected in series opposing, creating a positive and a negative power supply and common ground. This meets the power supply requirements of the two amplifiers used in the circuit. Ground is referred to the reference level that the components define as 0V. All components should have the same reference level.

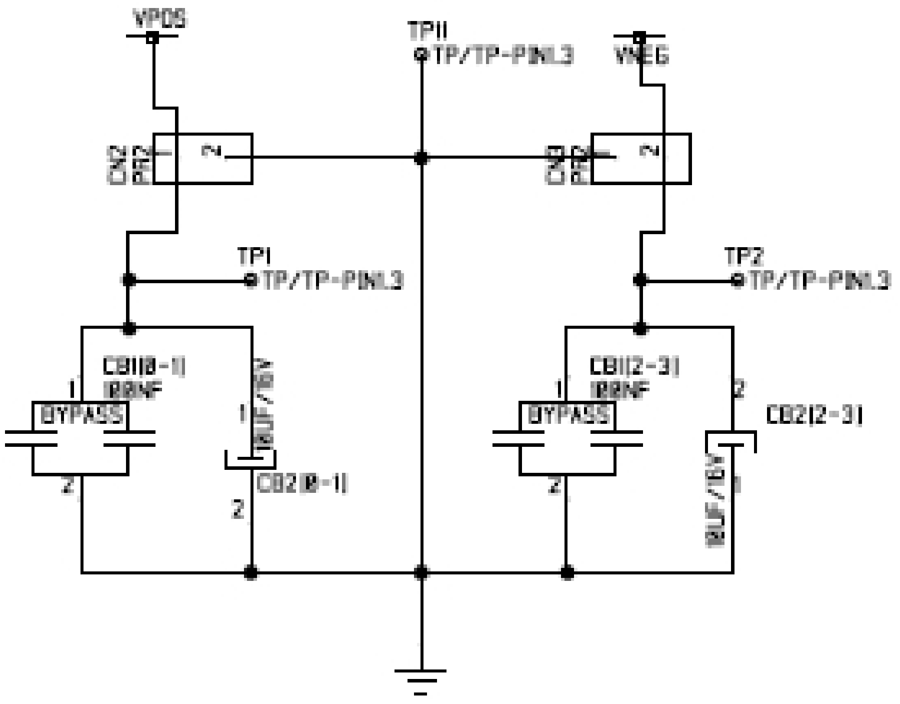
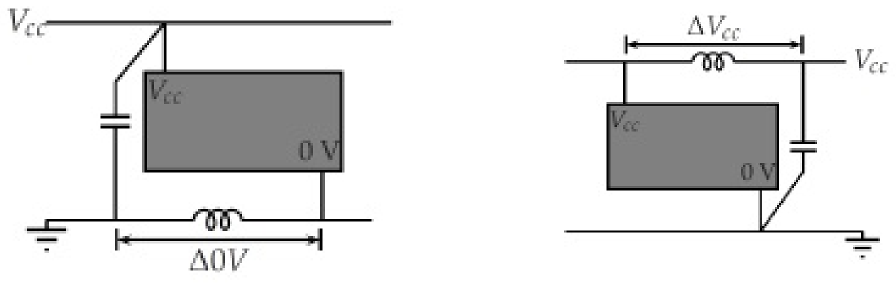


Figure 3.10: **Power Supply for the ECG Front-End**

The power and ground signals are mainly distributed on the bottom layer of the PCB. To keep the ground signal as clean as possible in a PCB, the use of a ground plane is ideal. With a two-layer PCB, however, this is not possible. The ground and power supply distribution is done by separate lines to each part of the circuit. Difference in length of the power and ground lines, may increase electrical noise from the returning currents.

Bypass capacitors

To reduce noise and current transients from the power supply, bypass capacitors are used. To protect against transient currents, high frequency bypassing is used. For this purpose a ceramic capacitor is best, due



(a) Wrong placement (b) Correct placement

Figure 3.11: **IC placement as described in Grødal (1997).** In (a) voltage variation will occur on ground. In (b) the noise of the transient current will occur in V_{CC}, and ground will stay clean.

to its good high frequency characteristics. To stabilize low frequent voltage variations from the power supply, low frequency bypassing is used. For this purpose, electrolytes, often tantals, are used due to their good low frequency performance. The two are placed in parallel. For best performance, the capacitors are placed as close to the IC as possible, preserving a stable voltage supply and ground to the component.

Due to inductance in every conductor, a voltage difference in the conductor will occur in case of transient currents. The conductors must therefore be as short as possible. To achieve this, the ideal placement of the bypass capacitors is to route the power and ground distribution in a manner such that power and ground connections enter the pad on the bypass capacitor on the one side, and exits on the other. This routing ensures that no current can go around the capacitor. (Grødal 1997). An example of bypassing is seen in figure 3.11

3.2 Data Acquisition Card

The ECG signal requires one ADC to convert the analogue signal into digital, for processing by the FPGA. A Data Acquisition Card (DAQ) was made with the possibility for the ECG module to be added to work from Aamodt (2013). The DAQ was therefore made to fulfill the requirements of the complete platform.

3.2.1 Circuit Design

The DAQ consists of 4 ADCs and 1 DAC programmed from the Altera FPGA. The card is connected to the FPGA's HSMC connector through an Altera Terasic THDB-HTG daughter card, which also provides power supplies of 3.3V, 5V, and 12V.

Analog to Digital Converter

For Analog-to-Digital conversion a high performance, 24-bit oversampled successive approximation (SAR) ADC (AD7766-2) is used. The output of the oversampled SAR is filtered using a linear-phase FIR filter. The fully filtered data is output in a 24-bit serial format with twos compliment, using MSB.

The internal circuitry of the AD776-2 is run by the SCLK. The MCLK sets the ADC sample rate. The MCLK can only be a fraction of the SCLK, due to the successive-approximation algorithm. The maximum frequency of the MCLK is 1.024MHz By employing oversampling, the quantization noise of the converter is spread across the bandwidth of 0 to f_{MCLK} .

The AD7766-2 has an output decimation rate of 32. The high rate increases the noise performance while decreasing the usable input bandwidth. The MCLK was set to 31.25kHz, with the decimation rate of AD7766-2 at 32, this gives an output data rate of 976Hz. The SCLK driving the internal circuit was set to 3.125MHz. By setting SCLK as a multiple of the MCLK, the rising clock edge won't skew (5.3 on page 55). The output data rate scales linearly with the MCLK frequency.

The AD7766-2 has excellent performance at ultralow power. Both digital and analog current consumption scales with the MCLK frequency applied to the device. The actual throughput equals the MCLK frequency divided by the decimation rate, seen in figure 3.12

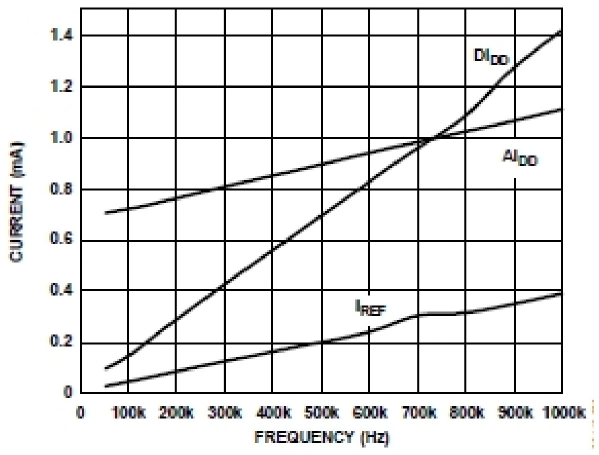


Figure 3.12: **AD7766-2 Current vs MCLK frequency**
The current throughput equals the MCLK frequency divided by the decimation rate of 32

The analog input is of differential structure. This provides rejections of signals common to both V_{in+} and V_{in-}. The signal to the ADC comes from a single-ended source. The single-ended-to-differential driver ADA4941-1 creates a fully differential input to the AD7766-2. It is a low power, low noise differential driver for ADCs in systems that are sensitive to power. The choice of the driver was based on the suggestion from the AD7766-2 datasheet, as shown in figure 3.13.

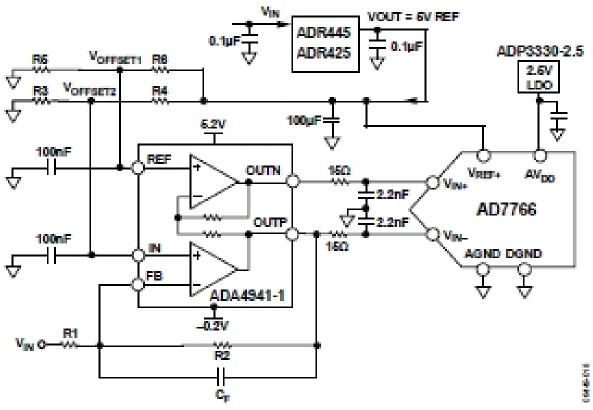


Figure 3.13: **Schematic for driving the AD7766-2 from a single-ended source**
R1, R2 and C_F set the attenuation ratio between the input range and the ADC range (V_{ref}). R3 and R4 set the common mode on the V_{in-} input, and R5 and R6 set the common mode for the V_{in+} input of the ADC.

R1 and C_F are set to work as an anti-aliasing filter before the ADC.

When digitized at a sampling rate of f_s, the original analog signal is replicated throughout the spectrum at intervals of f_s. Because of the replication, there may occur corruption of the frequency components of the original signal by components of the replicated signal. This is known as *aliasing*, and the corrupted frequencies are *alias* frequencies.

To capture all of the important information in the original signal, as well as eliminate the effect of aliasing, the sampling frequency must be at least twice as high as the highest frequency in the original signal. (Thede 2004). This relationship stems from the Nyquist-Shannon sampling theorem, that states: *If a function $x(t)$ contains no frequencies higher than B Hz, it is completely determined by giving its ordinates at a series of points spaced $1/(2B)$ seconds apart.* (Wikipedia 2001). From this follows function (3.3)

$$f_s > 2 \cdot f_h \tag{3.3}$$

A representation of analog and digital signals is illustrated in figure 3.14

On the DAQ, the values of the components in the anti-aliasing lowpass filter are 1 kΩ and 1.5 μF respectively. This gives an f_c of 106 Hz with an attenuation of 20 dB pr decade. In addition, the on-board digital filter provides a 38 dB attenuation for any possible aliasing frequency in the range from the filter stop band to where the image of the digital filter pass

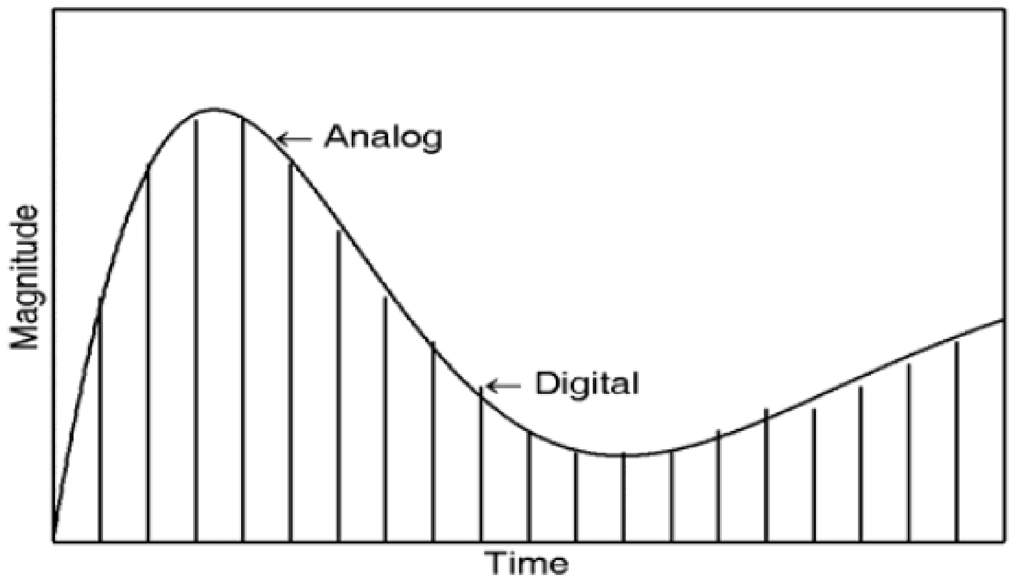


Figure 3.14: **Comparison of Analog and Digital Signals**
The digital replication of the analog signal increases accuracy with the sampling frequency

band occurs. With a sampling rate of 31.25 kHz the anti-aliasing protection is in the frequency range of 534 Hz to 30.7 kHz. This can be seen in figure 3.15

The signal is sampled as 24 bits. To fulfill the Nyquist-Shannon theorem for anti-aliasing, the signal must have an attenuation of $\approx$ 144 dB at $\frac{f_s}{2}$ = 15.625 kHz. The lowpass filter in the ECG ciruit, as shown in 3.7 on page 29 has an attenuation of $\approx$ 90 dB at 15.625 kHz. This, with anti-aliasing filter on the DAQ and protection of the digital filter, give an attenuation well within the Nyquist frequency.

Power supply to the ADC originates at the 12V, 5V, and 3.3V pins on the Terasic THDB-HTG connector, connected to the Cyclone Development kit. From there the voltage is regulated to ensure the correct supply to the ADC. The voltage applied to the reference input operates both as a reference supply and as the power supply for the AD7766-2. As a nominal reference voltage a 5V reference input was chosen, as this gives the full-scale differential input range of 10V. To ensure a 5V power supply, the ADR445 XFET voltage reference was used. It was chosen because of its features of ultralow noise, high accuracy, and low temperature drift. The analog voltage of 2.5V supplies the AV_{DD} pin on the AD7766-2. The voltage is ensured by a low dropout regulator, ADP3330. The regulator achieves high accuracy at room temperature. It also include safe current limit and a shutdown feature.

During initial power-up, power-down, synchronization and reset, a

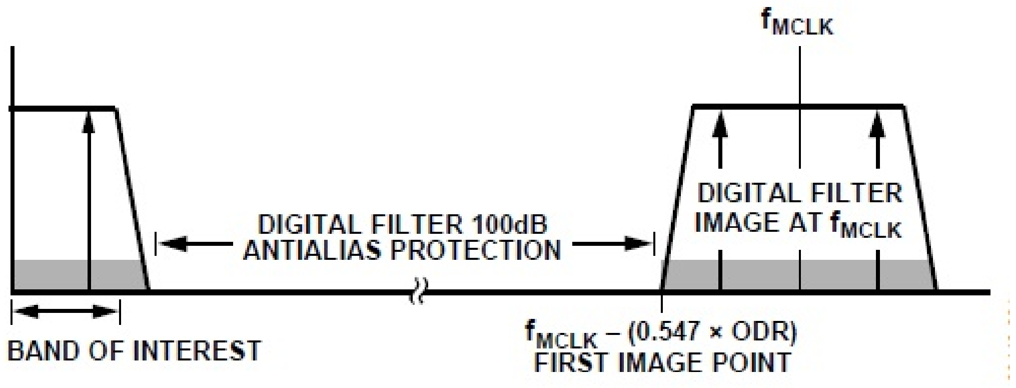

fMCLK
DIGITAL FILTER 100dB ANTIALIAS PROTECTION
DIGITAL FILTER IMAGE AT fMCLK
BAND OF INTEREST
fMCLK – (0.547 × ODR) FIRST IMAGE POINT

Figure 3.15: **Frequency Spectrum for Anti-aliasing protection in the AD7766**

With a first-order anti-aliasing filter, the digital filter in the AD7766 provides a 38 dB attenuation in the frequency spectrum of (0.547 × ODR) to (f_{mclk} - 0.547 × ODR). ODR is the Output Data Rate.

settling time $t_{settling}$, from the filter reset must elapse before valid data is output by the device. The timing diagram for this is shown in figure 3.16

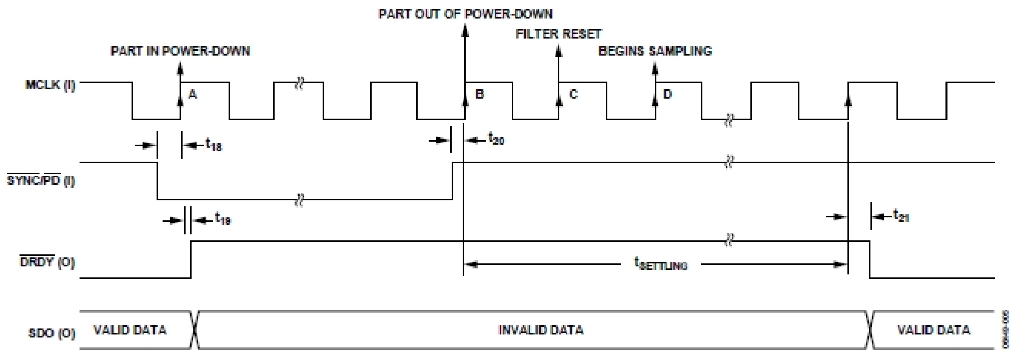


Figure 3.16: **Timing diagram for reset, synchronization, power-up, and power-down of the AD7766-2**

After the filter has settled, data is read based on the timing diagram in figure 3.17

Digital to Analog Converter

For Digital-to-Analog Conversion, the AD5340 by Analog Devices was used. It is a single resistor-string CMOS DAC of 12 bits.

The voltage at the V_{ref} is set to 5V by an ADR445 to provide the reference voltage for the DAC. The input coding to the DAC is straight binary, and the ideal output voltage is given by (3.4)

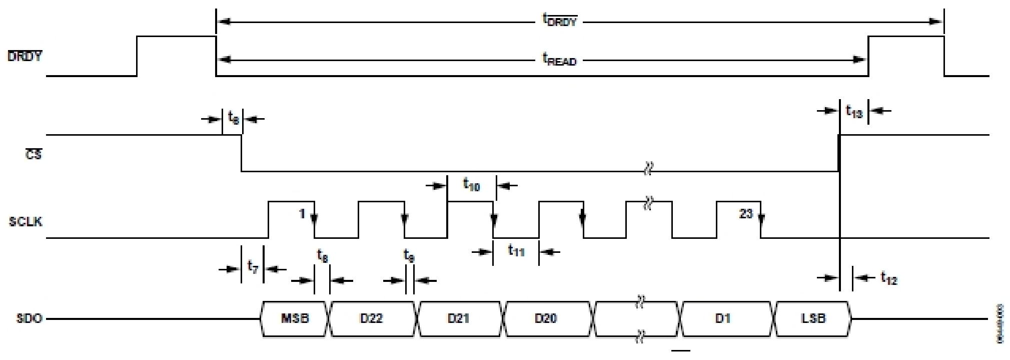


Figure 3.17: **Serial timing diagram for reading using $\overline{CS}$**

$$V_{out} = V_{ref} \times \frac{D}{2^N} \times Gain \tag{3.4}$$

where D is the decimal equivalent of the binary code, which is loaded to the DAC register 0 to 4095 (12 bit). N is the DAC resolution. Gain is the output amplifier gain, set to 1 or 2.

The low power consumption of the AD5340 can be reduced even further by setting $\overline{PD}$ low. The output stage is then internally switched from the output of the amplifier, making it open-circuit.

The schematics of the complete Data Acquisition Card is found in Appendix **??**.

3.2.2 PCB realization

The DAQ PCB was drawn and routed in Cadstar PCD Design tool. It was designed as a two-layer PCB to be produced at the ELAB at the University of Oslo. The last version of the PCB was, however, produced outhouse. SMD components were mounted on the top of the card. Connectors for connection to the FPGA were mounted on the back for practical purposes. Schematics of the circuit is found in Appendix B.3. PCB print is found in Appendix B.4.

Power and Ground Distribution

Power is supplied to the DAQ from the Altera Terasic THDB-HTG connected to the Cyclone III development kit. It offers 3.3V, 5V, and 12V from separate pins. The HMSC connector also offer 3.3V, 5V and ground signal from the pin connector. The power is regulated through voltage references and regulators as described in section 3.2.1. The power tracks were placed on the bottom layer of the PCB, to reduce interference to the signal tracks in the top layer.

Ground is referred to as the reference level the components in the system defines as 0V. It is therefore important to have the same ground potential throughout the circuit. The return current takes the path of the least inductance to common ground. To reduce any noise the return current may affect on the surrounding components, it would have been preferable to have a separate ground plane on the PCB. However, since the DAQ was designed as a two-layered PCB, this was not possible. The bottom layer was filled to be as fully a ground plane as the design allowed, making the ground potential the same for all components.

Bypass Capacitors

See section 3.1.2

Chapter 4

MATLAB and Modelsim Simulation

4.1 MATLAB Simulation

Before implementing the algorithm into the FPGA, it was simulated in MATLAB. To be able to test on a realistic signal, an ECG generating program by McSharry & Clifford (2003) was downloaded from physionet.org. This program generates an ECG signal and allows for adjustments of parameters like noise, number of heart beats, and sampling rate to make a more realistic signal to work with. The program is otherwise randomized, making each simulation slightly different from the other. The program proved to be a valuable tool to verify the Pan-Tompkins algorithm in J.Pan (1985). The program can be found in Appendix C.1.

A function was made for each step in the Pan-Tompkins algorithm. The code for the function are in Appendix C.2. The whole program was run, giving the outputs seen in figures 4.1 - 4.8

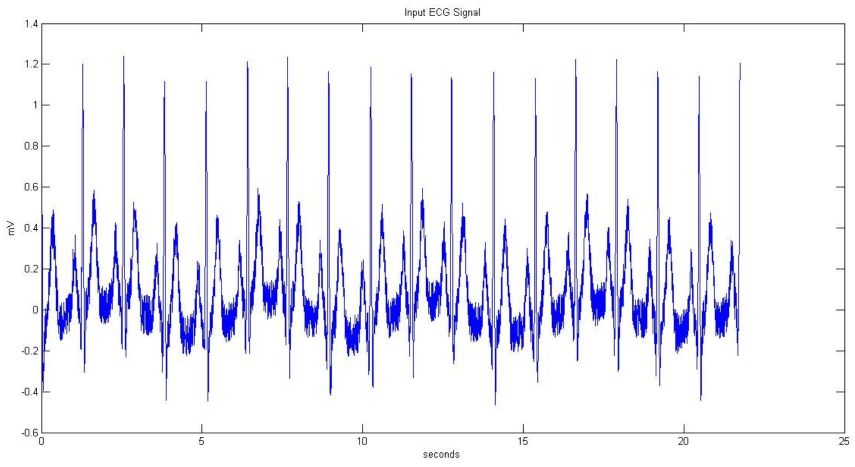


Figure 4.1: **Output from MATLAB generated ECG signal**
ECG signal generated from MATLAB code from McSharry et al. 2003.
The approximate number of heart beats was set to 10. Noise is 0.1mV
Other parameters were set to default.

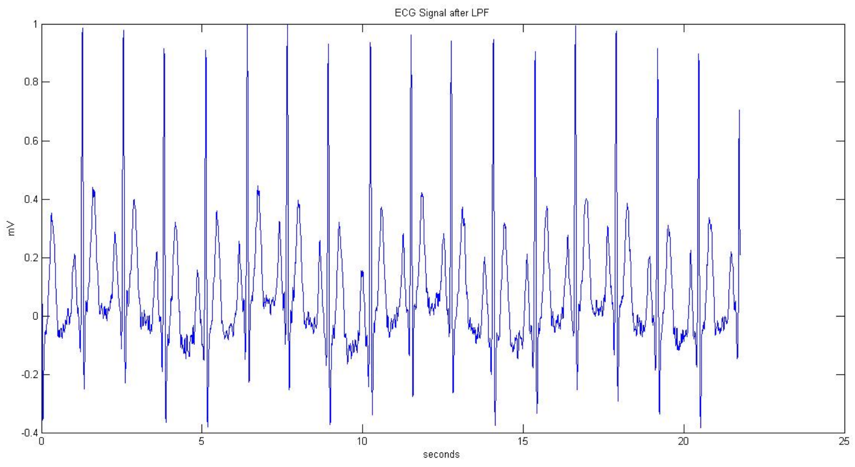


Figure 4.2: **Simulated ECG signal after Lowpass Filter**
Output of the lowpass filter. Some noise reduction is achieved.

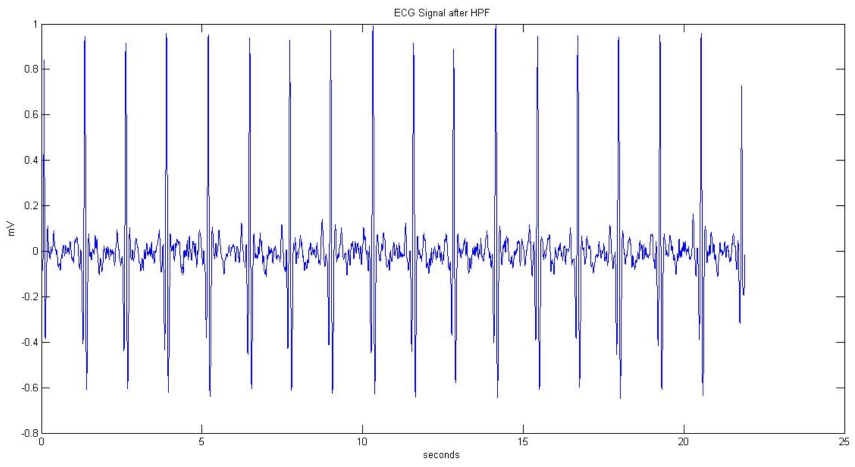


Figure 4.3: **Simulated ECG signal after Highpass Filter**
Output of the highpass filter. Further noise reduction is achieved, and DC offset is removed.

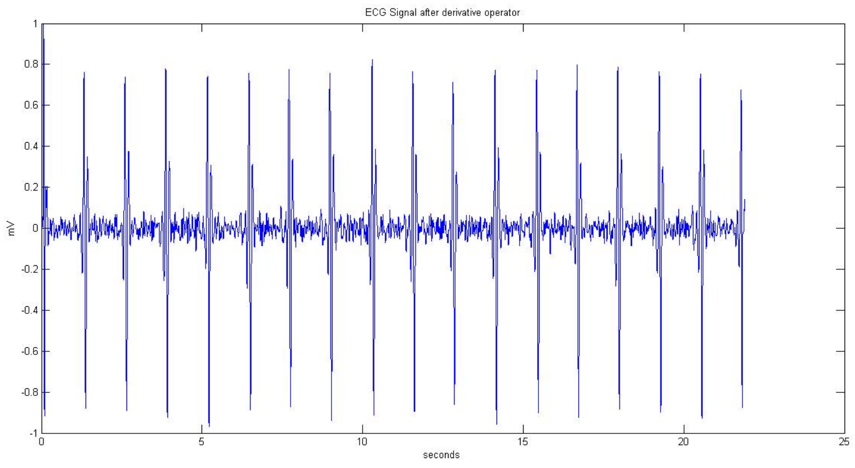


Figure 4.4: **Simulated ECG signal after Derivative**
Output of differentiator. The signal provides the QRS-complex slope information.

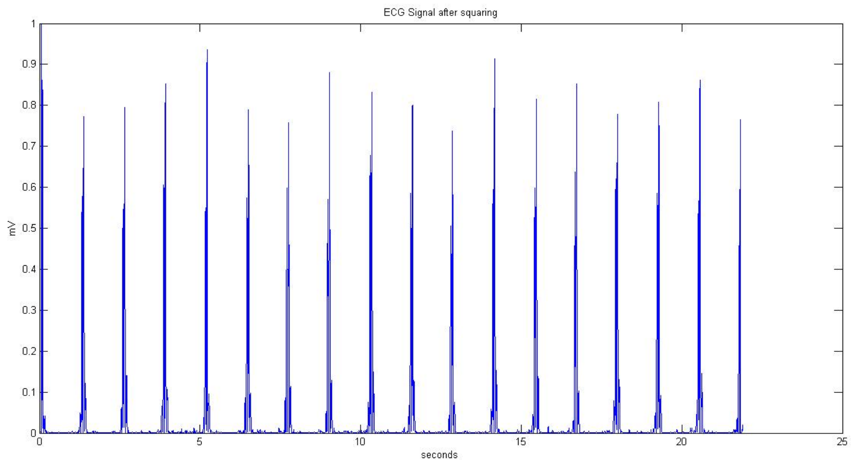


Figure 4.5: **Simulated ECG signal after Squaring Function**
Output of the squaring process. A nonlinear amplification of the derivative signal, emphasizing the higher frequencies. All data becomes positive

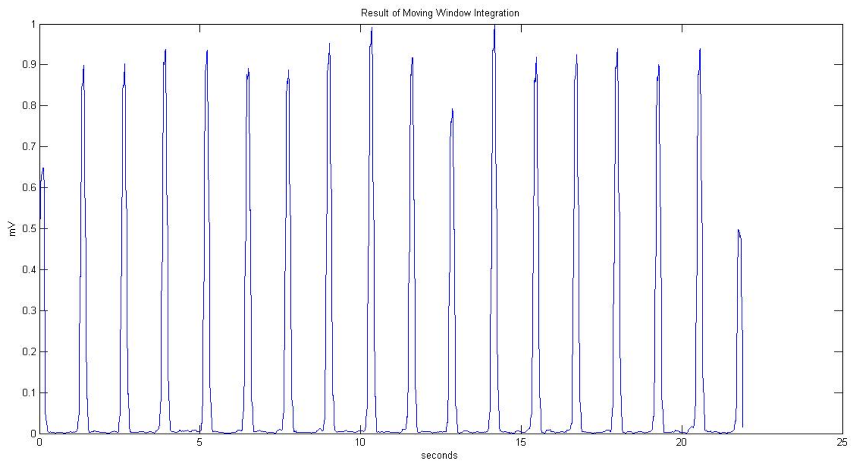


Figure 4.6: **Simulated ECG signal after Moving Window Integration**
Result of the moving window integration. Provides waveform feature information and the slope of the R wave.

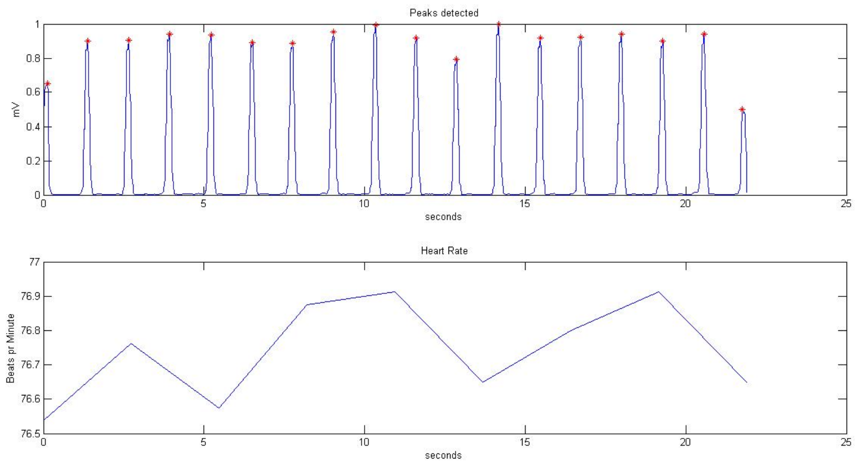


Figure 4.7: **Peak detection and calculated heart rate of generated ECG signal**
The top figure shows the peaks of the integrated signal. These are acknowledged as the peaks of the QRS-complex. The bottom figure shows the heart rate, calculated from the averaged RR interval from the eight most-recent beats.

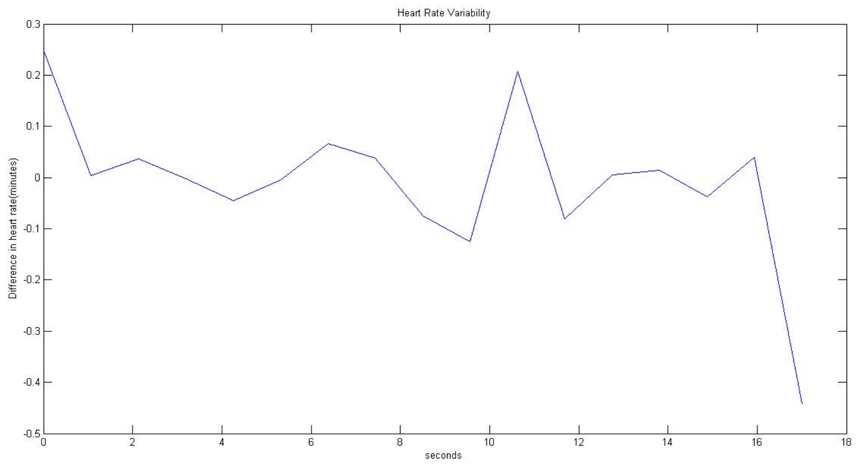


Figure 4.8: **Simulated Heart Rate Variability from generated ECG signal**
The heart rate variability calculated as the difference in heart rate from one heart beat to the next.

4.2 Modelsim Simulation

Each of the logic blocks were written in VHDL. Each block was tested in Modelsim to verify that the output was consistent with the arithmetic for each difference equation. For practical purposes, the size of the signals tested were set to 4 or 8 bits. This was changed during implementation to fit the requirements of the design. To generate results in Modelsim, testbenches were created. The testbench simulates input data to the logic block. The same testbench was used for all but one of the functions. For the peak detection, a separate on was made for simulation of a more realistic signal. The VHDL code for the testbenches are found in Appendix D.1. For practical purposes not all the input stimuli is included in the code file, since that will take up several pages in the appendix. The results from the simulations is seen in figures 4.9 to 4.14

The remaining calculations after peak detection were not tested in Modelsim. They involve floating-point arithmetic, and floating-point in VHDL requires more resources in the FPGA. This is further explained in section 5.1 on page 54.

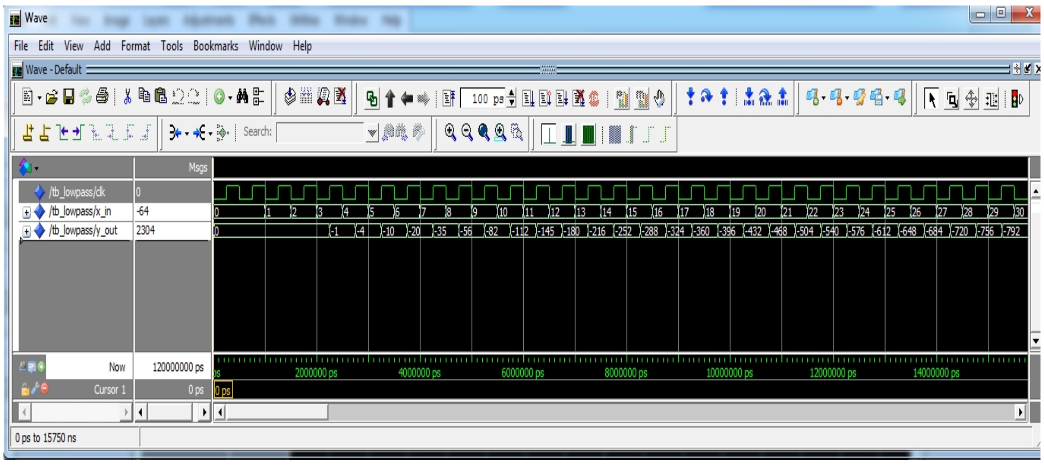


Figure 4.9: **Modelsim results for lowpass filter**
x_in sends in a signal counting up. y_out outputs the result from the difference equation for the lowpass filter

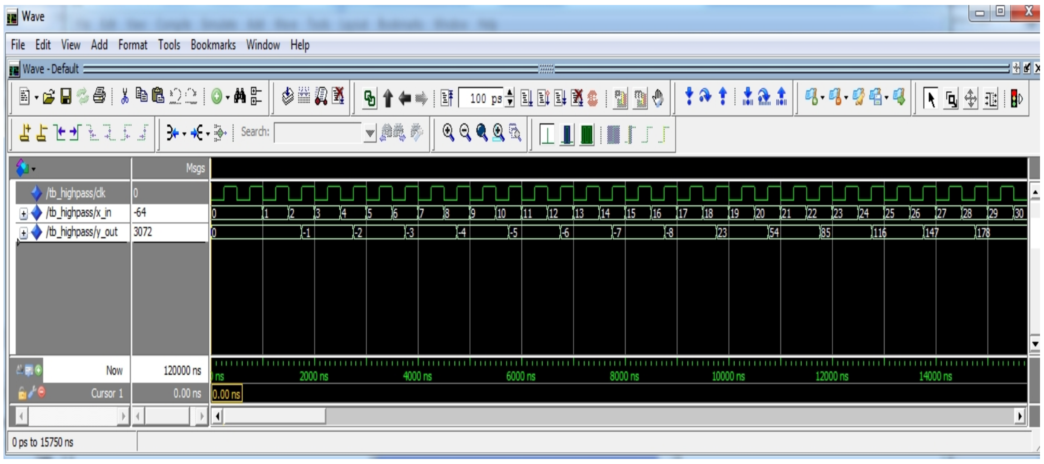


Figure 4.10: **Modelsim results for highpass filter**
x_in sends in a signal counting up. y_out outputs the result from the difference equation for the highpass filter

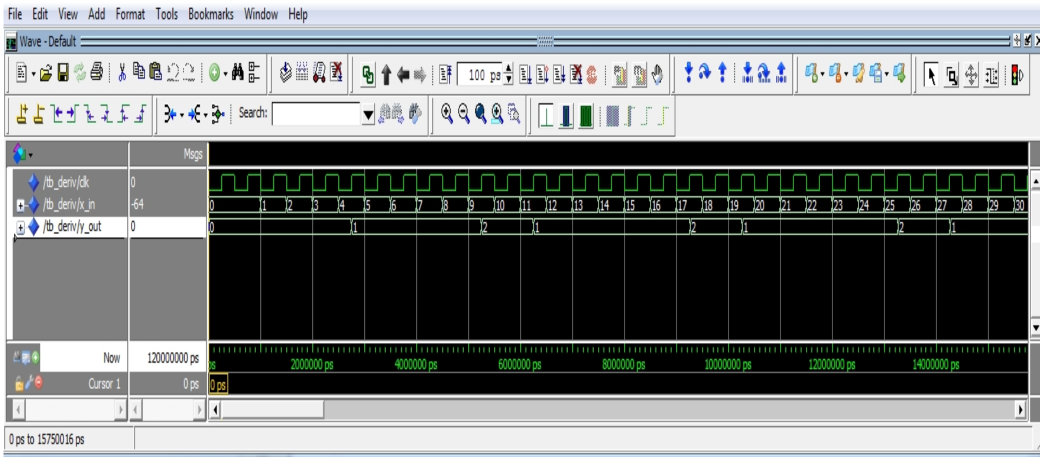


Figure 4.11: **Modelsim results for derivation**
x_in sends in a signal counting up. y_out outputs the result from the difference equation for the derivation

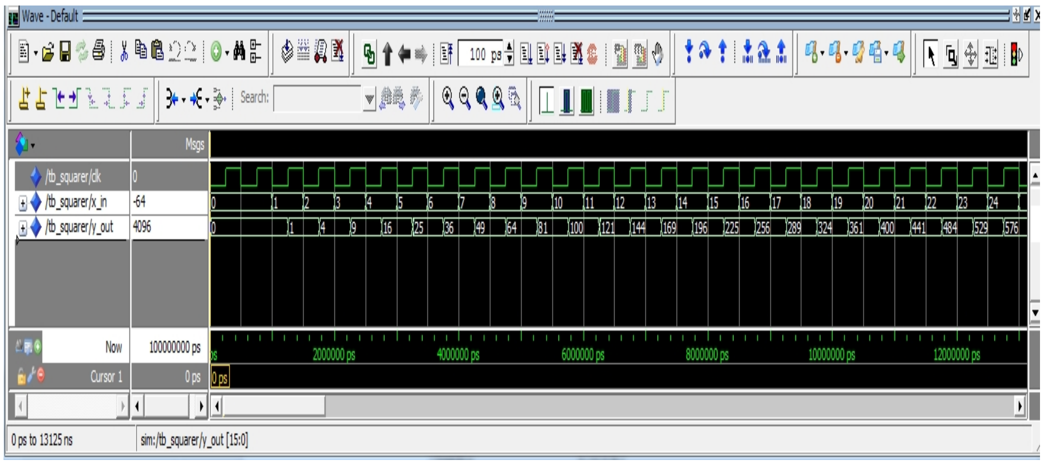


Figure 4.12: **Modelsim results for squaring process**
x_in sends in a signal counting up. y_out outputs the result from the difference equation for the squaring process

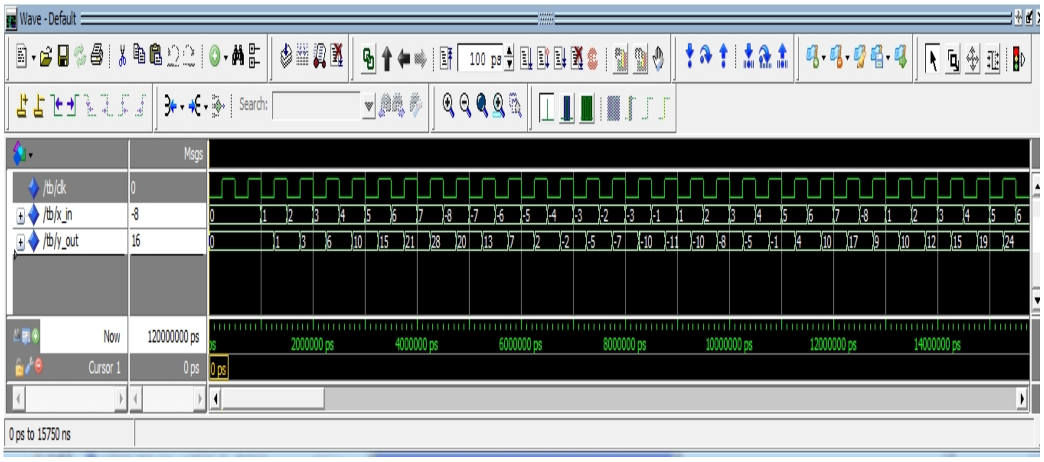


Figure 4.13: **Modelsim results for moving window integration**
x_in sends in a signal counting up. y_out outputs the result from the difference equation for the moving window integrator

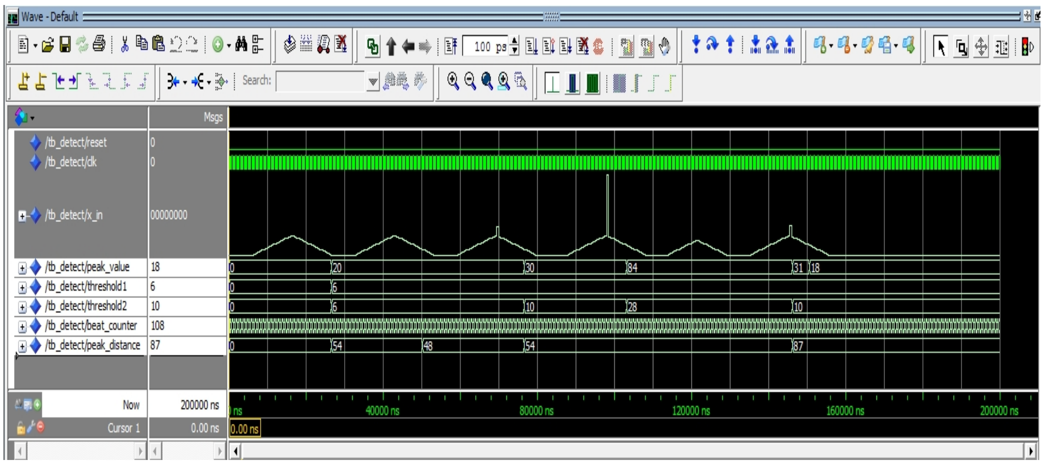


Figure 4.14: **Modelsim results for peak detection**
x_in sends in a signal constructed as a wave. Threshold 1 displays the first threshold. Threhsold2 is the adaptive threshold. Peak_value is the the peak value. Peak_distance is the number of samples between two consecutive peaks.

Chapter 5

Hardware Implementation

The microcomputer platform chosen for digital signal processing of the ECG signal is an Field-Programmable Gate Array (FPGA). FPGAs are reprogrammable chips comprised of a two-dimensional array of logical cells and programmable routing resources, conceptually visualized in figure 5.1. The logic cells can be programmed to perform a function, and a programmable switch can be customized to provide connections among the cells. (Chu 2008). The custom design is written in software, synthesized, compiled, and then programmed on to the FPGA. The reprogrammable aspect of the FPGA makes it a great choice when developing new technology in a fast and safe manner. The FPGA holds the same logic as an ASIC. A fully tested and verified FPGA can therefore be transferred to an ASIC for large scale production purposes.

5.1 Altera Cyclone III FPGA

The FPGA used for the development of the development platform is an Altera Cyclone III EP3C120F780C7 FPGA. This is placed on a DSP Development Kit, Cyclone III Edition by Altera Corporations. At the initiation of this thesis, this was the newest device family available. At the time, the Cyclone III offered the lowest power consumption combined with high functionality. Today, both Cyclone IV and Cyclone V device families exceeds the Cyclone III on both properties. The development kit also holds two HSMC connectors, used to connect the FPGA to the DAQ and the UART interface for the Bluetooth module. The FPGA includes the Nios II soft processor, for use in embedded systems.

The Cyclone III uses SRAM cells to store configuration data, which is downloaded to the device each time it powers up (*Cyclone III Device Handbook* 2011). The Cyclone III features are listed in table 5.1

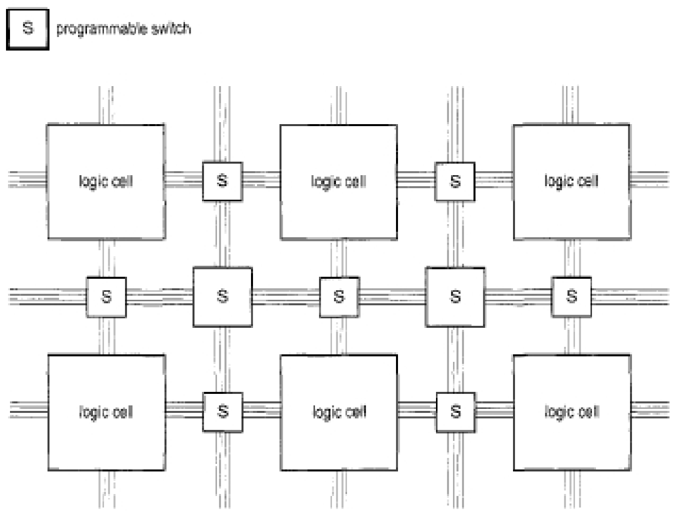


Figure 5.1: **Conceptual Structure of an FPGA Device**
The logic cells are programmed to perform functions, while the programmable switches provide customized connections between the cells. (©Chu 2008)

Table 5.1: Cyclone III Device Features

Feature	**Quantity**
Logic Elements	119 088
Memory (Kbits)	3888
Multipliers	288
PLLs	4
Global clock networks	20

Logic Elements

Logic Elements (LE) are the smallest units of logic in the FPGA architecture. They are constructed as shown in figure 5.2. The LEs are programmed using the Quartus II software. The software automatically chooses the appropriate mode for the LE, to optimize its resource usage.

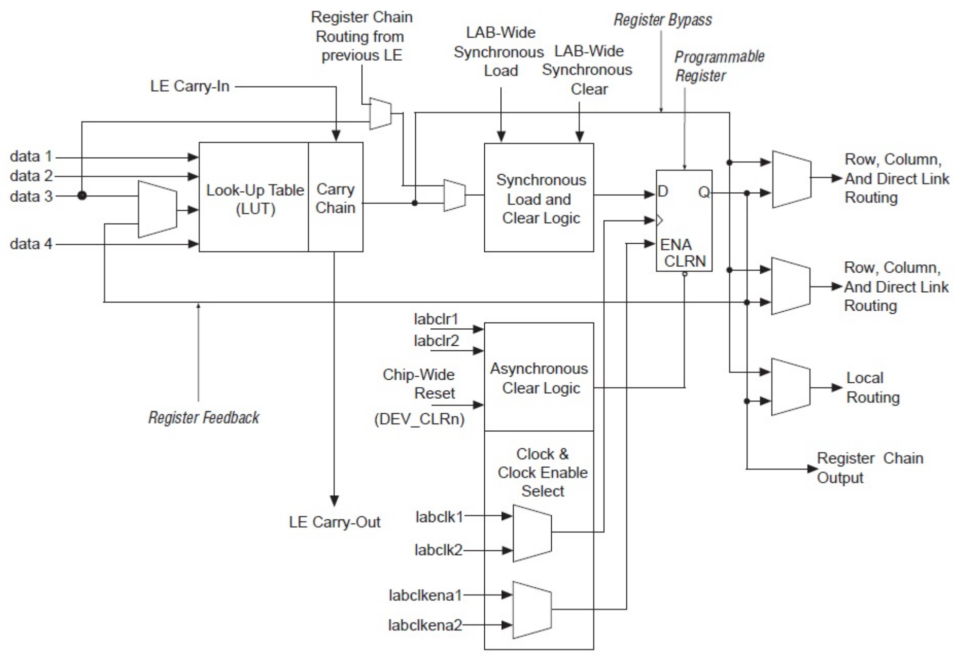

Register Chain Routing from previous LE
LAB-Wide Synchronous Load
LAB-Wide Synchronous Clear
Register Bypass
Programmable Register
LE Carry-In
data 1
data 2
data 3
data 4
Look-Up Table (LUT)
Carry Chain
Synchronous Load and Clear Logic
D
Q
ENA
CLRN
Row, Column, And Direct Link Routing
Row, Column, And Direct Link Routing
Local Routing
Register Chain Output
labclr1
labclr2
Chip-Wide Reset (DEV_CLRn)
Asynchronous Clear Logic
Clock & Clock Enable Select
Register Feedback
LE Carry-Out
labclk1
labclk2
labclkena1
labclkena2

Figure 5.2: **Logic Elements for the Cyclone III FPGA**
Logic Elements are the smallest units of logic in the FPGA. They provide advanced features with efficient logic usage (©*Cyclone III Device Handbook* 2011)

Quartus ®II Design Software

The language used to program the FPGA in this thesis is VHDL. It arose from the US government Very High Speed Integrated Circuits (VHSIC) program to became a standardized language for describing the structure and functions of ICs. The name VHDL is an acronym for VHSIC Hardware Description Language (Ashenden 2008). For the design of the FPGA ECG Module, the Quartus design software was used. This is a development environment that offers many tools for the development of complex FPGA systems. In the Quartus software, Altera offers ready-made and parameterized Intellectual Property (IP) blocks called megafunctions. The megafunctions help access to architecture-specific features within simple and complex functions and are optimized for efficient logic synthesis in the FPGA (*Introduction to Megafunction IP Cores* 2013).

For DSP system design, a DSP Builder interface tool between the Quartus II software and MathWorks' Simulink and MATLAB design environments can be used. This tool makes it possible to do arithmetic and design in either Simulink or MATLAB, and convert to the HDL environment in Quartus II (*Quartus II Handbook* 2013). For simulation and verification of the design, the Quartus has interface to Modelsim and SignalTap II Logic Analyzer.

Floating-point Numbers in VHDL

Floating-point numbers are great for precision in a wide dynamic range. They are, however, very resource demanding when used in hardware. Compared to fixed-point math, they take up almost 3 times as much hardware. In VHDL, the floating-point is defined by the IEEE-754 specifications (Bishop 2008).

The Quartus II megafunctions library offers ready-made floating-point functions for the designer to customize. Every floating-point operation has a significant cost of either hardware resource or timing. If timing is critical, more resources are used. And if hardware resources is critical, timing will suffer. Either way, a floating point operation uses more time than a fixed-point operation. The output of the floating-point module will consequently always add a delay to the signal propagation.

5.2 Nios II Processor and Bluetooth

The Nios II processor is a configurable soft IP core processor embedded in the Altera FPGA. The Nios II processor enables the possibility of software emulation of the design. As explained in Aamodt (2013) a Nios II was added to the platform. It is mainly used for interfacing the Bluetooth module for the transmitting of data to the mobile application through an Universal Asynchronous Receiver/Transmitter (UART).

The Nios II processor library holds an RS-232 UART peripheral for sending and receiving data over two external pins; *RxD* and *TxD*. These were assigned to one of the two HSMC connectors on the Cyclone III development kit. A Digilent PmodBT2 Bluetooth Interface was connected to the *TxD* for transmission of data to the mobile application. This specific interface was chosen due to its UART interface. The PmodBT2 is compatible for Bluetooth 2.1, 2.0, 1.2 and 1.1(Digilent 2004).

(*Nios II Prosessor Reference Handbook* 2011)

5.3 Clocks and Reset

In sequential logic circuits, clocks are important to "drive" the signal through it. Sequential circuits are defined as those that maintain internal state. The output produced depends on the history of inputs received previously. They are known as *synchronous* or *clocked* circuits, and "they use a rising or falling edge of a clock or a level of an enable signal, to control advance of state or storage of data." (Ashenden 2008)

The Cyclone III Development Board has two crystal oscillators driving reference clocks to the FPGA. Their frequencies are 125 MHz and 50 MHz. The 50 MHz reference clock has lower power consumption than the 125 MHz one. In the ECG Module, the 50 MHz reference clock was chosen mainly due to fit the chosen clock design by Aamodt (2013).

In a multiclock system, it is important that all the clocks are aligned on either the rising or falling edge, depending on the trigger. Clocks that are not aligned can introduce a skew effect in the circuit, which again can produce a metastable state. "A *metastable state* is created when the flip-flop's timing requirements are violated. The resulting output of the flip-flop is unknown, and can make the entire design nondeterministic" (Behne 2003).

5.3.1 Phase-Locked Loop

The reference clock of 50 MHz is much faster than is applicable for the logic in the ECG Module. It is therefore necessary to reduce the frequency to a manageable ones. The Quartus software has a megafunction for phase-locked loops (PLL) for clock management. When deciding the frequencies on the output of the PLL, the requirements for the whole design was taken into consideration. The system propagates on the rising edge of a clock. This means that all clocks need to align on a rising edge to prevent skew.

From section 3.2 on page 34 the ADC has a maximum MCLK of 1.024 MHz. The PLL was set up with two output clocks; SCLK and MCLK. SCLK is the clock driving the ADC, and MCLK is the sample frequency for the ADC. The division factor in the PLL was set to 16 and 1600, respectively. This gives

$$\text{SCLK} = 3.125\ \text{MHz}$$

$$\text{MCLK} = 31.25\ \text{kHz}$$

5.3.2 Clock Division

The sequential logic in the implementation of the Pan-Tompkins algorithm in 5.5 require that a clock set their sampling frequency of 200 Hz.

However, as the PLL was not able to divide the reference clock to the required frequency, a clock divider was implemented. The clock divider is referenced as clock160div in the ECG Module. It works as a small counter. It uses MCLK as the clock reference. The counter counts to the division factor, here 156. At every half of this the output is set either high or low, creating a clock output of 200 Hz. VHDL code for the clock160div is found in Appendix D.2

5.3.3 Reset

An asynchronous reset was implemented in the design for use in the ADC and the clock divider. It was assigned to a switch on the Cyclone III development kit. The logic with reset implemented is programmed to reset when the switch is set. The sequential logic of the ECG Module does not need a reset, as the signal is meant to just clock through the different logic. It is therefore not implemented in them.

The complete clock module for the ECG design is found in figure 5.3.

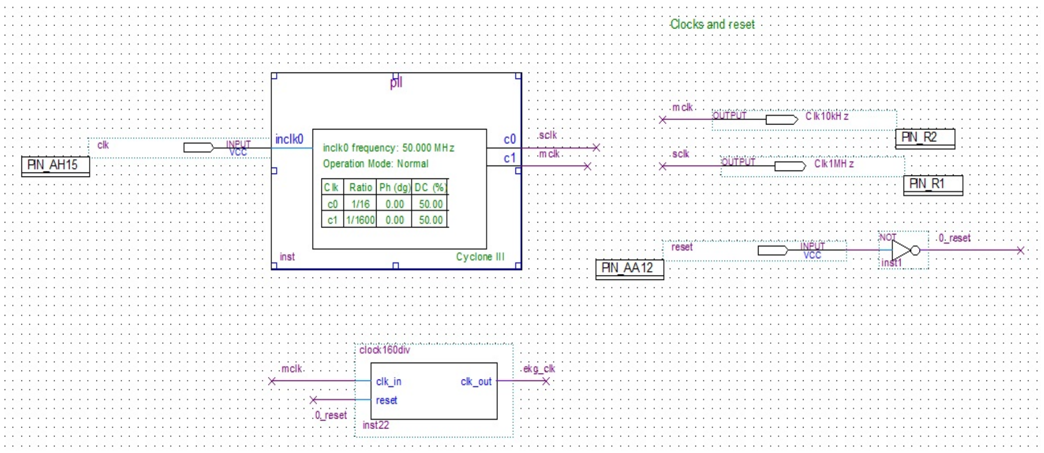


Figure 5.3: **Clock system for the ECG design**
Reference clock from the development kit is 50 MHz. It goes to the PLL, where it is divided into SCLK and MCLK. The clock160div uses the MCLK to divide to ECG_clk of 200 Hz. Reset is active high.

The clock signals were tested in the SignalTap II Logic Analyzer. It was set to trigger on the ekg_clk, so the signals could be best observed. As shown in figure 5.4, the clocks aligned on the triggered rising edge.

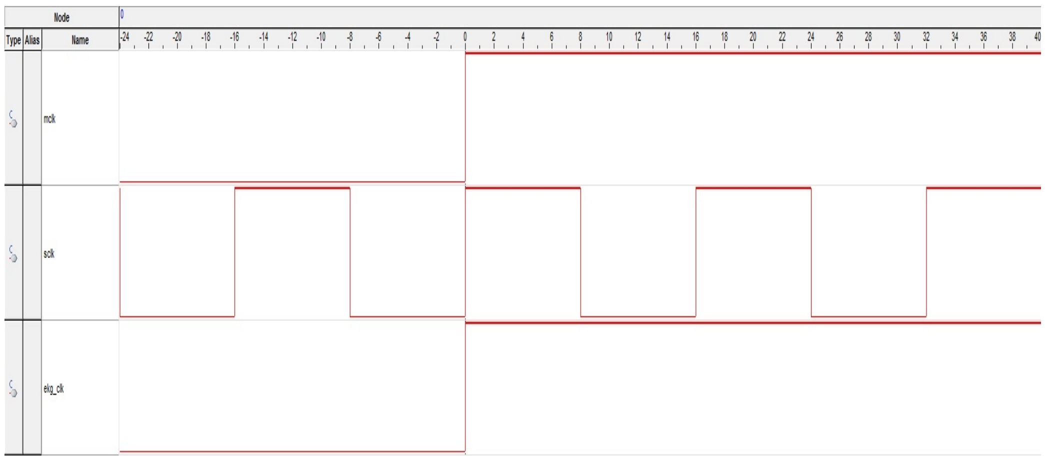


Figure 5.4: **Clock signal alignment in the clock module**
The clocks used in the clock module, have no skew on the rising edge trigger

5.4 ADC

The ADC module was built as a Moore Finite State Machine (FSM). The Moore FSM architecture means that "the outputs are solely a function of the present state" (Zwolinski 2004). The state machine was programmed according to the timing diagrams in figures 3.16 on page 38 and 3.17 on page 39. Figure 5.5 shows the flow chart for the FSM for the AD7766, as generated by Quartus. A drawing of the ADC block in the design is found in figure 5.6

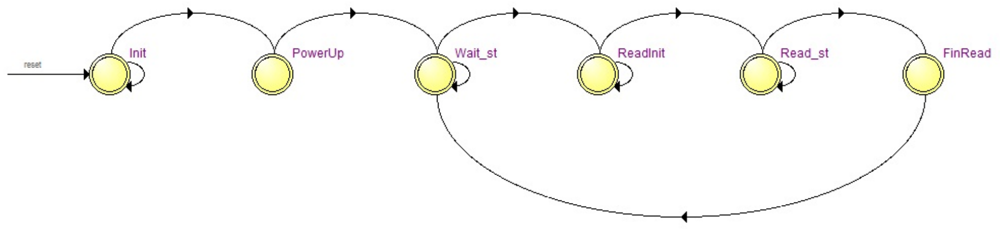


Figure 5.5: **Flow Chart for FSM for AD7766**
A flow chart of the states in the Finite State Machine as the signal is read in the AD7766

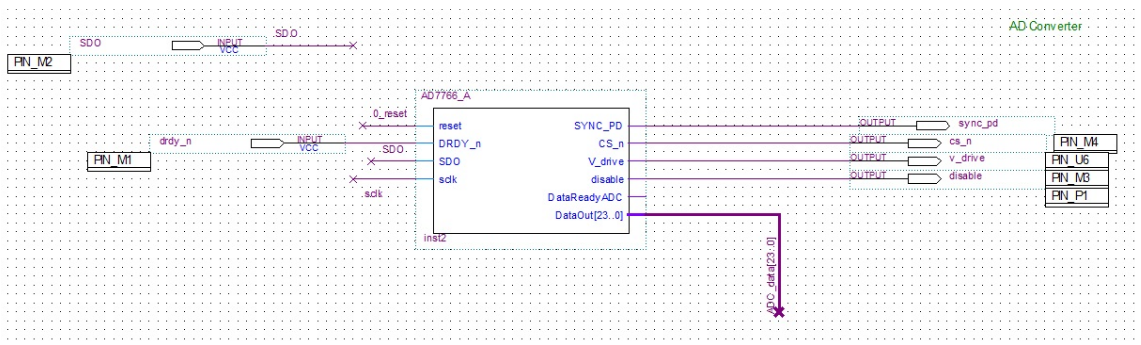


Figure 5.6: **Drawing of the ADC block in the ECG design**
Signals and pin assignments for the ADC in the ECG design

The DAQ holds a dedicated ADC for the ECG Module. Pin connections from the FPGA to the DAQ is found in Appendix **??**

5.5 ECG Module

This section describes the design and functionality for each of the FPGA logic blocks included in the ECG module Design VHDL files for the FPGA

ECG Module is found in Appendix D.2. Code generated by Quartus II Megafunctions are not included.

Filters

5.5.1 Lowpass and Highpass Filters

The difference equations of the lowpass and the highpass filters as described in section 2.4 on page 18 are recursive. The digital filters for these was therefore designed as Infinite Impulse Response (IIR) filters. The filter structure of the IIR filter is of Direct-Form I as shown in figure 5.7 The signal goes through a D Flip-Flop for delay. It is then multiplied by the filter coefficient in a multiplier. The signal is then added in an adder, before it is sent to the output. In the IIR filter, the output is delayed, multiplied and added to the signal using feedback. The delay can be seen at the output in figures 4.9 and 4.10 on page 47.

5.5.2 Derivation

The difference equation of the derivation as described in section 2.4 on page 19 is not recursive. The digital filter for this operation was therefore designed as a Finite Impulse Response (FIR) filter. The structure of the FIR filter is Direct-Form I is shown in figure 5.8. The signal goes through a D Flip-Flop for delay. It is then multiplied by the filter coefficient in a multiplier, here referred to as h(M). The signals are then added in an adder, before sent to the output.

5.5.3 Squared

The squaring of the signal was done by using a multiplier, where the input signal was multiplied with itself. The output of the multiplier was sent to the output of the squaring logic block.

5.5.4 Moving Window Integration

The structure of the moving window integration is a Direct-Form I FIR filter where all filter coefficients are 1. The filter was designed like the previously implemented derivation filter. The filter structure is described in 5.5.2.

5.5.5 Peak Detection

The peak detection module is built up by two sequential processes. The first process runs through the first 400 samples of the signal. It determines

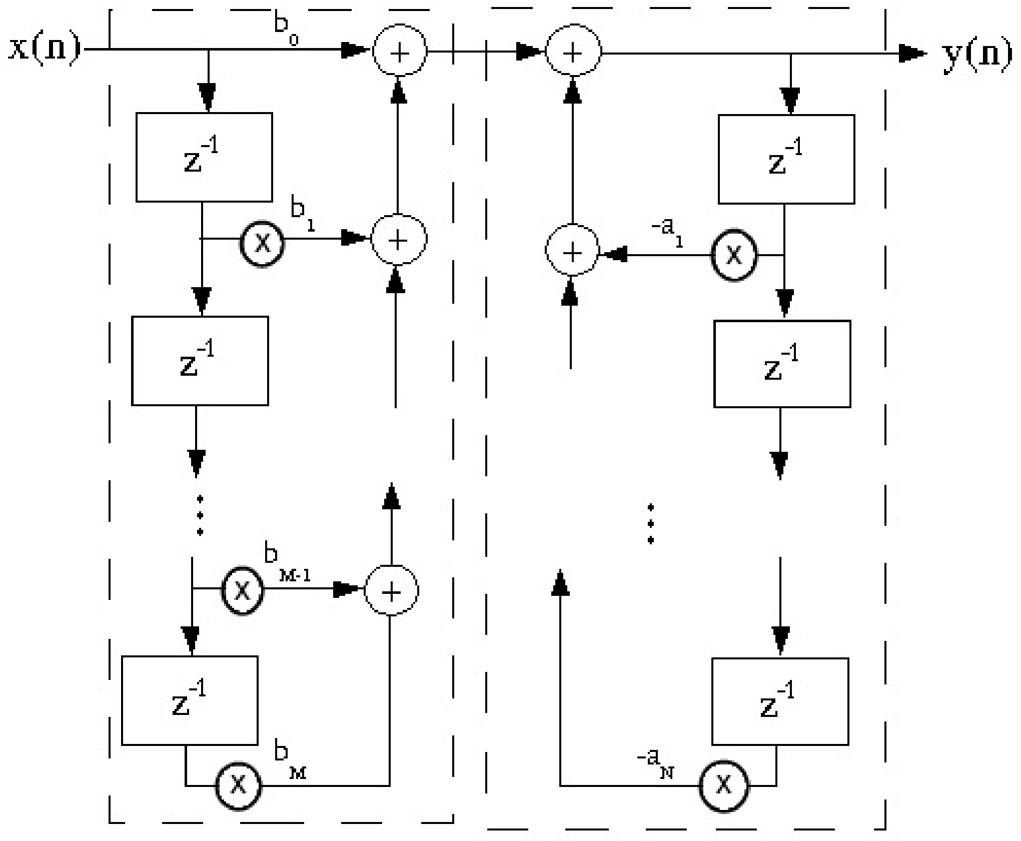


Figure 5.7: **Direct-Form I IIR Filter Structure**
(Jones 2004b) The signal goes through a D Flip-Flop for delay. It is then multiplied by the filter coefficient in a multiplier. The signals are then added in an adder, before sent to the output. In the IIR filter, the output is delayed, multiplied and added to the output signal using feedback.

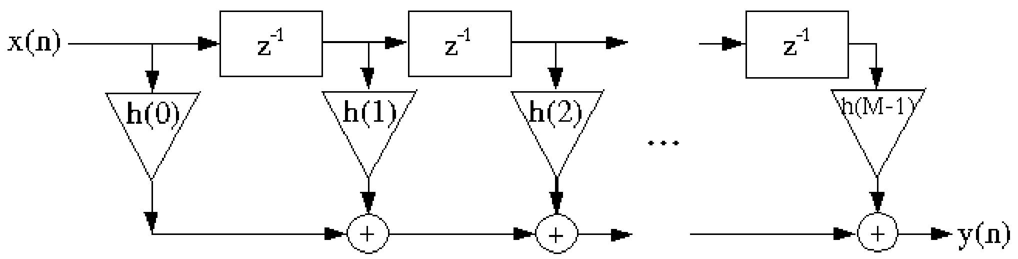


Figure 5.8: **Direct-Form I FIR Filter Structure**
The signal goes through a D Flip-Flop for delay. It is then multiplied by the filter coefficient in a multiplier, here referred to as h(M). The signal is then added in an adder, before it is sent to the output.

the peak value of all the samples taken. From this peak value, an initial threshold is set as

$$\text{Threshold} = \frac{peak}{3}$$

With a sampling frequency of 200 Hz, a maximum is found within 2 seconds of measuring. Depending of the heart rate of the subject measured, at least two QRS-complexes will have been detected within this time frame.

The next process finds peaks over the set threshold, and counts the number of samples between each peak. A peak is identified as a peak when the signal passes the threshold value on its descending slope. When a peak is identified as a peak, a new threshold is set based on that peak. The signal-to-noise ratio is high due to the previous processing of the signal. A threshold of $\frac{1}{3}$ of the peak value, is sufficient to ignore noise peaks.

Any peak detected within 50 samples of the previously detected peak is rejected. At a sampling frequency of 200 Hz, 50 samples equals 250 ms. As described in 2.1.4 on page 8, the cardiac refractory period is 250 ms long, and any peaks detected before that will be a false detection. The output of this module is the distance between peaks detected.

5.5.6 Heart Rate I

The heart rate calculates the first part of the RR Interval from the RR AVERAGE1 as described in section 2.4. It is structured as a moving window integration with a length of 8 samples. The output of this module is the integrated window. To get the accurate value of the averaged value, calculations with floating point numbers are needed. Floating point algorithms in FPGA requires extra resources, as described in section 5.1. Floating point calculations are therefore separated from the integer calculations.

Floating-Number Operations

Floating-point HDL design is very complex and complicated. However, Quartus II offers floating-point arithmetic as megafunctions. Such functions were used to do the floating-point operations in the rest of the ECG module.

5.5.7 Heart Rate II

To get the correct number for the average heart rate, the output of the moving window is divided by its length of 8. The output is the heart rate measured in samples. As we wish to represent it in beats per minute, a new floating-point operation is done from equation (5.1)

$$HeartRate = \frac{60s \cdot f_s}{RRAVERAGE1} = \frac{60s \cdot 200Hz}{RRAVERAGE1} = \frac{12000}{RRAVERAGE1} \tag{5.1}$$

5.5.8 Heart Rate Variability

Heart rate variability is the variation in the time between heartbeats. The difference between each of these outputs, will give the heart rate variability measured over time. The signal is sent to a floating-point subtraction function. To be able to subtract the signal from itself by one delay, the signal is split into to routes. One route is directly to the subtraction function, while the other goes through a box with output = input, creating the necessary delay. The heart rate variability is calculated from equation (5.2)

$$HRV = HeartRate_{i+1} - HeartRate_i \tag{5.2}$$

Chapter 6

iPad Application

As tablets are becoming more and more common in the work place, I opted to code for the iPad as Aamodt (2013) chose the Android platform. This way we covered the two largest suppliers of tablets in the market place.

Tablets provide an excellent screen real estate, and todays tablets have fast GPU (Graphic Processing Units) making them ideal for displaying real time graphs and measurements.

6.1 Application Overview

The iOS application receives data via TCP/IP. A game engine is used to render the graph to the screen, as game engines supplies fast graphics. Using this, up to four graphs can be displayed simultaneously. However, the application can be extended to display more if so was required.

It displays both graphs and numerical data with very low latency due to the TCP/IP protocol and high bandwidth of the wifi card in the device.

6.2 Programming Language and Frameworks

Several frameworks were tested in the development of the application. The first choice was QuartzCore as it is part of the iOS SDK. However the draw calls were to slow, causing the graph to flicker and lag.

Open GL was then considered as it supplies fast rendering of graphics. However, the learning curve was too steep and out of the scope of this thesis.

For high FPS (Frames per second), the Coco2d game engine was chosen as it provides an easy to understand API and a good frame rate.

6.3 Communications Protocol

As the iPad does not support Bluetooth 2.0 at the time of writing, TCP/IP was chosen for the communications protocol.

The TCP/IP protocol has the advantage of having multiple devices read from the same data source, only limited by the number of open sockets the data provider (or server) can support. It is also considerably faster and can handle a higher throughput than Bluetooth. TCP/IP also has guaranteed delivery, and scales incredibly well.

A TCP/IP server was written in Python to transmit simulated biomedical signal as proof of concept. The code for the server is found in Appendix **??**

6.4 Designing the Application

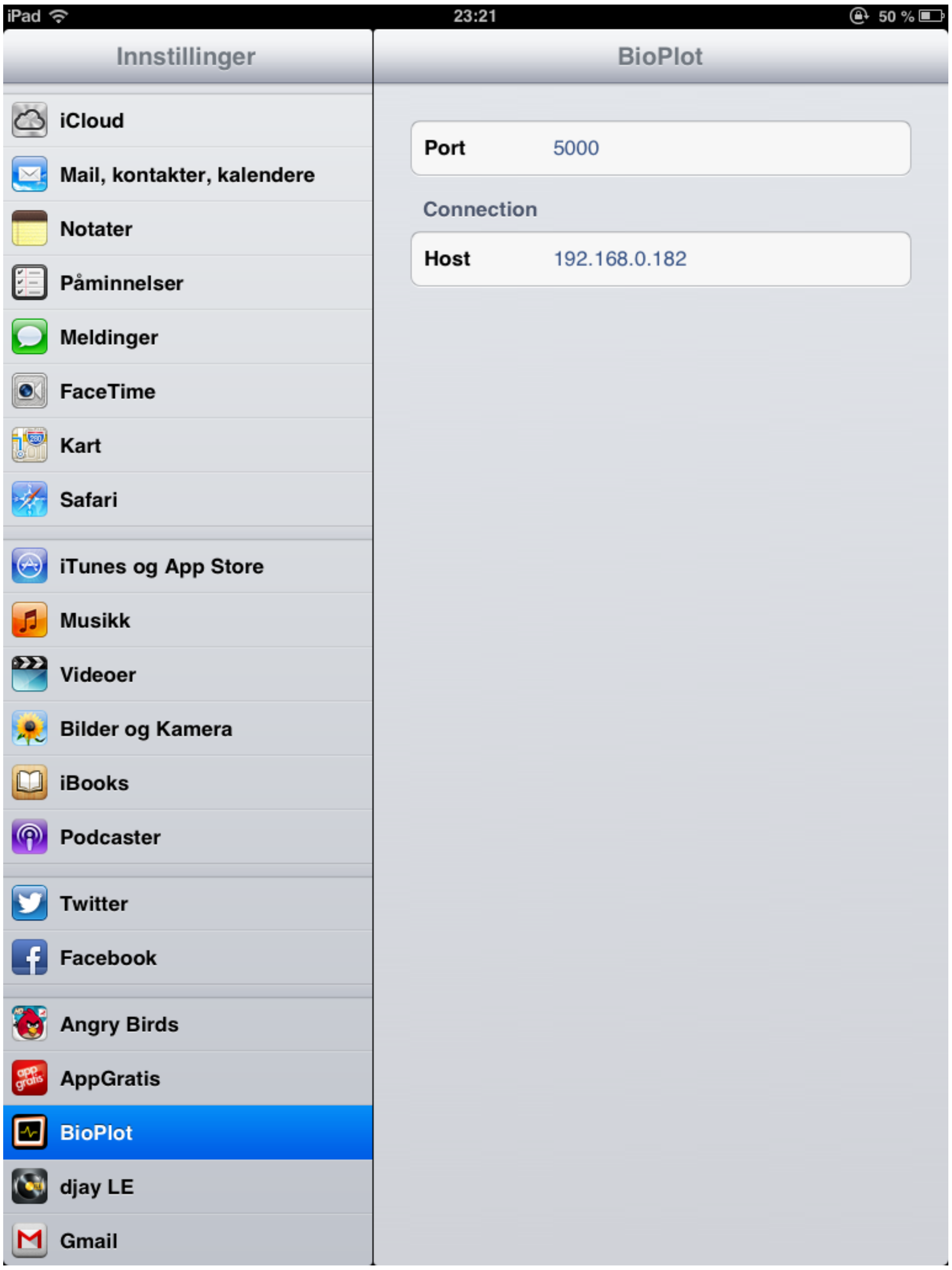
The application design was broken down into three main challenges

Communication

The iOS SDK features a streamreader/streamwriter, which was suitable for the application as they support callbacks. This makes it possible to update the graphics in realtime, as callbacks can be handled asynchronously.

Settings

The host and port will most likely change depending on the network of the device is currently connected to. It was therefore necessary to be able to configure these as needed without recompiling the source. The iOS SDK comes with support for user settings, and a wrapper was written around this to simply set defaults and give the user access to change the port and host without the application configuration. The port and host configuration in the iPad settings menu is seen in figure **??**



Port 5000

Connection

Host 192.168.0.182

Figure 6.1: **Port and host configuration setup in the iPad settings menu**

The Graph Renderer

Cocos2d game engine was used to render the graphs. A couple of simple api calls will draw lines on the screen, not too different from the way QuartzCore handles line drawing. However, the Cocos2d provides a lot higher FPS rate. The lines were rendered by supplying data to an array with a visible section and a buffer section. The visible section would be read, drawn and the updated with the data in the current buffer.

Figure 6.2 shows a screenshot of the running application.

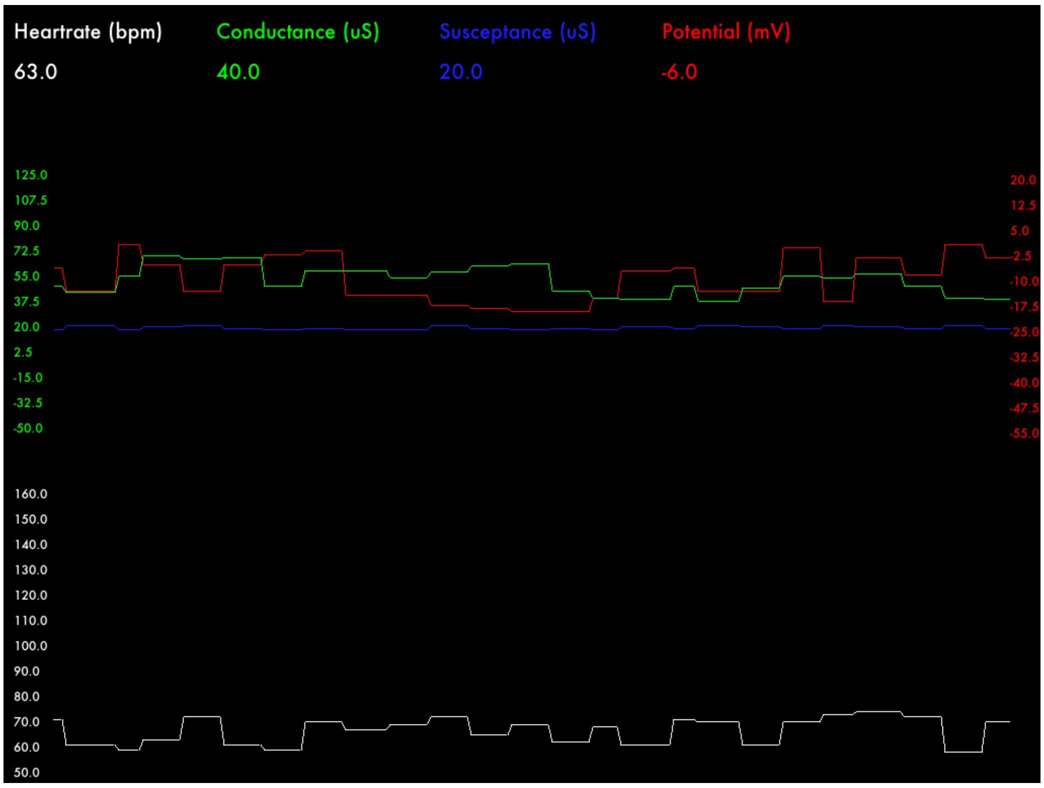


Figure 6.2: **Screenshot of running iPad application**

6.5 Future Prospects

With this technical stack, the possibilities of various applications are endless. This setup supports data distribution to multiple (authorized users in real time. An example application of this would be a hospital environment, where instant monitoring of patiens values could be achieved without having to move between the patients. Alerts could be triggered based on various rule sets within the application.

Using this setup, patients could even be monitored remotely. As a test, this application was run against a server in London, with very low latency.

The python server used in this thesis could be expanded to include Bluetooth communication through protocol packages. With such an expansion, the server can distribute the actual signal sent from the FPGA through the Bluetooth UART set up by Aamodt (2013. Other wireless communication protocols could be implemented in the FPGA, making it possible to bypass Bluetooth and communicate direclty via TCP/IP.

Chapter 7

Verification of the ECG Module

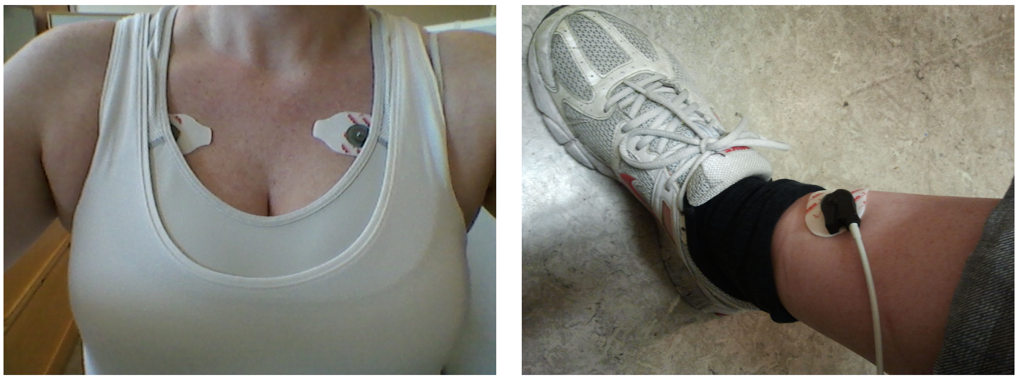
This chapter describes the measurements done to verify the ECG Module described in section 5.5 on page 58

7.1 Measurements

7.1.1 Setup

The measurements were done on the author of this thesis. Ag/AgCl electrodes were placed on LA and RA and RL, as illustrated in figure 7.1

The electrodes were connected to the ECG PCB with a 5-Lead ECG cable. The ECG PCB was connected to the DAQ, and the DAQ to the Cyclone III development board, as shown i figure 7.3 The output of the ECG PCB was connected to an oscilloscope for reference.



(a) LA and RA electrode placement (b) RL electrode placement

Figure 7.1: **Electrode Placement for testing the ECG FPGA Module**
(a) Electrodes are placed on the left (LA) and right (RA) side on the chest for measuring Lead I. (b) An electrode is placed on the right leg (RL) for grounding

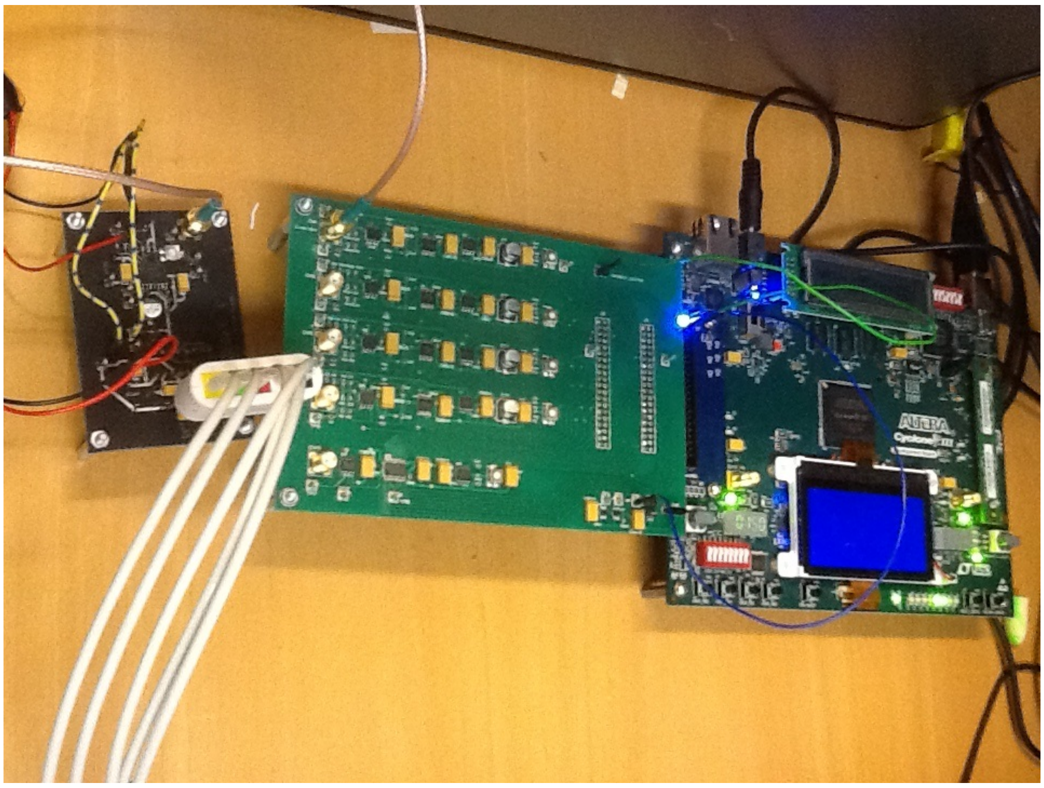


Figure 7.2: **System Setup for ECG Measurements**
Setup for measuring ECG. The ECG PCP is connected to the DAQ, and the DAQ is connected to the Cyclone III development board

7.1.2 Results

The output signal from the ECG PCB, has some interference. Figure 3.1 shows the ECG wave with interference.

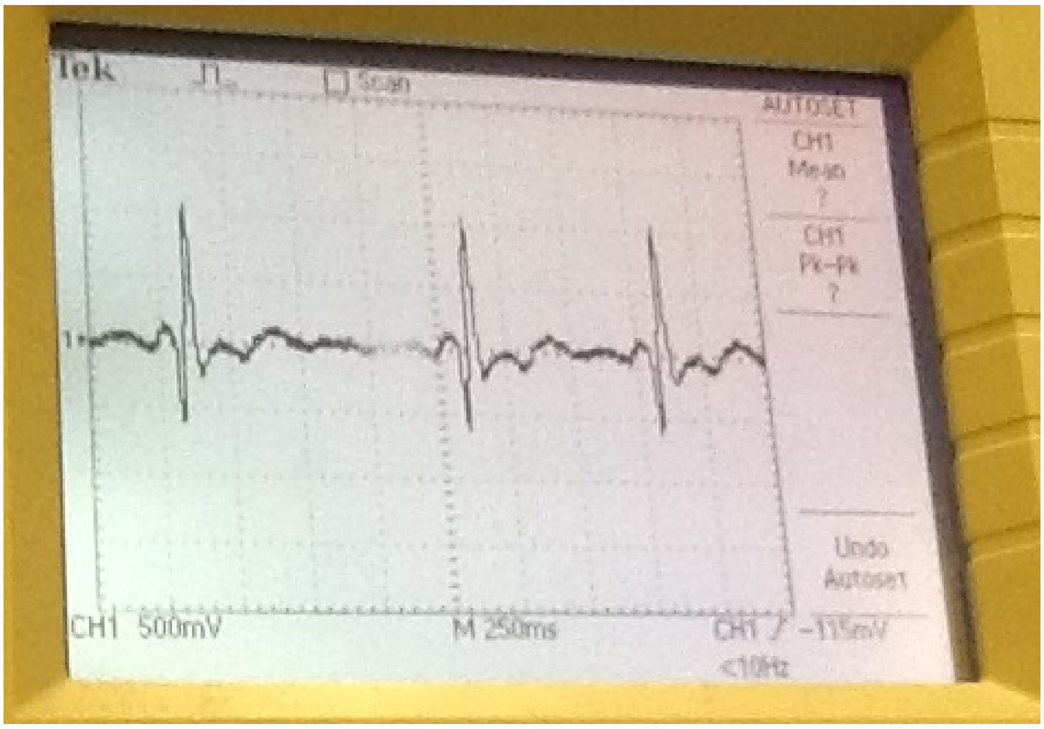


Figure 7.3: **ECG Wave from the output of the ECG PCB**
The ECG wave from the output of the ECG PCB. There is some 50 Hz noise on the signal

From the Fast Fourier Transform (FFT) in figure 7.4, there is a peak at the 50 Hz.

When the test subject touched a ground pin on the DAQ, the 50 Hz interference was attenuated with ≈ 10 dB. The FFT when properly grounded is shown in figure 7.5

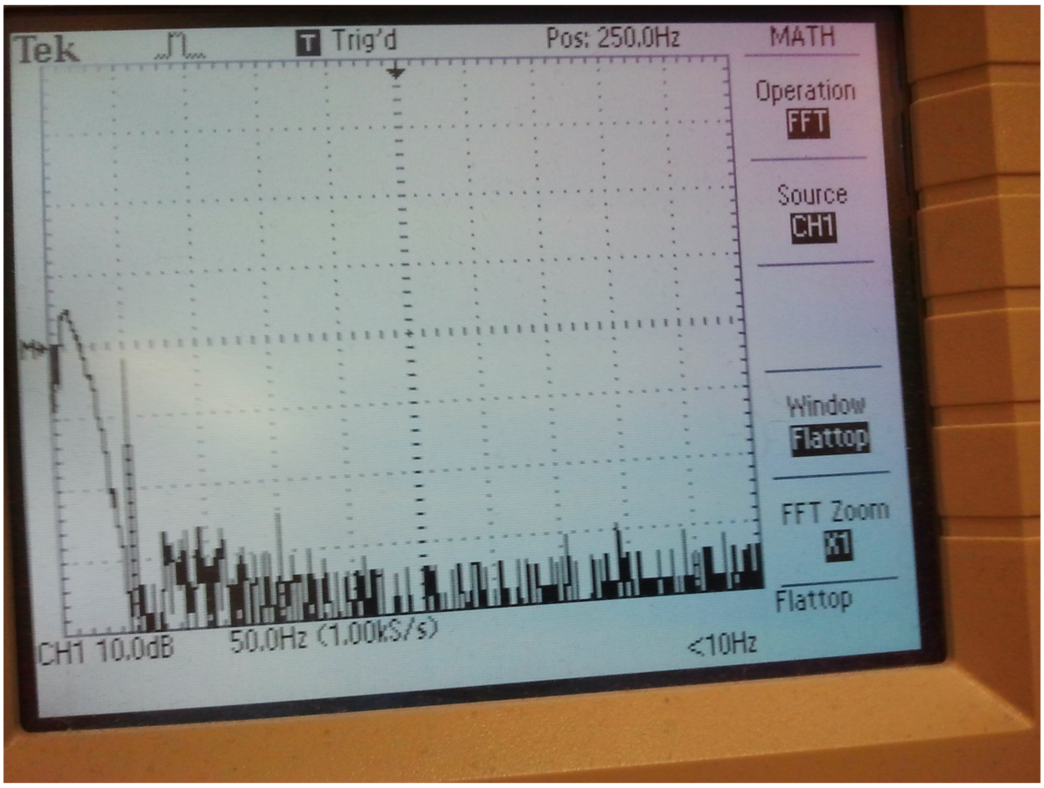


Figure 7.4: **FFT of the output from ECG PCB**
A spike at 50 Hz shows the interference on the signal

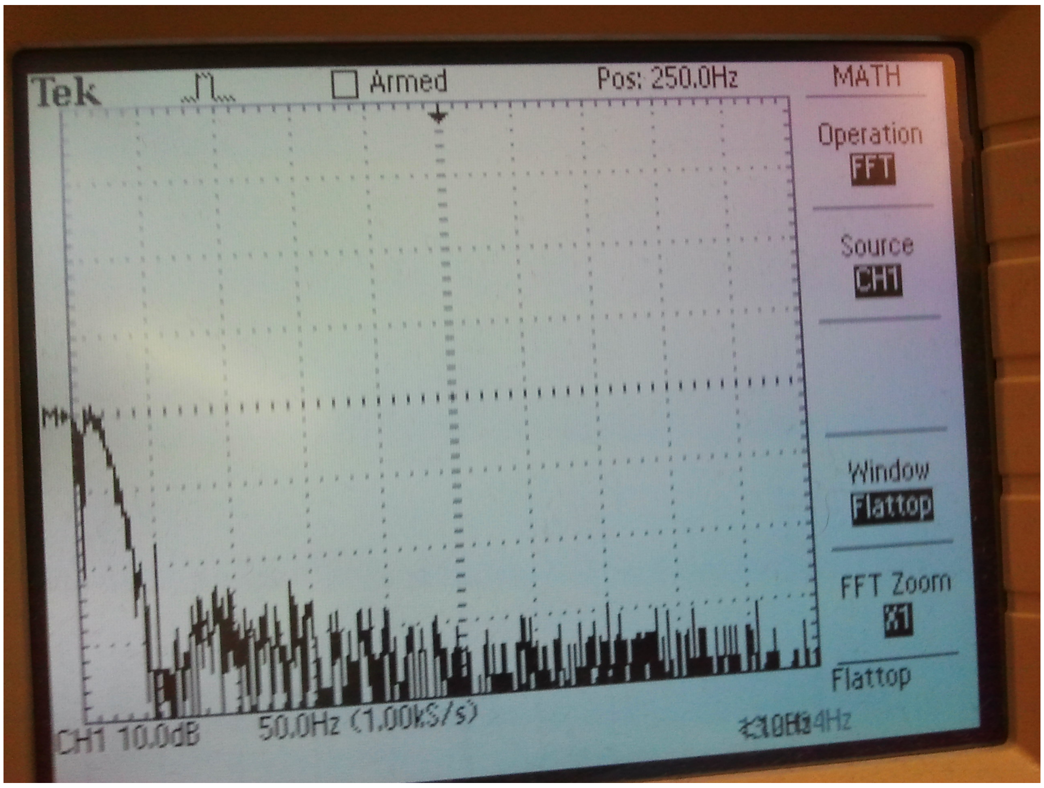


Figure 7.5: **FFT of the output of the signal when test subject touched a ground pin on the DAQ**

When test subject was properly grounded, the 50 Hz interference attenuated with 10 dB

The following figures show the signal outputs of the different modules in the FPGA

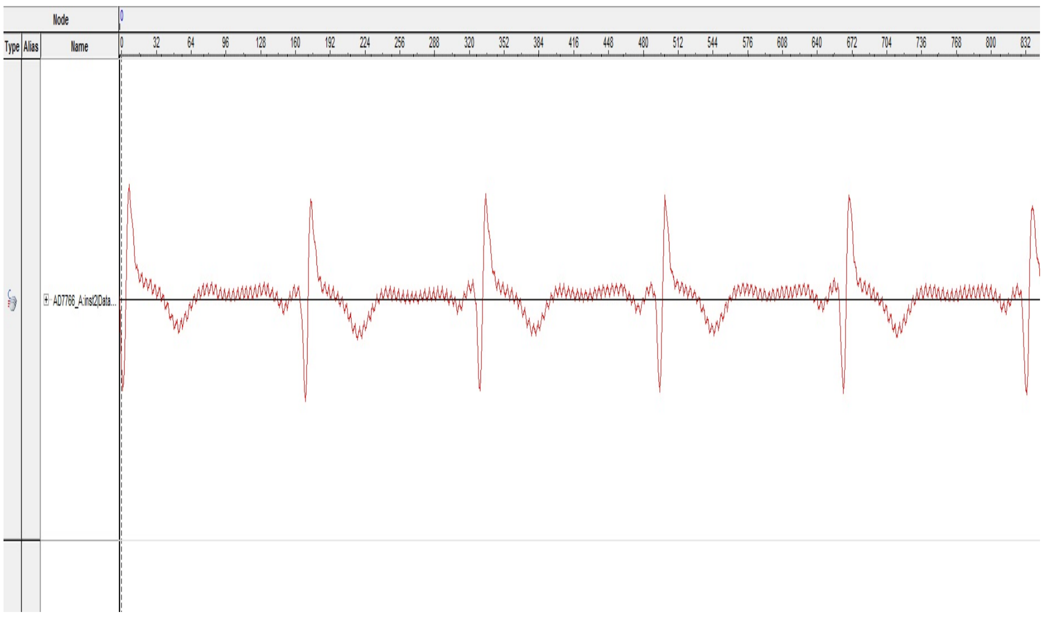


Figure 7.6: **ECG wave at the ADC output**
The ECG signal at the output of the ADC. The 50 Hz interference is noticable.

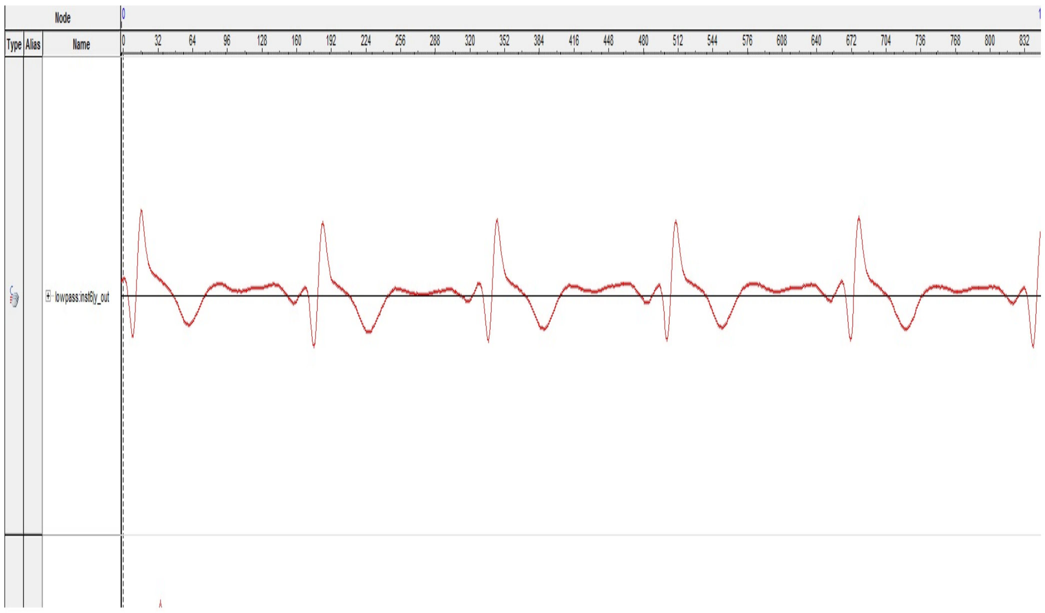


Figure 7.7: **Output of the lowpass filter**
The lowpass filter has a cutoff frequency of 11 Hz. The 50 Hz interference is removed. The signal is delayed 9 clock cycles

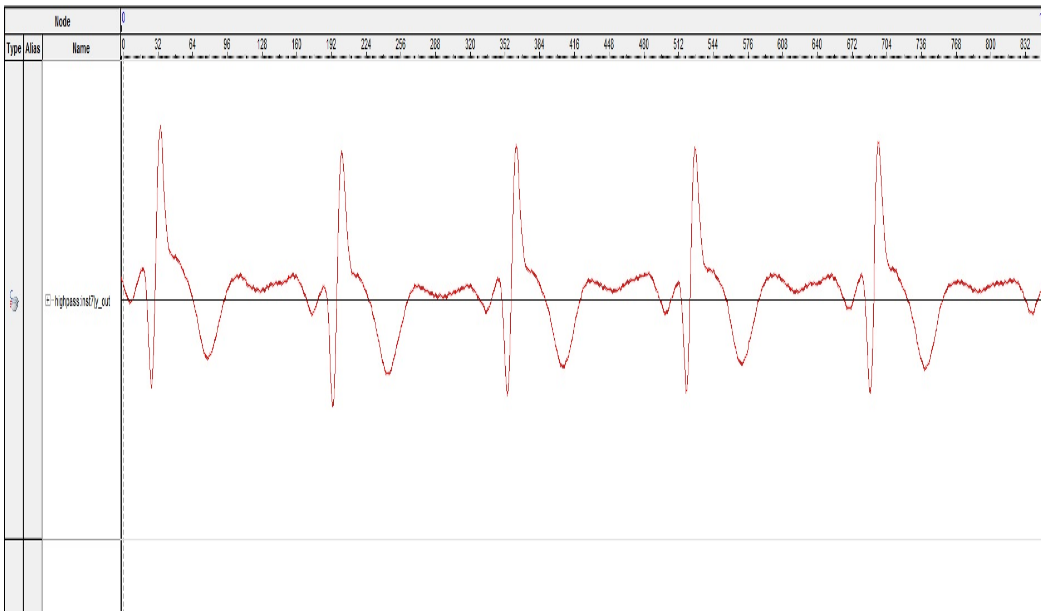


Figure 7.8: **Output of the highpass filter**
The highpass filter has a cutoff frequency of 5 Hz. The signal is delayed 17 clock cycles

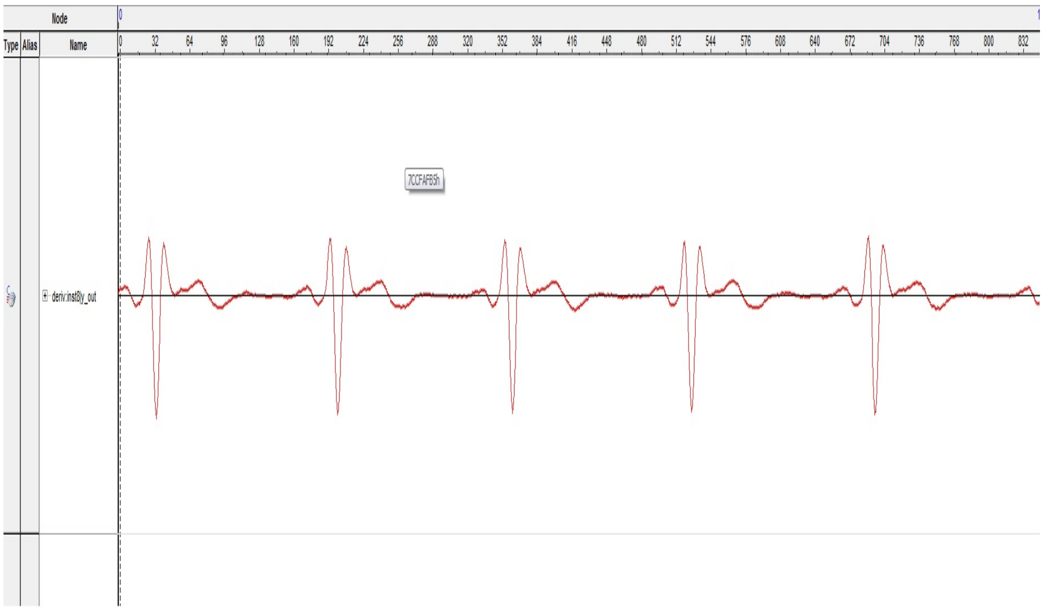


Figure 7.9: **Output of differentiator**
The output of the differentiator provides slope information for the QRS-complex

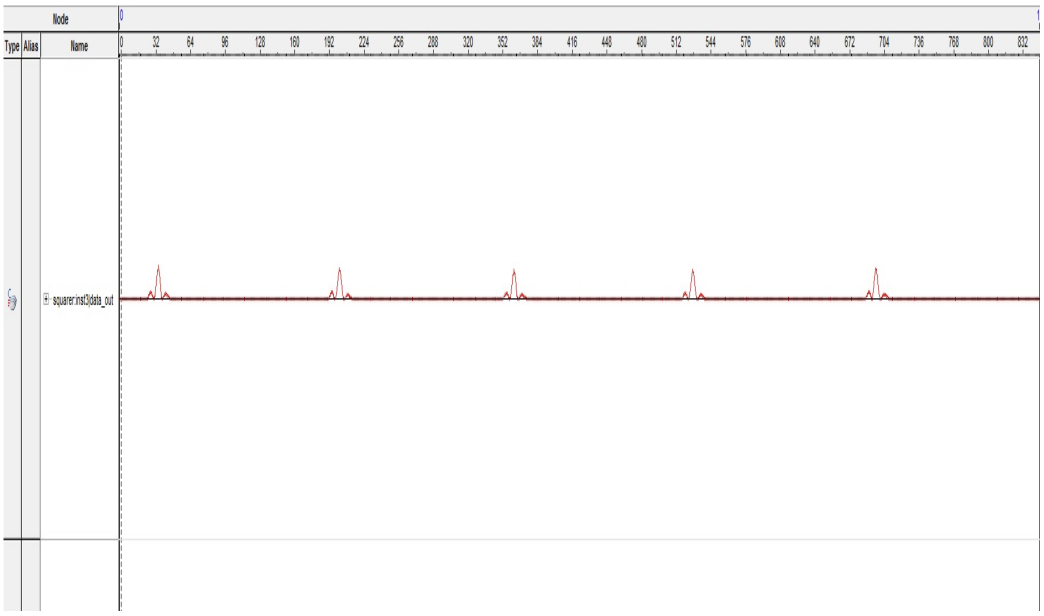


Figure 7.10: **Output of squaring process**
All data points are positive. The squaring does a nonlinear amplification of the derivative emphasizing the higher frequencies

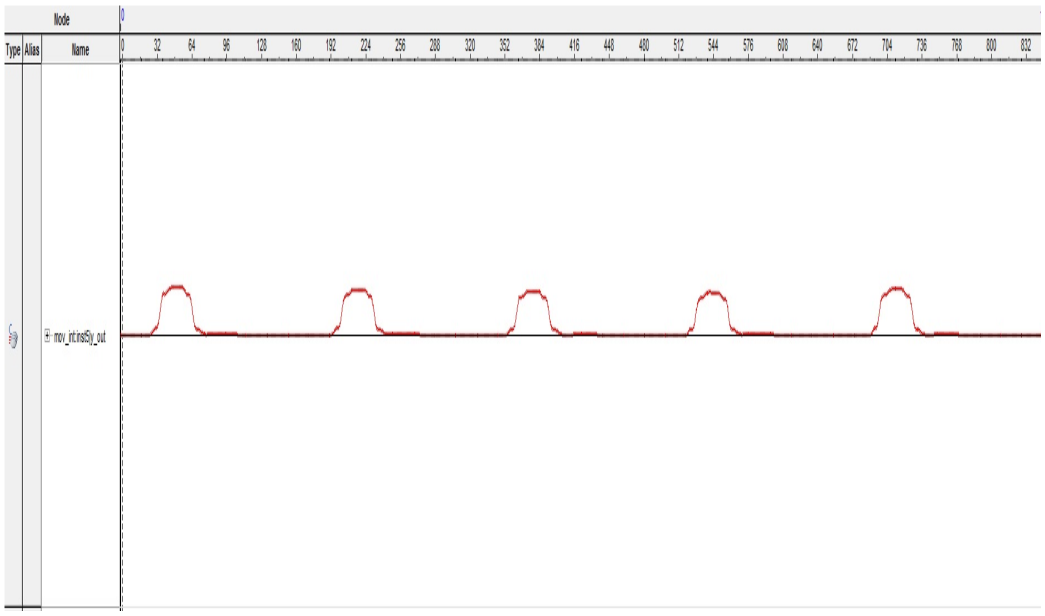


Figure 7.11: **Output of the moving integration**
The moving-window integration obtain information of the waveform and the R wave

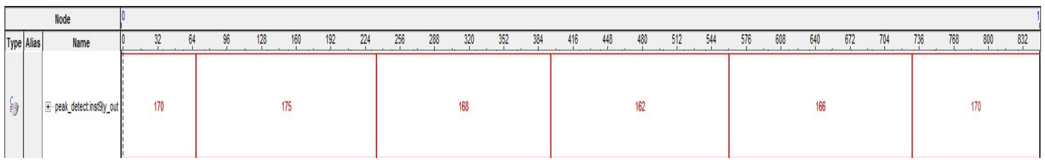


Figure 7.12: **Output of the peak detection**
Peak detection output is the number of samples between each detected peak

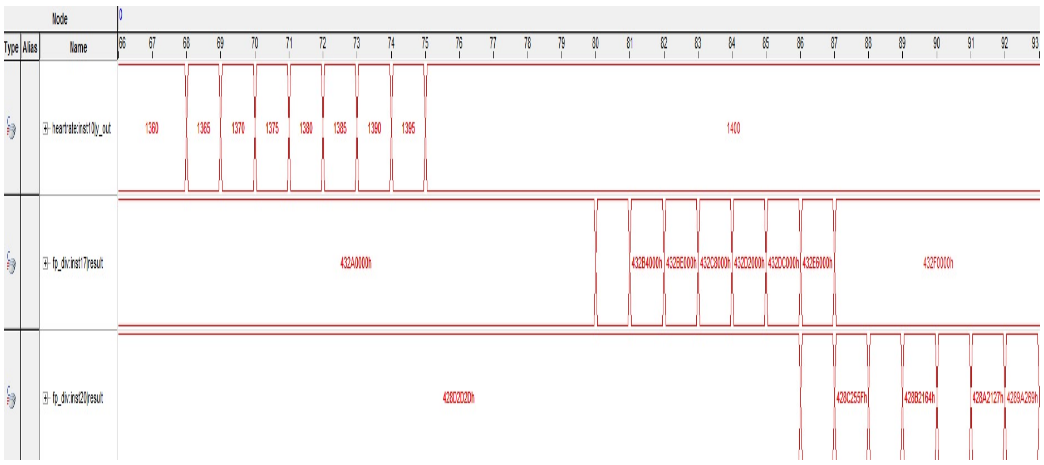


Figure 7.13: **Float-point calculations in FPGA**
The heartrate output is the integration filter for RR interval calculations. The heartrate goes through a floating-point converter, and then into the floating-point divider. Each floating-point operation is set with a latency of 6 clock cycles. The fp_div:inst17 is the floating-point average of the heartrate output, delayed 12 clock cycles. The fp_div:inst20 is the calculation of the actual heart rate. Because it is floating-point calculations, the output has a latency of 6 clock cycles.

7.2 Summary

The figures 7.6 to 7.13 show the output of the modules used from the Pan-Tompkins algorithm to help calculate the HRV. Compared to the results in figure 2.14 on page 22 by J.Pan (1985), and the MATLAB simulation results in Appendix **??**, the outputs are as expected.

There is some 50 Hz interference on the original signal. This was attenuated to an acceptable dB when touching a grounding pin on the DAQ. The notch filter in the ECG circuit takes care of most of the 50 Hz interference. Insufficient grounding of the test subject makes transient current go through the body, and in effect making an antenna. By adding a ground plane to the ECG PCB, this problem will most likely disappear.

Each of the modules contribute to a timing latency for the signal. Based on reading the diagrams from the output figures, the total delay from the QRS peak in the original ECG signal to the heart rate output from the floating-point division is $\approx$ 80 samples. With the sampling frequency of 200 Hz, this equals 395ms.

Chapter 8

Conclusion and Future Work

8.1 Conclusion of the Present Work

The work of this master thesis describes the design, development and analysis of an ECG measuring unit using the FPGA technology. According to the accomplishments and contributions, the following conclusions can be drawn:

- An analogue ECG circuit was developed and verified.
- A Data Acquisition Card developed, and verified
- An FPGA Module for ECG measurements was made, and verified. The ECG Module runs through every step of the Pan-Tompkins Algorithm for QRS peak detection. After the peaks are detected, Heartrate and Heartrate variability is calculated. The module is designed as such that the signal can be viewed and analyzed at any point during the digital processing. It is not limited to only the final output.
- The ECG FPGA Module is ready for implementation in the FPGA Development Platform Prototype. The implementation is done by connecting the output of the module to the Nios II PIO ports.
- An iPad application was created. As of today, it only works as a proof of concept. The application itself works, but only with generated data. The appropriate wireless communication protocol is missing.

8.2 Future Work

Based on the experience and results of this thesis, improvements and changes can be done.

- To remove all the 50Hz noise from the ECG signal, the circuit need better grounding.
- If the use of the ECG circuit is to be used with instruments not run on batteries, an isolation barrier should be implemented for the safety of the patient.
- A different wireless communication protocol needs to be added to the FPGA, so that the iPad application can receive real data.

Bibliography

Aamodt, Lars Jorgen Johnsen (2013). 'FPGA Based Development Platform for Biomedical Measurements'. Master Thesis. University of Oslo.

Anna Gruetzmann, Stefan Hansen et al. (2007). 'Novel dry electrodes for ECG monitoring'. In: *Physiol. Meas.* 28, pp. 1375–1390.

Ashenden, Peter J. (2008). *The Designer's Guide to VHDL*. Third Edition. Morgan Kaufmann Publishers, imprint of Elsevier.

B.S. Lipman, E. Massie et al. (1972). *Clinical Scalar Electrocardiography*. Chicago: Yearbook Medical Publishers, Inc.

Behne, Tim (2003). *FPGA Clock Schemes*. URL: http://www.embedded.com/design/configurable-systems/4024526/FPGA-Clock-Schemes.

Bekkeng, Tore Andre (2009). 'Prototype Development of a Multi-Needle Langmuir Probe System'. Master Thesis. University of Oslo.

Bishop, David (2008). *Floating Point Package User's Guide*. URL: http://www.vhdl.org/fphdl/Float_ug.pdf.

Bobbie, Patrick O. et al. (2004). 'Electrocardiogram (EKG) Data Acquisition and Wireless Transmission'. In: pp. 2665–2672.

Carlson, Shawn (2000). *Home is where the ECG is*. URL: http://www.scientificamerican.com/article.cfm?id=home-is-where-the-ecg-is.

Christos Pavlatos Alexandros Dimopoulos, G. Manis et al. *Hardware Implementation of Pan and Tompkins QRS Detection Algorithm*. URL: http://mule.cslab.ece.ntua.gr/docs/c8.pdf.

Chu, Pong P. (2008). *FPGA Prototyping by VHDL Examples [Xilinx Spartan-3 Version]*. John Wiley and Sons, Inc.

Cyclone III Device Handbook (2011). Altera Corporation.

Digilent (2004). *PmodBT2 - Bluetooth Interface*. URL: http://www.digilentinc.com/Products/Detail.cfm?Prod=PMOD-BT2.

Grødal, Agnar (1997). *Elektromagnetisk Kompabilitet for KonstruktÃ¸rer*. Tapir Forlag.

Introduction to Megafunction IP Cores (2013). URL: http://www.altera.com/literature/ug/ug_intro_to_megafunctions.pdf.

J.Pan, W. J. Tompkins (1985). 'A Real-Time QRS Detection Algorithm'. In: *IEEE Trans. Biomed. Eng.* 32, pp. 230–236.

John G. Webster, Editor (2010). *Medical Instrumentation [Application and Design]*. Fourth Edition. John Wiley and Sons, Inc.

Jones, Douglas L. (2004a). *FIR Filter Structures*. URL: http://cnx.org/content/m11918/latest/.

— (2004b). *IIR Filter Structures*. URL: http://cnx.org/content/m11919/latest/.

Kreuger, R.C.B (2006). URL: http://en.ecgpedia.org/wiki/Basics.

McSharry, Patrick et al. (2003). 'ecgsyn.m'. In: *IEEE Transactions On Biomedical Engineering* 50(3), pp. 289–294.

Nios II Prosessor Reference Handbook (2011). Version Design Suite Version 11.0. Altera Corporation.

Olav Sand Oystein V. Sjaastad, Egil Haug et al. (2009). *Menneskekroppen [Fysiologi og Anatomi]*. Second Edition. Gyldendal Akademisk.

Poole, Ian (2013). *Op amp notch filter circuit*. URL: http://www.radio-electronics.com/info/circuits/opamp_notch_filter/opamp_notch_filter.php.

Quartus II Handbook (2013). Version 13. Altera Corporation.

Rishi (2012a). *Action Potentials in Cardiac myocytes*. URL: https://www.khanacademy.org/science/healthcare-and-medicine/heart-depolarization/v/action-potentials-in-cardiac-myocytes.

— (2012b). *Action Potentials in Pacemaker Cells*. URL: https://www.khanacademy.org/science/healthcare-and-medicine/heart-depolarization/v/action-potentials-in-pacemaker-cells.

Shukla, Ashish (2008). 'Hardware Implementation of Real Time ECG Analysis Algorithms'. Master Thesis. University of Hawaii.

Thede, Les (2004). *Practical Analog and Digital Filter Design*. Artech House, Inc.

Uysal, Faruk (2011). *QRS Complex Detection and ECG Signal Processing*. URL: http://matlabz.blogspot.no/2011/04/contents-cancellation-dc-drift-and.html.

Vipinl (2011). *VHDL for 4-tap-fir-filter*. URL: http://vhdlguru.blogspot.no/2011/06/vhdl-code-for-4-tap-fir-filter.html.

Wikipedia (2001). URL: http://en.wikipedia.org/wiki/Nyquist-Shannon_sampling_theorem.

— (2004a). URL: https://en.wikipedia.org/wiki/Electrical_conduction_system_of_the_heart.

— (2004b). URL: http://en.wikipedia.org.

— (2007). URL: http://en.wikipedia.org/wiki/U_wave.

Zwolinski, Mark (2004). *Digital System Design with VHDL*. Second Edition. Pearson Education Limited.

Appendix A

Physical Constants

Table A.1: Physical Constants

$R = 8.31 \frac{J}{mol \cdot K}$	Gas Constant
$F = 96500 \frac{C \cdot valence}{mole}$	Faraday's Constant

Appendix B

Schematics and PCB Prints

B.1 Schematic drawing of the ECG Front-End

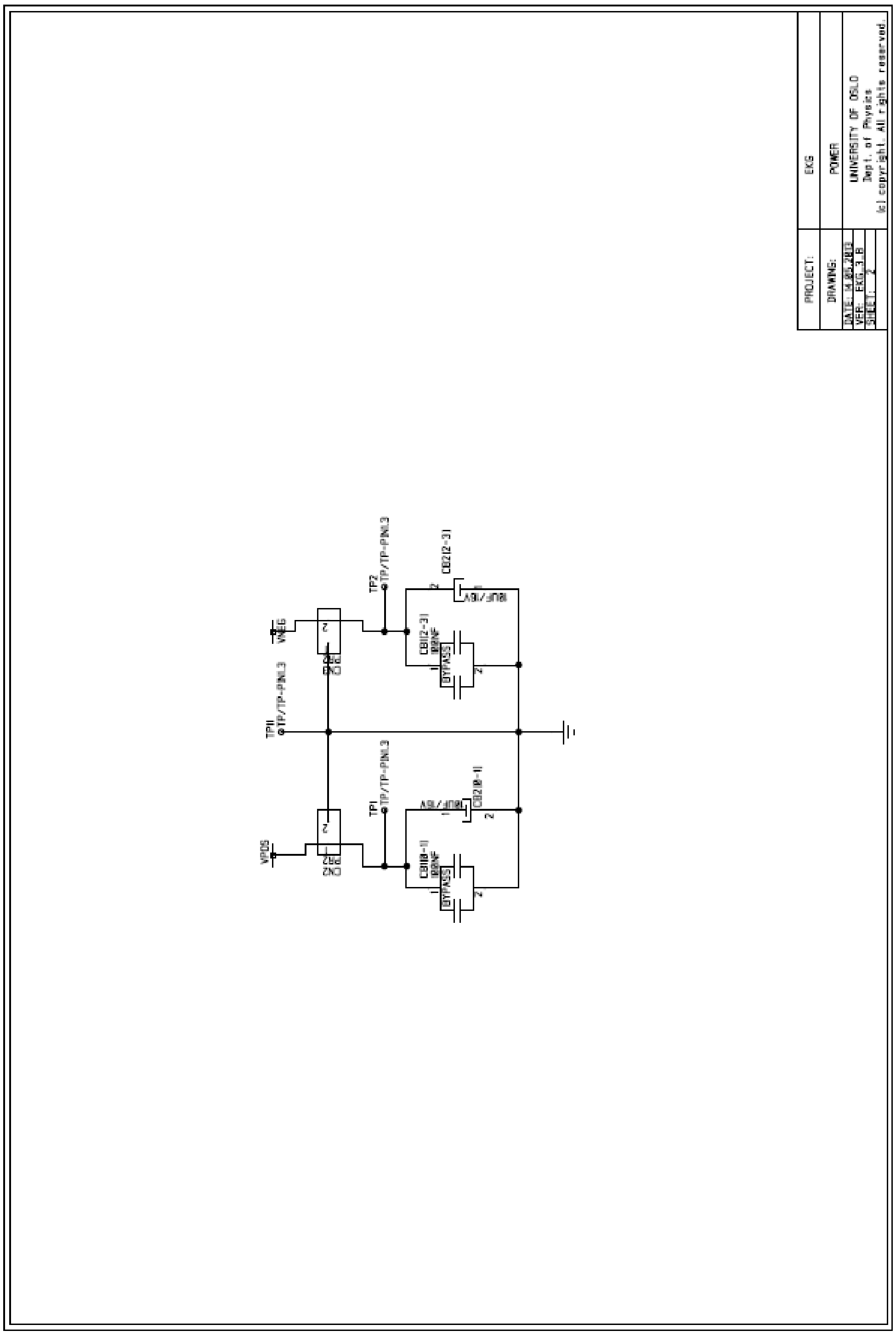


Figure B.1: Power Distribution for ECG Circuit

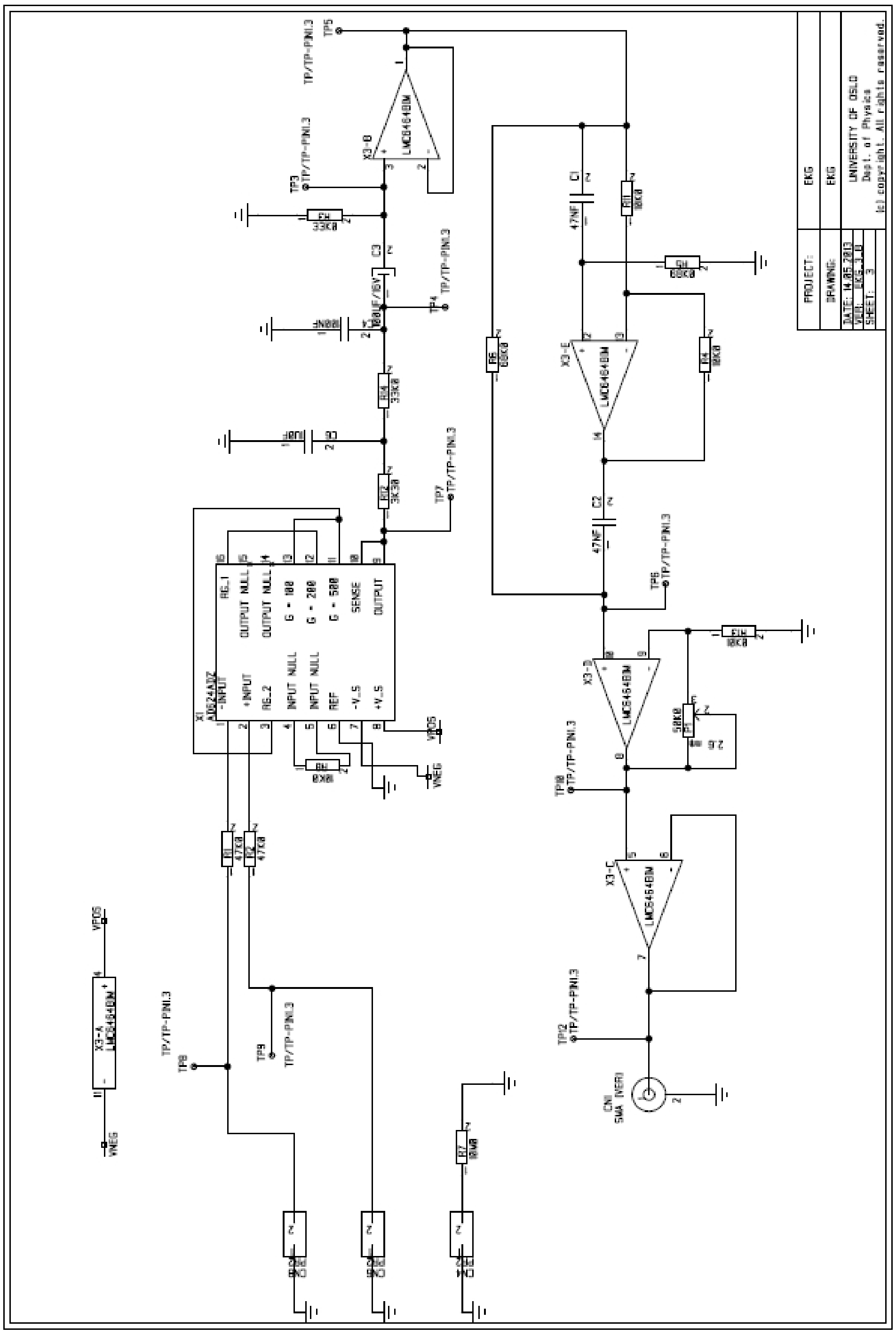


Figure B.2: ECG Circuit

B.2 PCB realization of the ECG Front-End

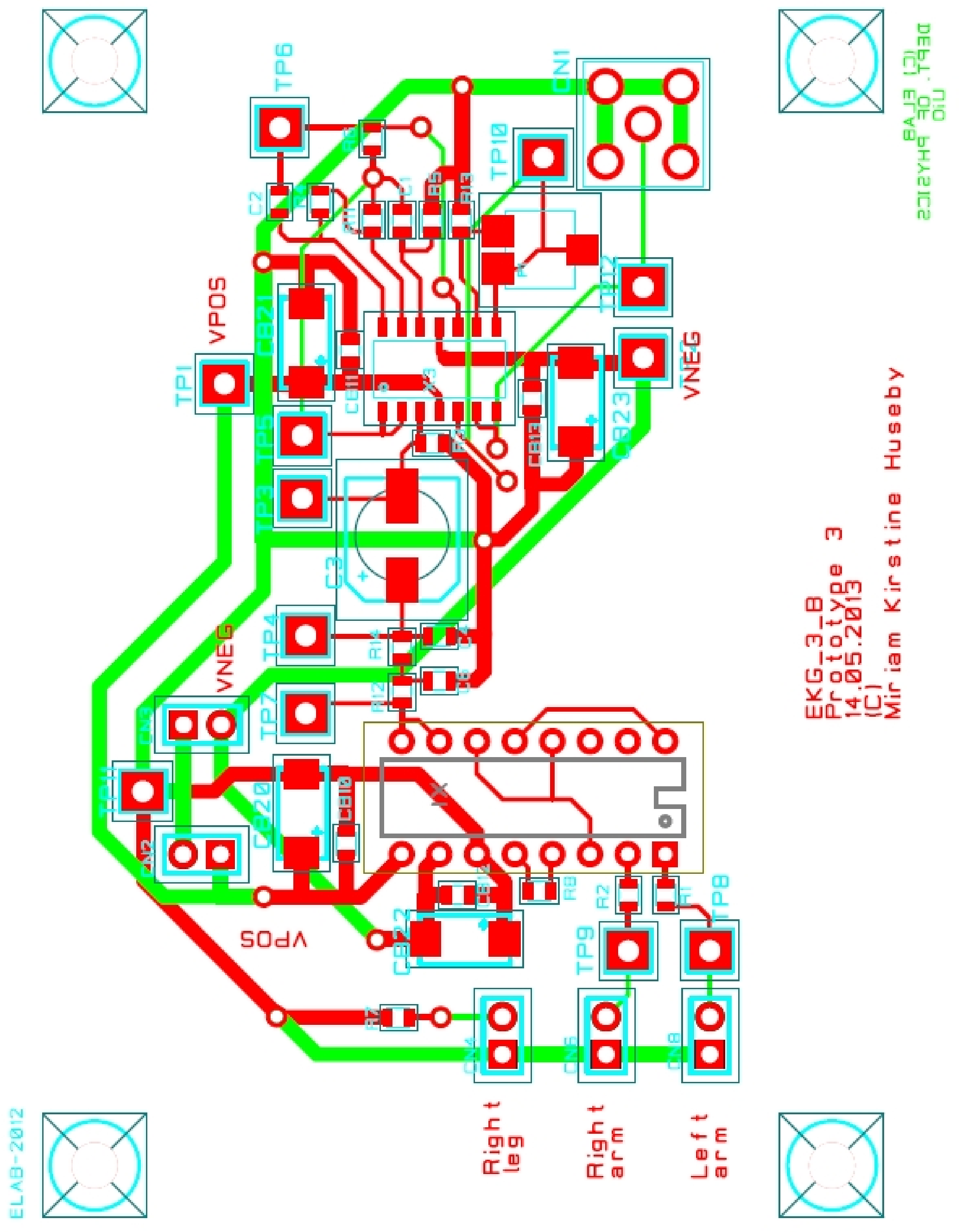


Figure B.3: PCB ECG Front-End

B.3 Schematic drawings of the Data Acquisition Card circuit

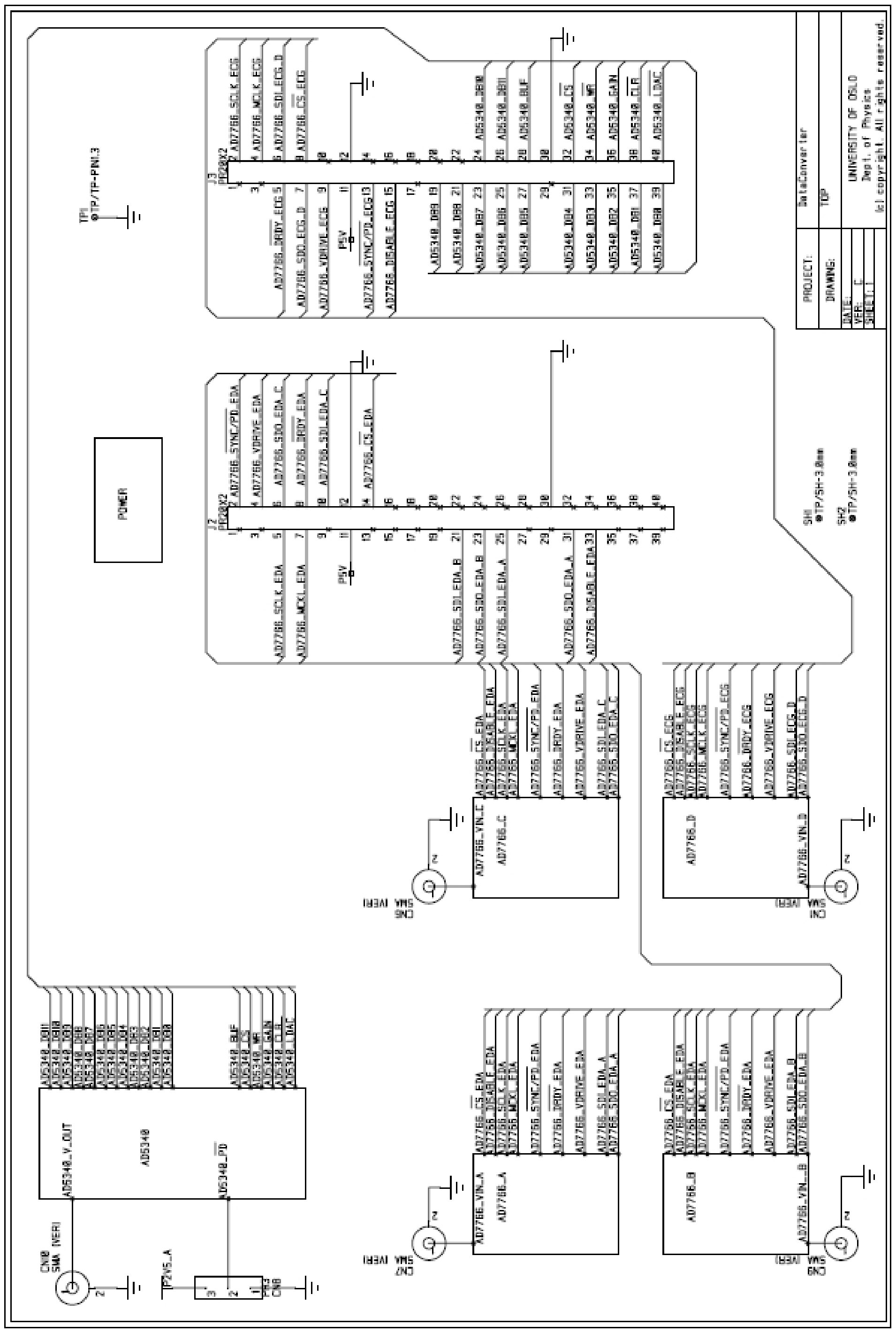


Figure B.4: Overview Schematic of DAQ

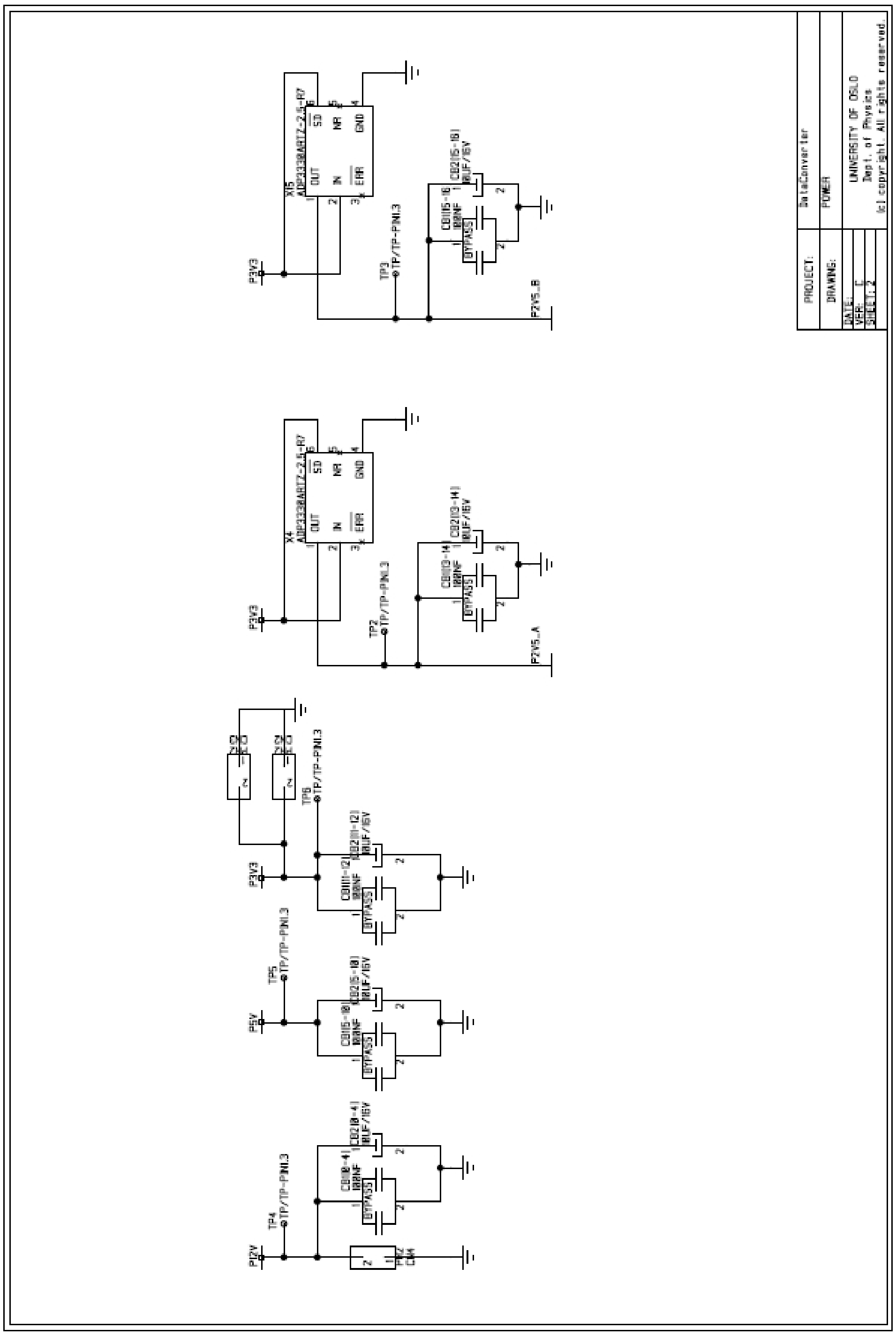


Figure B.5: Power distribution on DAQ

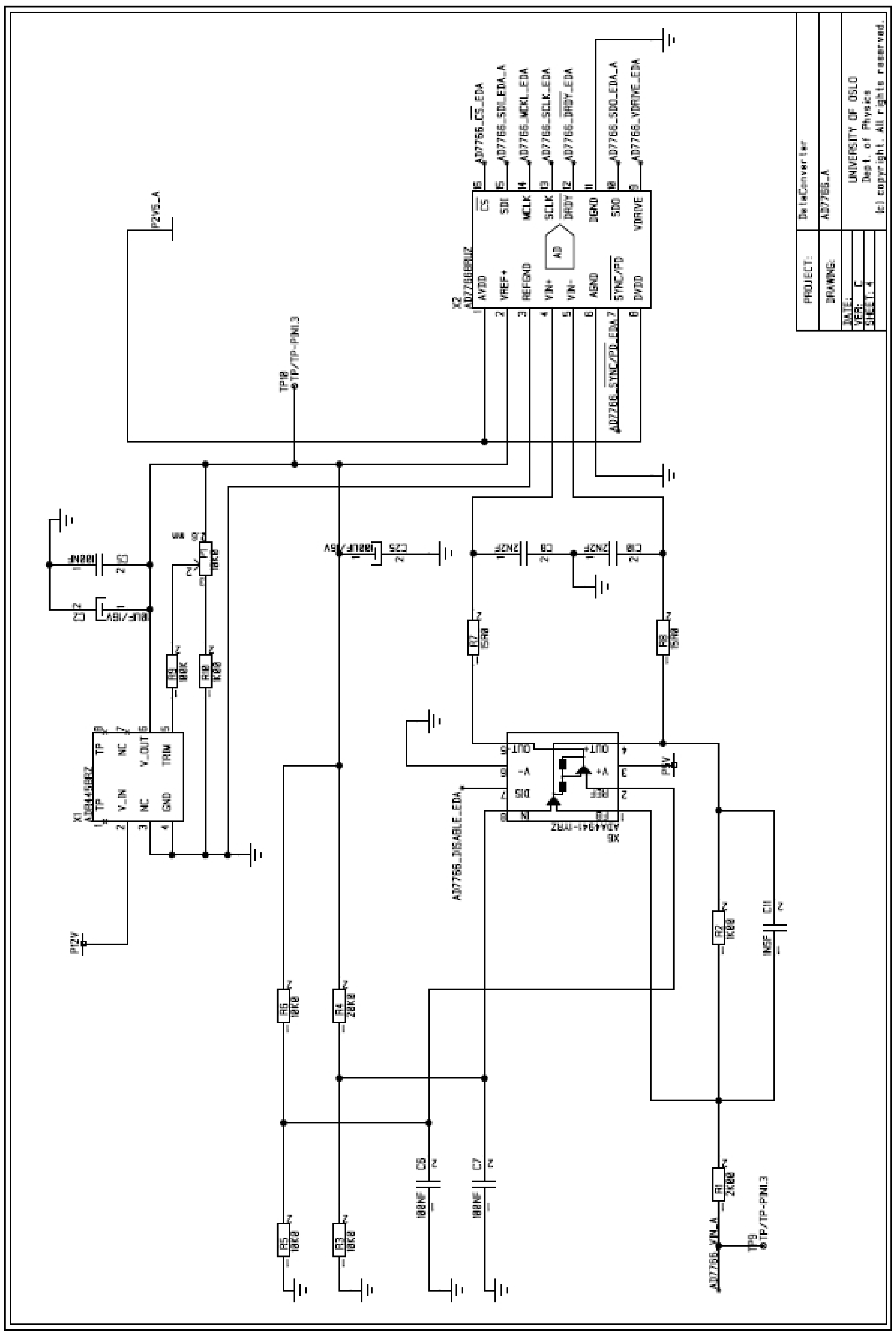


Figure B.6: Circuit for ADC for ECG Signal

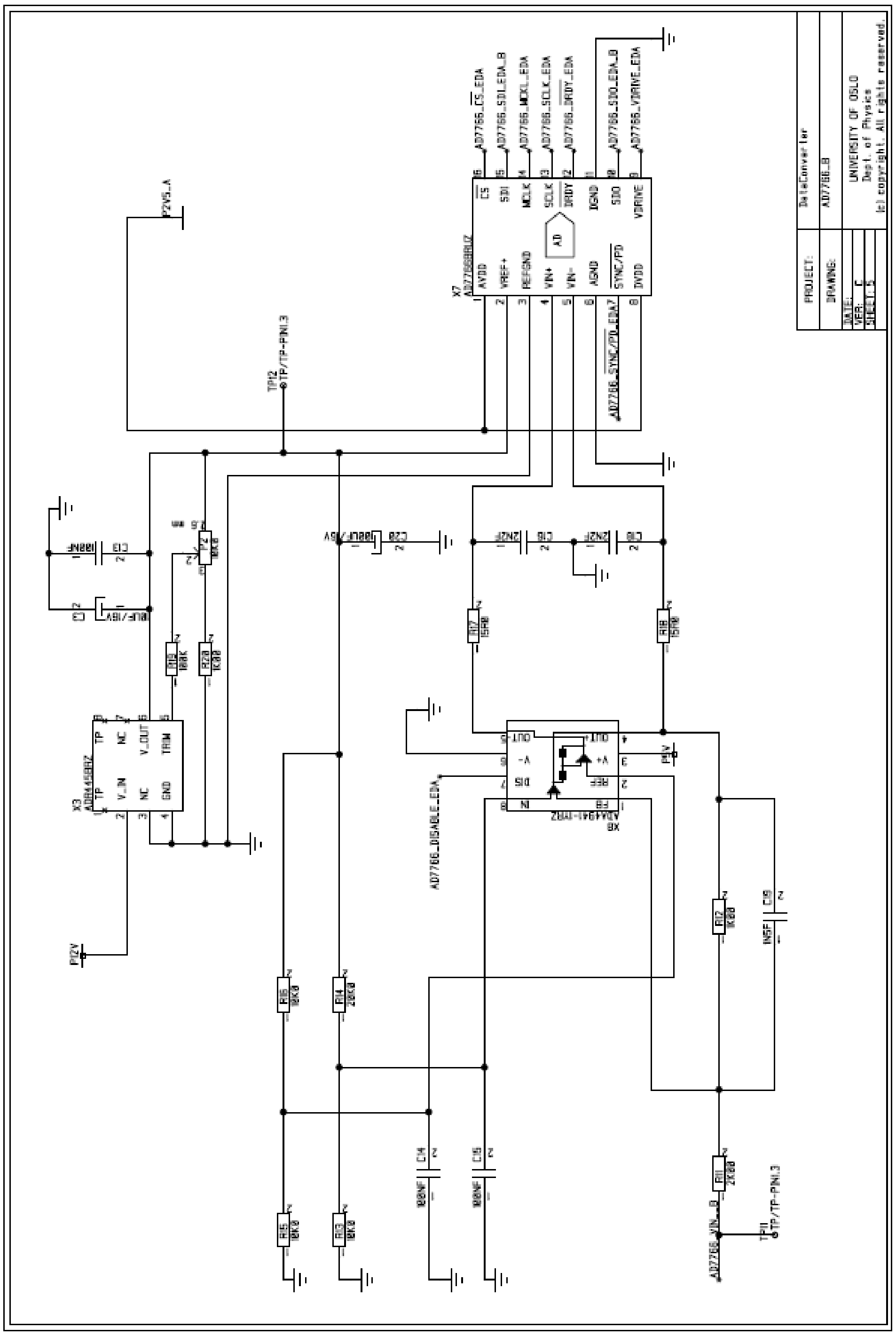


Figure B.7: Circuit for ADC for EDA Signal 1

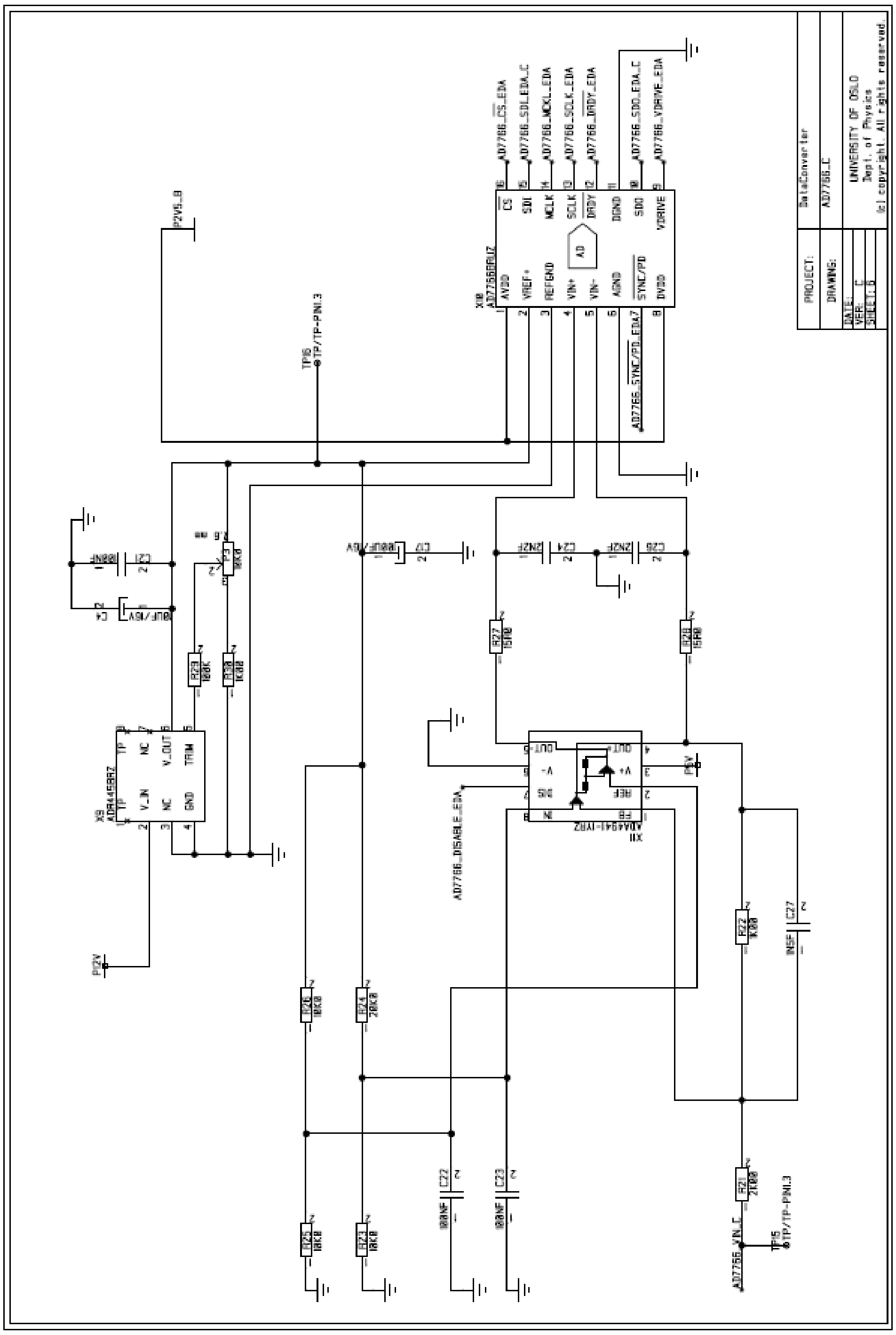


Figure B.8: Circuit for ADC for EDA Signal 2

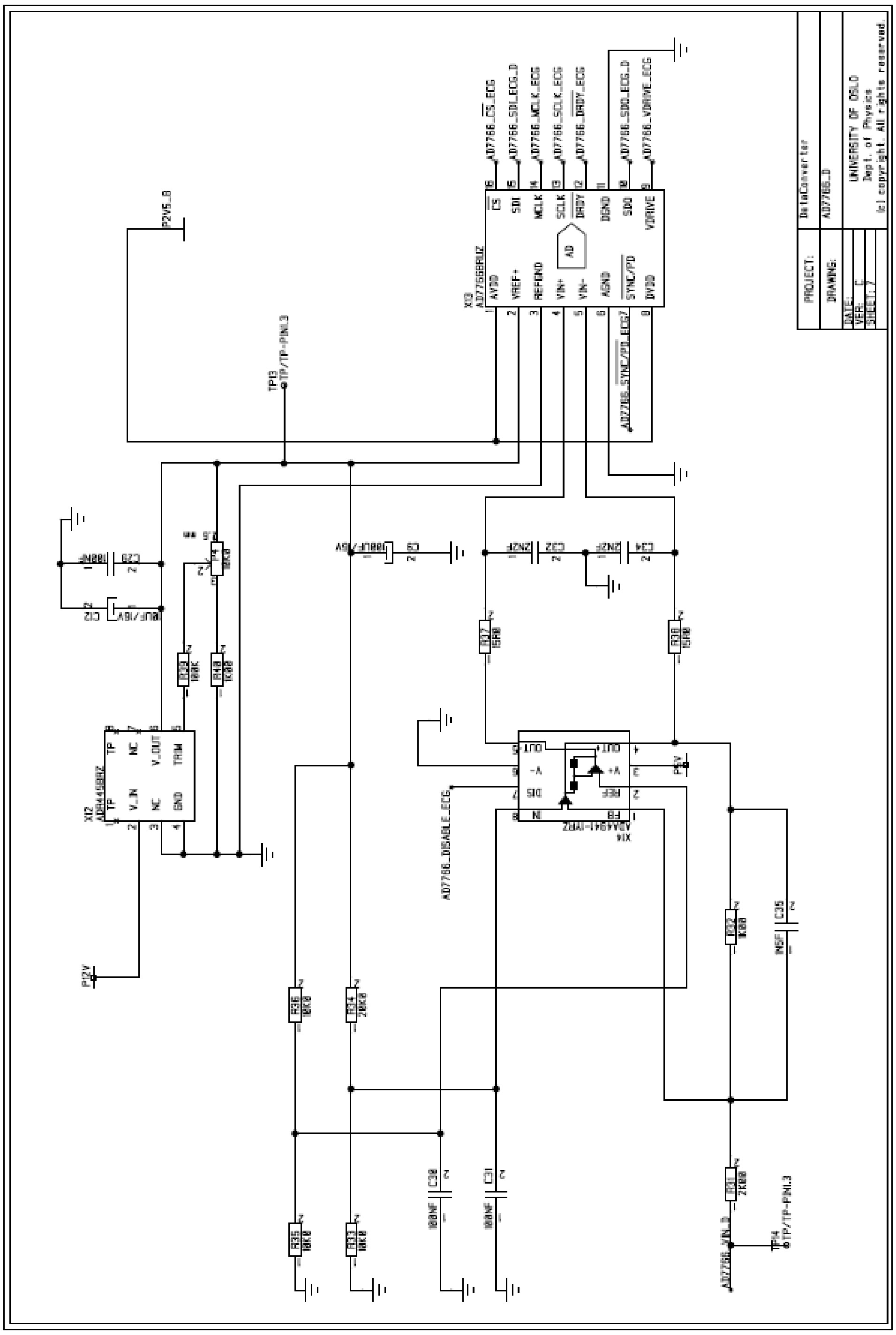


Figure B.9: Circuit for ADC for EDA Signal 3

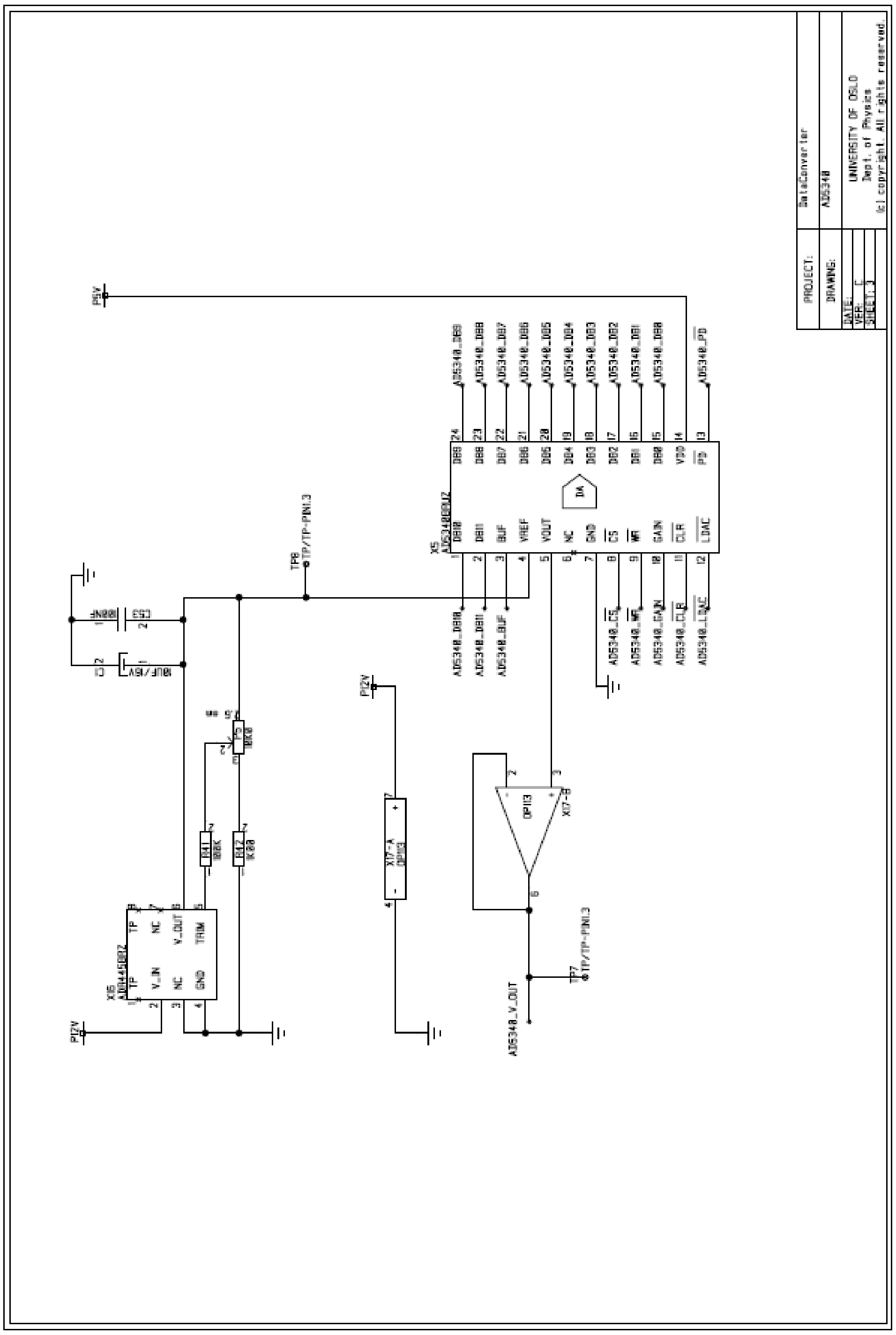


Figure B.10: Circuit for DAC for EDA Signal

B.4 PCB Prints of the Data Acquisition Card

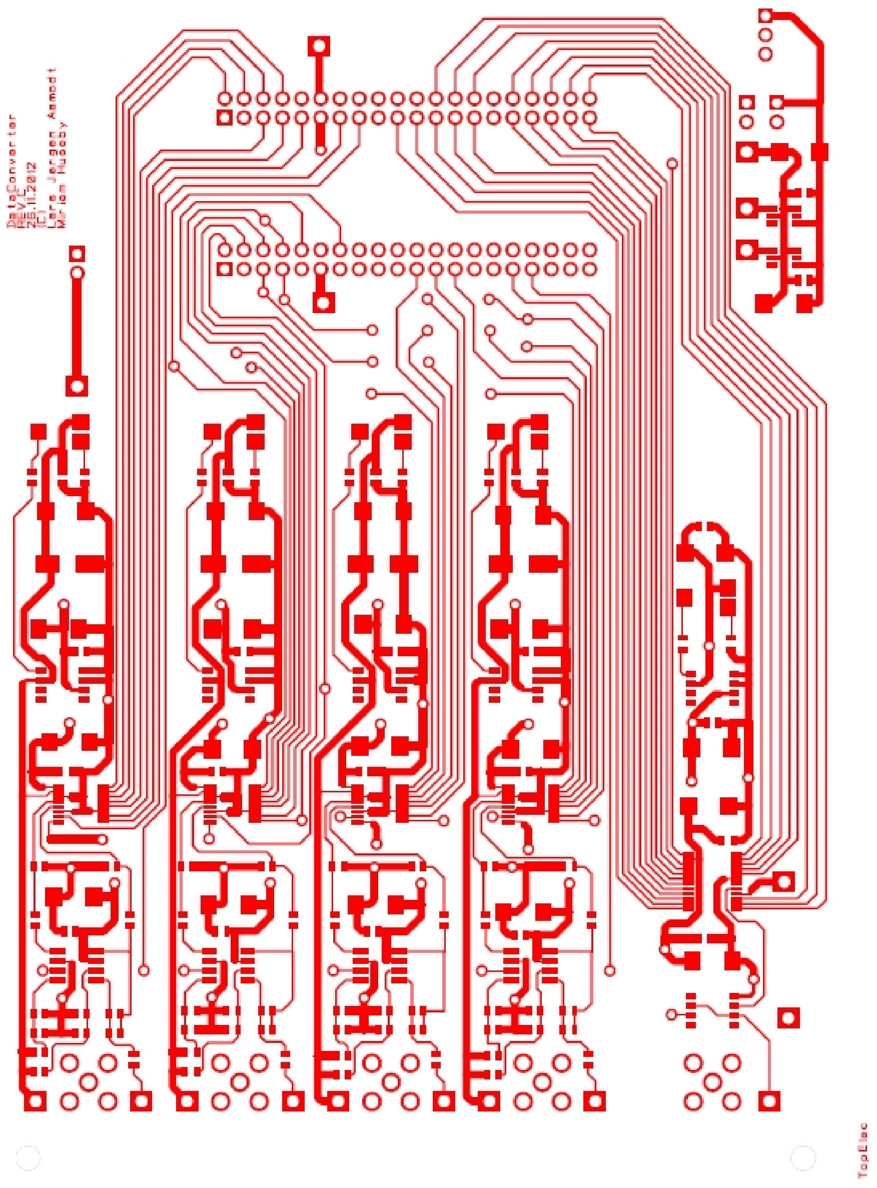


Figure B.11: DAQ PCB Top Layer

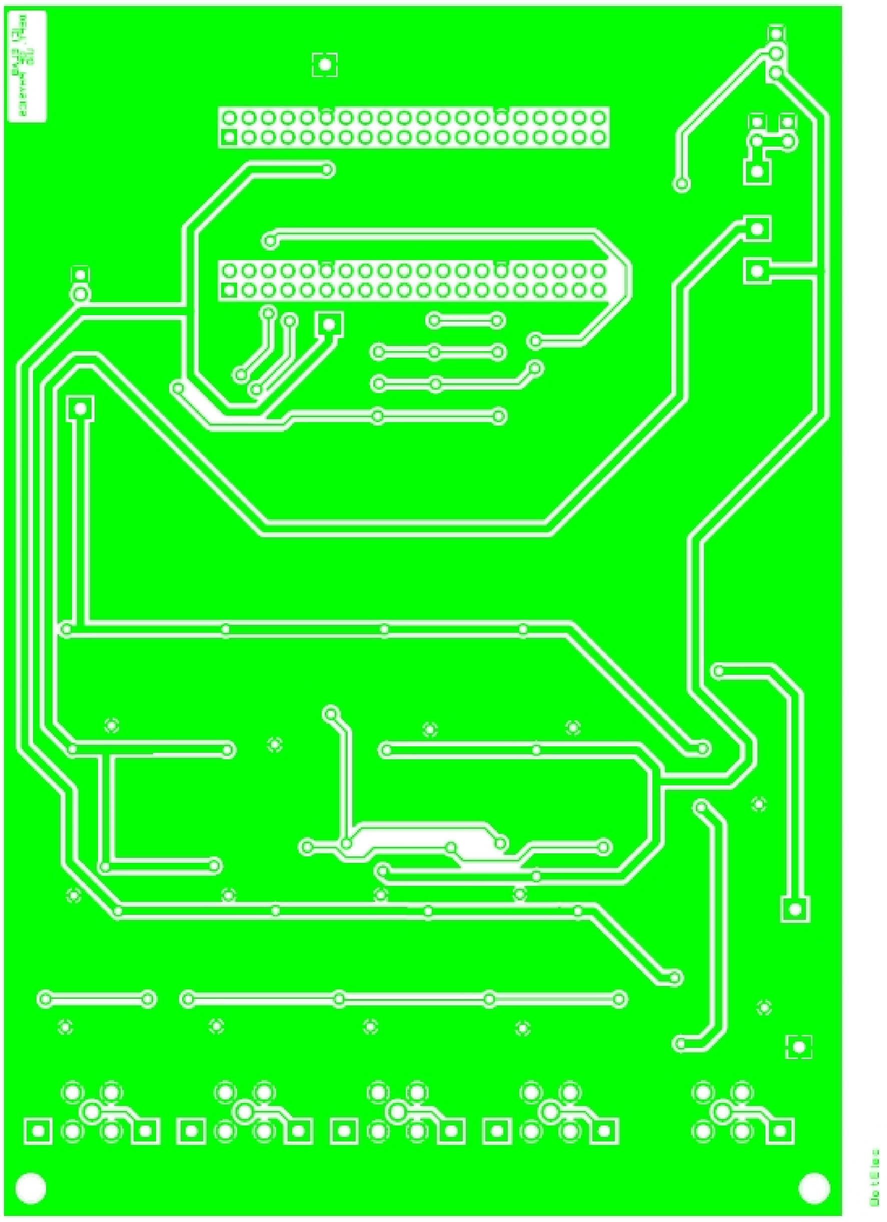


Figure B.12: DAQ PCB Bottom Layer

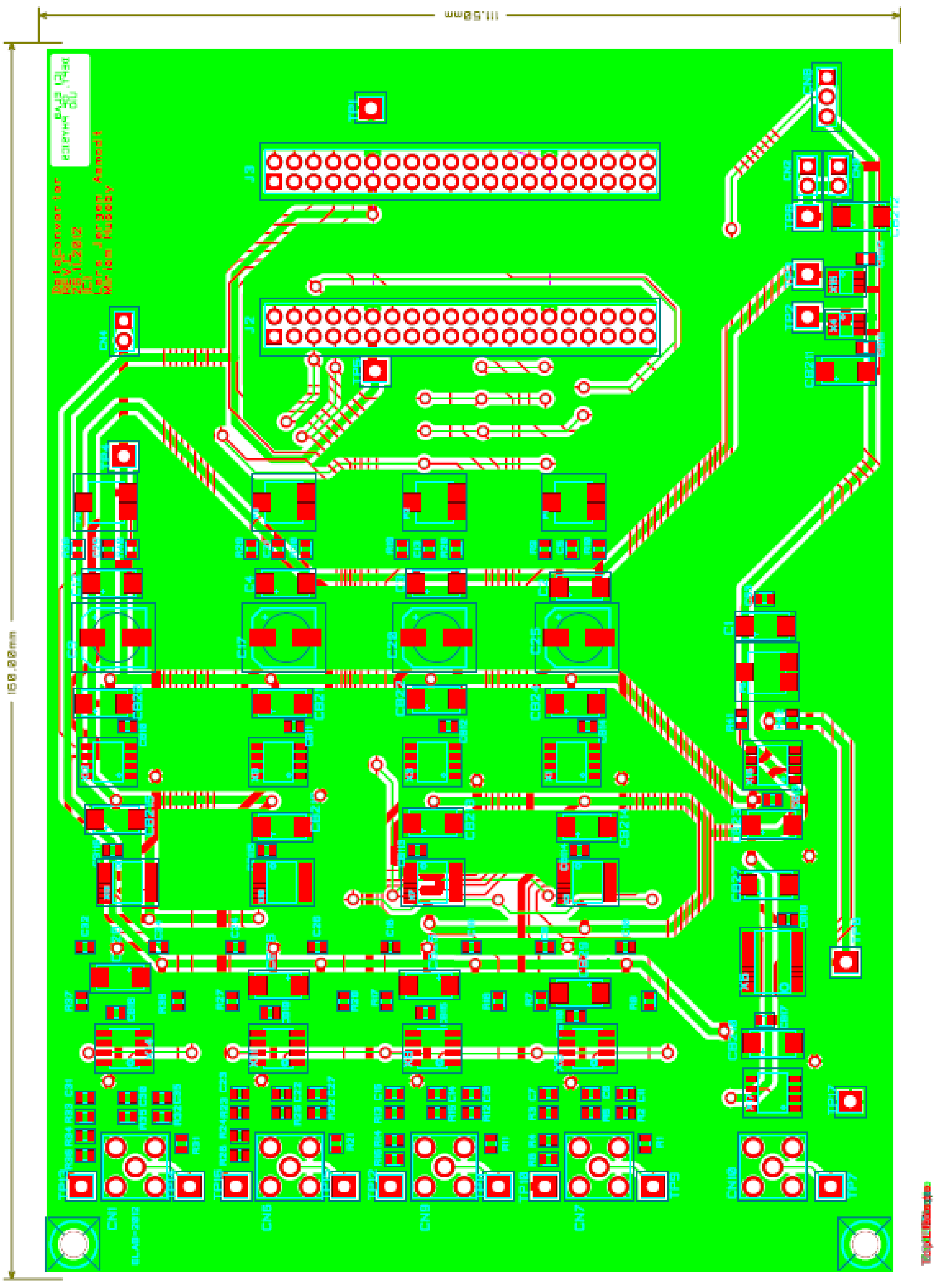


Figure B.13: DAQ PCB

Appendix C

Matlab Code

C.1 Program for generating ECG signal

Listing: MATLAB script for generating ECG signal

```
function [s, ipeaks] = ecgsyn(sfecg,M,Anoise,hrmean,hrstd,lfhfratio,sfint,ti,ai,bi)
% [s, ipeaks] = ecgsyn(sfecg,N,Anoise,hrmean,hrstd,lfhfratio,sfint,ti,ai,bi)
% Produces synthetic ECG with the following outputs:
% s: ECG (mV)
% ipeaks: labels for PQRST peaks: P(1), Q(2), R(3), S(4), T(5)
% A zero lablel is output otherwise ... use R=find(ipeaks==3);
% to find the R peaks s(R), etc.
%
% Operation uses the following parameters (default values in []s):
% sfecg: ECG sampling frequency [256 Hertz]
% M: approximate number of heart beats [256]
% Anoise: Additive uniformly distributed measurement noise [0 mV]
% hrmean: Mean heart rate [60 beats per minute]
% hrstd: Standard deviation of heart rate [1 beat per minute]
% lfhfratio: LF/HF ratio [0.5]
% sfint: Internal sampling frequency [256 Hertz]
% Order of extrema: [P Q R S T]
% ti = angles of extrema [-70 -15 0 15 100] degrees
% ai = z-position of extrema [1.2 -5 30 -7.5 0.75]
% bi = Gaussian width of peaks [0.25 0.1 0.1 0.1 0.4]
% Copyright (c) 2003 by Patrick McSharry & Gari Clifford, All Rights Reserved
% See IEEE Transactions On Biomedical Engineering, 50(3), 289-294, March 2003.
% Contact P. McSharry (patrick@mcsharry.net) or G. Clifford (gari@mit.edu)

%    This program is free software; you can redistribute it and/or modify
%    it under the terms of the GNU General Public License as published by
%    the Free Software Foundation; either version 2 of the License, or
%    (at your option) any later version.
%
%    This program is distributed in the hope that it will be useful,
%    but WITHOUT ANY WARRANTY; without even the implied warranty of
%    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
%    GNU General Public License for more details.
%
%    You should have received a copy of the GNU General Public License
%    along with this program; if not, write to the Free Software
%    Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
%
% ecgsyn.m and its dependents are freely availble from Physionet -
% http://www.physionet.org/ - please report any bugs to the authors above.

clc;
```

```
clear all
close all


% set parameter default values
if nargin < 1
   sfecg = 256;
end
if nargin < 2
   M =    10;%256;
end
if nargin < 3
   Anoise = 0;
end
if nargin < 4
   hrmean = 60;
end
if nargin < 5
   hrstd = 1;
end
if nargin < 6
   lfhfratio = 0.5;
end
if nargin < 7
   sfint = 512;
end
if nargin <8
    %     P  Q  R  S  T
   ti = [-70 -15 0 15 100];
end
% convert to radians
ti = ti*pi/180;
if nargin <9 % z position of attractor
       % P   Q   R   S   T
   ai = [1.2 -5 30 -7.5 0.75];
end
if nargin <10 % Gaussian width of each attractor
    %    P    Q    R    S    T
   bi = [0.25 0.1 0.1 0.1 0.4];
end

% adjust extrema parameters for mean heart rate
hrfact =  sqrt(hrmean/60);
hrfact2 = sqrt(hrfact);
bi = hrfact*bi;
ti = [hrfact2 hrfact 1 hrfact hrfact2].*ti;

% check that sfint is an integer multiple of sfecg
q = round(sfint/sfecg);
qd = sfint/sfecg;
if q ~= qd
   error(['Internal␣sampling␣frequency␣(sfint)␣must␣be␣an␣integer␣multiple␣' ...
'of␣the␣ECG␣sampling␣frequency␣(sfecg).␣Your␣current␣choices␣are:␣' ...
'sfecg␣=␣' int2str(sfecg) '␣and␣sfint␣=␣' int2str(sfint) '.']);
end

% define frequency parameters for rr process
% flo and fhi correspond to the Mayer waves and respiratory rate respectively
flo = 0.1;
fhi = 0.25;
flostd = 0.01;
fhistd = 0.01;

fid = 1;
fprintf(fid,'ECG␣sampled␣at␣%d␣Hz\n',sfecg);
fprintf(fid,'Approximate␣number␣of␣heart␣beats:␣%d\n',M);
fprintf(fid,'Measurement␣noise␣amplitude:␣%d␣\n',Anoise);
fprintf(fid,'Heart␣rate␣mean:␣%d␣bpm\n',hrmean);
fprintf(fid,'Heart␣rate␣std:␣%d␣bpm\n',hrstd);
fprintf(fid,'LF/HF␣ratio:␣%g\n',lfhfratio);
fprintf(fid,'Internal␣sampling␣frequency:␣%g\n',sfint);
fprintf(fid,'␣␣␣␣␣␣P␣␣Q␣␣R␣␣S␣␣T\n');
fprintf(fid,'ti␣=␣[%g␣%g␣%g␣%g␣%g]␣radians\n',ti(1),ti(2),ti(3),ti(4),ti(5));
fprintf(fid,'ai␣=␣[%g␣%g␣%g␣%g␣%g]\n',ai(1),ai(2),ai(3),ai(4),ai(5));
fprintf(fid,'bi␣=␣[%g␣%g␣%g␣%g␣%g]\n',bi(1),bi(2),bi(3),bi(4),bi(5));



% calculate time scales for rr and total output
sampfreqrr = 1;
trr = 1/sampfreqrr;
tstep = 1/sfecg;
rrmean = (60/hrmean);
Nrr = 2^(ceil(log2(M*rrmean/trr)));

% compute rr process
rr0 = rrprocess(flo,fhi,flostd,fhistd,lfhfratio,hrmean,hrstd,sampfreqrr,Nrr);

% upsample rr time series from 1 Hz to sfint Hz
rr = interp(rr0,sfint);

% make the rrn time series
dt = 1/sfint;
rrn = zeros(length(rr),1);
tecg=0;
i = 1;
while i <= length(rr)
   tecg = tecg+rr(i);
   ip = round(tecg/dt);
   rrn(i:ip) = rr(i);
   i = ip+1;
end
Nt = ip;

% integrate system using fourth order Runge-Kutta
fprintf(fid,'Integrating␣dynamical␣system\n');
x0 = [1,0,0.04];
Tspan = [0:dt:(Nt-1)*dt];
[T,X0] = ode45('derivsecgsyn',Tspan,x0,[],rrn,sfint,ti,ai,bi);

% downsample to required sfecg
X = X0(1:q:end,:);

% extract R-peaks times
ipeaks = detectpeaks(X, ti, sfecg);

% Scale signal to lie between -0.4 and 1.2 mV
z = X(:,3);
zmin = min(z);
zmax = max(z);
zrange = zmax - zmin;
z = (z - zmin)*(1.6)/zrange -0.4;

% include additive uniformly distributed measurement noise
eta = 2*rand(length(z),1)-1;
s = z + Anoise*eta;

%  plot(s)


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% MY CODE:
%
qrs = s;
fs = 200;                %sampling rate
M = length(qrs);         % Signal length
```

```
t = [0:M-1]/fs;          %time index
figure(1)
plot(t,qrs)
% plot(qrs)
xlabel('seconds');ylabel('mV');title('Input␣ECG␣Signal')

lp_qrs = lowpass(qrs);
hp_qrs = highpass(lp_qrs);
d_qrs = derive(hp_qrs);
s_qrs = squarer(d_qrs);
mi_qrs = mwi(s_qrs);
r_peaks = threshold(mi_qrs);
rate = hrv(r_peaks);

%END OF MY CODE
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function rr = rrprocess(flo, fhi, flostd, fhistd, lfhfratio, hrmean, hrstd, sfrr, n)
w1 = 2*pi*flo;
w2 = 2*pi*fhi;
c1 = 2*pi*flostd;
c2 = 2*pi*fhistd;
sig2 = 1;
sig1 = lfhfratio;
rrmean = 60/hrmean;
rrstd = 60*hrstd/(hrmean*hrmean);

df = sfrr/n;
w = [0:n-1]'*2*pi*df;
dw1 = w-w1;
dw2 = w-w2;

Hw1 = sig1*exp(-0.5*(dw1/c1).^2)/sqrt(2*pi*c1^2);
Hw2 = sig2*exp(-0.5*(dw2/c2).^2)/sqrt(2*pi*c2^2);
Hw = Hw1 + Hw2;
Hw0 = [Hw(1:n/2); Hw(n/2:-1:1)];
Sw = (sfrr/2)*sqrt(Hw0);

ph0 = 2*pi*rand(n/2-1,1);
ph = [ 0; ph0; 0; -flipud(ph0) ];
SwC = Sw .* exp(j*ph);
x = (1/n)*real(ifft(SwC));

xstd = std(x);
ratio = rrstd/xstd;
rr = rrmean + x*ratio;


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function ind = detectpeaks(X, thetap, sfecg)
N = length(X);
irpeaks = zeros(N,1);

theta = atan2(X(:,2),X(:,1));
ind0 = zeros(N,1);
for i=1:N-1
   a = ( (theta(i) <= thetap) & (thetap <= theta(i+1)) );
   j = find(a==1);
   if ~isempty(j)
      d1 = thetap(j) - theta(i);
      d2 = theta(i+1) - thetap(j);
      if d1 < d2
         ind0(i) = j;
      else
         ind0(i+1) = j;
      end
   end
end
```

```
d = ceil(sfecg/64);
d = max([2 d]); %UTSKRIFT
ind = zeros(N,1);
z = X(:,3);
zmin = min(z);
zmax = max(z);
zext = [zmin zmax zmin zmax zmin];
sext = [1 -1 1 -1 1];
for i=1:5
   clear ind1 Z k vmax imax iext;
   ind1 = find(ind0==i);
   n = length(ind1);
   Z = ones(n,2*d+1)*zext(i)*sext(i);
   for j=-d:d
      k = find( (1 <= ind1+j) & (ind1+j <= N) );
      Z(k,d+j+1) = z(ind1(k)+j)*sext(i);
   end
   [vmax, ivmax] = max(Z,[],2);
   iext = ind1 + ivmax-d-1;
   ind(iext) = i;
end
```

C.2 Matlab functions for Pan-Tompkins Algorithm

Listing: Lowpass filter

```
% Lowpass filter in Pan-Tompkins algorithm
% Author: Miriam Huseby
% Master Thesis 2013
% Reference: Faruk Uysal ; http://matlabz.blogspot.no/2011/04/contents-cancellation-dc-drift-and.ht

function lp_qrs = lowpass(qrs)


fs = 200;                 %sampling rate
N = length (qrs);         % Signal length
t = [0:N-1]/fs;           % time in seconds

% Low Pass Filter = (1/32) (1-z^-6)^2/(1-z^-1)^2
b = [1 0 0 0 0 -2 0 0 0 0 0 1]; %Numerator
a = [32 -64 32]; %Denominator
lp = filter(b,a,[1 zeros(1,12)]); % transfer function of LPF
lp_qrs = conv(qrs, lp);      %Convolves the vector
lp_qrs = lp_qrs(6+[1:N]); %cancle delay
lp_qrs = lp_qrs/ max( abs(lp_qrs )); % normalize , for convenience
figure(2)
plot(t, lp_qrs)
xlabel('seconds');ylabel('mV');title('ECG␣Signal␣after␣LPF')
```

Listing: Highpass filter

```
% Highpass filter in Pan-Tompkins algorithm
% Author: Miriam Huseby
% Master Thesis 2013
% Reference: Faruk Uysal ; http://matlabz.blogspot.no/2011/04/contents-cancellation-dc-drift-and.html

function hp_qrs = highpass(lp_qrs)


fs = 200;%256; %200;             %sampling rate
N = length (lp_qrs);          % Signal length
t = [0:N-1]/fs;               % time in seconds


%High Pass Filter
%Hp(z) = (-1 + z^-16 -z^-32)/(1-z^-1)

b = [-1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 32 -32 0 0 0 0 0 0 0 0 0 0 0 0 0 0 -1]; %Numerator
a = [32 -32];  %Denominator
hp = filter(b,a,[1 zeros(1,32)]); % transfer function of HPF
hp_qrs = conv(lp_qrs, hp);  %Convolves the vector
hp_qrs = hp_qrs/ max( abs(hp_qrs )); %Normalize
figure(3)
plot([0:length(hp_qrs)-1]/fs,hp_qrs)
xlabel('seconds');ylabel('mV');title('ECG␣Signal␣after␣HPF')
```

Listing: Derivative

```
% Derivative in Pan-Tompkins algorithm
% Author: Miriam Huseby
% Master Thesis 2013
% Reference: Faruk Uysal ; http://matlabz.blogspot.no/2011/04/contents-cancellation-dc-drift-and.ht

function d_qrs = derive(hp_qrs)


fs = 200;                  %sampling rate
N = length (hp_qrs);       %Signal length
t = [0:N-1]/fs;            %time index

% Derivative operator
% y(n) = 1/8[2x(n) + x(n-1) - x(n-3) - 2x(n-4)]

h = (1/8)*[2, 1, 0, -1, -2]; %Transfer function of the derivative
d_qrs = conv(hp_qrs, h); %convolution of the vector
d_qrs = d_qrs (2+[1: N]);
d_qrs = d_qrs/ max( abs(d_qrs )); %normalize
figure(4)
plot(t, d_qrs)
xlabel('seconds');ylabel('mV');title('ECG␣Signal␣after␣derivative␣operator')
```

Listing: Squaring process

```
% Squaring function in Pan-Tompkins algorithm
% Author: Miriam Huseby
% Master Thesis 2013
% Reference: Faruk Uysal ; http://matlabz.blogspot.no/2011/04/contents-cancellation-dc-drift-and.html

function s_qrs = squarer(d_qrs)


fs = 200;               %sampling rate
N = length (d_qrs);     % Signal length
t =  [0:N-1]/fs;        %time index


% Squaring the signal
% x^2
s_qrs = d_qrs .^2; %Squaring function
s_qrs = s_qrs/ max( abs(s_qrs )); %Normalize
figure(5)
plot(t, s_qrs)
xlabel('seconds');ylabel('mV');title('ECG␣Signal␣after␣squaring')
```

Listing: Moving window integrator

```
% Moving Window Integration in Pan-Tompkins algorithm
% Author: Miriam Huseby
% Master Thesis 2013
% Reference: Faruk Uysal ; http://matlabz.blogspot.no/2011/04/contents-cancellation-dc-drift-and.ht

function mi_qrs = mwi(s_qrs)

fs = 200;               %sampling rate
N = length (s_qrs);     % Signal length
t = [0:N-1]/fs;         % time index

% moving-window integration filter
% 30 samples long
h = ones (1 ,31)/31; %Transfer function
mi_qrs = conv (s_qrs ,h); %Convolutes the vector
mi_qrs = mi_qrs(15+[1: N]);
mi_qrs = mi_qrs/ max( abs(mi_qrs )); %Normalize

figure(6)
plot(t, mi_qrs)
xlabel('seconds');ylabel('mV');title('Result␣of␣Moving␣Window␣Integration')
```

Listing: Adaptive threshold and Peak detection

```
% Peak detection and calculate heart rate in Pan—Tompkins algorithm
% Author: Miriam Huseby
% Master Thesis 2013
function pks = threshold(mi_qrs)


fs = 200;                  %sampling rate
M = length (mi_qrs);       % Signal length
t  = [0:M—1]/fs;           %time index
%beats = [];

% Finds the peak of the integrated signal
h = ones (1 ,31)/31;
peak = max(mi_qrs);
thresh = 0.1;
y =mi_qrs;%
hb = 0;
l = length(y);
xx = (0:length(y)—1)/fs;

hh = length(h);
rrpos = [];

%Matlab function for peak detection in signal
[pks,locs] = findpeaks(y, 'minpeakdistance', hh, 'minpeakheight', thresh);
xx(locs);
figure(7), clf
subplot(2,1,1)
plot([0:length(y)—1]/fs,y)
%
hold on
plot(xx(locs),pks,'r*','markerfacecolor',[1 0 0])
xlabel('seconds');ylabel('mV');title('Peaks␣detected')


%Calculates beats pr minute

%Finds number of peaks
rrp = [];
 for i = 1:length(locs)—1
 rr1 = locs(i+1) — locs(i);
 rrp(i) = rr1;
 end

 %Finds the Average RR interval
 z = 1;
 hrv = [];
 len_of_rrp = length(rrp)
 while z + 8 <= len_of_rrp
      current_rrps = rrp(z:z+7)
      oo = mean(current_rrps);
      hrv(length(hrv)+1) = oo;
      z = z + 1;
 end

%Calculated beats pr minute, and plots is
beats = (hrv/fs)*60;
subplot(2,1,2)
K = length(beats);          % Signal length
t1 = linspace(0,max(t),K)
plot(t1, beats)
xlabel('seconds');ylabel('Beats␣pr␣Minute');title('Heart␣Rate')
```

Listing: Heart rate variability

```
% Calculate heart rate variability
% Author: Miriam Huseby
% Master Thesis 2013

function rate = hrv(r_peaks)


fs = 200;                 %sampling rate
N = length (r_peaks);     % Signal length
t =  [0:N-1]/fs;          % time index



%Calculates the heart rate variablity
rate = [];
for i = 1:N-1
    ra = r_peaks(i+1) - r_peaks(i);
    rate(i) = ra;
end

figure(8)
K = length(rate)          % Signal length
rate
t1 = linspace(0,K,K)
plot(t1, rate)
xlabel('seconds');ylabel('Difference␣in␣heart␣rate(minutes)');title('Heart␣Rate␣Variability')
```

Appendix D

VHDL Code

D.1 Testbench for Modelsim Simulation

Listing: Testbench for peak detection

```
-- Author:      Miriam Kirstine Huseby
-- Company:     University of Oslo
-- File name:   tb_detect.vhd
-- Date:        13.03.2013
-- Project:     Master thesis
-- Function:    testbench test for detection of ekg
-- From:        http://vhdlguru.blogspot.no/2011/06/vhdl-code-for-4-tap-fir-filter.html


library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
--use ieee.std_logic_unsigned.all;

entity tb_detect is
end tb_detect;

architecture beh of tb_detect is
    signal reset : std_logic := '0';
    signal clk : std_logic := '0';
    signal x_in : std_logic_vector(7 downto 0) := (others => '0');
    signal peak_value : std_logic_vector(7 downto 0) := (others => '0');
    signal threshold1 : std_logic_vector(7 downto 0) := (others => '0');
    signal threshold2 : std_logic_vector(7 downto 0) := (others => '0');
    signal beat_counter : std_logic_vector(9 downto 0) := (others => '0');
    signal peak_distance : std_logic_vector(9 downto 0) := (others => '0');


    constant clk_period : time := 500 ns;


    begin

    -- Instantiate the Unit Under Test (UUT)
    uut: entity work.FSM_detect port map(
        clk => clk,
        reset => reset,
        x_in => x_in,
        threshold2 => threshold2,
        beat_counter => beat_counter,
        threshold1 => threshold1,
      -- beatdist => beatdist,
 --    peak => peak ,
        peak_distance => peak_distance ,
        peak_value => peak_value

    );

    -- clock process definitions
    clk_process: process
    begin
        clk <= '0';
        wait for clk_period/2;
        clk <= '1';
        wait for clk_period/2;
    end process;

    -- Stimulus process
    stim_proc: process
    begin
        wait for clk_period*1;
        x_in <= "00000000";
        wait for clk_period*1;
        x_in <= "00000000";
        wait for clk_period*1;
        x_in <= "00000000";
        wait for clk_period*1;
        x_in <= "00000000";
        wait for clk_period*1;
        x_in <= "00000000";
        wait for clk_period*1;
        x_in <= "00000000";
        wait for clk_period*1;
        x_in <= "00000000";
        wait for clk_period*1;
        x_in <= "00000000";
        wait for clk_period*1;
        x_in <= "00000000";
        wait for clk_period*1;
        x_in <= "00000000";
        wait for clk_period*1;
        x_in <= "00000000";

        --------------------
        -- more input stimuli
        ---------------------
        wait for clk_period*1;
        x_in <= "00000000";

        wait;
    end process;
end beh;
```

Listing: Testbench for functions in the Pan-Tompkins Algorithm

```
-- Author:       Miriam Kirstine Huseby
-- Company:      University of Oslo
-- File name:    tb_lowpass.vhd
-- Date:         13.03.2013
-- Project:      Master thesis
-- Function:     teatbench 4 tap lowpass IIR filter
-- From:         http://vhdlguru.blogspot.no/2011/06/vhdl-code-for-4-tap-fir-filter.html


library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity tb_lowpass is
end tb_lowpass;

architecture beh of tb_lowpass is
    signal clk : std_logic := '0';
    signal x_in : std_logic_vector(7 downto 0) := (others => '0');
    signal y_out : std_logic_vector(15 downto 0) := (others => '0');
    constant clk_period : time := 500 ns;

    begin

    -- Instantiate the Unit Under Test (UUT)
    uut: entity work.lowpass port map(
        clk => clk,
        x_in => x_in,
        y_out => y_out
    );

    -- clock process definitions
    clk_process: process
    begin
        clk <= '0';
        wait for clk_period/2;
        clk <= '1';
        wait for clk_period/2;
    end process;

    -- Stimulus process
    stim_proc: process
    begin
        wait for clk_period*2;
        x_in <= "00000001";
        wait for clk_period*1;
        x_in <= "00000010";
        wait for clk_period*1;
        x_in <= "00000011";
        wait for clk_period*1;
        x_in <= "00000100";
        wait for clk_period*1;
        x_in <= "00000101";
        wait for clk_period*1;
        x_in <= "00000110";

        ------------
        -- more input
        ------------
        wait for clk_period*1;
        x_in <= "00001000";
        wait;
    end process;
end beh;
```

D.2 ECG Module

Listing: Clock Divider for 200Hz output

```
-- Author       : Miriam HUseby
-- Company      : University of Oslo
-- File name    : Clock160Div.vhd
-- Date         : 24.04.2013
-- Version      : 01
-- Project      : Master Thesis
-- Function     : Clock divider; Divides MCKL of 31.25kHz down to 200Hz
--                by dividing by 156

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity clock160div is
  port
    (
    clk_in  : in  std_logic;
    reset   : in  std_logic;
    clk_out : out std_logic
    );
end clock160div;

architecture clock_div_arch of clock160div is
    constant DivFactor : integer := 156;
    constant DivFactor_half : integer := 78;

    begin

-- Divide the input clock with DivFactor
    CLK_DIV:
    process (clk_in, reset)
    variable div_cnt : integer range 0 to DivFactor - 1;
    begin
        if reset = '1' then                  -- asynchronous reset
            div_cnt := 0;
            clk_out<= '0';
        elsif rising_edge(clk_in) then
            if (div_cnt = DivFactor - 1) then
                div_cnt := 0;
            else
                div_cnt := div_cnt + 1;
            end if;
            if (div_cnt >= DivFactor_half) then
                clk_out <= '0';
            else
                clk_out <= '1';
            end if;
        end if;
    end process CLK_DIV;
end clock_div_arch;
```

Listing: Finite State Machine for AD7766_A in DAQ

```
-- Author      : Lars JÃ¸rgen Aamodt
-- Company     : University of Oslo
-- File name   : AD7766.vhd
-- Date        : 15.08.2012
-- Project     : Master project
-- Function    : Control circuit for AD7766

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.std_logic_unsigned.all;

entity AD7766_A is

  port
    (
        reset                          : in                  std_logic;
        DRDY_n                         : in          std_logic;                        -- Data read
        SDO                                    : in                  std_logic;


        sclk                                   : in                  std_logic;

        SYNC_PD                        : out         std_logic;              -- Sync / powerdown
    CS_n                                       : out                 std_logic;
-- Chip select , active low

        V_drive                        : out         std_logic;
        disable                        : out         std_logic;
        DataReadyADC           : out                 std_logic;
        DataOut                                : out                 std_logic_vector(23 downto 0)



    );

end AD7766_A;

architecture AD7766_arch of AD7766_A is


signal bitcnt          : std_logic_vector(4 downto 0);
signal resetcnt        : std_logic_vector(11 downto 0);
signal shift_en        : std_logic;

--typedef.
type statetype is ( Init, PowerUp, ReadInit, Read_st, FinRead, Wait_st);

signal state : statetype;

begin


-- instantiate SR_Serin_redge
SRin_Pout_reg: entity work.SRin_Pout_reg
        port map (
                        clk      => sclk,
              DataIn   => SDO,
              shift_en=> shift_en,
              DataOut =>         DataOut
                        );

V_drive <= '1';
disable <= '0';
```

```
FSM_CONF_READ:
process(sclk, reset)
begin
  if (reset = '1') then
    state <= Init;


  elsif rising_edge(sclk) then
      -- set default values
            cs_n                    <= '1';
            sync_pd                                 <= '1';
            DataReadyADC            <= '0';
            shift_en                                <= '0';
            bitcnt                          <= (others => '0');
            resetcnt                                <= (others => '0');
            state <= Init;

    case state is

            when Init =>
                    sync_pd <= '0';
                    if (resetcnt= 4095) then
                            state <= PowerUp;
                    else
                            resetcnt <= resetcnt + 1;
                            state <= Init;
                    end if;

            when PowerUp =>
                    sync_pd <= '1';
                    state <= Wait_st;


            when  ReadInit =>

                    if drdy_n = '0' then
                            cs_n <= '0';
                            shift_en <= '1';
                            state <= Read_st;
                    else
                            state <= ReadInit;
                    end if;

            when  Read_st =>                                -- Reads 24 bits of valid data from
                    if (bitcnt = 23) then
                            cs_n <= '1';
                            shift_en <= '0';
                            state <= FinRead;
                    else
                            cs_n <= '0';
                            shift_en <= '1';
                            bitcnt <= bitcnt + 1;
                            state <= Read_st;
                    end if;


            when  FinRead =>
            -- Sets DataReady flag
                            shift_en <= '0';
                            DataReadyADC <= '1';
                            state <= Wait_st;

            when  Wait_st =>
                    if drdy_n = '1' then
                            state <= ReadInit;
                    else
```

```
                                state <= Wait_st;
                                --state <= ReadInit;
                        end if;

                when others =>
                        state <= Init;              -- Fault tolerance

    end case;

  end if;
end  process FSM_CONF_READ;

end AD7766_arch;
```

Listing: Register for AD7766. Serial in, Parallel out

```
-- Author      : Lars JÃ¸rgen Aamodt
-- Company     : University of Oslo
-- File name   : SR_SerIn_redge.vhd
-- Date        : 15.08.2012
-- Project     : Master project
-- Function    : Serial in parallel out shift register, rising edge.

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity SRin_Pout_reg is
  generic (
    width : integer := 24);

  port
    (
    clk       : in  std_logic;
    DataIn    : in  std_logic;
    shift_en  : in  std_logic;
    DataOut   : out std_logic_vector(width-1 downto 0)
    );

end SRin_Pout_reg;


architecture SRin_Pout_reg_arch of SRin_Pout_reg is
  signal data_int : std_logic_vector(width-1 downto 0);

begin

  SHIFT_REG:
  process (clk)
  begin

   if rising_edge(clk) then
      if (shift_en = '1') then
        data_int(0) <= DataIn;
        for i in 0 to width-2 loop
          data_int(i+1) <= data_int(i);
        end loop;
      end if;

    end if;
  end process SHIFT_REG;

  DataOut <= data_int;

end SRin_Pout_reg_arch;
```

Listing: Lowpass Filter

```
-- Author:      Miriam Kirstine Huseby
-- Company:     University of Oslo
-- File name:   lowpass.vhd
-- Date:        13.03.2013
-- Project:     Master thesis
-- Function:    Direct-Form I IIR lowpass filter
-- From:        http://vhdlguru.blogspot.no/2011/06/vhdl-code-for-4-tap-fir-filter.html


library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity lowpass is
    port(
        clk     : in std_logic;       -- clock signal
        x_in    : in signed(23 downto 0);   --input signal
        y_out   : out signed(28 downto 0)  -- filter output 29 bit
    );
end lowpass;

architecture beh of lowpass is

    component d_ff_lp is
        port(
            clk  : in std_logic;             -- clock input
            D    : in signed(47 downto 0);   -- data input from the mcm block
            Q    : out signed(47 downto 0) -- output connected to the adder
        );
    end component;

    signal H0, H1,H6, H12 : signed(23 downto 0) := (others => '0');
    signal M0, M1, M2, M3, M4, M5, M6,
           M7, M8, M9, M10, M11, M12 : signed(47 downto 0) := (others => '0');
    signal add1, add2, add3, add4, add5, add6,
           add7, add8, add9, add10, add11, add12 : signed(47 downto 0) := (others => '0');
    signal Q1, Q2, Q3, Q4, Q5, Q6, Q7,
           Q8, Q9, Q10, Q11, Q12, Y1, Y2 : signed(47 downto 0) := (others => '0');

    begin

-- filter coefficient initializations
-- y(n) = 2y(n-1)-y(n-2)+x(n)-2x(n-6)+x(n-12)
    H0 <= to_signed(1,24);
    H1 <= to_signed(0,24);
    H2 <= to_signed(0,24);
    H3 <= to_signed(0,24);
    H4 <= to_signed(0,24);
    H5 <= to_signed(0,24);
    H6 <= to_signed(-2,24);
    H7 <= to_signed(0,24);
    H8 <= to_signed(0,24);
    H9 <= to_signed(0,24);
    H10 <= to_signed(0,24);
    H11 <= to_signed(0,24);
    H12 <= to_signed(1,24);

--multiple constant multiplications
    M12 <= H12*x_in;
    M11 <= H1*x_in;
    M10 <= H1*x_in;
    M9 <= H1x_in;
    M8 <= H1*x_in;
    M7 <= H1*x_in;
    M6 <= H6*x_in;
    M5 <= H1*x_in;
    M4 <= H1*x_in;
    M3 <= H1*x_in;
    M2 <= H1*x_in;
    M1 <= H1*x_in;
    M0 <= H0*x_in;

-- adders
    add1 <= Q1 + M11;
    add2 <= Q2 + M10;
    add3 <= Q3 + M9;
    add4 <= Q4 + M8;
    add5 <= Q5 + M7;
    add6 <= Q6 + M6;
    add7 <= Q7 + M5;
    add8 <= Q8 + M4;
    add9 <= Q9 + M3;
    add10 <= Q10 + M2;
    add11 <= Q11 + M1;
    add12 <= Q12 + M0;


-- flip-flops (for introducing a delay)
    d_ff_lp1 : d_ff_lp
    port map(clk, M12, Q1);

    d_ff_lp2 : d_ff_lp
    port map(clk, add1, Q2);

    d_ff_lp3 : d_ff_lp
    port map(clk, add2, Q3);

    d_ff_lp4 : d_ff_lp
    port map(clk, add3, Q4);

    d_ff_lp5 : d_ff_lp
    port map(clk, add4, Q5);

    d_ff_lp6 : d_ff_lp
    port map(clk, add5, Q6);

    d_ff_lp7 : d_ff_lp
    port map(clk, add6, Q7);

    d_ff_lp8 : d_ff_lp
    port map(clk, add7, Q8);

    d_ff_lp9 : d_ff_lp
    port map(clk, add8, Q9);

    d_ff_lp10 : d_ff_lp
    port map(clk, add9, Q10);

    d_ff_lp11 : d_ff_lp
    port map(clk, add10, Q11);

    d_ff_lp12 : d_ff_lp
    port map(clk, add11, Q12);

-- an output produced at every positive edge of clock cycle
    process(clk)
    begin
        if(rising_edge(clk)) then
            Y1 <= add12 + Y2;
            Y2 <= (Y2+Y2) - Y1;
            y_out <= not Y2(28 downto 0);
        end if;
    end process;
end beh;
```

Listing: D Flip-Flop for Lowpass Filter

```
-- Author:       Miriam Kirstine Huseby
-- Company:      University of Oslo
-- File name:    d_ff_lp.vhd
-- Date:         13.03.2013
-- Project:      Master thesis
-- Function:     D Flip-Flop for LowPass IIR Filter
-- From:         http://vhdlguru.blogspot.no/2011/06/vhdl-code-for-4-tap-fir-filter.html


library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity d_ff_lp is
    port(
        clk  : in std_logic;          -- clock input
        D    : in signed(47 downto 0);   -- data input from the mcm block
        Q    : out signed(47 downto 0) -- output connected to the adder
    );
end d_ff_lp;

architecture beh of d_ff_lp is

    signal qt : signed(47 downto 0) := (others => '0');

    begin
    Q <= qt;
    d_ff_lp:
    process(clk)
    begin
        if(rising_edge(clk)) then
            qt <= D;
        end if;
    end process;
end beh;
```

Listing: Highpass Filter

```
-- Author:       Miriam Kirstine Huseby
-- Company:      University of Oslo
-- File name:    highpass.vhd
-- Date:         13.03.2013
-- Project:      Master thesis
-- Function:     Direct-Form I IIR Highpass filter
-- From:         http://vhdlguru.blogspot.no/2011/06/vhdl-code-for-4-tap-fir-filter.html


library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity highpass is
    port(
        clk     : in std_logic;      -- clock signal
        x_in    : in signed(28 downto 0);   --input signal
        y_out   : out signed(31 downto 0)  -- filter output 32 bit
    );
end highpass;

architecture beh of highpass is

    component d_ff_hp is
        port(
            clk  : in std_logic;            -- clock input
            D    : in signed(57 downto 0);   -- data input from the mcm block
            Q    : out signed(57 downto 0) -- output connected to the adder
        );
    end component;

    signal H0, H1, H16, H32 : signed(28 downto 0) := (others => '0');
    signal M0, M1, M16, M32 : signed(57 downto 0) := (others => '0');
    signal add1, add2, add3, add4, add5, add6, add7, add8, add9, add10, add11, add12,
           add13, add14, add15, add16, add17, add18, add19, add20, add21, add22, add23, add24,
           add25, add26, add27, add28, add29, add30, add31, add32 : signed(57 downto 0) := (others
    signal Q1, Q2, Q3, Q4, Q5, Q6, Q7, Q8, Q9, Q10, Q11, Q12, Q13, Q14, Q15,
           Q16, Q17, Q18, Q19, Q20, Q21, Q22, Q23, Q24, Q25, Q26,
           Q27, Q28, Q29, Q30, Q31, Q32 : signed(57 downto 0) := (others => '0');
    signal Y1 : signed(57 downto 0) := (others => '0');

    begin

-- filter coefficient initializations
-- y(n) = 2y(n-1)-y(n-2)+x(n)-2x(n-6)+x(n-12)
    H0 <= to_signed(-1,29);
    H1 <= to_signed(0,29);
    H16 <= to_signed(32,29);
    H32 <= to_signed(1,29);




--multiple constant multiplications
    M32 <= H32*x_in;
    M16 <= H16*x_in;
    M1 <= H1*x_in;
    M0 <= H0*x_in;




-- adders
    add1 <= Q1 + M1;
    add2 <= Q2 + M1;
    add3 <= Q3 + M1;
    add4 <= Q4 + M1;
    add5 <= Q5 + M1;
    add6 <= Q6 + M1;
    add7 <= Q7 + M1;
    add8 <= Q8 + M1;
    add9 <= Q9 + M1;
    add10 <= Q10 + M1;
    add11 <= Q11 + M1;
    add12 <= Q12 + M1;
    add13 <= Q13 + M1;
    add14 <= Q14 + M1;
    add15 <= Q15 + M1;
    add16 <= Q16 + M16;
    add17 <= Q17 + M1;
    add18 <= Q18 + M1;
    add19 <= Q19 + M1;
    add20 <= Q20 + M1;
    add21 <= Q21 + M1;
    add22 <= Q22 + M1;
    add23 <= Q23 + M1;
    add24 <= Q24 + M1;
    add25 <= Q25 + M1;
    add26 <= Q26 + M1;
    add27 <= Q27 + M1;
    add28 <= Q28 + M1;
    add29 <= Q29 + M1;
    add30 <= Q30 + M1;
    add31 <= Q31 + M1;
    add32 <= Q32 + M0;


-- flip-flops (for introducing a delay)
    d_ff_hp1 : d_ff_hp
    port map(clk, M32, Q1);

    d_ff_hp2 : d_ff_hp
    port map(clk, add1, Q2);

    d_ff_hp3 : d_ff_hp
    port map(clk, add2, Q3);

    d_ff_hp4 : d_ff_hp
    port map(clk, add3, Q4);

    d_ff_hp5 : d_ff_hp
    port map(clk, add4, Q5);

    d_ff_hp6 : d_ff_hp
    port map(clk, add5, Q6);

    d_ff_hp7 : d_ff_hp
    port map(clk, add6, Q7);

    d_ff_hp8 : d_ff_hp
    port map(clk, add7, Q8);

    d_ff_hp9 : d_ff_hp
    port map(clk, add8, Q9);

    d_ff_hp10 : d_ff_hp
    port map(clk, add9, Q10);

    d_ff_hp11 : d_ff_hp
    port map(clk, add10, Q11);

    d_ff_hp12 : d_ff_hp
    port map(clk, add11, Q12);
    d_ff_hp13 : d_ff_hp
    port map(clk, add12, Q13);

    d_ff_hp14 : d_ff_hp
    port map(clk, add13, Q14);

    d_ff_hp15 : d_ff_hp
    port map(clk, add14, Q15);

    d_ff_hp16 : d_ff_hp
    port map(clk, add15, Q16);

    d_ff_hp17 : d_ff_hp
    port map(clk, add16, Q17);

    d_ff_hp18 : d_ff_hp
    port map(clk, add17, Q18);

    d_ff_hp19 : d_ff_hp
    port map(clk, add18, Q19);

    d_ff_hp20 : d_ff_hp
    port map(clk, add19, Q20);

    d_ff_hp21 : d_ff_hp
    port map(clk, add20, Q21);

    d_ff_hp22 : d_ff_hp
    port map(clk, add21, Q22);

    d_ff_hp23 : d_ff_hp
    port map(clk, add22, Q23);

    d_ff_hp24 : d_ff_hp
    port map(clk, add23, Q24);

    d_ff_hp25 : d_ff_hp
    port map(clk, add24, Q25);

    d_ff_hp26 : d_ff_hp
    port map(clk, add25, Q26);

    d_ff_hp27 : d_ff_hp
    port map(clk, add26, Q27);

    d_ff_hp28 : d_ff_hp
    port map(clk, add27, Q28);

    d_ff_hp29 : d_ff_hp
    port map(clk, add28, Q29);

    d_ff_hp30 : d_ff_hp
    port map(clk, add29, Q30);

    d_ff_hp31 : d_ff_hp
    port map(clk, add30, Q31);

    d_ff_hp32 : d_ff_hp
    port map(clk, add31, Q32);


-- an output produced at every positive edge of clock cycle
    process(clk)
    begin
        if(rising_edge(clk)) then
            Y1 <= -Y1 + add32;
            y_out <= Y1(31 downto 0);
        end if;
    end process;
end beh;
```

Listing: D Flip-Flop for Highpass Filter

```
-- Author:       Miriam Kirstine Huseby
-- Company:      University of Oslo
-- File name:    d_ff_hp.vhd
-- Date:         13.03.2013
-- Project:      Master thesis
-- Function:     D Flip-Flop for HighPass IIR Filter
-- From:         http://vhdlguru.blogspot.no/2011/06/vhdl-code-for-4-tap-fir-filter.html


library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity d_ff_hp is
    port(
        clk  : in std_logic;           -- clock input
        D    : in signed(57 downto 0);   -- data input from the mcm block
        Q    : out signed(57 downto 0) -- output connected to the adder
    );
end d_ff_hp;

architecture beh of d_ff_hp is

    signal qt : signed(57 downto 0) := (others => '0');

    begin
    Q <= qt;
    d_ff_hp:
    process(clk)
    begin
        if(rising_edge(clk)) then
            qt <= D;
        end if;
    end process;
end beh;
```

Listing: Derivative

```
-- Author:        Miriam Kirstine Huseby
-- Company:       University of Oslo
-- File name:     deriv.vhd
-- Date:          13.03.2013
-- Project:       Master thesis
-- Function:      5 tap FIR filter for Derivative
-- From:          http://vhdlguru.blogspot.no/2011/06/vhdl-code-for-4-tap-fir-filter.html


library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity deriv is
    port(
        clk     : in std_logic;      -- clock signal
        x_in    : in signed(31 downto 0);   --input signal
        y_out   : out signed(30 downto 0)  -- filter output 31
    );
end deriv;

architecture beh of deriv is

    component d_ff_d is
        port(
            clk  : in std_logic;             -- clock input
            D    : in signed(63 downto 0);   -- data input from the mcm block
            Q    : out signed(63 downto 0) -- output connected to the adder
        );
    end component;

    signal H0, H1, H2, H3, H4 : signed(31 downto 0) := (others => '0');
    signal M0, M1, M2, M3, M4, M5 : signed(63 downto 0) := (others => '0');
    signal add1, add2, add3, add4 : signed(63 downto 0) := (others => '0');
    signal Q1, Q2, Q3, Q4 : signed(63 downto 0) := (others => '0');

    begin

-- filter coefficient initializations
-- H = 1/8[2 1 0 -1 -2]
    H0 <= to_signed(2,32);
    H1 <= to_signed(1,32);
    H2 <= to_signed(0,32);
    H3 <= to_signed(-1,32);
    H4 <= to_signed(-2,32);


--multiple constant multiplications
    M4 <= (H4)*x_in;
    M3 <= (H3)*x_in;
    M2 <= (H2)*x_in;
    M1 <= (H1)*x_in;
    M0 <= (H0)*x_in;

    M5 <= M4/8;

-- adders
    add1 <= Q1 + M3/8;
    add2 <= Q2 + M2/8;
    add3 <= Q3 + M1/8;
    add4 <= Q4 + M0/8;

-- flip-flops (for introducing a delay)
    d_ff_d1 : d_ff_d
    port map(clk, M5, Q1);
    d_ff_d2 : d_ff_d
    port map(clk, add1, Q2);

    d_ff_d3 : d_ff_d
    port map(clk, add2, Q3);

    d_ff_d4 : d_ff_d
    port map(clk, add3, Q4);


-- an output produced at every positive edge of clock cycle
    process(clk)
    begin
        if(rising_edge(clk)) then
            y_out <= not add4(30 downto 0);
        end if;
    end process;
end beh;
```

Listing: D Flip-Flop for Derivative

```
-- Author:      Miriam Kirstine Huseby
-- Company:     University of Oslo
-- File name:   d_ff_d.vhd
-- Date:        13.03.2013
-- Project:     Master thesis
-- Function:    D Flip-Flop for Derivative filter
-- From:        http://vhdlguru.blogspot.no/2011/06/vhdl-code-for-4-tap-fir-filter.html


library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity d_ff_d is
    port(
        clk  : in std_logic;            -- clock input
        D    : in signed(63 downto 0);   -- data input from the mcm block
        Q    : out signed(63 downto 0) -- output connected to the adder
    );
end d_ff_d;

architecture beh of d_ff_d is
    signal qt : signed(63 downto 0) := (others => '0');

    begin
    Q <= qt;
    d_ff_d:
    process(clk)
    begin
        if(rising_edge(clk)) then
            qt <= D;
        end if;
    end process;
end beh;
```

Listing: Squaring Operation

```
-- Author      : Miriam Kirstine Huseby
-- Company     : University of Oslo
-- File name   : squarer.vhd
-- Date        : 11.03.2013
-- Version     : 01
-- Project     : Master project
-- Function    : Squaring operation

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity squarer is
  port
    (
    clk               : in std_logic;
        data_in       : in signed(30 downto 0);
        data_out      : out signed(61 downto 0) -- 61 bit out
    );

end squarer;

architecture squarer_arch of squarer is
    signal product : signed(61 downto 0);
    signal squared : signed(30 downto 0);

    begin
    squared <= data_in;
    product <= squared*squared; -- multiplies the signal

    SQUARE:
    process(clk)
    begin
        if rising_edge(clk) then
            data_out <= product;
        end if;
    end process SQUARE;
end squarer_arch;
```

Listing: Moving Window Integration

```
-- Author:        Miriam Kirstine Huseby
-- Company:       University of Oslo
-- File name:     mov_int.vhd
-- Date:          13.03.2013
-- Project:       Master thesis
-- Function:      30 tap FIR filter for Moving window integrator
-- From:          http://vhdlguru.blogspot.no/2011/06/vhdl-code-for-4-tap-fir-filter.html


library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity mov_int is
    port(
        clk     : in std_logic;       -- clock signal
        x_in    : in signed(61 downto 0);   --input signal
        y_out   : out signed(63 downto 0)  -- filter output 64 bit
    );
end mov_int;

architecture beh of mov_int is

    component d_ff is
        port(
            clk  : in std_logic;             -- clock input
            D    : in signed(123 downto 0);   -- data input from the mcm block
            Q    : out signed(123 downto 0) -- output connected to the adder
        );
    end component;

    signal H0, H1, H2, H3, H4, H5, H6, H7, H8, H9, H10, H11, H12, H13, H14, H15, H16, H17, H18, H19,
           H20, H21, H22, H23, H24, H25, H26, H27, H28, H29 : signed(61 downto 0) := (others => '0');
    signal M0, M1, M2, M3, M4, M5, M6, M7, M8, M9, M10, M11, M12, M13, M14, M15, M16, M17, M18, M19,
           M20, M21, M22, M23, M24, M25, M26, M27, M28, M29 : signed(123 downto 0) := (others => '0');
    signal add1, add2, add3, add4, add5, add6, add7, add8, add9, add10, add11, add12, add13, add14, add15,
           add16, add17, add18, add19, add20, add21, add22, add23, add24, add25, add26, add27, add28,
           add29 : signed(123 downto 0) := (others => '0');
    signal Q1, Q2, Q3, Q4, Q5, Q6, Q7, Q8, Q9, Q10, Q11, Q12, Q13, Q14, Q15, Q16, Q17, Q18, Q19,
           Q20, Q21, Q22, Q23, Q24, Q25, Q26, Q27, Q28, Q29 : signed(123 downto 0) := (others => '0');

    begin

-- filter coefficient initializations
-- H = [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
    H0 <= to_signed(1,62);

--multiple constant multiplications
    M0 <= H0*x_in;

-- adders
    add1 <= Q1 + M0;
    add2 <= Q2 + M0;
    add3 <= Q3 + M0;
    add4 <= Q4 + M0;
    add5 <= Q5 + M0;
    add6 <= Q6 + M0;
    add7 <= Q7 + M0;
    add8 <= Q8 + M0;
    add9 <= Q9 + M0;
    add10 <= Q10 + M0;
    add11 <= Q11 + M0;
    add12 <= Q12 + M0;
    add13 <= Q13 + M0;
    add14 <= Q14 + M0;
    add15 <= Q15 + M0;
    add16 <= Q16 + M0;
    add17 <= Q17 + M0;
    add18 <= Q18 + M0;
    add19 <= Q19 + M0;
    add20 <= Q20 + M0;
    add21 <= Q21 + M0;
    add22 <= Q22 + M0;
    add23 <= Q23 + M0;
    add24 <= Q24 + M0;
    add25 <= Q25 + M0;
    add26 <= Q26 + M0;
    add27 <= Q27 + M0;
    add28 <= Q28 + M0;
    add29 <= Q29 + M0;

-- flip-flops (for introducing a delay)
    d_ff1 : d_ff
    port map(clk, M29, Q1);

    d_ff2 : d_ff
    port map(clk, add1, Q2);

    d_ff3 : d_ff
    port map(clk, add2, Q3);

    d_ff4 : d_ff
    port map(clk, add3, Q4);

    d_ff5 : d_ff
    port map(clk, add4, Q5);

    d_ff6 : d_ff
    port map(clk, add5, Q6);

    d_ff7 : d_ff
    port map(clk, add6, Q7);

    d_ff8 : d_ff
    port map(clk, add7, Q8);

    d_ff9 : d_ff
    port map(clk, add8, Q9);

    d_ff10 : d_ff
    port map(clk, add9, Q10);

    d_ff11 : d_ff
    port map(clk, add10, Q11);

    d_ff12 : d_ff
    port map(clk, add11, Q12);

    d_ff13 : d_ff
    port map(clk, add12, Q13);

    d_ff14 : d_ff
    port map(clk, add13, Q14);

    d_ff15 : d_ff
    port map(clk, add14, Q15);

    d_ff16 : d_ff
    port map(clk, add15, Q16);

    d_ff17 : d_ff
    port map(clk, add16, Q17);

    d_ff18 : d_ff
    port map(clk, add17, Q18);

    d_ff19 : d_ff
    port map(clk, add18, Q19);

    d_ff20 : d_ff
    port map(clk, add19, Q20);

    d_ff21 : d_ff
    port map(clk, add20, Q21);

    d_ff22 : d_ff
    port map(clk, add21, Q22);

    d_ff23 : d_ff
    port map(clk, add22, Q23);

    d_ff24 : d_ff
    port map(clk, add23, Q24);

    d_ff25 : d_ff
    port map(clk, add24, Q25);

    d_ff26 : d_ff
    port map(clk, add25, Q26);

    d_ff27 : d_ff
    port map(clk, add26, Q27);

    d_ff28 : d_ff
    port map(clk, add27, Q28);

    d_ff29 : d_ff
    port map(clk, add28, Q29);


-- an output produced at every positive edge of clock cycle
    process(clk)
    begin
        if(rising_edge(clk)) then
            y_out <= add29(63 downto 0);
        end if;
    end process;
end beh;
```

Listing: D Flip-Flop for Moving Window Integration

```
-- Author:        Miriam Kirstine Huseby
-- Company:       University of Oslo
-- File name:     d_ff.vhd
-- Date:          13.03.2013
-- Project:       Master thesis
-- Function:      D Flip-Flop for Moving window integrator
-- From:          http://vhdlguru.blogspot.no/2011/06/vhdl-code-for-4-tap-fir-filter.html


library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity d_ff is
    port(
        clk  : in std_logic;            -- clock input
        D    : in signed(123 downto 0);    -- data input from the mcm block
        Q    : out signed(123 downto 0) -- output connected to the adder
    );
end d_ff;

architecture beh of d_ff is
    signal qt : signed(123 downto 0) := (others => '0');

    begin
    Q <= qt;
    d_ff:
    process(clk)
    begin
        if(rising_edge(clk)) then
            qt <= D;
        end if;
    end process;
end beh;
```

Listing: Peak Detection

```
-- Author       : Miriam Kirstine Huseby
-- Company      : University of Oslo
-- File name    : peak_detect.vhd
-- Date         : 02.04.2013
-- Project      : Master project
-- Function     : Peak detection.

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity peak_detect is
  port
    (
        clk      : in  std_logic;
        x_in     : in  signed(63 downto 0);
        y_out    : out unsigned(8 downto 0)
    );
end peak_detect;

architecture peak_detect_arch of peak_detect is
    signal data_int, data_q : signed(63 downto 0) := (others => '0');
    signal data_max : signed(63 downto 0) := (others => '0');
    signal thr1: signed(63 downto 0);

begin

-- Finds the first threshold for the incoming signal based on the first 400 samples.
    threshold:
    process(clk)
        variable cnt :integer := 0;
    begin
                if (rising_edge(clk)) then
            data_int <= x_in;
            if(cnt = 400) then --400
                thr1 <= data_max/3;
            else
                if(data_max < data_int) then
                    data_max <= data_int;
                else
                    data_max <= data_max;
                end if;
                cnt := cnt + 1;
            end if;
        end if;
    end process threshold;

    peak:
    process(clk)
    variable peakPos :integer := 0;
    variable peakDistance :integer := 0;
    variable totalDistance :integer := 0;
    variable beat :integer := 0;
    variable beatDistance :integer := 0;
        variable thr2 : signed(63 downto 0) := thr1;

    begin
        if(rising_edge(clk)) then

            beat := beat + 1;
            if(x_in > thr2) then -- Input is higher than treshold so we start checking for peaks
                if(x_in > data_q) then -- This is a possible peak, but it is not yet confirmed
                    data_q <= x_in;
                    peakPos := beat; -- We set the peak position to the current beat every time
                end if;             --- we reach a value higher then the previous value
            end if;
            if (thr2 >= x_in and data_q >= thr2) then
                if (peakPos > 0) then
                    peakDistance := beat - peakPos;
                    beatDistance := beat - peakDistance;
                    totalDistance := beatDistance + peakDistance;
                    if(totalDistance > 50) then
                        beat := 1; -- Resets the beat counter
                                                peakPos := 0; -- Set the peak distance to zero
                                                thr2 := data_q/3;  -- Adaptive threshold
                                                data_q <= (others => '0'); -- Reset the data_q
                                                y_out <= to_unsigned(totalDistance, 9); --output
                                        else
                                                totalDistance := totalDistance + 1;
                                                data_q <= (others => '0');
                                        end if;
                end if;
            end if;
        end if;
    end process peak;

end peak_detect_arch;
```

```
-- Author:        Miriam Kirstine Huseby
-- Company:       University of Oslo
-- File name:     heartrate.vhd
-- Date:          13.03.2013
-- Project:       Master thesis
-- Function:      8 tap FIR filter for Moving window integrator
-- From:          http://vhdlguru.blogspot.no/2011/06/vhdl-code-for-4-tap-fir-filter.html


library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity heartrate is
    port(
        clk     : in std_logic;      -- clock signal
        x_in    : in unsigned(8 downto 0);   --input signal from peak_detector
        y_out   : out unsigned(31 downto 0)  -- filter output 18 bit, satt opp til 32 bit for float conversi
    );
end heartrate;

architecture beh of heartrate is

    component d_ff_hr is
        port(
            clk  : in std_logic;            -- clock input
            D    : in unsigned(31 downto 0);   -- data input from the mcm block
            Q    : out unsigned(31 downto 0) -- output connected to the adder
        );
    end component;

    signal H0 : unsigned(22 downto 0) := (others => '0');
    signal M0, M1, M2, M3, M4, M5, M6, M7 : unsigned(31 downto 0) := (others => '0');
    signal add1, add2, add3, add4, add5, add6, add7 : unsigned(31 downto 0) := (others => '0');
    signal Q1, Q2, Q3, Q4, Q5, Q6, Q7 : unsigned(31 downto 0) := (others => '0');

    begin

-- filter coefficient initializations
-- H = [1 1 1 1 1 1 1 1]
    H0 <= to_unsigned(1,23);

--multiple constant multiplications
    M7 <= H0*x_in;
    M6 <= H0*x_in;
    M5 <= H0*x_in;
    M4 <= H0*x_in;
    M3 <= H0*x_in;
    M2 <= H0*x_in;
    M1 <= H0*x_in;
    M0 <= H0*x_in;

-- adders
    add1 <= Q1 + M6;
    add2 <= Q2 + M5;
    add3 <= Q3 + M4;
    add4 <= Q4 + M3;
    add5 <= Q5 + M2;
    add6 <= Q6 + M1;
    add7 <= Q7 + M0;


-- flip-flops (for introducing a delay)
d_ff1 : d_ff_hr
port map(clk, M7, Q1);

d_ff2 : d_ff_hr
port map(clk, add1, Q2);
```

```
d_ff3 : d_ff_hr
port map(clk, add2, Q3);

d_ff4 : d_ff_hr
port map(clk, add3, Q4);

d_ff5 : d_ff_hr
port map(clk, add4, Q5);

d_ff6 : d_ff_hr
port map(clk, add5, Q6);

d_ff7 : d_ff_hr
port map(clk, add6, Q7);




-- an output produced at every positive edge of clock cycle
    process(clk)
    begin
        if(rising_edge(clk)) then
            y_out <= add7;
        end if;
    end process;
end beh;
```

```
-- Author:      Miriam Kirstine Huseby
-- Company:     University of Oslo
-- File name:   d_ff_hr.vhd
-- Date:        07.05.2013
-- Project:     Master thesis
-- Function:    D Flip-Flop for Moving window integrator to determine heartrate
-- From:        http://vhdlguru.blogspot.no/2011/06/vhdl-code-for-4-tap-fir-filter.html


library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity d_ff_hr is
    port(
        clk  : in std_logic;            -- clock input
        D    : in unsigned(31 downto 0);   -- data input from the mcm block
        Q    : out unsigned(31 downto 0) -- output connected to the adder
    );
end d_ff_hr;

architecture beh of d_ff_hr is
    signal qt : unsigned(31 downto 0) := (others => '0');

    begin
    Q <= qt;
    d_ff_hr:
    process(clk)
    begin
        if(rising_edge(clk)) then
            qt <= D;
        end if;
    end process;
end beh;
```

Appendix E

iPad Application Code

E.1 Python script for ECG Server

Listing: TCP/IP Server for generating data to iPad app

```
# Server to transmit simulated biomedical signals to iPad application
# Project: Master Thesis
# Written by Jonas Hagstedt & Miriam Huseby
# Date: 28.04.2013

import random
from gevent import server
from gevent import event
from gevent.monkey import patch_all; patch_all()
import gevent
from multiprocessing import Process
import os
from datetime import datetime


class EkgClient(object):
    def __init__(self, socket, address, server_handler):
        self.socket = socket
        self.address = address
        self.server_handler = server_handler
        gevent.spawn(self.listen)

    def listen(self):
        f = self.socket.makefile()
        while True:
            line = f.readline()
            if not line:
                print 'client␣died'
                del self.server_handler.users[self.address]
                break
            line = line.rstrip()
            for addr, c in self.server_handler.users.iteritems():
                msg = '<%s>␣says:␣%s' % (self.address, line)
                c.send(msg)

            print '<%s>:␣%s' % (self.address, line)

    def send(self, msg):
        f = self.socket.makefile()
        f.write('%s\r\n' % msg)
        f.flush()
```

```
class EkgServerHandler(object):
    def __init__(self):
        self.users = {}
        self.generate_metrics_event = event.Event()
        gevent.spawn_later(1, self.generate_metric_data)
        self.command_list = ['RANDOM', 'MAX']
        self.command = 'RANDOM'

    def __call__(self, socket, address):
        client = EkgClient(socket, address, self)
        self.users[address] = client
        self.socket = socket
        self.address = address
        print 'connection␣made'

    def set_command(self, cmd):
        if cmd in self.command_list:
            self.command = cmd
        print cmd

    def generate_metric_data(self):
        print 'starting␣ekg'
        while True:
            data = '%s␣%s␣%s␣%s' % (
                self.generate_ekg_data(),
                self.generate_conductance_data(),
                self.generate_potential_data(),
                self.generate_susceptance_data()
            )
            self.send_data(data)

            self.generate_metrics_event.wait(0.3)

    def generate_ekg_data(self):
            ekg_data = [str(int(17*random.random() + 58)) for i in xrange(1)]
            t = datetime.now().time()
            ekg_data = [str(t.second + 50) for i in xrange(11)]
            ekg_data = 'E%s' % ','.join(ekg_data)
            return ekg_data

    def generate_conductance_data(self):
        con_data = [str(int(32.5*random.random() + 37.5)) for i in xrange(1)]
        # con_data = ['103' for i in xrange(11)]
        con_data = 'C%s' % ','.join(con_data)
        return con_data

    def generate_susceptance_data(self):
        sus_data = [str(int(4*random.random() + 18)) for i in xrange(1)]
        # sus_data = ['70' for i in xrange(11)]
        sus_data = 'S%s' % ','.join(sus_data)
        return sus_data

    def generate_potential_data(self):
        pot_data = [str(int(23*random.random() + -20)) for i in xrange(1)]
        # pot_data = ['-10' for i in xrange(11)]
        pot_data = 'P%s' % ','.join(pot_data)
        return pot_data

    def send_data(self, data):
        for addr, c in self.users.iteritems():
            c.send(data)
#           print 'sent to %s clients' % len(self.users)


def clear_screen():
    os.system(['clear', 'cls'][os.name == 'nt'])
```

```
def handle_command(cmd):
    clear_screen()
    print cmd
    server.handle.set_command(cmd)

    print('\r\n')
    print('Command␣list:\r\n')
    print('\r\n')

if __name__ == '__main__':
    server = server.StreamServer(('0.0.0.0', 5000), EkgServerHandler())
    p = Process(target = server.serve_forever)
    p.start()
    cmd = raw_input()
    while cmd != 'q':
        cmd = raw_input()
        handle_command(cmd)
```

E.2 iPad Application

Listing: main file

```
//
//  main.m
//  BioPlot
//
//  Created by Jonas Hagstedt on 27/04/2013.
//  Copyright __MyCompanyName__ 2013. All rights reserved.
//

#import <UIKit/UIKit.h>

int main(int argc, char *argv[]) {

    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
    int retVal = UIApplicationMain(argc, argv, nil, @"AppController");
    [pool release];
    return retVal;
}
```

Listing: IntroLayer header file

```
//
//  IntroLayer.h
//  BioPlot
//
//  Created by Jonas Hagstedt on 27/04/2013.
//  Copyright __MyCompanyName__ 2013. All rights reserved.
//


// When you import this file, you import all the cocos2d classes
#import "cocos2d.h"

// HelloWorldLayer
@interface IntroLayer : CCLayer
{
}

// returns a CCScene that contains the HelloWorldLayer as the only child
+(CCScene *) scene;

@end
```

Listing: IntroLayer m file

```
//
//  IntroLayer.m
//  BioPlot
//
//  Created by Jonas Hagstedt on 27/04/2013.
//  Copyright __MyCompanyName__ 2013. All rights reserved.
//


// Import the interfaces
#import "IntroLayer.h"
#import "HelloWorldLayer.h"


#pragma mark - IntroLayer

// HelloWorldLayer implementation
@implementation IntroLayer

// Helper class method that creates a Scene with the HelloWorldLayer as the only child.
+(CCScene *) scene
{
        // 'scene' is an autorelease object.
        CCScene *scene = [CCScene node];

        // 'layer' is an autorelease object.
        IntroLayer *layer = [IntroLayer node];

        // add layer as a child to scene
        [scene addChild: layer];

        // return the scene
        return scene;
}

//
-(void) onEnter
{
        [super onEnter];

        // ask director for the window size
        CGSize size = [[CCDirector sharedDirector] winSize];
        CCSprite *background;

        if( UI_USER_INTERFACE_IDIOM() == UIUserInterfaceIdiomPhone ) {
                background = [CCSprite spriteWithFile:@"Default.png"];
                background.rotation = 90;
        } else {
                background = [CCSprite spriteWithFile:@"Default-Landscape~ipad.png"];
        }
        background.position = ccp(size.width/2, size.height/2);

        // add the label as a child to this Layer
        [self addChild: background];

        // In one second transition to the new scene
        [self scheduleOnce:@selector(makeTransition:) delay:0.1];
}

-(void) makeTransition:(ccTime)dt
{
        [[CCDirector sharedDirector] replaceScene:[CCTransitionFade transitionWithDuration:1.0 scer
}
@end
```

Listing: HelloWorldLayer header file

```
//
//  HelloWorldLayer.h
//  BioPlot
//
//  Created by Jonas Hagstedt on 27/04/2013.
//  Copyright __MyCompanyName__ 2013. All rights reserved.
//


#import <GameKit/GameKit.h>

// When you import this file, you import all the cocos2d classes
#import "cocos2d.h"
#import "Connectivity.h"
#import "GraphDetail.h"
#import "GetSettings.h"

// HelloWorldLayer
@interface HelloWorldLayer : CCLayer <ConnectionDelegate> {
    NSTimer *timer;
    NSMutableArray *newData;
    NSMutableDictionary *graphs;
    Connectivity *connection;

    CCLabelTTF* ekgLabelVal;
    CCLabelTTF* conLabelVal;
    CCLabelTTF* susLabelVal;
    CCLabelTTF* potLabelVal;
}

// returns a CCScene that contains the HelloWorldLayer as the only child
+(CCScene *) scene;
- (void)receivedNewData:(NSArray *)someData forGraph:(NSString*)graphId;

@end
```

Listing: HelloWorldLayer m file

```
//
//  HelloWorldLayer.m
//  DrawLineNonsense
//
//  Created by Jonas Hagstedt & Miriam Huseby on 27/04/2013.
//  Copyright __MyCompanyName__ 2013. All rights reserved.
//


// Import the interfaces
#import "HelloWorldLayer.h"

// Needed to obtain the Navigation Controller
#import "AppDelegate.h"

#pragma mark - HelloWorldLayer

// HelloWorldLayer implementation
@implementation HelloWorldLayer

// Helper class method that creates a Scene with the HelloWorldLayer as the only child.
+(CCScene *) scene
{
        // 'scene' is an autorelease object.
        CCScene *scene = [CCScene node];

        // 'layer' is an autorelease object.
        HelloWorldLayer *layer = [HelloWorldLayer node];
        // add layer as a child to scene
        [scene addChild: layer];

        // return the scene
        return scene;
}

static int pointCount = 0;
static int screenMiddle = 0;
static CGSize winSize;

-(id) init
{
        if( (self=[super init]) ) {
        winSize = [[CCDirector sharedDirector] winSize];
        pointCount = winSize.width;
        screenMiddle = winSize.height / 2;
        newData = [NSMutableArray new];
        graphs = [[NSMutableDictionary new] retain];

        GraphDetail *ekg = [[GraphDetail alloc] initWithName:@"ekg" andVisibleBufferSize:kVisibleBu
        ekg.r = 255;
        ekg.g = 255;
        ekg.b = 255;
        ekg.shouldShowLabel = YES;
        ekg.labelFromVal = 50;
        ekg.labelToVal = 160;
        ekg.zeroPointPosition = -40;
        ekg.labelPointIncrement = 10;
        ekg.labelText = @"Heartrate:␣%.1f␣bpm";
        for (int i = 0; i < kDrawBufferSize; i++) {
            [ekg.graphData addObject:[NSValue valueWithCGPoint:CGPointMake(i*kSpeedMultiplier + kL
        }
        [graphs setObject:ekg forKey:ekg.graphId];
        [ekg release];

        GraphDetail *conductance = [[GraphDetail alloc] initWithName:@"conductance" andVisibleBuffe
        conductance.r = 0;
        conductance.g = 255;
        conductance.b = 0;
        conductance.shouldShowLabel = YES;
        conductance.labelFromVal = -50;
        conductance.labelToVal = 125;
        conductance.zeroPointPosition = 400;
        conductance.labelPointIncrement = 17.5;
        conductance.labelText = @"Conductance:␣0.0␣uS";

        for (int i = 0; i < kDrawBufferSize; i++) {
            [conductance.graphData addObject:[NSValue valueWithCGPoint:CGPointMake(i*kSpeedMultipl
        }
        [graphs setObject:conductance forKey:conductance.graphId];
        [conductance release];

        GraphDetail *susceptance = [[GraphDetail alloc] initWithName:@"susceptance" andVisibleBuffe
        susceptance.r = 30;
        susceptance.g = 30;
        susceptance.b = 255;
        susceptance.shouldShowLabel = NO;
        susceptance.labelFromVal = -50;
        susceptance.labelToVal = 125;
        susceptance.zeroPointPosition = 400;
        susceptance.labelPointIncrement = 17.5;
        susceptance.labelText = @"wa:␣0.0␣uS";

        for (int i = 0; i < kDrawBufferSize; i++) {
            [susceptance.graphData addObject:[NSValue valueWithCGPoint:CGPointMake(i*kSpeedMultipl
        }
        [graphs setObject:susceptance forKey:susceptance.graphId];
        [susceptance release];

        GraphDetail *potential = [[GraphDetail alloc] initWithName:@"potential" andVisibleBufferSize:kVisib
        potential.r = 255;
        potential.g = 0;
        potential.b = 0;
        potential.shouldShowLabel = YES;
        potential.labelRightAlign = YES;
        potential.labelFromVal = -55;
        potential.labelToVal = 20;
        potential.zeroPointPosition = 400;
        potential.labelPointIncrement = 7.5;
        potential.labelText = @"ro:␣0.0␣uS";

        for (int i = 0; i < kDrawBufferSize; i++) {
            [potential.graphData addObject:[NSValue valueWithCGPoint:CGPointMake(i*kSpeedMultiplier + kLeftF
        }
        [graphs setObject:potential forKey:potential.graphId];
        [potential release];

        // create and initialize labels
        [self drawLabels];

        [self scheduleUpdate];

        connection = [Connectivity new];
        connection.delegate = self;
        GetSettings *gs = [[GetSettings alloc] init];
        [connection connect:[gs getHost] port:[gs getPort]];
        [gs release];
//          [connection connect:@"192.168.1.104" port:5000];
//          timer = [NSTimer scheduledTimerWithTimeInterval:0.3 target:self selector:@selector(timerFired) us
        }
        return self;
}

- (void)timerFired {
    float r = getRandomSmall();
    float val = getRandom();
    NSMutableString *dataString = [NSMutableString stringWithString:@"E"];
    for (int i = 0; i < r; i++) {
        val = 150;
        if (i < r-1) {
            [dataString appendString:[NSString stringWithFormat:@"%f,", val]];
        } else {
            [dataString appendString:[NSString stringWithFormat:@"%f", val]];
        }
    }
    [dataString appendString:@"␣C"];
    for (int i = 0; i < r; i++) {
        val = 100;
        if (i < r-1) {
            [dataString appendString:[NSString stringWithFormat:@"%f,", val]];
        } else {
            [dataString appendString:[NSString stringWithFormat:@"%f", val]];
        }
    }
    [self dataAvailable:dataString];
}

// This happens when the socket receives new data (or when test timer has done it's job)
- (void)dataAvailable:(NSString *)data {
    NSArray *rawGraphs = [data componentsSeparatedByString:@"␣"];

    for (NSString *graphString in rawGraphs) {
        NSString *graphId = [graphString substringToIndex:1];
        graphString = [graphString substringFromIndex:1];
        for (id key in graphs) {
            GraphDetail *graphDetail = [graphs objectForKey:key];
            if ([graphId isEqualToString:graphDetail.graphId]) {
                NSArray *rawGraphData = [graphString componentsSeparatedByString:@","];
                NSMutableArray *graphData = [NSMutableArray new];
                for (NSString *graphValue in rawGraphData) {
                    [graphData addObject:[NSNumber numberWithFloat:[graphValue floatValue]]];
                }
                [self receivedNewData:graphData forGraph:graphDetail.graphId];
            }

            if ([graphDetail.graphId isEqualToString:@"E"])
                ekgLabelVal.string = [graphDetail getLabelText];
            else if ([graphDetail.graphId isEqualToString:@"C"])
                conLabelVal.string = [graphDetail getLabelText];
            else if ([graphDetail.graphId isEqualToString:@"S"])
                susLabelVal.string = [graphDetail getLabelText];
            else if ([graphDetail.graphId isEqualToString:@"P"])
                potLabelVal.string = [graphDetail getLabelText];
        }
    }
}


// Process data
- (void)receivedNewData:(NSArray *)someData forGraph:(NSString*)graphId {
    // 1. Check what graph the data belongs to
    for (id key in graphs) {
        GraphDetail *graphDetail = [graphs objectForKey:key];
        if ([graphDetail.graphId isEqualToString:graphId]) {
            if ([someData count] > kReceivedBufferSize) {
                NSUInteger x = [someData count];
                graphDetail.receivedGraphData = [[someData subarrayWithRange:(NSRange) {x-kReceive
            } else {
                graphDetail.receivedGraphData = [someData copy];
            }
        }


    }
}

-(void)draw{
    [self drawGraph];
}

- (void)drawGraph {
    glLineWidth(1.0f);

    for (id key in graphs){
        GraphDetail *graphDetail = [graphs objectForKey:key];
        ccDrawColor4B(graphDetail.r, graphDetail.g, graphDetail.b, 255);
        for (int i = 0; i < graphDetail.visibleBufferSize; i++) {
            NSMutableArray *drawBuffer = graphDetail.graphData;
            CGPoint from = [[drawBuffer objectAtIndex:i] CGPointValue];
            CGPoint to = [[drawBuffer objectAtIndex:i+1] CGPointValue];
            from.y = ((from.y - graphDetail.labelFromVal) * kSignalZoom) * [graphDetail doStuff] +
            // to.y = ((to.y + graphDetail.zeroPointPosition) * kSignalZoom) * [graphDetail doStuf
            to.y = ((to.y - graphDetail.labelFromVal) * kSignalZoom) * [graphDetail doStuff] + gra
            //labelFromVal
            ccDrawLine(from, to);
        }
    }
}

- (void)drawLabels {
    CGFloat fontSize = 20;
    CGFloat labelTopPadding = 40;
    CCLabelTTF *ekgLabel = [CCLabelTTF labelWithString:@"Heartrate␣(bpm)" fontName:@"Futura" fontSize:fontS
    ekgLabel.anchorPoint = ccp(0, 0);
    ekgLabel.position = ccp(10, winSize.height - labelTopPadding);
    [self addChild:ekgLabel];

    ekgLabelVal = [CCLabelTTF labelWithString:@"0.0" fontName:@"Futura" fontSize:fontSize];
    ekgLabelVal.anchorPoint = ccp(0, 0);
    ekgLabelVal.position = ccp(10, winSize.height - labelTopPadding*2);
    [self addChild:ekgLabelVal];

    CCLabelTTF *conLabel = [CCLabelTTF labelWithString:@"Conductance␣(uS)" fontName:@"Futura" fontSize:font
    conLabel.anchorPoint = ccp(0, 0);
    conLabel.position = ccp(210, winSize.height - labelTopPadding);
    conLabel.color = ccc3(0, 255, 0);
    [self addChild:conLabel];

    conLabelVal = [CCLabelTTF labelWithString:@"0.0" fontName:@"Futura" fontSize:fontSize];
    conLabelVal.anchorPoint = ccp(0, 0);
    conLabelVal.position = ccp(210, winSize.height - labelTopPadding*2);
    conLabelVal.color = ccc3(0, 255, 0);
    [self addChild:conLabelVal];

    CCLabelTTF *susLabel = [CCLabelTTF labelWithString:@"Susceptance␣(uS)" fontName:@"Futura" fontSize:font
    susLabel.anchorPoint = ccp(0, 0);
    susLabel.position = ccp(430, winSize.height - labelTopPadding);
    susLabel.color = ccc3(30, 30, 255);
    [self addChild:susLabel];

    susLabelVal = [CCLabelTTF labelWithString:@"0.0" fontName:@"Futura" fontSize:fontSize];
    susLabelVal.anchorPoint = ccp(0, 0);
    susLabelVal.position = ccp(430, winSize.height - labelTopPadding*2);
    susLabelVal.color = ccc3(30, 30, 255);
    [self addChild:susLabelVal];

    CCLabelTTF *potLabel = [CCLabelTTF labelWithString:@"Potential␣(mV)" fontName:@"Futura" fontSize:fontSiz
    potLabel.anchorPoint = ccp(0, 0);
    potLabel.position = ccp(650, winSize.height - labelTopPadding);
    potLabel.color = ccc3(255, 0, 0);
    [self addChild:potLabel];

    potLabelVal = [CCLabelTTF labelWithString:@"0.0" fontName:@"Futura" fontSize:fontSize];
    potLabelVal.anchorPoint = ccp(0, 0);
    potLabelVal.position = ccp(650, winSize.height - labelTopPadding*2);
    potLabelVal.color = ccc3(255, 0, 0);
    [self addChild:potLabelVal];

    for (id key in graphs) {
        GraphDetail *graph = [graphs objectForKey:key];
        if (graph.shouldShowLabel) {
            CGFloat labelPosition = graph.zeroPointPosition + graph.labelFromVal;
            CGFloat x = 10;
            if (graph.labelRightAlign)
                x = winSize.width - 30;
            for (CGFloat i = graph.labelFromVal; i <= graph.labelToVal; i+=graph.labelPointIncrement) {
                NSString *text = [NSString stringWithFormat:@"%.1f", i];
                CCLabelTTF *l = [CCLabelTTF labelWithString:text fontName:@"Futura" fontSize:12];
                l.anchorPoint = ccp(0, 0.5);
                l.position = ccp(x, labelPosition*kSignalZoom);
                l.color = ccc3(graph.r, graph.g, graph.b);
                labelPosition += kGraphSpacing;// graph.labelPointIncrement;
                [self addChild:l];
            }
        }
    }

    // Lower labels
//     int labelPosition = 0 + kBottomPadding;
//     for (int i = 50; i <= 160; i+=10) {
//         NSString *text = [NSString stringWithFormat:@"%d", i];
//         CCLabelTTF *l = [CCLabelTTF labelWithString:text fontName:@"Futura" fontSize:12];
//         l.color = ccc3(255, 255, 255);
//         l.anchorPoint = ccp(0, 0.5);
//         l.position = ccp(10, labelPosition*kSignalZoom);
//         labelPosition+=10;
//         [self addChild:l];
//     }
//
//     labelPosition = 130 + kBottomPadding;
//     for (float i = -55; i <= 25; i+=7.5) {
//         NSString *text = [NSString stringWithFormat:@"%.1f", i];
//         CCLabelTTF *l = [CCLabelTTF labelWithString:text fontName:@"Futura" fontSize:12];
//         l.anchorPoint = ccp(0, 0.5);
//         l.position = ccp(winSize.width-30, labelPosition*kSignalZoom);
//         l.color = ccc3(255, 0, 0);
//         labelPosition+=13;
//         [self addChild:l];
//     }
//
//     labelPosition = 130 + kBottomPadding;
//     for (float i = -50; i <= 130; i+=17.5) {
//         NSString *text = [NSString stringWithFormat:@"%.1f", i];
//         CCLabelTTF *l = [CCLabelTTF labelWithString:text fontName:@"Futura" fontSize:12];
//         l.anchorPoint = ccp(0, 0.5);
//         l.position = ccp(10, labelPosition*kSignalZoom);
//         l.color = ccc3(0, 0, 255);
//         labelPosition+=13;
//         [self addChild:l];
//     }

}

- (void)update:(ccTime)delta {
    for (id key in graphs) {
        GraphDetail *graphDetail = [graphs objectForKey:key];
        [graphDetail updateGraph];
    }
}

float getRandom() {
    float y = arc4random() % 250;
    y += 50;
    return y;
}

float getRandomSmall() {
    float y = arc4random() % 20;
    return y;
}


- (void) dealloc
{
        [super dealloc];
    [timer release];
    [newData release];
    [graphs release];
    [connection release];
}


@end
```

Listing: GraphDetail header file

```
//
//  GraphDetail.h
//  BioPlot
//
//  Created by Jonas Hagstedt & Miriam Huseby on 28/04/2013.
//
//

#import <Foundation/Foundation.h>

#define kLeftPadding 50
#define kRightPadding 10
#define kSpeedMultiplier 4
#define kDrawBufferSize ((1024)/kSpeedMultiplier)-kRightPadding
#define kReceivedBufferSize 10
#define kVisibleBufferSize kDrawBufferSize - kReceivedBufferSize
#define kSignalZoom 1
#define kBottomPadding 10
#define kGraphSpacing 25

@interface GraphDetail : NSObject

@property (nonatomic, retain) NSMutableArray *graphData;
@property (nonatomic, retain) NSMutableArray *receivedGraphData;
@property (nonatomic, retain) NSString *name;
@property (nonatomic, retain) NSString *graphId;

@property (nonatomic, assign) NSInteger visibleBufferSize;

@property (nonatomic, assign) NSInteger r;
@property (nonatomic, assign) NSInteger g;
@property (nonatomic, assign) NSInteger b;
@property (nonatomic, assign) CGFloat zeroPointPosition;

@property (nonatomic, assign) CGFloat labelFromVal;
@property (nonatomic, assign) CGFloat labelToVal;
@property (nonatomic, assign) CGFloat labelPointIncrement;
@property (nonatomic, assign) BOOL shouldShowLabel;
@property (nonatomic, assign) BOOL labelRightAlign;

@property (nonatomic, retain) NSString *labelText;

- (id)initWithName:(NSString*)theName andVisibleBufferSize:(NSInteger)aVisibleBufferSize andGraphId:(NSStrin
- (void)updateGraph;
- (CGFloat)doStuff;
- (NSString *)getLabelText;
@end
```

Listing: GraphDetail m file

```
//
//  GraphDetail.m
//  BioPlot
//
//  Created by Jonas Hagstedt & Miriam Huseby on 28/04/2013.
//
//

#import "GraphDetail.h"

@implementation GraphDetail

@synthesize graphId, graphData, name;
@synthesize receivedGraphData;
@synthesize visibleBufferSize;
@synthesize r, g, b;
```

```
@synthesize zeroPointPosition;
@synthesize labelFromVal, labelToVal, shouldShowLabel, labelPointIncrement, labelRightAlign;
@synthesize labelText;

- (id)initWithName:(NSString*)theName andVisibleBufferSize:(NSInteger)aVisibleBufferSize andGraphId
    if (self = [super init]) {
        self.graphData = [[NSMutableArray alloc] init];
        self.receivedGraphData = [[NSMutableArray alloc] init];
        self.name = [[theName copy] retain];
        self.graphId = [[aGraphId copy] retain];
        self.visibleBufferSize = aVisibleBufferSize;
    }
    return self;
}

- (void)updateGraph {
    for (int i = 0; i < [self.graphData count]-1; i++) {
        NSValue *oldVal = [self.graphData objectAtIndex:i];
        NSValue *newVal = [self.graphData objectAtIndex:i+1];
        NSValue *v = [NSValue valueWithCGPoint:CGPointMake(oldVal.CGPointValue.x, newVal.CGPointVal
        [self.graphData removeObjectAtIndex:i];
        [self.graphData insertObject:v atIndex:i];
    }

    if (self.receivedGraphData) {
        for (NSInteger j = 0; j < [self.receivedGraphData count]; j++) {
            CGFloat y = [[self.receivedGraphData objectAtIndex:j] floatValue];
//            int index = self.visibleBufferSize + j;
            int index = kDrawBufferSize - [self.receivedGraphData count] + j;

            int alal = [self.receivedGraphData count];
            CGFloat x = ((NSValue*)[self.graphData objectAtIndex:index]).CGPointValue.x;
            NSValue *newValue = [NSValue valueWithCGPoint:CGPointMake(x, y)];
            [self.graphData removeObjectAtIndex:index];
            [self.graphData insertObject:newValue atIndex:index];
        }
        self.receivedGraphData = nil;
    }
}

- (CGFloat)doStuff {
    return kGraphSpacing / self.labelPointIncrement;
}

- (NSString *)getLabelText {
    NSNumber *n = [self.receivedGraphData lastObject];
    return [NSString stringWithFormat:@"%.1f", [n floatValue]];
}

- (void)dealloc {
    [super dealloc];
    [self.graphData release];
    [self.name release];
    [self.graphId release];
    [self.receivedGraphData release];
}

@end
```

Listing: GetSettings header file

```
//
//  GetSettings.h
//  BioPlot
//
//  Created by Jonas Hagstedt on 29/04/2013.
//
//

#import <Foundation/Foundation.h>

@interface GetSettings : NSObject {
    NSUserDefaults *userDefaults;
}

- (NSString *)getHost;
- (NSInteger)getPort;
+ (void)registerDefaultsFromSettingsBundle;
@end
```

Listing: GetSettings m file

```
//
//  GetSettings.m
//  BioPlot
//
//  Created by Jonas Hagstedt on 29/04/2013.
//
//

#import "GetSettings.h"

@implementation GetSettings

- (id)init {
    if (self = [super init]) {
        userDefaults = [NSUserDefaults standardUserDefaults];
    }
    return self;
}

- (NSString *)getHost {
    NSString *host = [userDefaults stringForKey:@"host_preference"];
    return host;
}

- (NSInteger)getPort {
    NSInteger port = [userDefaults integerForKey:@"port_preference"];
    return port;
}

+ (void)registerDefaultsFromSettingsBundle {
    NSString *settingsBundle = [[NSBundle mainBundle] pathForResource:@"Settings" ofType:@"bundle"];
    if(!settingsBundle) {
        NSLog(@"Could␣not␣find␣Settings.bundle");
        return;
    }

    NSDictionary *settings = [NSDictionary dictionaryWithContentsOfFile:[settingsBundle stringByAppendingPat
    NSArray *preferences = [settings objectForKey:@"PreferenceSpecifiers"];

    NSMutableDictionary *defaultsToRegister = [[NSMutableDictionary alloc] initWithCapacity:[preferences co
    for(NSDictionary *prefSpecification in preferences) {
        NSString *key = [prefSpecification objectForKey:@"Key"];
        if(key) {
            [defaultsToRegister setObject:[prefSpecification objectForKey:@"DefaultValue"] forKey:key];
        }
    }
    [[NSUserDefaults standardUserDefaults] registerDefaults:defaultsToRegister];
    [defaultsToRegister release];
}


@end
```

Listing: Connectivity header file

```
//
//  connectivity.h
//  statusThingyTest
//
//  Created by Jonas Hagstedt on 25/01/2013.
//  Copyright (c) 2013 Jonas Hagstedt. All rights reserved.
//

#import <Foundation/Foundation.h>

@protocol ConnectionDelegate <NSObject>
- (void)dataAvailable:(NSString*)data;
@end

@interface Connectivity : NSObject <NSStreamDelegate>

@property (strong) NSInputStream *inputStream;
@property (strong) NSOutputStream *outputStream;


@property (strong) id<ConnectionDelegate> delegate;

- (void)connect:(NSString *)host port:(float)port;

@end
```

Listing: Connectivity m file

```
//
//  connectivity.m
//  statusThingyTest
//
//  Created by Jonas Hagstedt on 25/01/2013.
//  Copyright (c) 2013 Jonas Hagstedt. All rights reserved.
//

#import "Connectivity.h"

@implementation Connectivity

@synthesize delegate, inputStream, outputStream;

- (void)connect:(NSString *)host port:(float)port {
    CFReadStreamRef readStream;
    CFWriteStreamRef writeStream;

    CFStreamCreatePairWithSocketToHost(NULL, (__bridge CFStringRef)host, port, &readStream, &writeStream);

    inputStream = (__bridge_transfer NSInputStream *)readStream;
    outputStream = (__bridge_transfer NSOutputStream *)writeStream;

        [inputStream setDelegate:self];
        [outputStream setDelegate:self];
    [inputStream scheduleInRunLoop:[NSRunLoop currentRunLoop] forMode:NSDefaultRunLoopMode];
    [outputStream scheduleInRunLoop:[NSRunLoop currentRunLoop] forMode:NSDefaultRunLoopMode];
    [inputStream open];
    [outputStream open];
}

- (void)stream:(NSStream *)theStream handleEvent:(NSStreamEvent)streamEvent {

        switch (streamEvent) {

                case NSStreamEventOpenCompleted:
                        NSLog(@"Stream␣opened");
                        break;
                case NSStreamEventHasBytesAvailable:
                        if (theStream == inputStream) {
                                uint8_t buffer[1024];
                                int len;

                                while ([inputStream hasBytesAvailable]) {
                                        len = (int)[inputStream read:buffer maxLength:sizeof(buffer
                                        if (len > 0) {
                                                NSData *data = [NSData dataWithBytes:buffer length:
                        NSString *output = [[NSString alloc] initWithData:data encoding:NSASCIIStri
                        [self.delegate dataAvailable:output];
                                        }
                                }
                        }
                        break;


                case NSStreamEventErrorOccurred:
                        NSLog(@"Can␣not␣connect␣to␣the␣host!");
                        break;

                case NSStreamEventEndEncountered:
            [theStream close];
            [theStream removeFromRunLoop:[NSRunLoop currentRunLoop] forMode:NSDefaultRunLoopMode];
            theStream = nil;

                        break;
                default:
                        NSLog(@"Unknown␣event");
        }

}

@end
```

Listing: AppDelegate header file

```
//
//  AppDelegate.h
//  BioPlot
//
//  Created by Jonas Hagstedt on 27/04/2013.
//  Copyright __MyCompanyName__ 2013. All rights reserved.
//

#import <UIKit/UIKit.h>
#import "cocos2d.h"

@interface AppController : NSObject <UIApplicationDelegate, CCDirectorDelegate>
{
        UIWindow *window_;
        UINavigationController *navController_;

        CCDirectorIOS   *director_;                                           // weak ref
}

@property (nonatomic, retain) UIWindow *window;
@property (readonly) UINavigationController *navController;
@property (readonly) CCDirectorIOS *director;

@end
```

Listing: AppDelegate m file

```
//
//  AppDelegate.m
//  BioPlot
//
//  Created by Jonas Hagstedt on 27/04/2013.
//  Copyright __MyCompanyName__ 2013. All rights reserved.
//

#import "cocos2d.h"

#import "AppDelegate.h"
#import "IntroLayer.h"
#import "GetSettings.h"

@implementation AppController

@synthesize window=window_, navController=navController_, director=director_;

- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOption
{
    [GetSettings registerDefaultsFromSettingsBundle];
        // Create the main window
        window_ = [[UIWindow alloc] initWithFrame:[[UIScreen mainScreen] bounds]];


        // Create an CCGLView with a RGB565 color buffer, and a depth buffer of 0-bits
        CCGLView *glView = [CCGLView viewWithFrame:[window_ bounds]
                                       pixelFormat:kEAGLColorFormatRGB565     //k
                                       depthFormat:0             //GL_DEPTH_COMPONEN
                                preserveBackbuffer:NO
                                        sharegroup:nil
                                     multiSampling:NO
                                   numberOfSamples:0];

        director_ = (CCDirectorIOS*) [CCDirector sharedDirector];

        director_.wantsFullScreenLayout = YES;

        // Display FSP and SPF
        [director_ setDisplayStats:NO];

        // set FPS at 60
        [director_ setAnimationInterval:1.0/60];

        // attach the openglView to the director
        [director_ setView:glView];

        // for rotation and other messages
        [director_ setDelegate:self];

        // 2D projection
        [director_ setProjection:kCCDirectorProjection2D];
//      [director setProjection:kCCDirectorProjection3D];

        // Enables High Res mode (Retina Display) on iPhone 4 and maintains low res on all other de
        if( ! [director_ enableRetinaDisplay:YES] )
                CCLOG(@"Retina␣Display␣Not␣supported");

        // Default texture format for PNG/BMP/TIFF/JPEG/GIF images
        // It can be RGBA8888, RGBA4444, RGB5_A1, RGB565
        // You can change anytime.
        [CCTexture2D setDefaultAlphaPixelFormat:kCCTexture2DPixelFormat_RGBA8888];

        // If the 1st suffix is not found and if fallback is enabled then fallback suffixes are goi
        // On iPad HD  : "-ipadhd", "-ipad",  "-hd"
        // On iPad     : "-ipad", "-hd"
        // On iPhone HD: "-hd"
        CCFileUtils *sharedFileUtils = [CCFileUtils sharedFileUtils];
        [sharedFileUtils setEnableFallbackSuffixes:NO];                    // Default: NO. No
        [sharedFileUtils setiPhoneRetinaDisplaySuffix:@"-hd"];        // Default on iPhone Retina
        [sharedFileUtils setiPadSuffix:@"-ipad"];                              // Default
        [sharedFileUtils setiPadRetinaDisplaySuffix:@"-ipadhd"];      // Default on iPad RetinaD

        // Assume that PVR images have premultiplied alpha
        [CCTexture2D PVRImagesHavePremultipliedAlpha:YES];

        // and add the scene to the stack. The director will run it when it automatically when the
        [director_ pushScene: [IntroLayer scene]];


        // Create a Navigation Controller with the Director
        navController_ = [[UINavigationController alloc] initWithRootViewController:director_];
        navController_.navigationBarHidden = YES;

        // set the Navigation Controller as the root view controller
//      [window_ addSubview:navController_.view];       // Generates flicker.
        [window_ setRootViewController:navController_];

        // make main window visible
        [window_ makeKeyAndVisible];

        return YES;
}

// Supported orientations: Landscape. Customize it for your own needs
- (BOOL)shouldAutorotateToInterfaceOrientation:(UIInterfaceOrientation)interfaceOrientation
{
        return UIInterfaceOrientationIsLandscape(interfaceOrientation);
}


// getting a call, pause the game
-(void) applicationWillResignActive:(UIApplication *)application
{
        if( [navController_ visibleViewController] == director_ )
                [director_ pause];
}
```

```
// call got rejected
-(void) applicationDidBecomeActive:(UIApplication *)application
{
        if( [navController_ visibleViewController] == director_ )
                [director_ resume];
}

-(void) applicationDidEnterBackground:(UIApplication*)application
{
        if( [navController_ visibleViewController] == director_ )
                [director_ stopAnimation];
}

-(void) applicationWillEnterForeground:(UIApplication*)application
{
        if( [navController_ visibleViewController] == director_ )
                [director_ startAnimation];
}

// application will be killed
- (void)applicationWillTerminate:(UIApplication *)application
{
        CC_DIRECTOR_END();
}

// purge memory
- (void)applicationDidReceiveMemoryWarning:(UIApplication *)application
{
        [[CCDirector sharedDirector] purgeCachedData];
}

// next delta time will be zero
-(void) applicationSignificantTimeChange:(UIApplication *)application
{
        [[CCDirector sharedDirector] setNextDeltaTimeZero:YES];
}

- (void) dealloc
{
        [window_ release];
        [navController_ release];

        [super dealloc];
}
@end
```

Appendix F

FPGA Design

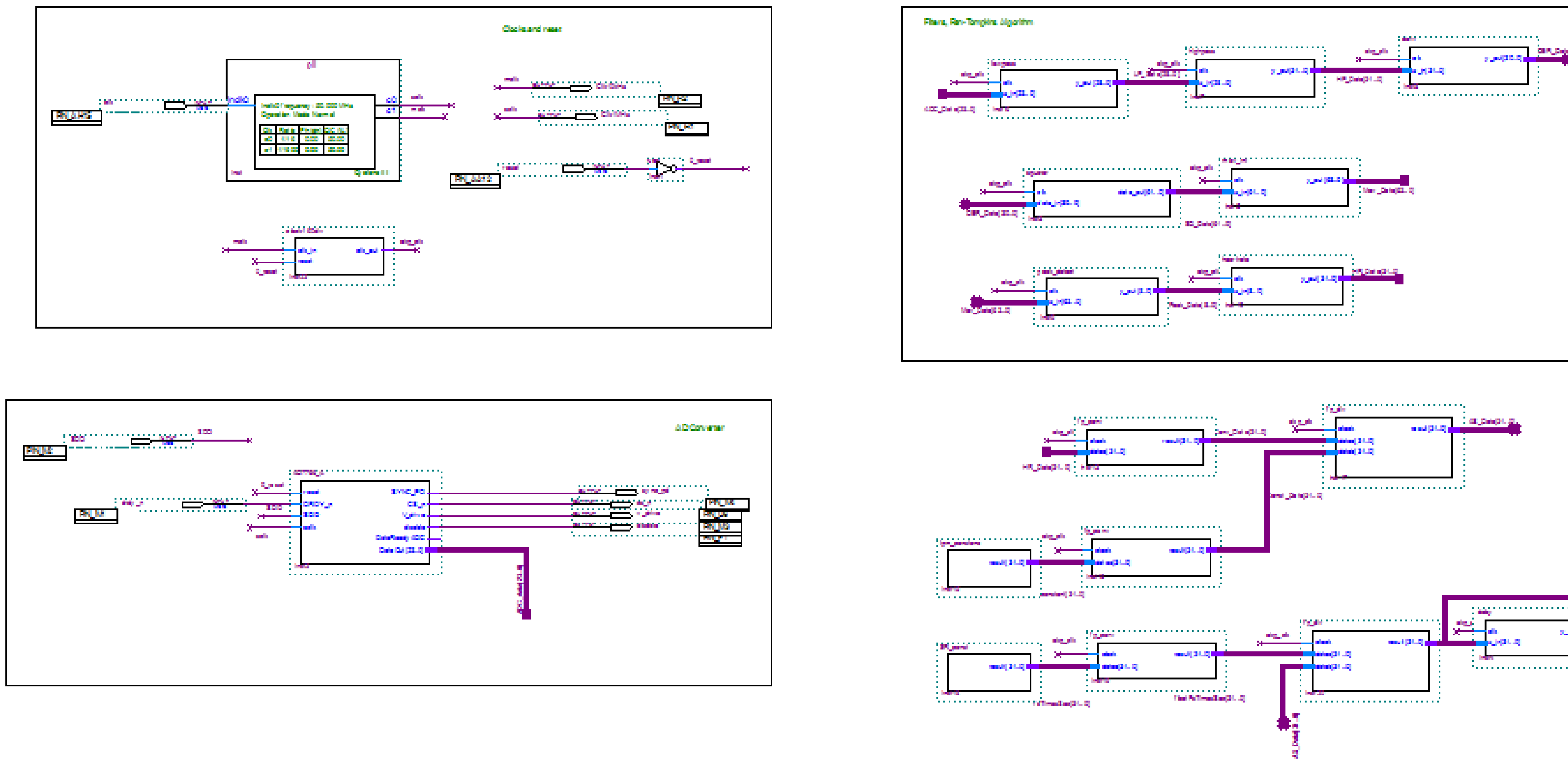


Figure F.1: Complete ECG FPGA Design

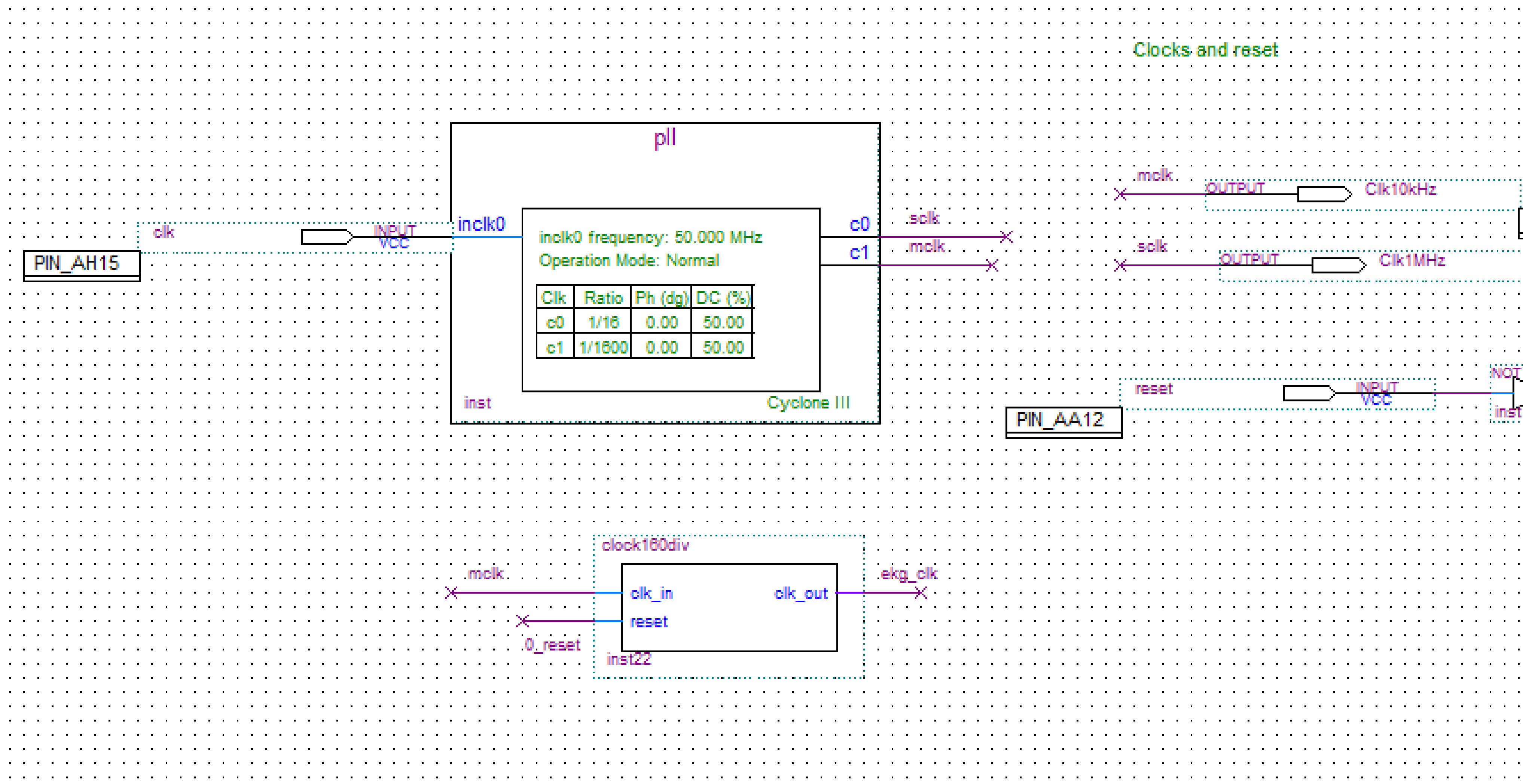


Figure F.2: Clocks and Reset in ECG FPGA Design

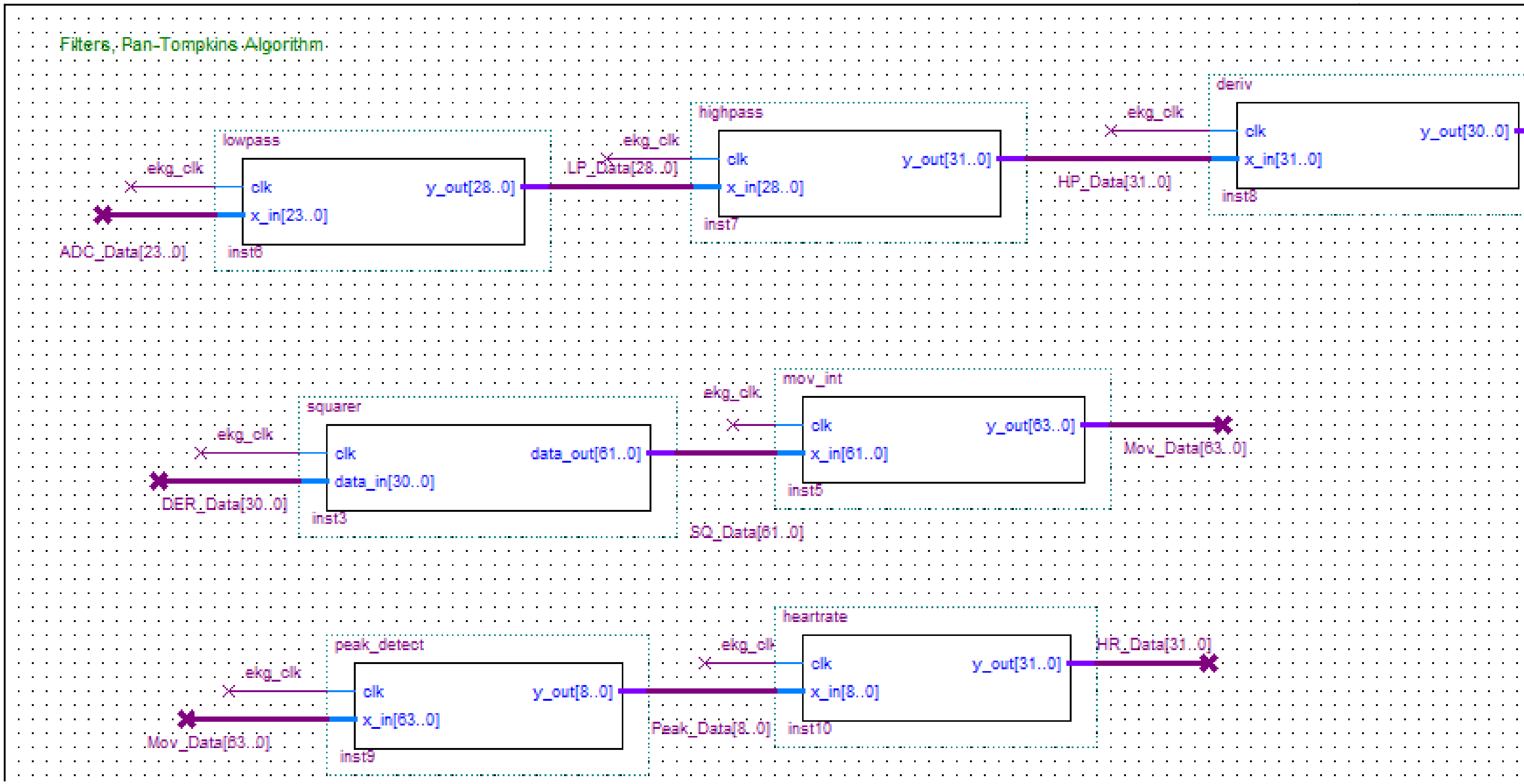


Figure F.3: Pan-Tompkins Algorithm implementation

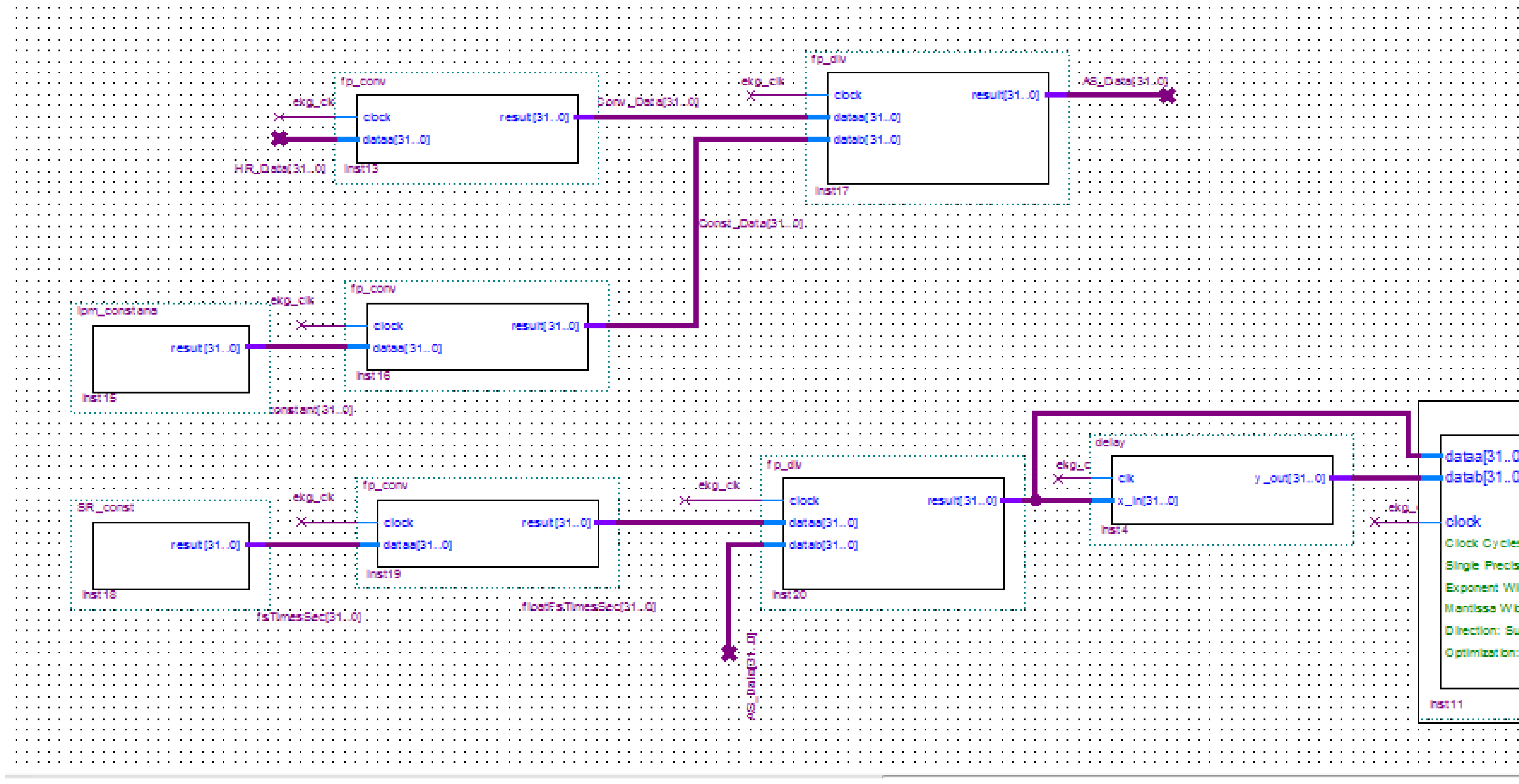


Figure F.4: Floating-Point Operations