UiO : Department of Physics
University of Oslo

FPGA Based Development Platform for Biomedical Measurements

Master Thesis

Lars Jørgen Johnsen Aamodt

4th June 2013

Abstract

This thesis deals with prototype development of an FPGA based development platform for biomedical measurements. The system uses a custom built front-end to measure electrodermal activity. The digital signal processing is performed on an FPGA, and the data is transferred via Bluetooth to an Android application. The digital signal processing, Bluetooth communication and the Android application has been tested and verified, and the potential, current and resistance measurement chains of the front-end show a high degree of linearity. The reactance measurement chain was found to be inoperable, and further testing is required to get the front-end fully functional.

Acknowledgments

This thesis is the fulfilling of the Master of Science in Electronics and Computer Technology at the Department of Physics, University of Oslo. This work 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 Ph.D Christian Tronstad at the OUS Rikshospitalet, Department of Clinical and Biomedical Engineering. I' m very grateful to Ørjan G. Martinsen for giving me the opportunity to work with this interesting topic. I would like to thank Martinsen for his support and guidance during this work. I would also like to thank Christian Tronstad for shearing his insight on skin measurements. Special thanks go to Tore Andrè Bekkeng for his guidance and motivation throughout the entire project. I also want to thank the Electronics lab at the Department of Physics and Stein Lyng Nilsen for valuable help during circuit design and PCB production. To all former and current students at room 333V at the Department of Physics, this would not have been the same without you. To Bent and Espen, thanks for all discussions, help and nonsense during this time. Last but not lest I would like to give a special thanks to my amazing girlfriend Tine Paulsen, and my family for all their support and care during my work on this project.

Oslo, Norway, June 2013
Lars Jørgen Johnsen Aamodt

Nomenclature

FFT Fast Fourier Transform

AC Alternating Current

ADC Analog-to-Digital Converter

AOSP Android Open Source Project

ASIC Application Specific Integrated Circuit

CMR Common-mode Rejection

CMRR Common-mode Rejection Ratio

CORDIC Coordinate Rotation Digital Computer

CPE Constant Phase Element

EA Electrode Area

ECG Electrocardiogram

EDA Electrodermal Activity

EEA Effective Electrode Area

FIR Finite Impulse Response

FPGA Field-Programmable Gate Array

FSM Finite State Machine

HDL Hardware Description Language

HSMC High-Speed Mezzanine Connector

IC Integrated Circuit

In-Amp Instrumentation Amplifier

ISM Industrial, Science and Medical Radio Band

LE Logic Element

LUT Look-up Table

NCO Numerically Controlled Oscillator

Op-Amp Operational Amplifier

PCB Printed Circuit Board

PIO Programmed Input/Output

PLL Phase-Locked Loop

Redox Oxidation-reduction reactions

SDK Software Development Kit

SNR Signal-to-Noise Ratio

Sudomotor Movement of sweat in the sweat duct

UART Universal Asynchronous Receiver/Transmitter

VCCS Voltage-controlled Current Source

Contents

List of Figures

List of Tables

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 performing 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 modern FPGAs 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 marked 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 interfaced 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 main goal of this thesis is to produce an FPGA based development platform for biomedical measurments. The system should be based on general modules that perform tasks like: measurements, pre-amplification, digital signal processing, communication and data representation. In addition to the specifications already mentioned, the system should also meet the following requirements:

- The digital signal processing should be implemented on the FPGA in such a way that the system is expandable and scalable.
- The data representation should be done on an Android mobile application.
- The communication between the Android application and the hardware platform should be done using Bluetooth technology.
- The biomedical measurement used to demonstrate the capabilities of the platform should be electrodermal activity.
- The system should facilitate for the implementation of an ECG signal processing module, which has been developed by Huseby (2013).

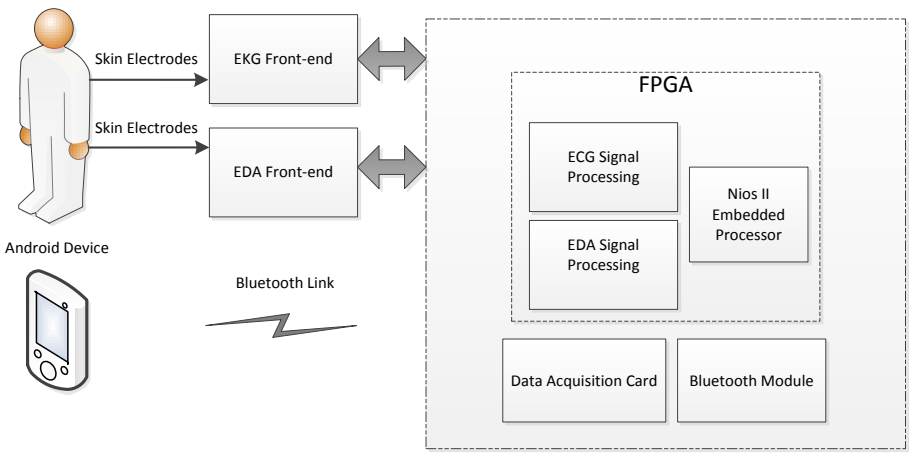


Figure 1.1: FPGA based development platform for biomediacal measurments.

Chapter 2

Basic Theory

This chapter will give a review of the basic theory needed to understand the purpose and measuring principle of the system presented in this thesis.

2.1 Bioimpedance

Bioimpedance is a term used to describe the passive electrical properties of biological materials. It serves as an indirect transducing mechanism for physiological events, and in its simplest form it only requires the application of two or more electrodes. Measurements of bioimpedance can be performed on a vast specter of biological materials such as dead tissue, living tissue or organic material related to any organism such as plant, microbe, cell, animal or human. (Grimnes et al. 2006)

Since body fluids contains ions like Na^+ and Cl^- the conductivity of the body is electrolytic. This means that the charge carriers are ions, and not electrons like in metal wires. Body tissue is composed of cells; these cells have thin cell membranes with poor conductive properties. This gives tissue a capacitive property that is frequency dependent, and as a result, tissue can be regarded as a dielectric. (Grimnes et al. 2006)

Since tissue can be regarded as both a volume conductor and a dielectric it can be described as a complex impedance.

Impedance **Z** `[ohm, Ω]` is a term used to denote a materials ability to oppose AC current flow. In an electrical circuit or a biomaterial it is expressed as the complex ratio of an AC voltage to an AC current:

$$Z = \frac{v}{i} \tag{2.1}$$

where v is the voltage and i is the current.

Impedance can also be expressed by its Cartesian form:

$$Z = R + jX \tag{2.2}$$

where the real part R is the resistance and the imaginary part X is the reactance.

Or on its polar form:

$$Z = |Z| \, e^{jarg(Z)} \tag{2.3}$$

where the magnitude |Z| represents the ratio between the voltage amplitude and the current amplitude and ϕ = arg(Z) represents the phase difference between voltage and current.

Admittance **Y** `[siemens, S]` is the inverse of impedance $Y = \frac{1}{Z}$ and is a measure of how easily a circuit or biomaterial will allow a current to flow:

$$Y = G + jB \tag{2.4}$$

where the real part G is the conductance, and imaginary part B is the suseptance.

Immittance is a term that combines both impedance and admittance of a circuit or biomaterial. Since immittance is a term that applies to both impedance and admittance which have different units, it dose not have its own unit.

2.2 Anatomy of Human Skin

To better understand the electrical properties of human skin and how its activity is measured, we will start with a review of the basics of skin anatomy.

2.2.1 Skin Anatomy

As one can see from Figure 2.1 on the facing page the skin consists of two distinct layers; epidermis which is the outermost layer, and dermis which is located underneath. In addition to these two main layers, there is also a third layer called hypodermis. This layer is sometimes included, even though it is not strictly a part of the skin. In addition to these layers, the skin also contains several different appendages such as, sweat glands, sebaceous glands and hair follicles.

Epidermis

The epidermis is the skins outer structure and serves as a protective layer between the body and its environment. The epidermis is divided into two main layers, an inner layer of viable cells called stratum Malpighii and an outer layer of anucleate horny cells called stratum corneum. As

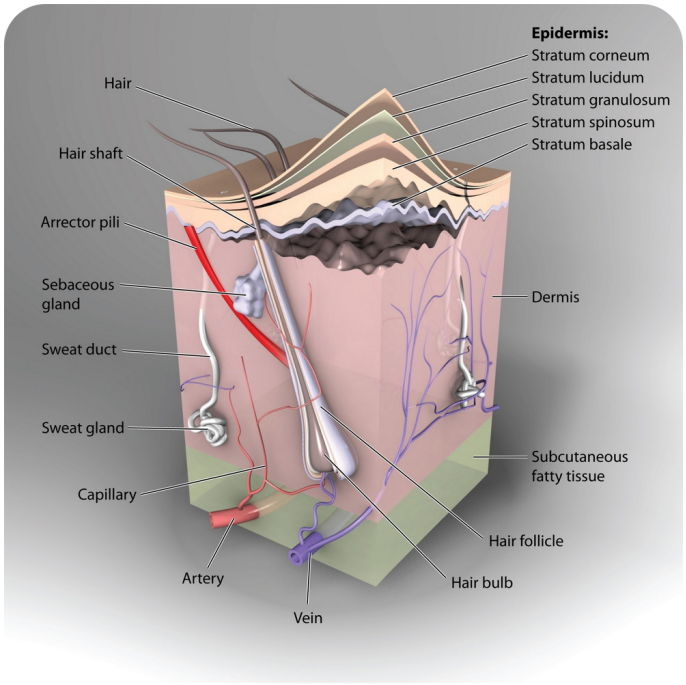


Figure 2.1: Cross section showing the anatomy of the human skin (reprinted from *The Integumentary System*).

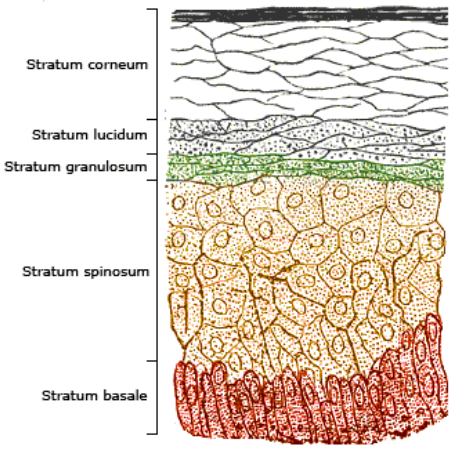


Figure 2.2: Cross section showing a section of epidermis with its epidermal layers (reprinted from *Skin layers*).

one can see form Figure 2.2 the stratum Malpighii is subdivided into four different layers. The first of these is the stratum granulosum located underneath stratum corneum containing various-sized keratohyalin. A spinous layer called stratum spinosum and the stratum basale in contact with the dermis. In friction surfaces or in areas where the epidermis is thick, like the hands and feet, there are also an additional layer of hyaline called stratum lucidum located between stratum corneum and stratum granulosum.(Montagna et al. 1974)

In humans the skin is continually being renewed in a process referred to as keratinization. The desquamation of horny cells on the skins surface is replaced by cell proliferation of the basal epidermal cells. As the keratinocytes divides into two daughter cells in the basal layer, one remains static, and the other migrates to the upper layer. Here the cell undergoes a number of morphological and biochemical changes. In the next layer the keratinocytes grows and flatten, and the generation of keratin will progressively fill the cell. As the kreatinocytes move towards the upper layers they become flatter, and there nucleus begins to degenerate. They also secrete a cement which increases cohesion between the cells. When the cells arrive at the stratum corneum they have become corneocytes, anucleate flattened cells filled with keratin.(Montagna et al. 1974)

Dermis

The dermis is located under the epidermis and it consists of a matrix of loosely connective tissue composed of the fibrous proteins collagen, elastin and reticulin. The matrix of the dermis contains blood vessels, nerves and lympatics. The epidermal appendages eccrine sweat glands, apocrine glands and the pilosebaceous unit are also penetrated into it. Compared to the epidermis, the dermis is much thicker, and it is divided into two dermal layers. The two layers are distinguished according to there density, and by the arrangement of their collagen fibers.(Boucsein 2012)

The layer next to the epidermis is called stratum papillare, and it forms a fingerlike pattern which fits into cavities on the underside of the epidermis. Thus forming an intimate intermeshed junction referred to as the epidermal-dermal junction. In addition to a possibly adhesive effect this greatly increases the area of the basal layer, and results in an enlargement of the area that produces new epidermal cells.(Montagna et al. 1974)

The inner dermal layer called stratum reticulare is thicker than stratum papillare, and it is made up bye strong collagenus fibers. This gives the skin a high resistance against damage and rupture.

Hypodermis

The hypodermis is located under the dermis, and connects the skin with the connective tissue covering the muscles. It allows good horizontal mobility of the skin across its surface, and consists of loose connective tissue. The hypodermis also contains the nerves and vessels which supplies the skin. Because it can store fat it will be working as a thermal layer.(Boucsein 2012)

2.2.2 The Distribution and Structure of Sweat Glands

The human body is covered with more than three million sweat glands. These glands are considered to be exocrine which means that they secrete directly onto the skin's surface. The greatest density of sweat glands are found on the palms, soles, and on the forehead. The lowest densities are found on the arms, legs, and trunk.(Kuno 1956) Millington(1983) states the following mean numbers of sweat gland per m^2 on adult skin: 233 on the palms, 620 on the soles, 360 on the forehead and 120 on the thighs.

There are two different types of sweat glands, apocrine and eccrine. As one can see form Figure 2.1 on page 5 the apocrine glands are large in size, and discharge into the hair follicle. These glands are mainly found in the areola region of the breast, the genital and the axillary regions. According to Herrmann et al. 1973 the apocrine sweat glands only play a negligible role with respect to the total amount of sweating.

The eccrine glands, which make up the majority in human skin have there greatest densities on the palms, soles and on the forehead. From Figure 2.1 on page 5 one can see the eccrine gland are divided into two different subparts; the secretory segment and the duct. The secretory segment originates in the hypodermis or in the dermis and has the shape of an irregularly coiled rounded mass. From this structure there is a duct that goes relatively straight through the dermis and epidermis before it spirals through the stratum corneum and opens on to the skin's surface through a pore.(Boucsein 2012)

2.3 Electrical Properties of Human Skin

As mentioned in Section 2.2.1 the stratum corneum is composed of a protective barrier of keratinized cells called corneocytes. This layer has a high ionic impedivity compared to the viable deeper layers of the skin.(Grimnes 1982) The impedivity is influenced by the moister content of stratum corneum. It is at its largest on the surface which is in direct contact with the ambient humidity.(Tronstad et al. 2008)

The stratum corneum is perforated with sweat ducts. Since sweat is an electrolyte solution the filling and absorption of sweat will mainly

influence the admittance of the outer layers of skin. Grimnes (1982) shows that the filling of a sweat duct results in a doubling of the stratum corneum admittance within few seconds.

In Grimnes (1984) it is argued that the ions flow through the stratum corneum is negligible and that the dominant ionic path through ordinary stratified stratum corneum is through the sweat ducts.

Since sweat is an electrolyte the filling of the sweat ducts mainly contributes to the conductive part of the admittance, the capacitive part of the admittance represents the moisture content of the stratum corneum. (Martinsen et al. 2001)

2.3.1 Electrodermal Activity

The sweat activity on friction surfaces such as palmar and plantar skin sites is very sensitive to psychological stimuli or conditions.(Grimnes et al. 2006) Because of this the psychological effects on the electrical changes in human skin have been studied for more then 100 years. Throughout this history there have been different theories on the function and mechanism of electrodermal activity (EDA). (Cacioppo et al. 2007)

From a physiological point of view, active sweating in humans can be divided into two categories: thermal and mental / emotional. Thermal sweating appears over the whole body surface, and plays an important role in keeping the body temperature constant. The other part of active sweating is influenced by emotion, mental stress and sensory stimulation. The palmar and plantar sweat glands are innervated by the sympathetic chain of the autonomic nervous system. In Kerassidis (1994) the thermoregulatory effect of palmar and plantar sweat was investigated, and the paper concludes that palmar and plantar sweating is not thermoregulatory. As a result of this EDA is said to reflect sympathetic activation. This type of sweating is usually present at some level during a conscious state of mind, but disappears during sleep. (Kuno 1956)

2.3.2 Electrical Models of Human Skin

As mentioned in Section 2.1 on page 3 tissue have resistive and capacitive properties. In order to investigate different aspects of these properties one can use substitute models for the tissue, or in this instance the skin. All models of biological tissue are separated into descriptive and explanatory.

Descriptive Model

The descriptive model is meant to describe the electrical properties of the skin, and thus characterize the skin by both known electrical components and algorithms. This category of models primarily reflects the measured

values and time courses, and it does not necessarily correspond to the microanatomy of the skin. (Grimnes et al. 2008)

For the skin an descriptive electrical model is an electronic circuit. If this circuit is constructed out of one ideal resistor a one ideal capacitor, the model would be able to represent measuring results of one frequency. It would not be able to mimic the whole immittance spectrum found in the skin. To compensate for this, a second resistor is usually added, and the capacitor is replaced with a more general Constant Phase Element (CPE). The CPE is not a physical component, but a mathematical concept that can have any constant phase.(Grimnes et al. 2006)

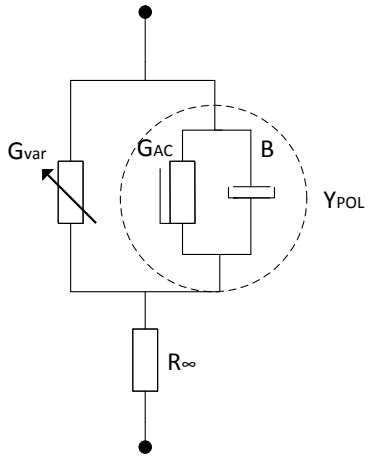


Figure 2.3: Descriptive skin equivalent model.

The circuit in Figure 2.3 is a skin equivalent model from Grimnes (2005). It consists of the resistor G_{var} connected in parallel with the CPE (Y_{POL}). The CPE consists of a frequency dependent capacitor B connected in parallel with a frequency dependent resistor G_{AC}. I addition to the parallel connection of G_{var} and CPE there is also a resistor R_∞ connected in series.

Explanatory Model

The explanatory model is based on electrical theory and is composed of discrete components like resistors, conductors, voltage sources, etc. The model uses knowledge about basic electrical concepts, and relates them to anatomical structures of physical processes. In this way the model is meant to explain the physical processes or anatomy by the properties of its electrical components.(Grimnes et al. 2008)

Figure 2.4 on the next page shows a simplified electrical equivalent model of the skin, the model is based on the Fowles model (D.C. 1974), and modified for AC measurements by Christian Tronstad.

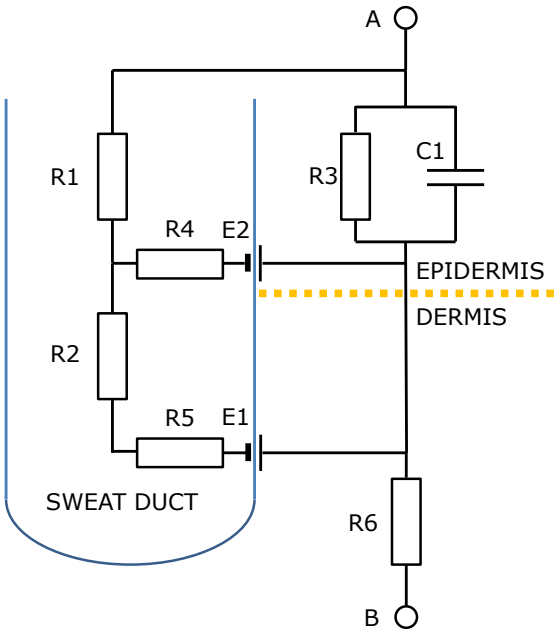


Figure 2.4: Explanatory sweat duct model (reprinted from Tronstad 2012).

The following explanation originates from the description given by Tronstad (2012), and is based on many years of bioimpedance research by S Grimnes and Ø G Martinsen.

There are several different sources influencing the electrical properties of the skin, but this model focuses on the sudomotor (movement of sweat). The sudomotor activity is the mechanism responsible for the largest change in the electrical admittance. As already mentioned in Section 2.3.2, the skin has both conductive and capacitive properties. One can see from Figure 2.4, that sweating mainly influenc the conductive part of the admittance. The resistors R1 and R2 represent the conductive pathway through the sweat duct, and they change conductance as the sweat fills the duct. This means that the sum R1 + R2 is of interest when measuring the sudomotor sweat activity. The terminal A is for the measuring electrode and terminal B is for the counter-electrode. The resistor R3 and capacitor C1 represent the epidermal admittance. The resistance from the deeper layers of the skin, to the counter-electrode, is represented by the resistor R6. Resistors R4 and R5 represents the resistance of the duct wall at the epidermal and sub-epidermal levels, and are connected to the biopotentials E2 and E1, where E1 < E2.

Since the impedance of a capacitor decreases with higher frequencies, C1 will influence the sensitivity depth of the measurement. At higher excitation frequencies the sensitivity depth will increase. This means that a low excitation frequency results in a measurement sensitive to

epidermal skin properties and the highly resistive stratum corneum. The contribution form the resistor R6 will becomes negligible. If the excitation frequency is too low, the AC measurement signal will fall within the same range as the skin biopotential activity. The signals will then become difficult or impossible to distinguish.

2.4 Electrodes

In order to measure bioimpedance, it is necessary to provide some sort of interface between the body and the instrumentation. This interface is obtained by the use of electrodes. These electrodes interacts with the ionic charge carriers, and transduce them into electric current for the instrumentation. To achieve this transducing function the electrodes consist of an electrical conductor in contact with some sort of aqueous ionic solution from the body.

The interaction between electrons in the electrodes and the ions in the body can influence the performance of the measurement. It is therefore important that the electrodes are selected according to the intended application.(Grimnes et al. 2008)

Electrode-electrolyte Interface

In Figure 2.5 on the following page, the transfer between the body and the electrode is shown. The body is represented by the electrolyte. When a net current crosses this interface the electrons (e^-) and anions (A^-) move in the opposite direction of the current in the electrode. The cations (C^+) move in the same direction as the electrode current. These chemical reactions are called redox (reduction-oxidation) , and can be presented in general by the following reactions :(Webster 2009)

$$C \rightleftharpoons C^{n+} + ne^- \tag{2.5}$$

$$A^{m-} \rightleftharpoons A + me^- \tag{2.6}$$

where n is the valence of cation material C, and m is the valence of anion material A.

In Reaction 2.5 it is assumed that the electrode is made up of some atoms of the same material as the cations. This material can become oxidized at the interface to form a cation and one ore more free electrons. The cation is then discharged into the electrolyte, and the electron remains in the electrode as a charge carrier. Reaction 2.6 gives the process of the anions. In this case the anions at the electrode-electrolyte interface can become oxidized to a neutral atom and give off one or more electrons to the electrode. The reactions described for Reactions 2.5 and 2.6 are

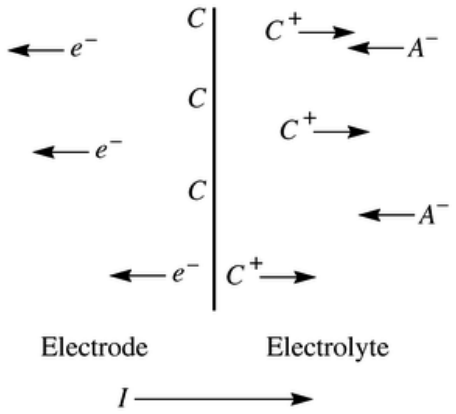


Figure 2.5: Current crossing an electrode-electrolyte interface from left to right. The electrode consists of metallic atoms C. The electrolyte is an aqueous solution containing cations of the electrode metal C^+ and anions A^-. The figure is reprinted from Webster (2009).

often reversible, and when the reaction happens from right to left; it is referred to as reduction. If no current is crossing the interface the oxidation and reduction reactions cancel each other out. When current flow from electrode to electrolyte the oxidation dominates, and when it flows in the opposite direction the reduction dominates.(Webster 2009)

Another important aspect of the electrode-electrolyte interface is the half-cell potential. When metal is placed in a solution containing the same ions as it self, there will be a local change in the concentration of ions in the solution near the surface of the metal. This means that the charge in this region is not neutral, and there will be a difference in potential between the region close to the metal and the rest of the solution.(Webster 2009) It is important to use the same metals for an electrode pair. If different metals are used, the different half-cell potentials may create DC voltage outputs of more than 1 V. This can contribute to the noise of the measurement, and saturate the input of the biopotential amplifier.(Grimnes et al. 2008)

Polarization

If two ionic solutions of different concentrations are separated bye an ion-selective semi permeable membrane, there will be an electric potential E across the membrane. This relationship is described bye the Nernst equation:

$$E = -\frac{RT}{nF} ln \frac{a_1}{a_2} \tag{2.7}$$

where a_1 an a_2 are the activities of the ions on either side of the membrane. R is the universal gas constant, T is the absolute temperature, n is the valence of the ions and F is the Faraday constant.

The half-cell potential or Nernst potential mentioned in the previous section is described for conditions where there are no electric current between the electrode and the electrolyte. For conditions where there is a current between the two, the observed half-cell potential is often altered. This alteration is due to the polarization of the electrode and can influence the electrode performance.(Grimnes et al. 2008)

The difference in potential between the zero current half-cell potential and the observed potential are known as overpotential. The overpotential phenomenon is composed of three different mechanisms: the ohmic, the concentration, and the activation overpotentials. The ohmic overpotential is due to the resistance of the electrolyte, and it results in a voltage drop along the path of the current in the electrolyte. The concentration overpotential results from changes in the ion distributions of the electrolyte near the electrode-electrolyte interface. When no current is flowing across the electrode-electrolyte interface, the reactions described by 2.5 on page 11 and 2.6 on page 11 reach equilibrium. If a current is established, this equality will be disturbed resulting in a change of the half-cell potential of the electrode. The difference between this change of potential and the equilibrium potential, is the concentration overpotential. The last mechanism of polarization is the activation overpotential. This mechanism results in a difference in voltage between the electrode and the electrolyte, and happens because the redox has different energy barriers. This results in different activation energies and the redox reaction is therefore not entirely reversible.(Bronzino 2000)

Polarizable and Nonpolarizable Electrodes

Theoretically there are two types of electrodes: perfectly polarizable, and perfectly nonpolarizable. The difference between these two classifications refers to what happens to the electrode when a current passes between it and the electrolyte.

For perfectly polarizable electrodes there are no actual charge crossing the electrode-electrolyte interface when a current is applied. These electrodes work by changing the distribution within the ion solution near the electrode.

Perfectly nonpolarizable electrodes allow the current to pass freely across the electrode-electrolyte interface without changing the charge distribution in the electrolyte near the electrode. The two electrode classifications are only theoretical, and they can not be fabricated. It is however possible to fabricate electrodes that approximate the theoretical descriptions.(Webster 2009)

Electrodes made from noble metals such as platinum come close to

behaving like polarizable electrodes. For this type of electrodes a charge distribution different from that of the bulk electrolytic solution is found close to the electrode-electrolyte interface. This effect makes the electrodes sensitive to situations where movement is present, and in measurements that involve low frequency or dc signals. If the electrode moves with respect to the electrolytic solution, the charge distribution in the solution adjacent to the electrode surface will change. This will result in a voltage change in the electrode (motion artifacts).

Because of this, for most biomedical measurements, nonpolarizable electrodes are preferred. A typical example of an electrode that comes close to having nonpolarizable characteristics is the silver-silver chloride electrode (Ag/AgCl).

The Skin Surface Electrode

In order to couple the electrode to the skin an ionic conductor is positioned between the tissue and the electronic conductor. According to Grimnes (2008) the purpose of the contact electrolyte is to:

- Control the metal-electrolyte interface
- Form a high conductance salt bridge from the metal to the skin
- Ensure small junction potentials
- Enable the metal-electrolyte interface to separate form the tissue
- Fill out space between the electrode plate and tissue

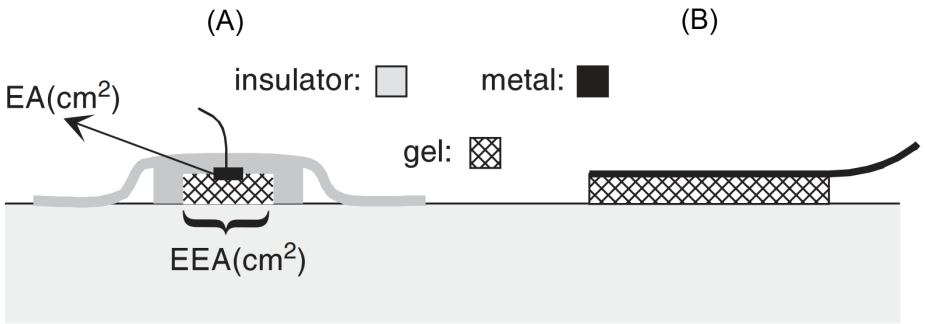


Figure 2.6: Two common designs for skin surface electrodes: (a) recessed metal with gel in cup; (b) electrode with hydrogel contact electrolyte. The figure is reprinted from Grimnes (2006).

Figure 2.6 shows the two most common types of skin contact electrodes. These electrodes are constructed with a controlled distance

between the metal part of the electrode and the skin. This volume is filled with contact electrolyte. For electrode (A) the electrolyte is a wet gel that is restrained by a container for mechanical support and to prevent evaporation. For electrode (B) the electrolyte is a solid contact gel that is sticky so it also serves as fixation. (Grimnes et al. 2006)

Variations of the design used for electrode (B) are common for disposable electrodes. According to Electrodermal Measurespsyp et al. (2012) they can have the following advantages: hygienic, antiseptic, hypoallergenic, and latex free. They can have good systems for fixation, and be stored for many months in an unopened package. The electrodes can be produced in large production runs with uniform electrical characteristics, and be pre-gelled so that the metal-electrolyte interface is stabilized and ready for use.

There are two surface areas that are important fore the immittance of an electrode. The first of these, the metal-electrolyte interface area, referred to as the electrode area (EA). This area determines the polarization impedance of the EA, and is dependent on redox reactions and sorption processes at the interface, and diffusion processes in the electrolytic solution. The second area is the interface between the electrolytic solution and the skin called the effective electrode area (EEA). This area determines the measured skin Impedance. The type and concentration of the electrolyte is therefore important in order to avoid interference with the sweat process.(Grimnes et al. 2006)

The use of sintered silver-silver chloride electrodes is standard for measurements of EDA. This is because they minimize electrode polarization, and the bias potential between the electrodes. Another advantage is that they are also commercially available both as reusable and disposable versions.(Psychophysiological Research et al. 2012)

2.4.1 Electrode Noise

Noise is an unwanted AC voltage superimposed on its equilibrium DC potential in the electrodes. This noise is dependent on the frequency range and it can take the form of pulse noise, white noise or $1/f$ noise [1].(Grimnes et al. 2008) According to Grimnes et al. (2008) there are tree rules that are important for the electrode noise:

- Larger electrode area gives less noise because of averaging effects.
- The more polarizable the electrode is, the more noise it generates.
- A more diluted contact electrolyte will generate more noise.

There are also other sources of disturbance present when using skin electrodes, sudden voltage spikes with amplitudes of hundreds of

[1] Fore more information on the different types of noise see Section 3.6

microvolt's and millisecond duration may occur. Non-uniform electrode surface can have local current exchanges between impurity centers that will contribute to the electrode noise. An AgCL electrode in 0.9% saline may generate noise in the order of 10 μVp-p when used to measure ECG at a bandwidth of 0.1-100 Hz. Pure silver plate electrodes may generate 10 times this value.(Grimnes et al. 2008)

2.5 Measuring Principles

This section looks further into the different methods used to measure EDA and goes through the electrode system used in this thesis, its recording sites and its measurement artifacts.

2.5.1 Endosomatic Measurements

An endosomatic measurement of EDA refers to the potential difference that can be measured across the palmar and plantar skin in the absence of any applied voltage or current. In this type of measurement one electrode is placed on an active site, and the reference electrode is placed on a relatively inactive site. The parameter measured is called skin potential SP, and the signal amplitude usually have a range from $0\ to\ \pm 20\,mV$.(Dittmar et al. 1991)

2.5.2 Exosomatic Mesurments

Exosomatic measurements of EDA are conducted by applying a constant external voltage or current through the electrodes. This current can be either AC or DC. With the application of constant current source, the result will be a resistance measurement. If the voltage is kept constant, the result will be a conductance measurement. With the application of an AC voltage signal with a constant voltage, the measured result will be admittance, and for a constant current the result will be impedance.(Boucsein 2012)

2.5.3 Three-electrode Systems

Figure 2.7 on the facing page shows the general principle behind a three-electrode impedance system, and its sensitivity field. In this system there is separate current carrying and signal pick-up electrodes, so the signal is not picked up from the site of current application. This means that the impedance measured is transfer impedance. For two-electrode systems it can be difficult to estimate the contribution from the neutral electrode. This problem can be compensated by using a large neutral electrode, but the distal volume segment of the current path through the tissue, will still influence the measurement. To better control the measured tissue zone,

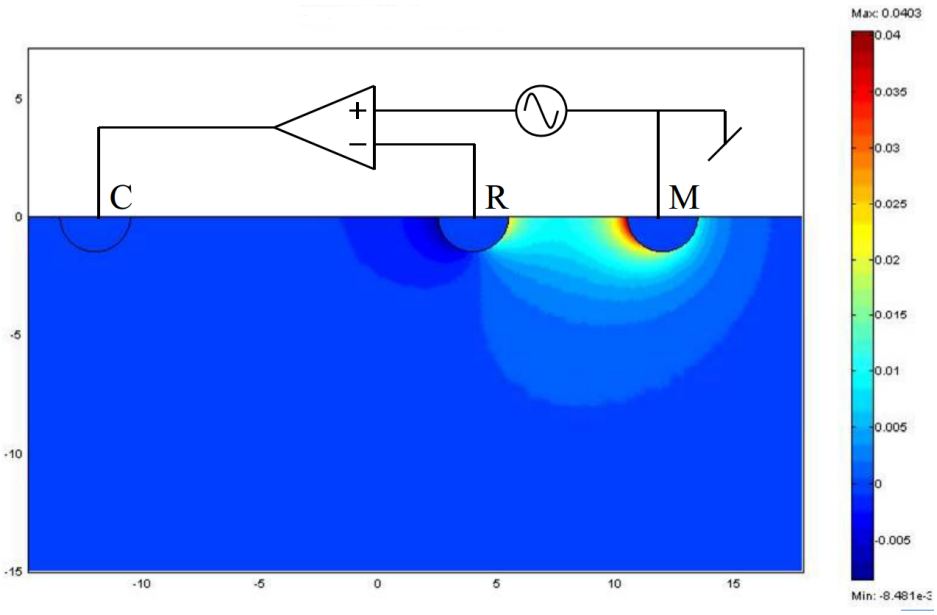


Figure 2.7: Three-electrode system and its sensitivity field.(*Oslo Bioimpedance Group* 2010)

a third electrode is therefore added. The electrode M is used for both current injection and voltage recording. The current in this set-up flows from electrode M to electrode C, and the electrode R is used as a pick-up electrode. If electrode R was not present the measured result would be dependent on the impedance of both the electrodes and the tissue between them. In order to avoid this, the voltage u is measured on R with respect to M. The impedance is then given as $Z = u/i$ and the measurement is dependent on the polarization impedance of M, and a tissue volume zone proximal to M. (Grimnes et al. 2008)

In Martinsen et al.(1999) it is shown that the sensitivity is also dependent on the measurement frequency. High frequency measurements are dominated by the deeper viable layer of the skin, and low frequency measurement are dominated by the stratum corneum.

2.5.4 DC Potential and AC Conductance Measured at the Same Skin Site

The two parameters skin conductance and skin potential are important in the EDA tradition. The standard way to measure these parameters is to use an exosomatic DC current for the conductance and an endosomatic method for the potential. The use of DC current in the exosomatic measurement results in a disturbance in the endosomatic DC potential. The two measurements can therefore not be conducted at the same time. In addition to this the DC current flow will polarize the electrodes,

electrolyze the skin, and possibly disturb the conductance du to varying electromotive forces in the circuit.(Grimnes et al. 2010) To avoid these problems a new measuring method was presented in Grimnes et al. (2010). This method uses an AC constant current system to measure AC conductance and DC potential simultaneously at the same skin site. Figure 2.8 shows a modified version of the original measuring system. This system was design by professor Sverre Grimnes.

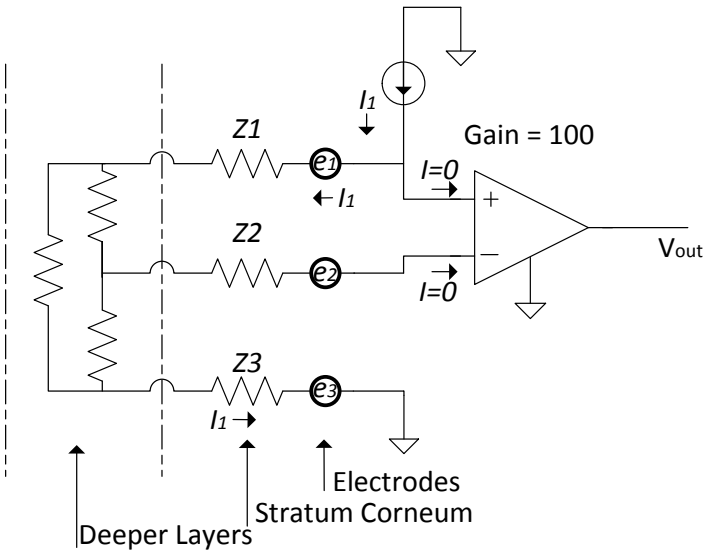


Figure 2.8: Measuring Principle

The circuit in Figure 2.8 uses a three electrode system, and performs both an exosomatic and an endosomatic measurement. The DC potential is measured through electrode 1 and electrode 2, and is amplified by the instrumentation amplifier. The AC impedance measurement is also measured between the same electrodes, and the signals must therefore be separated by signal processing. The DC potential is extracted by the use of a lowpass filter that removes the AC components, and leaves the DC potential. In order to find the AC conductance G, the complex impedance must be separated into resistance R and reactance X. This is done with a lock-in amplifier, and has the added benefit of removing unwanted noise. Since we now have the resistance and the reactance, both the conductance and the susceptance can be calculated by the following formulas:

$$G = \frac{R}{R^2 + X^2} \quad [S] \tag{2.8}$$

$$B = \frac{-X}{R^2 + X^2} \quad [S] \tag{2.9}$$

As mentioned in Section 2.3, the low frequency skin susceptance is measure of the stratum corneum hydration, and is therefore an interesting parameter to include in the instrument.

2.5.5 Recording Sites

EDA is recoded on the smooth and hairless skin of the palms and the soles. The palms are often used for convenience. Figure 2.9 shows an overview of the recommended electrode sites for EDA measurements on the palms. Active electrodes are placed either on the volar phalanges of the fingers, or on the thenar and hypothenar sites of the palms on the nondominant hand. The nondominant hand is used to reduce the chance of movement, and to leave the dominant hand free to be used.

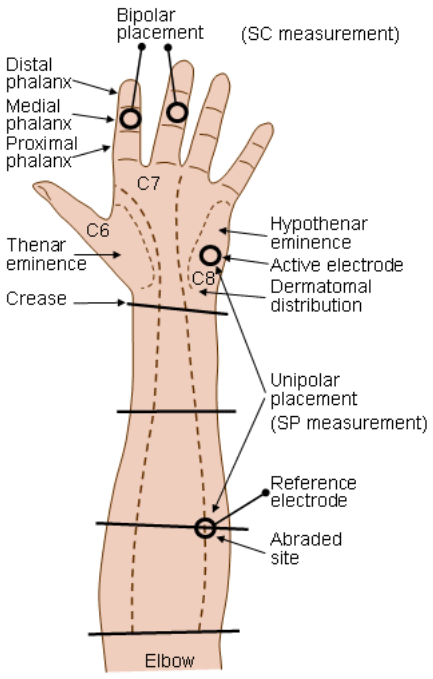


Figure 2.9: Suggested electrode sites fore EDA measurements on the left arm (reprinted from Malmivuo (1995)).

Chapter 3

Electronics Theory

This chapter reviews the electronics theory used to implement the measuring system presented in this thesis.

3.1 Voltage-to-Current Conversion

In bioimpedance applications impedance is often found by injecting an AC current into the subject and measuring the voltage across it. In order to get a reliable measurement it is therefore important to have a stable and reliable current source. The stimulating AC signal is normally generated as a voltage by direct syntheses or a digital-to-analog converter. The AC signal is then converted by a voltage-to-current generator into a current of known amplitude.

The impedance of the subject is a value that changes with time, and results in a variable load. It is therefore important that the current source is able to cope with these variations without going into saturation or any other form of overload. The current source should also have a high output impedance in order to affect the measurement in the smallest possible way.

This section will take a look at some of the most commonly used current sources in bioimpedance and their pros and cons.

3.1.1 The Howland Current Source

The Howland current source as shown in Figure 3.1 on the next page is a well known and widely used circuit for linear voltage-controlled current sources (VCCS) with variable loads.

The circuit was invented by Prof. Bradford Howland at MIT, and first published in (Sheingold 1964). In its original form the Howland current source consists of one op-amp and four resistors, and its operation is based on both negative and positive feedback. According to Chen et al. (2010), the circuit can be described by current analysis giving the following equations.

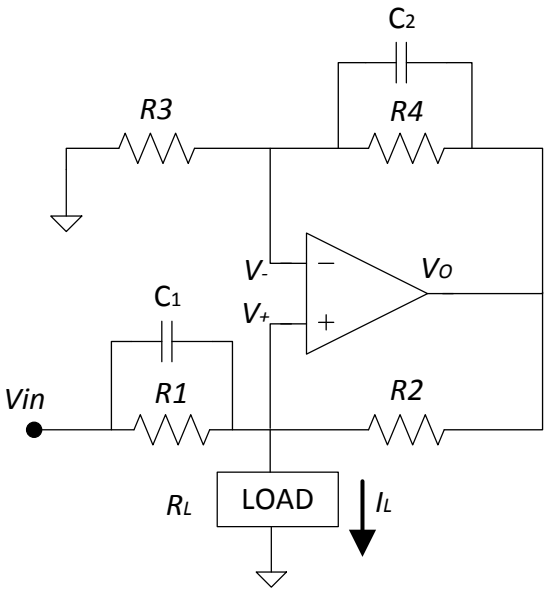


Figure 3.1: The basic Howland current source.

$$I_L \approx \frac{V_{in} - V_+}{R_1} + \frac{V_o - V_+}{R_2} \tag{3.1}$$

$$V_+ \approx V \tag{3.2}$$

$$\frac{V_-}{R_3} \approx \frac{V_o - V_-}{R_4} \tag{3.3}$$

If one then sets $R_1 R_4 = R_2 R_3$, the current flowing through the variable load R_L is given by

$$I_L \approx \frac{V_{in}}{R_1} \tag{3.4}$$

From Formula 3.4 one can see that the load current I_L is not a part of the output current expression. The current I_L through the load is therefore independent of the load, and for an ideal op-amp the output impedance is infinite.

The main problem with this configuration is that in order to guarantee high output impedance, the resistors $R_1 - R_4$ must be closely matched. According to Xiaoke et al. (2012) it has been shown that the maximum mismatch should not exceed 0.1%

3.1.2 The Enhanced Howland Current Source

The enhanced Howland current source is shown in Figure 3.2. As the name implies this configuration is a modified version of the original Howland. By neglecting the stabilizing capacitors C_1 and C_2, and assuming an ideal op-amp, current analysis gives the following equations.(Chen et al. 2010)

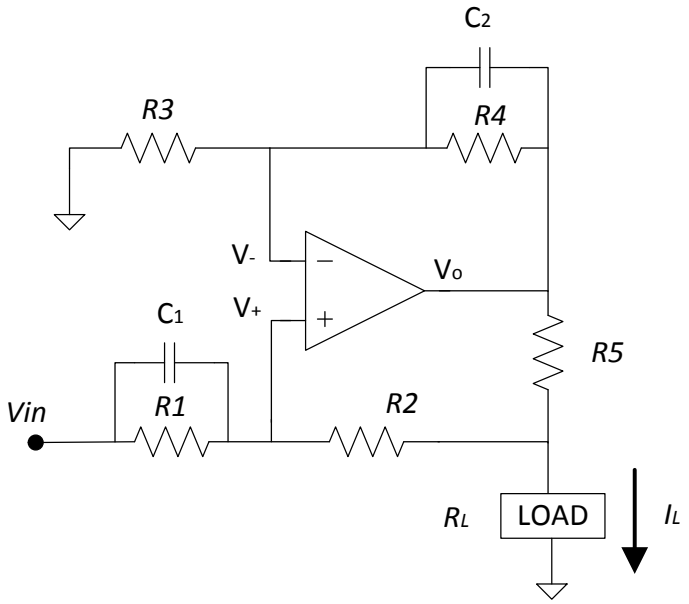


Figure 3.2: The modified Howland current source.

$$I_L \approx \frac{V_o - V_L}{R_4} + \frac{V_L - V_+}{R_3} \tag{3.5}$$

$$\frac{V_{in} - V_-}{R_1} \approx \frac{V_- - V_o}{R_5} \tag{3.6}$$

$$\frac{V_L - V_+}{R_3} \approx \frac{V_+}{R_2} \tag{3.7}$$

$$V_+ \approx V_- \tag{3.8}$$

If one then sets $R_1 = R_5$ and $R_2 = R_3 + R_4$, the current flowing through the variable lode R_L is given by:

$$R_I \approx \frac{V_{in}}{R_4} \tag{3.9}$$

From Formula 3.9 on the preceding page one can see that the value of R_L is not included in the output current expression. The load current I_L is therefore independent of the load resistance and assuming an ideal op-amp the output impedance is infinite.

One of the benefits of this configuration is that by splitting R_2 from the original Howland into R_3 and R_4, the power consumption of R_1 is reduced. This means that unwanted heating of R_1 is reduced, resulting in a more stable value for the resistor due to its temperature coefficient, resulting in a better balance in the resistor bridge formed by R_1, R_2, R_3, R_4.

Another advantage of the enhanced Howland current source is that it enables the designer to use larger resistors for the same output current. This results in a design that is less influenced by the resistance represented by the length of the Printed Circuit Board PCB tracks, and results in better balancing of the resistors.

3.1.3 The Dual op-amp Current Source

Figure 3.3 shows the dual op-amp current source. This current source differs from the two Howland configurations mentioned earlier by being composed of two op-amps, thereby increasing the circuit complexity. By neglecting the stabilizing capacitors C_1 and C_2, and assuming an ideal op-amp, current analysis gives the following equations.(Chen et al. 2010)

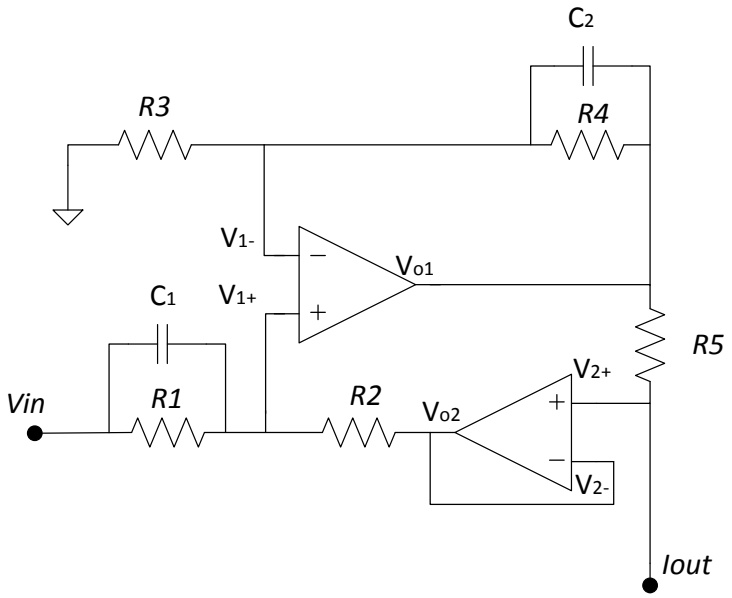


Figure 3.3: The dual op-amp current source.

$$V_{1+} \approx V_{1-} \tag{3.10}$$

$$I_L \approx \frac{V_{o1} - V_{2+}}{R_5} \tag{3.11}$$

$$\frac{V_{in} - V_{1-}}{R_3} \approx \frac{V_{1-} - V_{o1}}{R_4} \tag{3.12}$$

$$\frac{V_{2-} - V_{1+}}{R_2} \approx \frac{V_{1+}}{R_1} \tag{3.13}$$

$$V_{2+} \approx V_{2-} \tag{3.14}$$

If one then sets $R_1 R_4 = R_2 R_3$, the current flowing through the variable lode R_L is given by:

$$R_I \approx \frac{R_2}{R_1} \cdot \frac{V_{in}}{R_5} \tag{3.15}$$

In this configuration the introduction of an additional op-amp results in an advantage. For an ideal op-amp the current flowing form R_5 will see the theoretically infinite input resistance of the op-amp, and it will therefore flow through the load resistance R_L.

3.2 Current Measurements

According to KEITHLEY (2004) there are two basic techniques for low current measurements: the shunt ammeter technique, and the feedback ammeter technique. This section will look at the differences between the two methods, and go through their strengths and weaknesses.

3.2.1 Shunt Ammeter

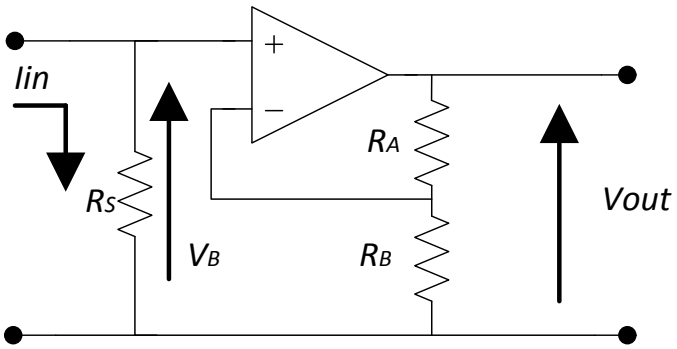


Figure 3.4: The shunt ammeter.

Figure 3.4 on the preceding page shows a shunt ammeter. The circuit is constructed by shunting the input of a voltmeter with the resistor R_s. When the current I_{in} flows through the shunt resistor R_s, there will be a voltage drop across it. This voltage is measured by the voltmeter, and the resulting output voltage is given by:

$$V_{out} = I_{in} R_S \left(1 + \frac{R_A}{R_B}\right) \tag{3.16}$$

To minimize the shunt ammeters influence on the system to be measured, the shunt resistance R_S should be kept as small as possible. This is because the voltage drop caused by the current flowing through the current measuring device can affect the circuit being measured. This is effect referred to as a voltage burden. Small resistor values also have the added benefit of better accuracy, voltage, time and temperature stability. In addition to this, a low value resistor also reduces the input time constant and thus results in a faster instrument response time. The disadvantage of small resistor values is that they degrade the signal-to-noise ratio of the measurement. This means that in order to measure small currents, the actual value of R_S must be large in order to get a sufficient voltage drop over the resistor.

3.2.2 Feedback Ammeter

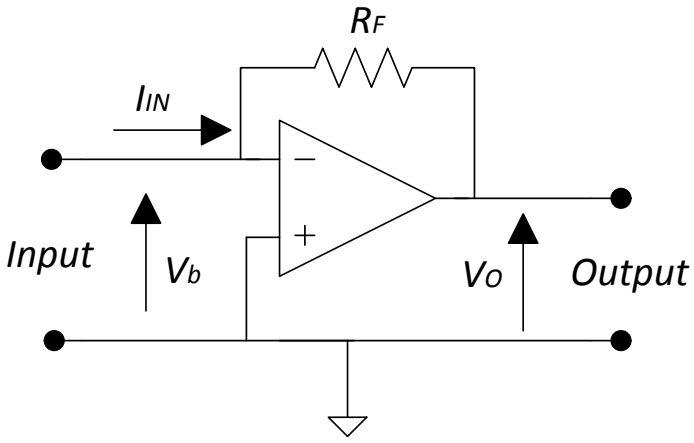


Figure 3.5: The feedback ammeter.

The feedback ammeter or transimpedance amplifier is shown in Figure 3.5. In this measurement technique the input current I_{in} flows through the feedback resistor R_F. If the op-amp has a low offset current, the output voltage is given as

$$V_O = -I_{IN} R_F \tag{3.17}$$

where the output voltage is a measure of the input current, and the overall sensitivity is given by the feedback resistor R_f.

The feedback ammeter does not have a shunt resistor, this means that the voltage burden V_b is reduced to practically zero. The feedback ammeter is therefore better suited for measuring low currents. Since the settling time of the instrument is proportional to the size of R_S multiplied by the total capacitance of the cabling and the op amp input capacitance, the feedback configuration results in faster measurements.

3.3 The Instrumentation Amplifier

Figure 3.6 shows the basic structure of an three op-amp instrumentation amplifier (in-amp). An in-amp is a device that amplifies the difference between two input voltage signals while rejecting signals that are common to both inputs. This makes the in-amp well suited for extracting small signals from signal sources, and its output is given by the following formula:

$$V_{out} = A(V_1 - V_2) = A\Delta V \tag{3.18}$$

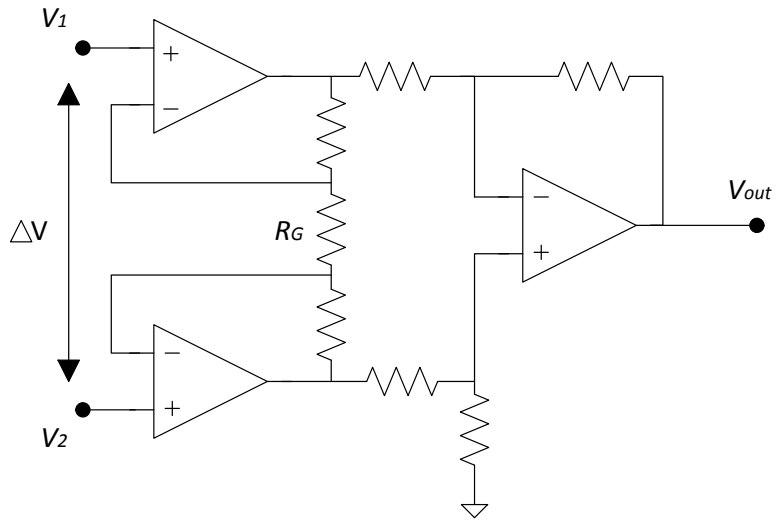


Figure 3.6: Basic three op-amp instrumentation amplifier schematic.

where V_1 is the noninverting input, V_2 is the inverting input, ΔV is the voltage difference and A is the gain.

An important property of the in-amp is its common-mode rejection (CMR) which is its ability to amplify signals that are differential and cancel out any signals that are common. The common-mode gain is the ratio of change in output voltage to change in common-mode input voltage.

The differential mode gain is the gain between the input and output for voltages applied across the two inputs. The common-mode rejection ratio (CMRR) is then the ratio between the differential gain and the common-mode gain. (Kitchin et al. 2006)

Another important property when measuring a differential potential is the input impedance of the in-amp which should be high. This is to ensure that the instrument do not burden the measurand.

As one can see from Figure 3.6 on the previous page it is possible to build an in-amp from tree op-amps and seven resistors. This implies that the designer needs to select appropriate op-amps and balance the circuit manually to achieve high CMR. An often more suitable option are to select a monolithic IC op-amp produced by one of the big IC manufactures.

The advantages of selecting a monolithic IC in-amp are that the passive and active components will be located on the same die. This means that they can be closely matched by wafer laser trimming and other manufacturing techniques. Ensuring a high CMR, matched temperature characteristics, better performance, and low price. Using one IC will also reduce the PCB space and complexity.

3.4 Analog Isolation Techniques and Patient Safety

In medical applications it is often essential to provide some sort of protection between the patient and the electrical equipment. The main reason for this is to prevent possible hazards like electrical shock. To accomplish this, some sort of isolation barrier or galvanic isolation is used. Galvanic isolation can be accomplished by the use of three different technologies: transformer isolation, capacitor isolation and opto-isolation. In addition to safety, galvanic isolation also has the added benefit of breaking ground loops, and thus reducing line frequency interference.

An isolation transformer is a transformer where the primary and secondary windings are physically separated from each other. Isolation transformers are often 1:1, and the main purpose is not to provide voltage transformation, but signal isolation. The isolation transformer works by the same principle as a normal transformer, but they have an additional safety screen between the primary and secondary windings. This screen is connected to external ground, and there is no connection between external ground and the neutral connection used as a reference by the isolated system. This removes common mode noise between the external ground and the floating neutral.

Capacitor isolation use circuit-specific capacitors that are constructed so that they shunt the energy generated by high voltage impulses, transients or surges to ground.

An opto-isolator or optocupler use optical transmission to transfer an electrical signal, and in its simplest form it consists of a LED in combination with a photo diode. This configuration works for digital

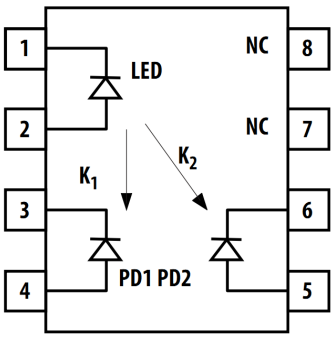


Figure 3.7: Schematic of the HCNR201

applications, but since the linearity and stability is a function of time and temperature, it is not suitable for analog applications.(Bronzino 2000) This problem can be solved by using a closely matched photo diode design like the one used in HCNR200/201 by Avago Technologies. For these types of linear optocuplers a LED is used to illuminate two matched photo diodes. One photo diode is then used to monitor and stabilize the circuit driving the LED. This results in a photo current on the output that is linearly related to the light output of the LED on the input.

3.4.1 Isolation Amplifiers

An isolation amplifier is a commonly used interconnection for analog signals, and they can be implemented using different technologies. Most integrated solutions rely on some sort of analog to digital, or voltage to frequency conversion to provide input/output and noise insulation. These designs often use transformers or high-speed digital optocouplers which often results in complex and expensive solutions. A flexible and low-cost alternative to the integrated solutions are to construct the amplifier around a linear optocoupler like the HCNR201 shown in Figure 3.7.

When constructing this type of isolation amplifier, the optocoupler can be configured as either photovoltaic or photoconductive. When a photovoltaic configuration is used the photo diode is unbiased, and when a photoconductive configuration is used the photo diode is reverse-biased. The biasing of the photo diode influence the current flow in the photo diode when no light shines on it, and this effect is refered to as dark current. In a photoconductive configuration the dark current is

proportional to the bias voltage. In a photovoltaic configuration the dark current is close to zero.

When low noise, high linearity, and drift performance are important design parameters, a photovoltaic configuration is the best alternative as it can meet or exceed the equivalent of 12 bit AD performance. When maximum signal bandwidth is desired a photoconductive configuration is more appropriate, and its linearity and drift characteristics are comparable to a 9 bit AD converter.(Vishay 2008)

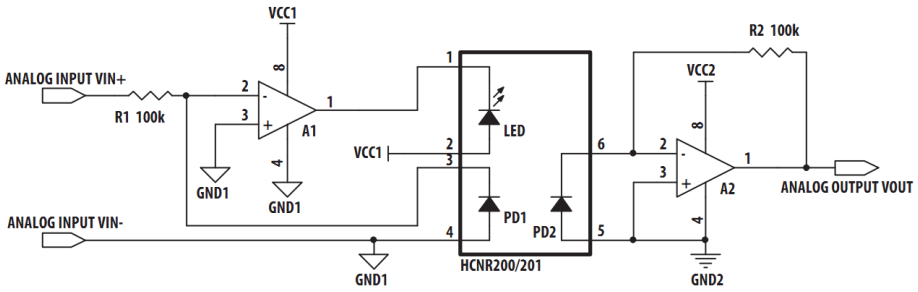


Figure 3.8: Unipolar photovoltaic amplifier (reprinted from Vishay emiconductors (2008)).

The Unipolar Photovoltaic Amplifier

Figure 3.8 shows the schematic of a photovoltaic isolation amplifier constructed from the optocoupler HCNR201 and two op-amps. On the input stage the external feedback amplifier is used in combination with PD1 to monitor the light generated from the LED, and adjust the LED current for nonlinearities. The op-amp A1 will always try to deliver zero volts across the photo diode PD1. This means that if a positive voltage is placed on the input, A1 will swing to the negative rail and create a flow of current through the LED. The positive input voltage will also create a current through R1. The light generated by the LED will be detected by PD1, and it will generate the photo current I_{PD1}. If one assumes A1 to be an ideal amplifier, all the current through R1 will flow through PD1. Since the + input of A1 is set to 0 V, the current through R1, and I_{PD1}, is equal to $I_{PD1} = V_{IN+}/R1$. From this relation we can see that I_{PD1} is dependent on the input voltage and the value of R1, and independent of the light output characteristics of the LED. If there is a change in the LED temperature, A1 will adjust, and a constant current in PD1 will be maintained. The current I_{PD1} is exactly proportional to V_{In+}, and this gives a linear relationship between the input voltage and the photo diode current. By stabilizing and linearizing I_{PD1} , the light from the LED is also stabilized and linearized. Since the LED shines on both of the photo diodes, I_{PD2} will be stabilized as well. In reality I_{PD1} and I_{PD2} are not

perfect, so the transfer gain coefficient K_3 is included: $I_{PD1} = K_3 \cdot I_{PD2}$ In the output stage the transimpedance amplifier converts I_{PD2} back into a voltage according to $V_{Out} = I_{PD2} \cdot R2$. The combination of the above equations yields the expression:(Technologies 2010)

$$\frac{V_{Out}}{V_{In}} = K_3 \cdot \frac{R2}{R1} \tag{3.19}$$

3.5 Lock-In Detection

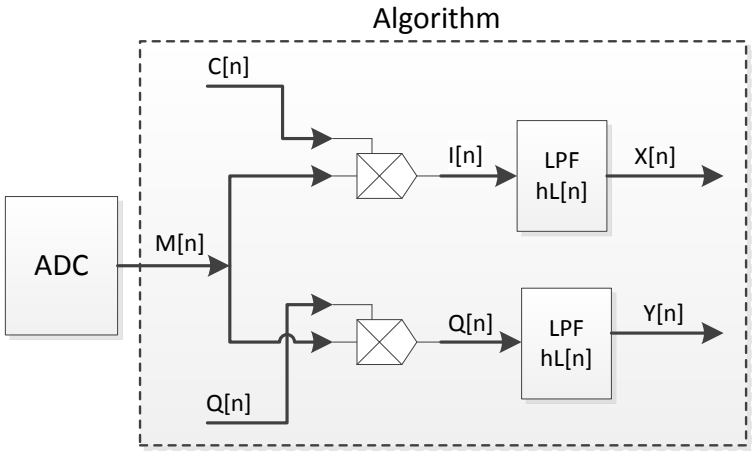


Figure 3.9: Digital quadrature demodulation hardware algorithm.

Lock-in detection, or synchronous detection is one of the most effective and widely used techniques for recovering signals dominated by noise. In principle the technique involves a reference signal that is demodulated at the same frequency as the signal of interest. The demodulated signal is then lowpass filtered to obtain the result. The demodulation and filtering enables the lock-in to focus on the signal of interest and ignore other frequencies.

The rest of this section is based on the article "Digital Lock-In Detection for Discriminating Multiple Modulation Frequencies With High Accuracy and Computational Efficiency" by Masciotti et al. (2008).

If a source signal S(t) is applied to a system that yields a measurable response M(t), it can be written mathematically as:

$$S(t) = dc_s + A_s cos(2\pi f_m t + \varphi_s) \tag{3.20}$$

$$M(t) = dc_m + A_m cos(2\pi f_m t + \varphi_m) \tag{3.21}$$

Where dc_s is the dc component of S(t), dc_m is the dc component of M(t), A_s is the amplitude of S(t), A_m is the amplitude of M(t), φ_s is the phase of S(t), φ_m is the phase of M(t), f_m is the modulation frequency and t is time.

In order to analyze the system, one measure the amplitude A_m and the phase φ_m.

In an analog lock-in system the measured signal M(t) is multiplied by a reference signal of the same frequency by a mixer, and the signal is then low-passed filtered. When both the phase and amplitude of the measured signal is of interest quadrature demodulation is used. Figure 3.9 on the preceding page shows a diagram that displays the general algorithm used in quadrature demodulation. The demodulator works by multiplying the signal M[n] by the in-phase reference C[n] (sin) and the quadrature reference S[n] (cos). This results in the two signals I[n] and Q[n] which is the in-phase and quadrature products of the multiplications. These two signals are then lowpass filtered to obtain the final result, which is proportional to the difference in amplitude and phase between the reference and the measured signal.

From Figure 3.9 an analog-to-digital converter (ADC) is used to transfer the analog signal over to the digital domain. If this ADC takes N_s samples of the signal M(t) at a sampling frequency of f_s that satisfies the Nyquist criterion $f_s > 2f_m$, the discrete signal is given by:

$$M[n] = dc_m + A_m cos\left[\frac{2\pi f_m n}{f_s} + \varphi\right], \quad 0 \leq n \leq N_s - 1 \tag{3.22}$$

The measurement time T_m, which is the time required to take the N_s samples is

$$T_m = \frac{N_s}{f_s} \tag{3.23}$$

If the monitoring of the source does not require rates as high as f_s the measurement time is given by the settling time of the lowpass filter. The noise reduction exhibited by the lock-in detector increases with T_m, and the value of N_s is therefore chosen as a trade off between the noise reduction and the measurement time T_m.

In order for the digital quadrature demodulation to work, the in-phase and quadrature signals C[n] and S[n] have to be represented as discrete signals on the following form:

$$C[n] = cos\left[\frac{2\pi f_m n}{f_s}\right] \tag{3.24}$$

$$S[n] = sin\left[\frac{2\pi f_m n}{f_s}\right] \tag{3.25}$$

The digital signals S[n] and M[n] are both generated internally by the system. This means that they are free of any other noise than the numerical precision errors as long the ADC is synchronized. With C[n] and S[n] on discrete form, the in-phase and quadrature signals I[n] and Q[n] multiplied by the measured M[n] can be written on the following form:

$$I[n] = M[n] \times cos\left[\frac{2\pi f_m n}{f_s}\right] \tag{3.26}$$

$$I[n] = \frac{1}{2}A_m cos(\varphi_m) + dc_m cos\left[\frac{2\pi f_m n}{f_s}\right] + \frac{1}{2}A_m cos\left[\frac{4\pi f_m n}{f_s} + \varphi_m\right] \tag{3.27}$$

$$Q[n] = M[n] \times sin\left[\frac{2\pi f_m n}{f_s}\right] \tag{3.28}$$

$$Q[n] = \frac{1}{2}A_m sin(\varphi_m) + dc_m sin\left[\frac{2\pi f_m n}{f_s}\right] + \frac{1}{2}A_m sin\left[\frac{4\pi f_m n}{f_s} + \varphi_m\right] \tag{3.29}$$

When the sinusoids $I[n]$ and $Q[m]$ are multiplied their spectral components A_m and dc_m are frequency shifted. As one can see from Equation 3.27 and Equation 3.29 the equations are composed of three distinct components. The first is the dc component due to the phase and amplitude of the original sinusoidal signal. The second component has an amplitude of dc_m at the original modulation frequency. And the third component is at twice the modulation frequency and has an amplitude of $\frac{A_m}{2}$.

To obtain the real part X[n] and the imaginary part Y[n] of the measured sinusoid, one can filter out the unwanted ac term by convolving I[n] and Q[n] with a lowpass filter $h_L[n]$ as shown below:

$$X[n] = h_L[n] \otimes I[n] \tag{3.30}$$

$$X[n] \approx \frac{1}{2}A_m cos(\varphi_m) \tag{3.31}$$

$$Y[n] = h_L[n] \otimes Q[n] \tag{3.32}$$

$$Y[n] \approx \frac{1}{2}A_m sin(\varphi_m) \tag{3.33}$$

3.6 Inherent Noise

Noise is inherent in the circuit elements of analog systems and can not be totally eliminated. Since the noise of a system affects system performance metrics such as signal-to- noise ratio (SNR) it has to be taken into consideration. This section will take a closer look at the three most common sources of noise in analog systems.

Thermal noise

Thermal noise is often referred to as Johnson noise or Nyquist noise, and it is generated when thermal energy causes free electrons to move randomly in a resistive material. This movement happens regardless of any applied voltage or current flow, and is similar to the Brownian motion of particles. The random vibration of the electrons is dependent on temperature, and will increase for temperatures above absolute zero. When the electrons move, their charge contributes to many small current surges in the material and even though the average of these movements is zero they contribute to an instantaneous voltage across the conductor. The power density of thermal noise has a uniform or flat power distribution over all frequencies and is therefore referred to as white noise.(Motchenbacher et al. 1993)

At frequencies below 100 MHz, thermal noise can be calculated by using Nyquist's relation for the thermal noise voltage E_t in rms voltage(TI 2002):

$$E_t = \sqrt{4k_bTR\Delta f} \tag{3.34}$$

From Equation 3.34 the thermal current noise I_t is then given by:

$$I_t = \frac{E_t}{R} = \sqrt{\frac{4k_bT\Delta f}{R}} \tag{3.35}$$

where k_b is Boltzmann's constant, T is the absolute temperature in Kelvin, R is the resistance in ohms, and Δf is the noise in hertz over the measured bandwidth.

Shot Noise

Shot noise is generated by random fluctuations in the motion of charge carriers in a conductor. When electrons encounter a potential barrier, such as a pn junction in a semiconductor, each electron and hole carries a charge. When the electron encounters the barrier, energy is stored. When the hole arrives, it shoots across the barrier and the energy is released. This will result in an impulse of current and this pulsing flow creates a granule effect with variations referred to as shot noise.(Motchenbacher et al. 1993)

According to TI (2002) some of the characteristics of shot noise are:

- It is always associated with current flow, and it stops when the current flow stops.
- It is independent of temperature.
- It is spectrally flat or has a uniform power density which results in a constant value when plotted versus frequency.
- It is present in any conductor.

The RMS shot noise current is equal to:

$$E_s = \sqrt{2qI_{DC}\Delta f} \tag{3.36}$$

where q is the electronic charge (1.602×10^{-19} Coulombs), I_{DC} is the direct current in amperes, and Δf is the noise bandwidth in hertz.

Flicker Noise

Flicker noise or 1/f noise as it is often referred to, has a spectral density that increases without limit as frequency decreases. This type of noise is pervasive in nature, and it is present in all active and many passive devices. The origin of flicker noise is not completely understood, but the generation and recombination of carriers in surface energy states and the density of surface states are believed to be important factors. Improvements in the treatment of surfaces during manufacturing have reduced flicker noise. (Motchenbacher et al. 1993)

TI (2002) states that some of the most important characteristics of flicker noise are:

- It increases as the frequency decreases.
- It is associated with a dc current in electronic devices
- It has the same power content in each octave

The voltage noise E_f is given(TI 2002):

$$E_f = K_V \sqrt{\left(ln \frac{f_{max}}{fmin} \right)} \tag{3.37}$$

And the current noise I_f is given:

$$I_f = K_I \sqrt{\left(ln \frac{f_{max}}{fmin} \right)} \tag{3.38}$$

where E_f and I_f are proportionality constants in volts or amperes representing E_f and I_f at 1 Hz, and f_{max} and f_{min} are maximum and minimum frequencies in Hz.

3.7 Noise analysis of a Transimpedance Amplifier

Figure 3.10 shows the noise model analysis circuit for the feedback ammeter. To analyze noise in the frequency domain, equivalent generators for voltage noise are applied to the circuit. These generators are the square root of the noise power densities, and represent the noise density over frequency for the different elements:

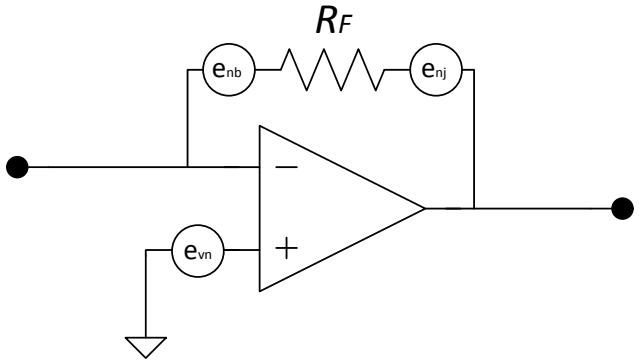


Figure 3.10: Noise model for the feedback ammeter.

- e_{nb} is the input current noise of the op-amp multiplied by the feedback resistor R_F
- e_{nv} is the input voltage noise of the operational amplifier
- e_{nj} is the thermal noise or the Johnson noise of the resistor

I order to calculate the total noise in the system; it is assumed that the different noise contributions are not correlated. One can then find the total rms noise signal e_{tot} by the following formula:

$$e_{tot} = \sqrt{e_{nb}^2 + e_{nv}^2 + e_{nj}^2} \tag{3.39}$$

In general analog design the resistors are normally kept relatively small in order to achieve a trade off between the thermal noise and the current consumption. For the transimpedance amplifier this is not possible as the gain is given by $V_O = -I_{IN}R_F$ which implies that in order to get a high gain, the value of the feedback resistor R_F also have to be high.

In order to calculate the total noise e_{tot} of the transimpedance amplifier, the operational amplifier OPA140 is chosen as an example. The OPA140 is a high precision JFET input amplifier that according to its data sheet features the following noise characteristics:

Input current noise density = 0.8 $fA/\sqrt{Hz}$

Input voltage noise density = 8 $nV/\sqrt{Hz}$

If one then assumes a bandwidth of 1 kHz, a feedback resistor of 1 MΩ and a room temperature of 22° C (or 295.15 K) the different noise contributions are given as:

$$e_{nb} = \frac{0.8\ fA}{\sqrt{1\ kHz}} \cdot 1\ M\Omega \approx 25.3\ pV$$

$$e_{nv} = \frac{8\ nV}{\sqrt{1\ kHz}} \approx 253.0\ pV$$

$$e_{nj} = \sqrt{4k_bTR\Delta f} = \sqrt{4 \cdot k_b \cdot 295.15\ K \cdot 1\ M\Omega \cdot 1\ kHz} \approx 4.04\ \mu V$$

$$e_{tot} = \sqrt{e_{nb}^2 + e_{nv}^2 + e_{nj}^2}$$

$$= \sqrt{(25.3\ pV)^2 + (253.0\ pV)^2 + (4.04\ \mu V)^2} \approx 4.05\ \mu V$$

As one can from e_{tot}, the thermal noise of the feedback resistor R_f is the dominant noise source. From e_{tot}, the thermal noise of the feedback resistor R_f is the dominant noise source. This is because the individual noise contributions are added by the square.

Signal-to-noise ratio is defined as the power ratio between a signal and the noise. For the transimpedace amplifier this is given by:

$$SNR[dB] = 10log_{10}\left(\frac{I \cdot R_f}{e_{tot}}\right) \tag{3.40}$$

To get a general understanding of the threshold of detectability for the transimpedace amplifier one can assume that the SNR is equal to 1. This results in SNR = 0 dB and Formula 3.40 can now be written as:

$$10log_{10}\left(\frac{I \cdot R_f}{e_{tot}}\right) = 0 \tag{3.41}$$

Solved for I we get:

$$I = \left(\frac{I \cdot e_{tot}}{R_f}\right) \tag{3.42}$$

With $e_{tot} = 4.05\ \mu V$ and $R_f = 1\ M\Omega$ and an SNR of 1 Formula3.42 gives the lowest detectable threshold in this example as 4.05 nA.

3.8 Sampling Analog Signals

By sampling a continuous analog signal we attempt to represent the time dependency of the signal by a set of discrete samples taken at fixed intervals t_s. The information between each sample in the original signal is lost. In order to ensure that the representation of the analog signal is accurate the intervals of t_s must therefore be carefully chosen.

The sampling theorem also referred to as the Shannon sampling theorem states that an input signal must be sampled at a minimum rate grater than twice the highest-frequency component in the signal. This critical sampling rate is called the Nyquist rate, and stated as a formula, it says that $\frac{f_s}{2} > f_a$, where f_a is the maximum frequency of the signal being sampled.

If one violates the Nyquist criterion it is referred to as undersampling and the result is a phenomena called aliasing.

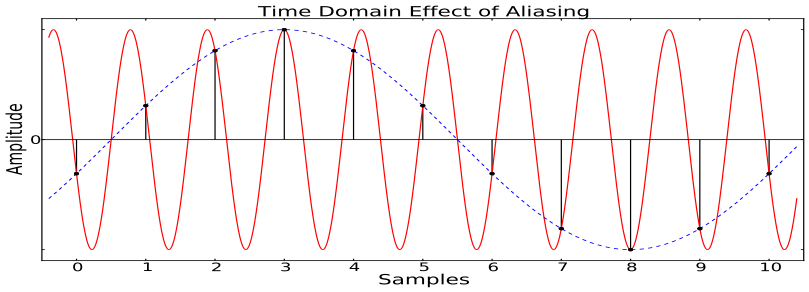


Figure 3.11: Aliasing making two different sinusoidal fit the same set of samples.

From Figure 3.11 one can see an example of aliasing, a sinusoid of frequency f_{red} = 0.9 (solid line) is sampled at f_s = 1.0 which is below the Nyquist rate.

From the samples taken of f_{red}, it looks as if the sinusoid f_{blue} = 1.0 (dotted line) is the correct representation.

This effect is called frequency fold back, and it occur because every frequency above the $\frac{f_s}{2}$ is folded back into the sub-$\frac{f_s}{2}$ range.

Until now the signals used to explain sampling theory have all been pure sinusoids of one frequency. However in the real world, this is rarely the case. Analog signals are often contaminated by different types of noise and spurious signals, and any unwanted interference with a frequency beyond $\frac{f_s}{2}$ will alias.

To avoid this it is necessary to filter the analog signal before it is sampled by the ADC. The influence of the disturbance on the signal being sampled is dependent on the power spectrum of the out-of-band noise. If

the noise power is much smaller then the resolution of the ADC it will be undetectable.

A lowpass filter is a filter that passes low-frequency signals, and reduces the amplitude of signals with frequencies above its cutoff frequency. This is why lowpass filters are well suited as antialiasing filters.

An ideal antialiasing filter should pass all the frequencies below its cutoff frequency, and block all undesired higher frequencies. However for a real analog filter this implies that the filter needs a very small transition band, resulting in a high number of poles (order), higher complexity, and higher price. An often used technique to lower the complexity and price of the analog antialiasing filter is to use oversampling. In an oversampling system the samples are taken at higher rates than what is strictly needed. This result in a spread in the quantization noise-energy. The noise outside the region of interest can then be filtered out by a digital lowpass filter, thereby reducing the noise level in the frequency band of interest.

Classes of ADCs called Sigma-Delta converters are inherently oversampling converters. These converters are slow, compared to other converters like successive approximation, but they have benefits as high resolution, high stability, low power and low cost.

Chapter 4

Digital Theory, Communication and Software

This chapter reviews the digital theory, communication and software used to create the hardware platform presented in this thesis.

4.1 FPGA Design

Field-Programmable Gate Arrays (FPGAs) are Integrated Circuits (ICs) that contain configurable blocks of logic with interconnects that can be configured to establish connections between them. The main difference between FPGAs and more conventional fixed logic circuits like Application Specific Integrated Circuits (ASICs) are that they can be programmed multiple times. An FPGA can be used to implement the same logic as an ASIC but with the added benefit of reprogrammability.

The smallest unit in an Altera Cyclone III FPGA is the Logic Element (LE) shown in Figure 4.1 on the following page. The LE has four inputs and contains a four-input Look-up Table (LUT), a register and output logic. A LUT can be regarded as a function generator that can be used to implement any function with four variables. More detailed information regarding the Altera Cyclone III family can be found in Altera (2012). In order to define the behavior of an FPGA, the designer uses a Hardware Description Language (HDL). In this thesis VHDL will be used to describe the digital system.

4.1.1 NiosII

The Nios II shown in Figure 4.2 on the next page is a 32-bit embedded soft processor that is designed to be instantiated on Altera FPGA devices. The processor is defined in a hardware description language, and it can therefore be implemented entirely in programmable logic and memory blocks. The development of a Nios II system consists

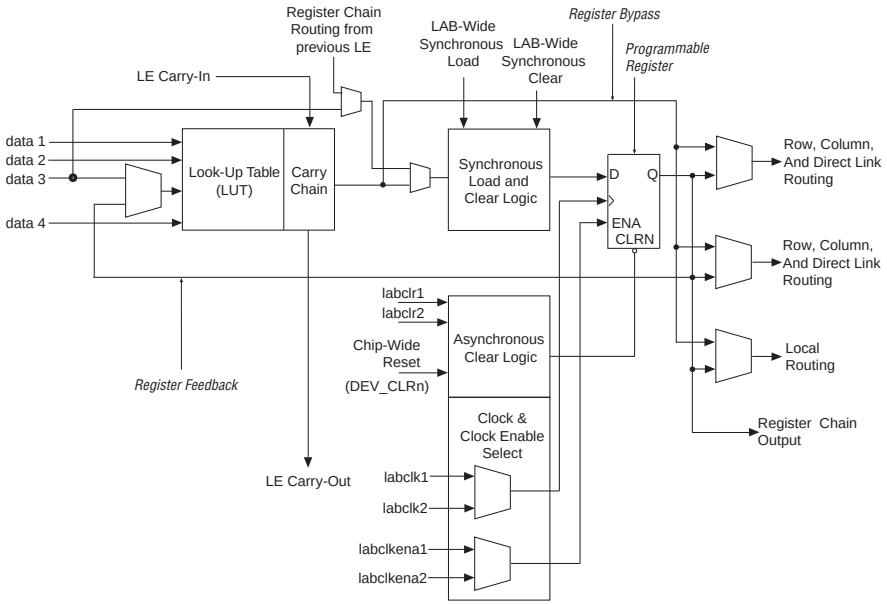


Figure 4.1: Cyclone III device family logic element (reprinted from Altera (2012)).

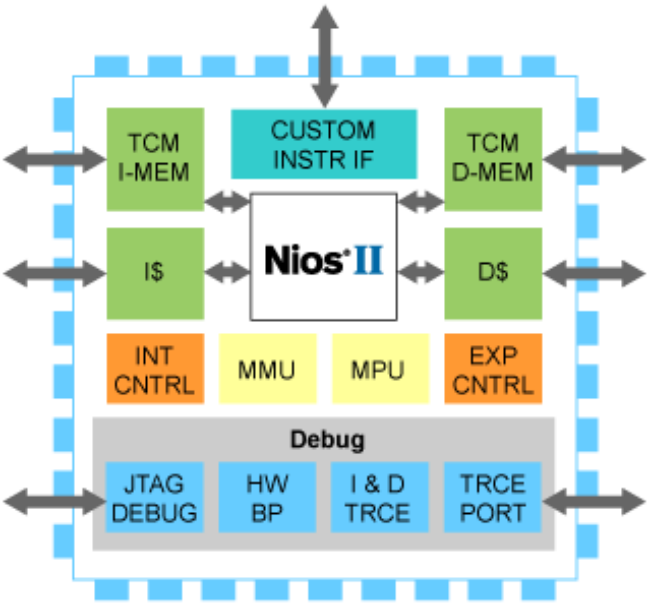


Figure 4.2: The Nios II 32-bit embedded soft processor (reprinted from Altera(2013)).

of two steps: hardware generation, and software design. The Altera Qsys integration tool is used to configure and integrate the system hardware. This tool allows the designer to choose from three different core configurations: Nios II/f (fast), Nios II/s (standard), and Nios II/e (economy). In addition to this, the designer can add any number of peripherals and memory interfaces. The Altera Embedded Design Suite (EDS) is used for software design. This package includes the Eclipse based development environment, C/C++ compiler, debugger and an instruction-set simulator.(Altera 2013b)

4.1.2 UART

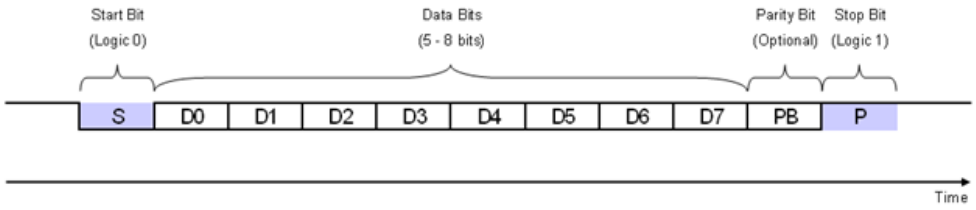


Figure 4.3: The basic UART packet format containing 1 Start bit, 8 Data bits, 1 Parity bit and 1 Stop bit (reprinted from Kong (2010)).

Universal Asynchronous Receiver/Transmitter (UART) is a character based protocol used for serial communication. When two UART devices exchange data, the transmitter takes bytes of data and transmits the individual bits sequentially. The receiving UART then reassemble the bits back into bytes. UART communication can be either full duplex or half duplex. The transmitting and receiving data lines are named Tx and Rx, and asynchronous transmission allows data to be transmitted without the use of a separate clock line. Figure 4.3 shows the basic UART packet format containing 1 Start bit, 8 Data bits, 1 optional Parity bit and 1 Stop bit. In order to synchronize the transmitter and the receiver, the two devices need to operate on the same transmission speed (Baud Rate). The Start bit is used to inform the receiver that a data word is about to be transmitted and it is used to synchronize the receiver. The stop bit marks the end of a transmission, and the optional parity bit is used for error correction.(Osborne 1980)

4.1.3 Megafunctions

Altera Megafunctions are ready-made, and tested blocks of intellectual property that is optimized for Altera devices. In Quartus the MegaWizard Plug-In tool is used to select, customize and implement a large variety of

different functions that range from standard flip-flops to FFT algorithms and FIR filters. There are mainly two types of megafunctions; the ones supplied from the Altera Megafunction Partners Program, and the ones created by Altera referred to as MegaCore Functions.(Altera 2013a)

The most basic functions are free to use, but some of the more complex functions require a license. If a license is needed the function can be tested and used according to the two operating modes. The untenthered mode is time restricted, and will disable the function after a set time period. In the tethered mode the development card or system needs to be connected to the host computer that runs the Altera design software. If this mode is supported by the implemented megafunctions, the design can operate for a long time or indefinitely.

4.1.4 SignalTap II Logic Analyzer

The SignalTap II Logic Analyzer is an Altera debugging tool used to examine the behavior of the internal signals of the FPGA in real-time. The analyzer works without extra I/O connections or probes, and it can be set up with different data gathering and trigging conditions. The captured data is stored in the memory blocks of the device, and the JTAG communication cable used to program the FPGA is used for data transfer.

4.2 Bluetooth Communication

Bluetooth is a wireless technology standard used to exchange data over short distances. It was first developed by Ericsson as a wireless alternative to RS-232 data cables, and it is now managed by the Bluetooth Special Interest Group. Bluetooth operates in the globally unrestricted Industrial, Science, and Medical radio band (ISM) that range from 2.400 GHz to 2.483 GHz. The hardware used for Bluetooth is specifically designed for small size, low cost and low power. When two bluetooth enabled devices connect to each other, it is called pairing. The device that initiates the connection is called the master and the other devices are called slaves. A master can have simultaneous connections with up to seven slaves in ad-hoc networks known as piconets. When several piconets are connected it is referred to as a scatternet. The devices in a piconet use a specific frequency hopping pattern which is determined by the Bluetooth specification address and clock of the master. In short the hopping pattern is a pseudo-random ordering of the 79 frequencies in the ISM band, and the pattern can be adapted to exclude a portion of the frequencies that are used by interfering devices. The range is application specific and can operate over a distance of 10 meters to 100 meters depending on the device class and desired power consumption.(Bluetooth 2013)

4.3 Android

Android is a Linux based open-source software stack containing an operation system, middel-ware and key applications. Android is created for devices like mobile phones and tablets, and it is maintained and developed by the Android Open Source Project (AOSP) led by Google. Applications for Android are written in the Java programming language, and the Android SDK (Software Development Kit) provides the tools and API libraries necessary to build, test and debug applications.

4.3.1 Android Architecture

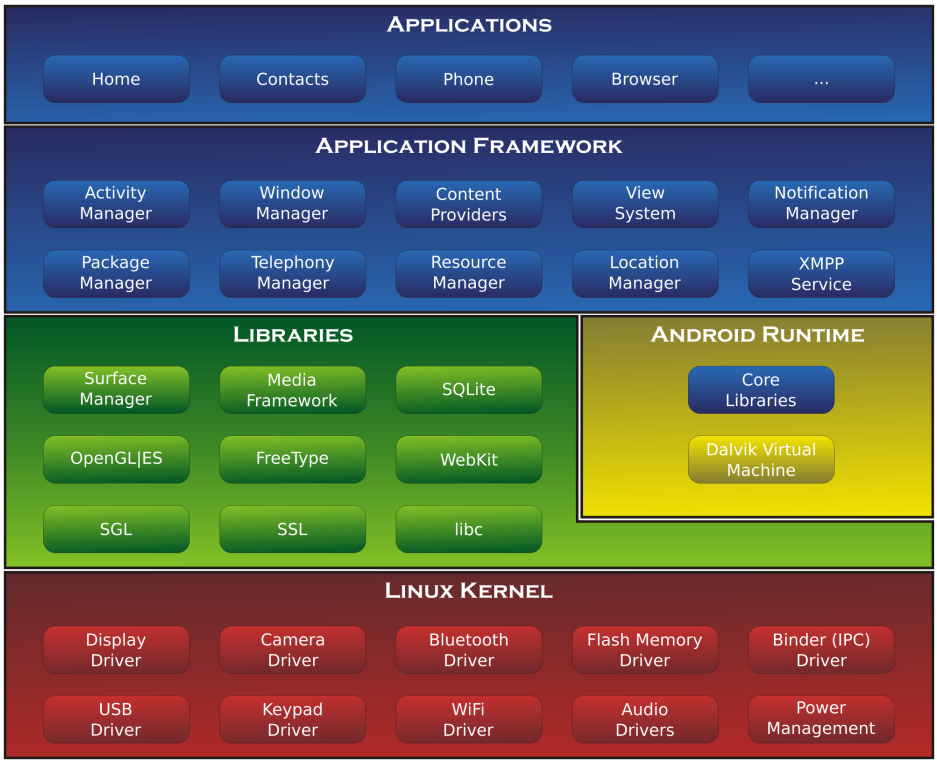


Figure 4.4: The architecture of the Android operating system. (reprinted from *Android-System-Architecture*)

Figure 4.4 shows a diagram of the major components of the Android operation system. The diagram is divided into different sections, and the top section is the Application layer. This layer consists of Java applications like the preinstalled core applications for SMS, contacts, email client, calendar, and other basic functions. The second layer is the Application Framework; this layer enables the developer access to the same framework APIs as the core application. This enables the developer to take advantage of the device hardware, run background services, set alarms, etc. The

third layer contains the C/C++ libraries used by the different components of the system. These libraries are exposed to the developer through the Application Framework. In addition to the C/C++ libraries this layer also contains the Android Runtime. The Android Runtime contains the core library and the Dalvik Virtual Machine. The core libraries provide most of the core functionality for the Java programming language. The Dalvik Virtual Machine enables an Android application to run in its own process, and it is optimized for a minimal memory footprint. The bottom layer is the Linux Kernel, and this layer is responsible for core system services, and serves as an abstraction layer between the hardware and the rest of the software stack.

4.3.2 Application Fundamentals

An Android application is constructed from four different basic application components. These components are activities, services, broadcast receivers and content providers, and the developer can choose to use them separately, or in combinations.

- Activities: An activity represents a single screen with a user interface that responds to events. An Activity can start other activities, and they have the ability to return values. This means that different activities can be combined to create a cohesive user experience.

- Services: A service is a component that runs in the background, and they have no user interface. These components are used to perform long-running operations like playing music or maintaining data transfer, without blocking the user interaction with the activity. A service can be started by other components like activities. In order for a different component to interact with a service, they need to bind to the service.

- Broadcast Receivers: A broadcast receiver is used to respond to system broadcast announcements like screen turn off or low battery. These components have no user interface, but they can be used to create a status bar notification, and in that way serve as a gateway to other components.

- Content Providers: A content provider enables an application to access shared application data through the file system, a SQLite database, or other storage locations available to the application. The content provider manages application data, and the application needs the proper permissions to query the content provider for information.

For more information on Android development see Android (2013).

Chapter 5

Design and Development

In this chapter the design and development process of the developmet platform is reviewed. Included is a look at the general system requirements and coverage of the implementation. Each subsystem and their design requirements are covered in their respective sections.

5.1 General System Requirements

To demonstrate its capabilities, the development platform presented in this thesis is going to conduct endosomatic and exosomatic measurements on skin. These measurements should be conducted using a custom built front-end connected to skin electrodes. The generation and sampling of analog signals needed for the measurements are to be generated by a custom built data acquisition card, and the signal processing are to be implemented using FPGA technology. The representation of the measurements should be displayed on an Android mobile application. The communication between the Android application and the hardware platform should be done by the use of Bluetooth technology. In addition to the specifications already mentioned, the system should also have the following requirements:

- The digital signal processing should be implemented on the FPGA in such a way that the system is expandable and scalable.
- The Android application should be able to store data for later analysis and processing.
- The hardware should satisfy safety requirements so that human measurements can be conducted.

5.2 System Overview

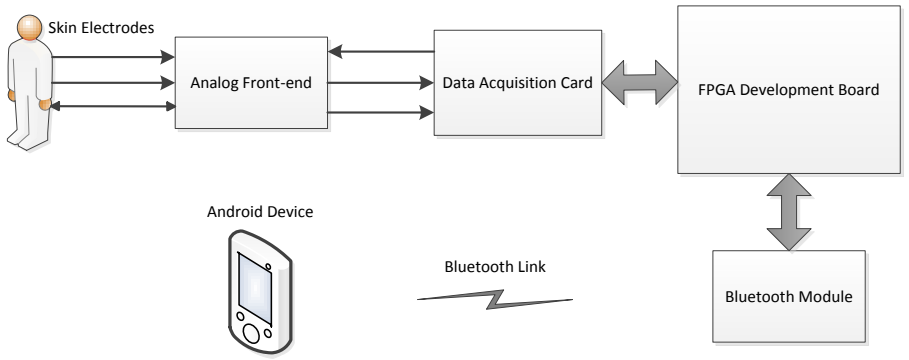


Figure 5.1: Overview of the FPGA based development platform.

An overview of the selected development platform and its main components are shown in Figure 5.1. As one can see from the diagram the system is divided into six distinct parts: the skin electrodes, the analog front-end, the Data Acquisition Card, the FPGA development board, the Bluetooth module and the Android platform. The arrows in the diagram show the direction of the control and communication signals.

In short the functions of the different modules are as follows:

- **Skin electrodes:** Transduce the analog signals to and from the skin.
- **Analog Front-End:** Apply a constant AC current to the skin electrodes, measure and amplify the analog signals from the electrodes and provide galvanic separation.
- **Data Acquisition Card:** Generate the analog signals needed for the front-end and sample the analog signals for the FPGA development board.
- **FPGA Development Bord:** Control the Data Acquisition Card and the Bluetooth communication module and perform signal processing algorithms.
- **Bluetooth module:** Establish a wireless connection between the hardware and the software platforms.
- **Android platforms:** Receive, display and save the transferred data.

5.3 Skin Electrodes

As discussed in Section 2.4 the electrode geometry and its materials influence the performance of the skin measurement. The electrodes used in this thesis are the pre-wired Kendall 1050NPSM Neonatal Electrodes from COVIDIEN. These electrodes are made of Ag/AgCL and use a conductive adhesive hydrogel to provide adhesion to the skin. The electrodes are delivered in packets of three color coded electrodes that have a diameter of 25 mm each. This gives an EEA of 4.9 cm^2 per electrode.

The Kendall 1050NPSM Neonatal Electrodes have previously been used for conductance measurements by Johnsen (2009), Tronstad et al. (2010) and Jabbari et al. (2010) with good results. According to Tronstad et al. (2013) the use of solid gels have the following advantages:

- Solid gels are less sensitive to motion artifacts.
- Solid gels have a better ability than wetter gels to return the skin conductance to the baseline during recovery after a period of sweating.
- Solid gels like the one used in Kendall 1050NPSM Neonatal Electrodes introduce a smaller change in the level of skin conductance over time, due to a smaller amount of free water in the gel and its lower viscosity.

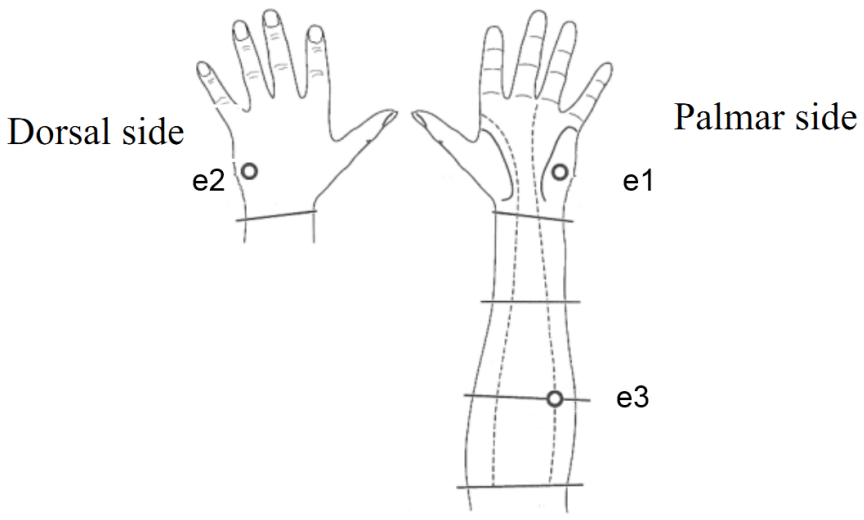


Figure 5.2: Placements of the electrodes (adapted from Jabbari et al. (2010)).

The selected electrode placement is shown in Figure 5.2.

5.4 Analog Front-end

The purpose of the Analog Front-end is to conduct endosomatic and exosomatic measurements as described in Section 2.5. In addition to this, the front-end also needs to supply galvanic separation between the parts of the circuit in contact with skin, and the rest of the system. In order to make the measurement system flexible and easy to modify the Analog Front-end will be implemented as a separate PCB.

Figure 5.3 shows an overview of the selected design. As one can see from the diagram the front-end is constructed of the following modules: an isolated DC-DC converter, an isolation amplifier, a dual op-amp Howland current source and a preamplifier. Operating power for the front-end is supplied by the FPGA development board.

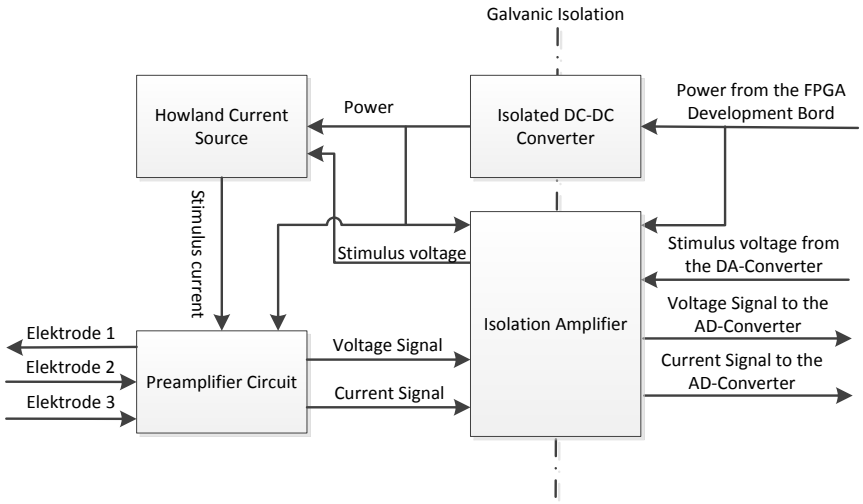


Figure 5.3: Overview of the Analog Front-end.

5.4.1 Power Isolation and Distribution

The Analog Front-end PCB is supplied with regulated 12 V DC from the FPGA development board. The dual op-amp Howland current source and the preamplifier needs a supply of $\pm$ 12 V DC and since they are connected to human skin the power to these modules needs to be isolated from the rest of the system. The power consumption of the modules protected by the isolation barrier including the drivers for the isolation amplifier was estimated to worst case value of 780 mW. Since DC-DC converters generally get more efficient for higher loads, the estimated

power consumption was used to select a converter with the correct power rating.

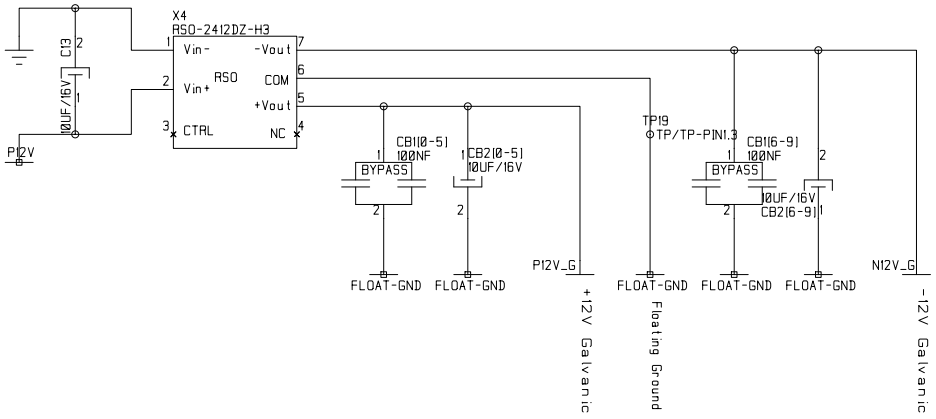


Figure 5.4: The isolated DC-DC converter RSO-2412DZ/H3 from Recom Power.

To satisfy the mentioned requirements the isolated dual output DC-DC converter RSO-2412DZ/H3 from Recom Power was selected. This converter delivers 1 Watt of power, has an output voltage of $\pm$ 12 V and accepts input voltages ranging from 4.5 V to 18 V. The converter is certified according to the EN-60601-1 standard for medical electrical equipment. According to its data sheet the isolation is tested for a voltage of 3000VDC for 1 second and is rated to 1500VAC@60Hz for 1 minute. In addition to this, the unit also has continuous short circuit protection.

The efficiency at full load is typically 75-80%, and it can deliver an output current of $\pm$42 mA. To make sure that the maximum capacitive load of $\pm$ 330μF was not exceeded, the capacitive contribution of the isolated part of the front end was calculated and found to be 101 μF. The maximum output ripple of the converter is limited to 20 MHz and has a maximum value of 50 mV_{P-P}. In accordance with the recommendations given in the data sheet a 10 μF capacitor was added between terminal 1 and 2 as shown in Figure 5.4

5.4.2 Isolation Amplifier

For safety reasons, the interface of the Analog Front-end PCB needs to be isolated from the rest of the system. As discussed in Section 3.4.1 this can be accomplished by integrated solutions, but in order to reduce costs an analog isolation design based on the optocoupler HCNR201 from Avago Technologies was selected. The schematic for this isolation amplifier is

shown in Figure 5.5, and its functionality is explained in Section 3.4.1. As one can see from the circuit, it consists of two OP413 op-amps from Analog Devices that are used to drive the input and output of the optocoupler. The OP413 op-amp were chosen for its low noise and drift characteristics and its availability in quad packs. The HCNR201 LED is supplied with +12V DC and in order to limit its current the 470k Ω resistor R7 was added. The 4.7 pF capacitor C9 and 330 pF capacitor C7 are added to increase stability and they have the added benefit of limiting the bandwidth and therby reducing noise. Because of its higher value C12 will be the determining factor. With the potentiometer adjusted to 150 kΩ (gain of 1 : 1) the cutoff frequency f_c of the lowpass filter is given by:

$$f_c = \frac{1}{2\pi RC} = \frac{1}{2\pi \cdot 150k\Omega \cdot 330pF} \approx 3.5\ kHz \tag{5.1}$$

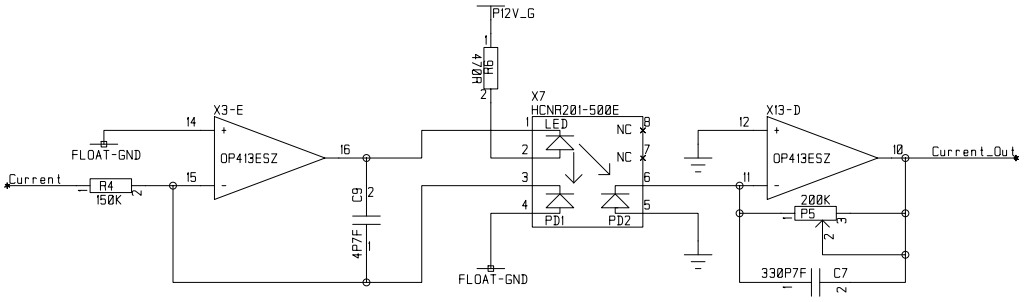


Figure 5.5: Isolation Amplifier

5.4.3 Howland Current Source

For exosomatic measurements a constant AC current is needed. This current is generated by a VCCS, which receive a 2.5V AC signal from the Data Acquisition Card.

As discussed in Section 3.1 the output impedance of the Howland circuit variations are limited by feedback resistor matching. This problem can be solved by using external precision resistors or by the use of potentiometers to calibrate the circuit. This will however add to the complexity of the design and calibration. To avoid these problems, a Dual op-amp Howland circuit constructed around a difference amplifier with internal precision resistors was chosen. This circuit is based on a design given in BURR-BROWN (1990). As one can see from Figure 5.6, it consists of the difference amplifier AD8276 in combination with the op-amp AD8512 and one resistor. In this configuration the error represented by the feedback resistors are significantly reduced. Unlike current sources made with a series pass element, this current source has a bipolar output.

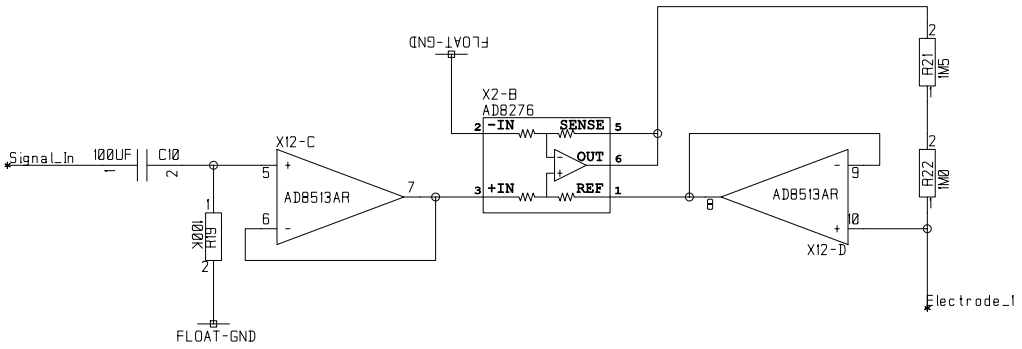


Figure 5.6: Dual Op-amp Howland Current Source

It can therefore both sink and source current. The circuit shown in Figure 5.6 has differential inputs. In order to attain a non-inverting transfer function the -IN input is grounded. The external op-amp AD8512 is used to drive the feedback resistor in the difference amplifier. This op-amp is a dual-supply precision JFET type, and it was selected for its low input bias current, noise and drift characteristics.

In addition to the components already mentioned, a passive high-pass filter and a voltage buffer is added in front of AD8276. The high-pass filter is used to shift the level of the signal from the isolation amplifier, and the voltage buffer is used to prevent unwanted loading on the input of AD8276. The cut-off frequency f_c of the high-pass filter is given by:

$$f_c = \frac{1}{2\pi RC} = \frac{1}{2 \cdot \pi \cdot 100\ k\Omega \cdot 100\ \mu F} \approx 0,016\ Hz \tag{5.2}$$

The main component of the Howland current source is the AD8276 precision dual supply JFET difference amplifier. This component was primarily selected for its laser-trimmed on-chip resistors that provide advantages like high output impedance, space-saving and improved gain accuracy and temperature drift. The AD8276 has a CMRR of 86 dB and it is able to sink and source a current of $\pm$ 70 mA.

The output current I_L of the current source is given as: (BURR-BROWN 1990)

$$I_L = \frac{V_{in}}{R} \tag{5.3}$$

where V_{in} is the input voltage and R is the external resistor.

According to Yamamoto et al. (1981) the limit for linear measurements on skin are limited to a current density of 20 $\mu A/cm^2$ for 25 Hz. This means that for the Kendall 1050NPSM Neonatal Electrodes with a diameter of 25 mm, the current can not exceed 92 μA. In addition to this, the selected current will also influence the magnitude of the measured

impedance and thus the signal-to-noise ratio. In order to balance the SNR against the range needed for the ADC, the current was set to 1 μ A. This gives the following value for the resistor R:

$$R = \frac{V_{in}}{I_L} = \frac{2.5V}{1\ \mu A} = 2.5\ M\Omega \tag{5.4}$$

Since 2.5 MΩ do not exist as a standard resistor size, one resistors of 1.5MΩ was connected in series with a 1MΩ resistor as shown in Figure 5.6.

According to BURR-BROWN (1990) an approximation of the output impedance Z_0 can be found by the following formula:

$$Z_0 = R_X \cdot 10^{\left(\frac{CMRR}{20}\right)} \tag{5.5}$$

where R_X is the external resistor and CMRR is the difference amplifiers common mode rejection ratio in dB. With a CMRR of 86 dB and an external resistor of 2.5 $M\Omega$ the output impedance of the current source is given by:

$$Z_0 = R_X \cdot 10^{\left(\frac{CMRR}{20}\right)} = 2.5\ M\Omega \cdot 10^{\left(\frac{86\ dB}{20}\right)} = 49.8\ G\Omega \tag{5.6}$$

5.4.4 Voltage Reference

The input of the isolation amplifier has a range of 0-5 V. To account for this the precision voltage reference ADR445 is used to bias the in-amp and the transimpedance amplifier. In order to allow for adjustments, the variable voltage source design shown in Figure 5.7 was selected. In this circuit the two 10 $k\Omega$ potentiometers P6 and P7 are used to set the desired voltage. The buffering amplifier provides impedance matching and current drive. The potentiometer P6 connected between V_{OUT} and GND, with its wiper connected to the op-amp, is used for coarse adjustments. The potentiometer with its wiper connected to trim pin of ADR445 is used for fine adjustments.

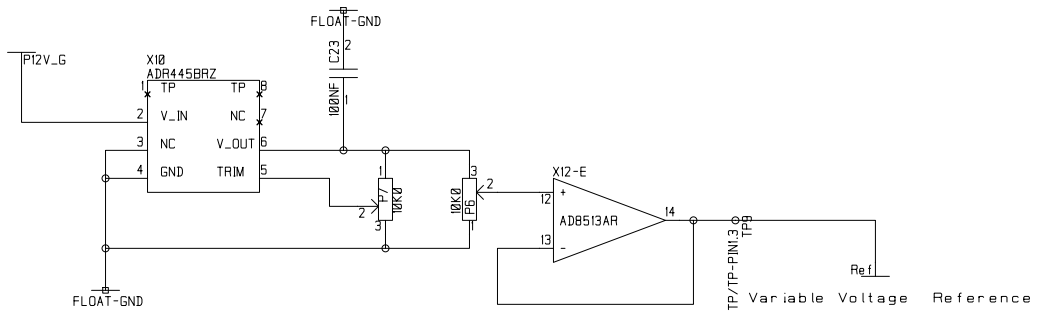


Figure 5.7: Variable precision voltage reference

5.4.5 Preamplifier Circuit

To conduct the endosomatic and exosomatic measurements explained in Section 2.5 an instrumentation amplifier is needed. The instrumentation amplifier selected is the FET-input INA 111 from BURR-BROWN. This in-amp has a high CMMR of 106 dB, a low input current of 20 pA and low offset voltage and drift characteristics. According to BURR-BROWN (1998) the gain of INA111 is set by the following formula:

$$G = 1 + \frac{50\ k\Omega}{R_G} \tag{5.7}$$

where R_G is an external resistor.

In order to determine the appropriate gain for the skin impedance measurement, the expected range must be determined. This was done by Johnsen (2009) as part of his Master thesis, and he found that for a frequency of 24Hz the skin impedance was approximately 30 $k\Omega$. However, this value is dependent on the skin and activity of the subject. It can vary from 10 $k\Omega$ during high activity, to several hundred $k\Omega$ for dry skin during low activity.

Another important critera for determining the gain is the range of the endosomatic voltage measurement. According to Boucsein (2012) endosomatic voltage measurements usually range from -0.1 to -20 mV, but for some subjects, recordings can become as low as -50 to -70 mV.

The range of the ADC used for the Data Acquisition Card is set from 0 to 10 volts. With an offset of 5 V supplied by the voltage reference, this means that the sum of the endosomatic and exosomatic signals can not exceed ± 5 V. If one assumes an impedance of 200 $k\Omega$, a skin potential of 70 mV and a current of 1 μA, the maximum combined signal is equal to 130 mV. This gives a maximum gain of $\frac{5V}{130\ \mu V} = 38.46$.

From Equation 5.7 this gives the following value for the external resistor R_X:

$$R = \frac{50\ k\Omega}{G - 1} = \frac{50\ k\Omega}{38.46 - 1} \approx 1.33\ k\Omega \tag{5.8}$$

As one can see from Figure 5.8 two resistors of 1 kΩ and 330 Ω was used to obtained the combined value of 1.330 kΩ. This gives the in-amp a gain of 38.6.

As one can see form Figure 5.8 a transimpedance amplifier has been added to the three electrode system. This modification is done so that the current applied to the skin can be monitored. The op-amp selected for this is the OPA129 from BURR BROWN. This amplifier has an ultra low bias current of 100 fA combined with low drift and noise characteristics. This makes it suitable for low current measurements.The 4.7; pF capacitor C5 is added to stabilize the op amp.

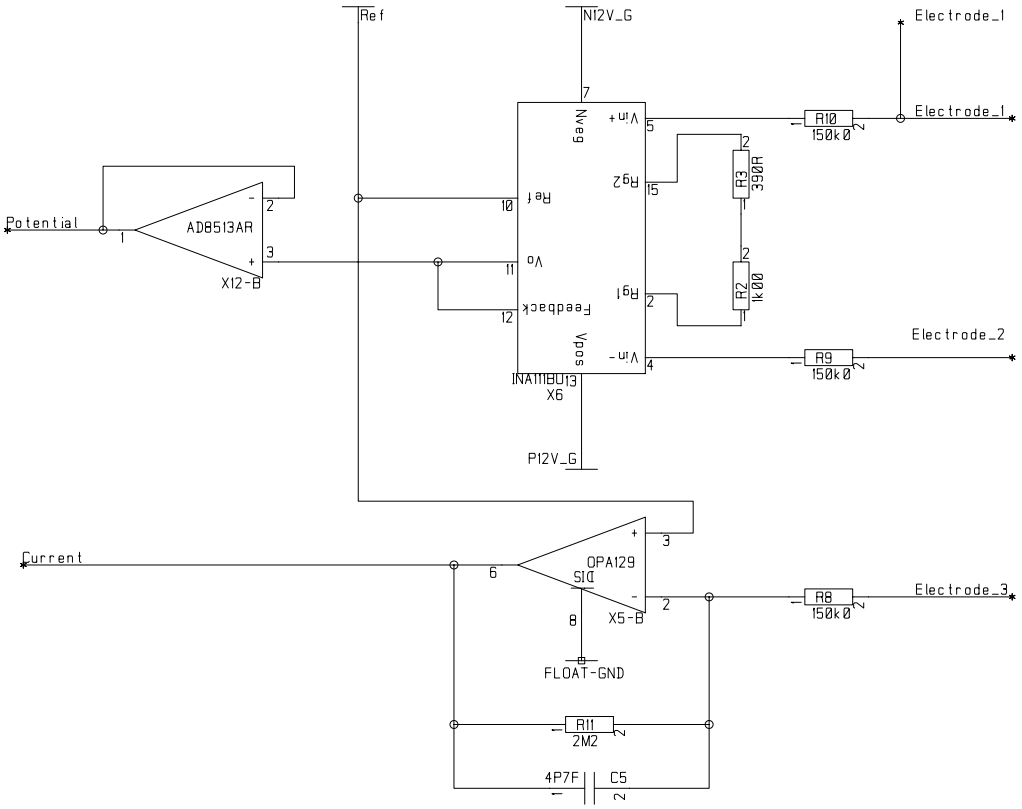


Figure 5.8: Preamplifier

5.5 The Data Acquisition Card

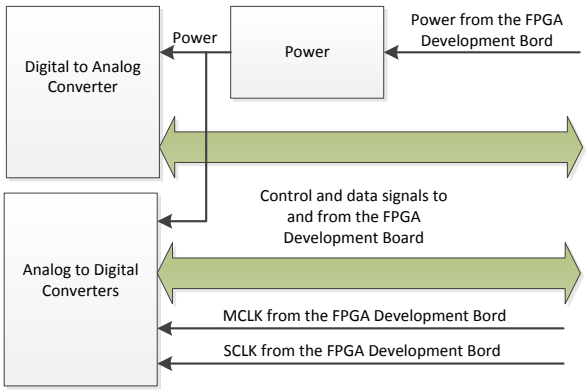


Figure 5.9: Overview of the Data Acquisition Card

The purpose of the Data Acquisition Card is to generate the sinusoid needed for the exosomatic measurements and sample the analog signals for the ECG and the EDA parts of the measurement system. The Data Acquisition Card is implemented as an extension card to the FPGA Development Board. The FPGA Development Board uses a High-Speed Mezzanine connector that is based on the Samtec 0.5 mm pitch, surface-mount QTH/QSH family of connectors. To simplify the interface the THDB-HTG board is used. This board serves as a converter for the High-Speed Mezzanine interface, and converts it to three 40-pin expansion prototype connectors. In addition to this the converter board also supplies regulated 3.3 V, 5 V and 12 V. For the Data Acquisition Card connector J2 and J3 are used. An overview of the Data Acquisition Card is shown in Figure 5.9, and it consists of the following main modules: one digital to analog converter, four analog to digital converter and a power module. The Data Acquisition Card was designed and produced in collaboration with Miriam Kirstine Huseby.

5.5.1 Analog-to-Digital Converters

For Analog-to-Digital Conversion a high performance, 24-bit over-sampling successive approximation (SAR) ADC (AD7766-2) was used. This converter was selected for its ac and dc accuracy, low power consumption and its on-chip linear-phase lowpass FIR filter. A wide dynamic range in combination with ac and dc accuracy makes the AD7766-2 suited to measure small variations in skin impedance over a wide dynamic range.

The filtered output data is in 24-bit serial format, two's compliment, with the MSB being clocked out first. The sample rate is set by the MCLK. By using oversampling, the quantization noise of the converter is spread across the bandwidth of 0 to f_{mclk} . Digital filtering following the converter output acts to remove the out-of-band quantization noise. The frequency response of the digital filter is shown in figure 5.10. Because of the digital filtering, the AD7766-2 has an output decimation rate of 32. With a maximum MCLK input frequency of 1024 MHz, the maximum output data rate is 32 kHz. The analog inputs of the AD7766-2 are differential. Which provide rejections of signals common to both V_{in+} and V_{in-}.

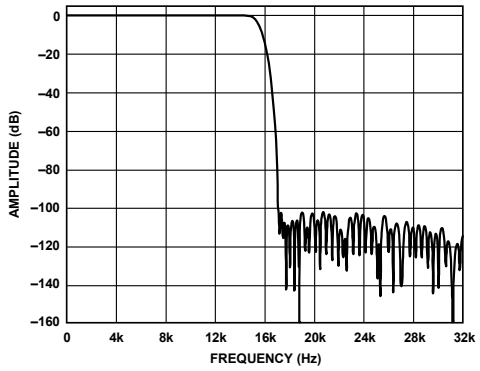


Figure 5.10: Digital Filter Frequency Response for AD7766-2.

Single-ended-to-differential Driver

In order to drive the differential inputs of AD7766-2, the single-ended-to-differential driver ADA4941-1 was selected. The ADA4941-1 is a low power, low noise differential driver for ADCs. The choice of the driver was based on an example circuit from the AD7766-2 datasheet, as shown in Figure 5.11.

Voltage Reference

As a nominal reference voltage, either 5 V or 2.5 V supply can be used. A 5 V reference input was chosen, as this gives the full-scale differential input range from 0 to 10 V. To ensure a stable 5 V reference , the ADR445 XFET voltage reference was used. It was chosen because of its ultralow noise, high accuracy, and low temperature drift. As a nominal reference voltage, either 5 V or 2.5 V supply can be used. A 5V reference input was chosen, as this enables a full-scale differential input range of 10 V.

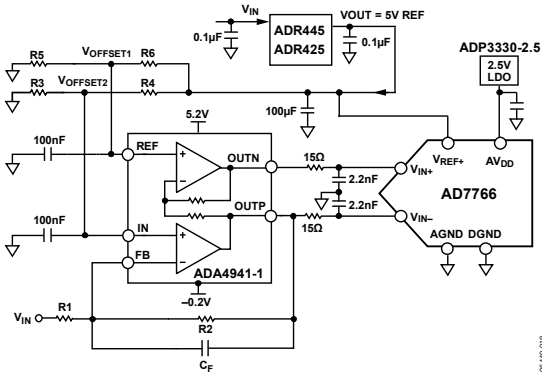


Figure 5.11: Schematic for driving the AD7766-2 from a single-ended source.

Recalculating the Input Range

To get an input range from 0 V to 10 V and an antialiasing cutoff frequency of 106 Hz the input driver needs to be recalculated. The AD7766-2 datasheet gives the following explanation of the different components:

R1 and R2 are used to set the attenuation ratio between the input range and the ADC range (V_{ref}). R1, R2 and C_F set the input resistance, bandwidth and antialiasing. 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.

The selected component values are shown in Table 5.1, and the LTspice simulation used to verify the values are located in Appendix E.

Table 5.1: Selected component values for the ADC driver ADA4941-1.

Component Name	*Value*
C_F	330 pF
$R1$	2 $k\Omega$
$R2$	1 $k\Omega$
$R3$	20 $k\Omega$
$R4$	10 $k\Omega$
$R5$	10 $k\Omega$
$R6$	10 $k\Omega$

5.5.2 Digital to Analog Converter

For Digital-to-Analog Conversion, the 12 bits single voltage-output AD5340 by Analog Devices is used. The AD5340 was selected for its low power operation and its parallel data interface. 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:

$$V_{out} = V_{ref} \times \frac{D}{2^N} \times Gain \tag{5.9}$$

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.

With a gain of 1 the maximum output-voltage is given as:

$$V_{out} = V_{ref} \times \frac{D}{2^N} \times Gain = 5\ V \times \frac{4095}{2^12} \times\ 1 = 4.99V \tag{5.10}$$

5.5.3 Power Module

Power for the Data Acquisition Card is supplied by the 12V, 5V, and 3.3V pins on the Terasic THDB-HTG connector connected to the FPGA Development Bord. In addition to the voltages supplied by the Terasic THDB-HTG connector, two ADP3330 regulators are used to supply 2.5 V to the AV_{DD} and DV_{DD} pins of the ADCs.

5.6 FPGA Development Board

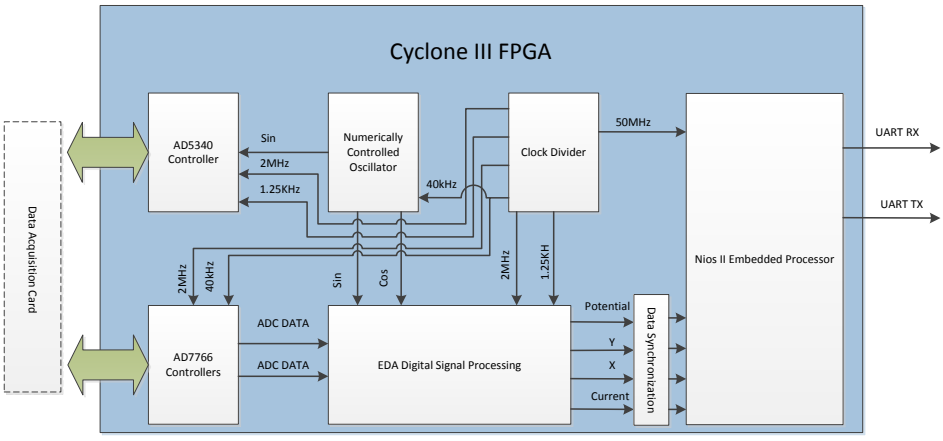


Figure 5.12: Overview of the digital design.

The purpose of the FPGA development board is to control the external modules and perform signal processing. Since the size and complexity of the final system is not completely known in advance. The development board had to be selected so that the entire prototype could

be implemented. In addition to this, the development board had to be equipped with an interface that allowed the use of external modules.

To satisfy these requirements, the Cyclone III DSP Development Board from Altera was selected. This board provides a hardware platform that is intended for low-power, high-volume and feature-rich design. The FPGA used on the development board is the low-cost series Cyclone III family FPGA EP3C120F780 with 119K logic elements, four phase locked loops and 288 embedded 18-bit x 18-bit multipliers. The Cyclone III device family is built on 65-nm low-power process technology, and they are designed to provide low static and dynamic power consumption. In addition to the FPGA, the development board is also expendable through two Altera High-Speed Mezzanine Connectors (HSMC) that each have 86 I/Os and can deliver up to 19.8 W per interface.

An overview of the digital design is shown in Figure 5.12. As one can see from the diagram the design consists of the following modules: AD5340 Controller, AD7766 Controllers, Numerically Controlled Oscillator, EDA Digital Signal Processing, Clock Dividers and Nios II Embedded Processor. The arrows in the diagram represent the main clock and control signals. The digital logic implemented on the FPGA is done by a combination of ready-made Altera Megafunctions and custom modules written in VHDL. The VHDL code and functions are found in Appendix F.

5.6.1 Clock Divider

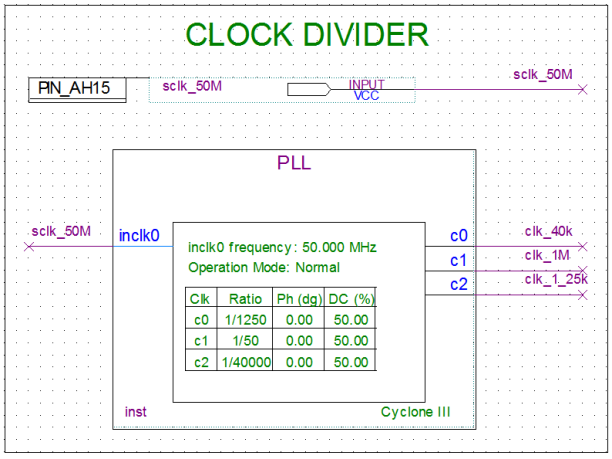


Figure 5.13: The Clock Divider.

The FPGA Development board is supplied with two crystal oscillators that serve as reference clocks for the Cyclone III FPGA. The oscillator frequencies are 125 MHz and 50 MHz. The dynamic power consumption

of a FPGA is proportional to the switching frequency that drives its logic gates (Rabaey 2009). The 50 MHz clock was therefore selected as the system master clock.

As shown in Figure 5.12 the digital modules have different frequency requirements. In order to meet these requirements, a Phase-Locked Loop (PLL) Megafuntcion was used. The PLL use one of the four integrated PLL blocks available in the Cyclone III FPGA to generate stable frequencies that are distributed throughout the design. The generated frequencies are given in Table 5.2. The principles used to chose the specific frequencies will be covered in later sections.

Table 5.2: PLL Frequencies

Signal Name	*Frequency*
clk_1M	2 MHz
clk_40k	40 kHz
clk_1_25k	1.25 kHz

5.6.2 Numerically Controlled Oscillator

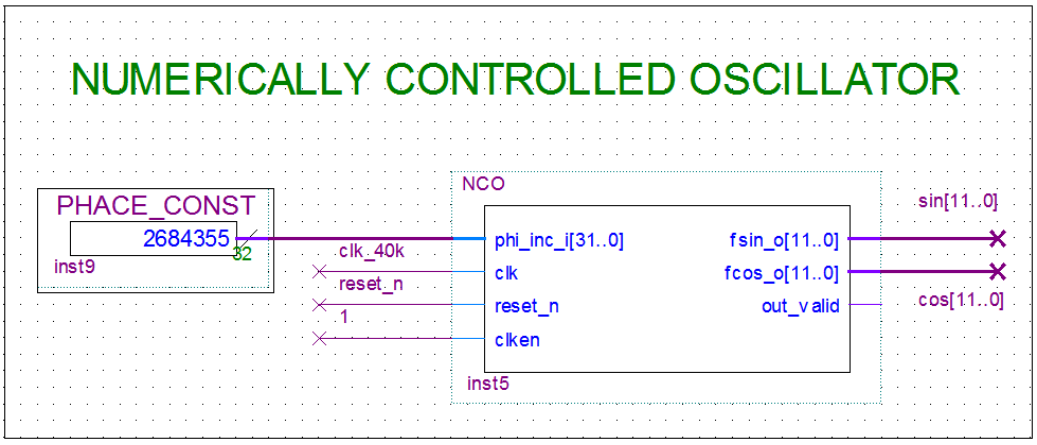


Figure 5.14: The Numerically Controlled Oscillator.

To perform the algorithm described in Section 3.5 and generate the sinusoid needed for the exosomatic measurement, a Numerically Controlled Oscillator (NCO) is used. The NCO is implemented as a MegaCore function, and its outputs are the discrete-time and valued in-phase and quadrature signals fsin() and fcos(). As shown in Figure 5.14 the NCO is driven by a 40 kHz clock signal. The 31-bit phase constant phi_inc_i is used to generate an output frequency of 25 Hz. The

Angular resolution and magnitude precision of the output waveforms are set to 16-bit and 12-bit respectively. The NCO supports small ROM, large ROM, CORDIC and Multiplier based algorithms. In order to obtain the highest possible SNR, the different algorithms were compared in the design software, and a large ROM based algorithm was selected. This gives a SNR of 89 dB. According to Electrodermal Measurespsyp et al. (2012) a frequency between 20 and 30 Hz is recommended to reduce the capacitive current without giving time for electrolysis during each half cycle. Because of this and the type of filter used by the DSP algorithm, a frequency of 25Hz was selected.

5.6.3 AD5340 Controller

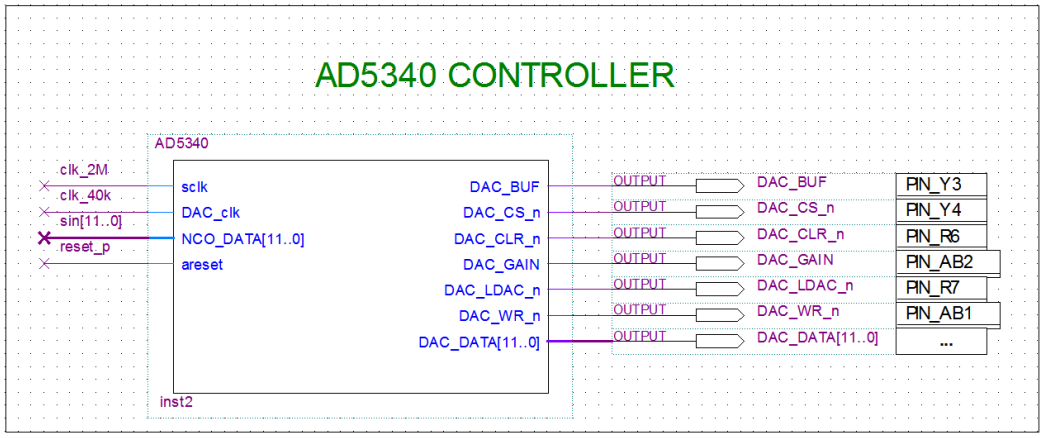


Figure 5.15: The AD5340 Controller.

The AD5340 module shown in Figure 5.15 is used to control the external digital-to-analog converter located on the Data Acquisition Card. The data input signal NCO_Data is connected to the sine wave signal produced by the NCO. The AD5340 is used to supply a continuous analog signal, and it is therefore configured in synchronous mode. This enables the DAC registers to be updated by the rising edge of WR_n as shown in Table 5.3. Before the data word is shifted out to the DAC it is converted to offset binary.

5.6.4 AD7766 Controllers

The Data Acquisition Card is equipped with four AD7766 analog-to-digital converters. These ADCs are controlled by the AD7766_A and AD7766_B modules. The AD7766_A module controls one ADC that is dedicated to the ECG part of the system. The remaining ADCs are controlled by AD7766_B, and these ADCs are dedicated to the EDA part

Table 5.3: AD5340 Truth Table

$\overline{CLR}$	$\overline{LDAC}$	$\overline{CS}$	$\overline{WR}$	*Function*
1	1	1	X	No data transfer
1	1	X	1	No data transfer
0	X	X	X	Clear all registers
1	1	0	$0 \rightarrow 1$	Load input register
1	0	0	$0 \rightarrow 1$	Load input register and DAC registers
1	0	X	X	Update DAC register

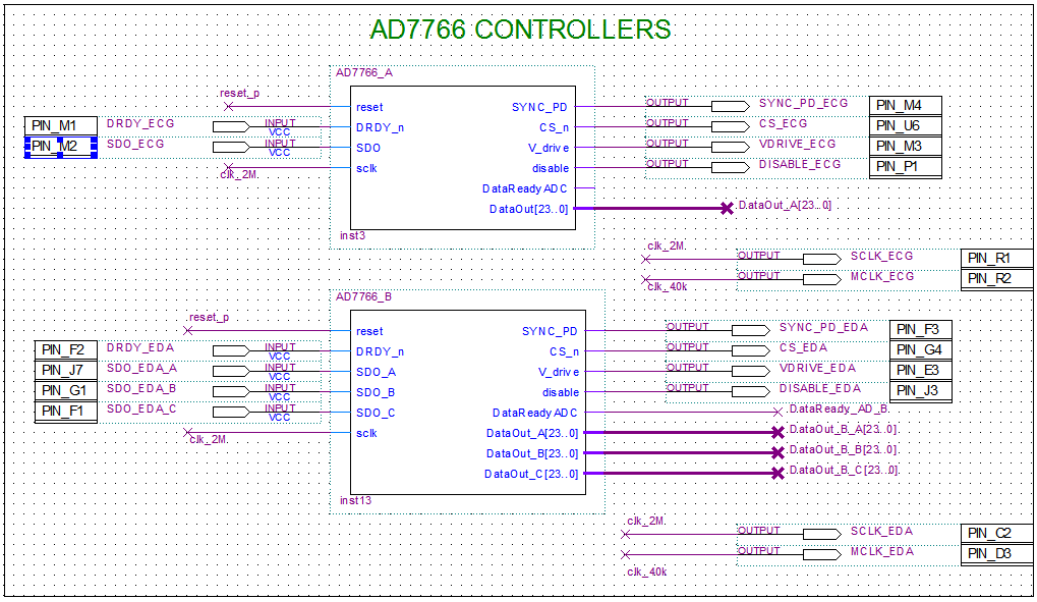


Figure 5.16: The AD7766 Controllers.

of the system. The ADC data rate is set by the clock signals MCLK_EDA and MCLK_ECG. As explained in Section 5.5 the AD766 ADC use a digital filter and oversampling to reduce the total signal noise. This results in a decimation ratio of 32. If the ADC is run at a sampling rate f_s of 40 kHz the data rate $f_d = \frac{40\,kHz}{31} = 1.25\,kHz$. The sampling rate was selected in order to achieve a data rate that gives an integer number of samples per period for the 25 Hz signal. Since the data rate used by the different EDA DSP modules is the same, the ADCs are run in parallel.

The AD7766 Controller is implemented according to the finite state machine (FSM) shown in Figure 5.18. The FSM diagram is designed according to the AD7766 timing diagram shown in 5.17.

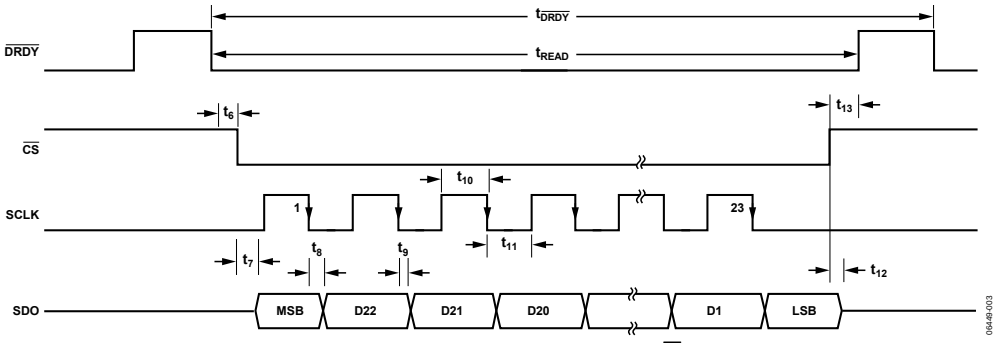


Figure 5.17: Serial timing diagram, reading data using $\bar{CS}$.

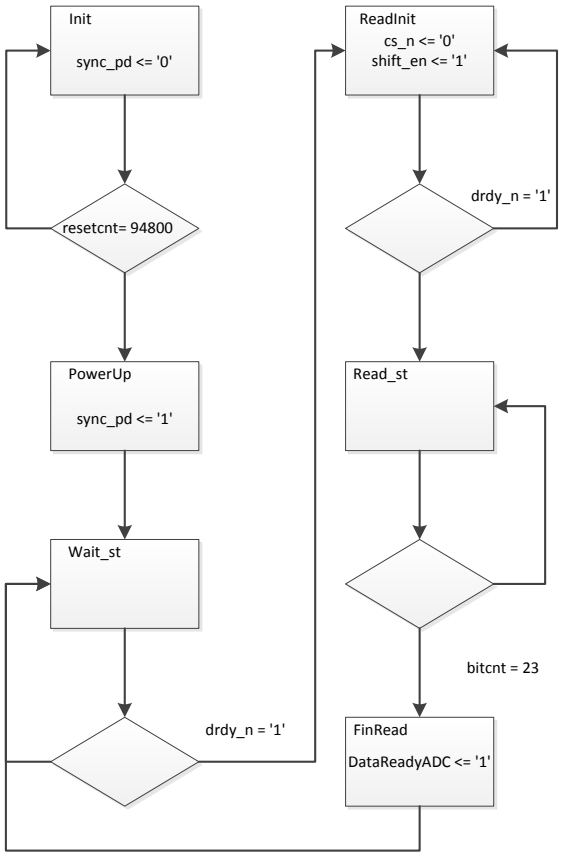


Figure 5.18: Diagram of the FSM used to control the AD7766.

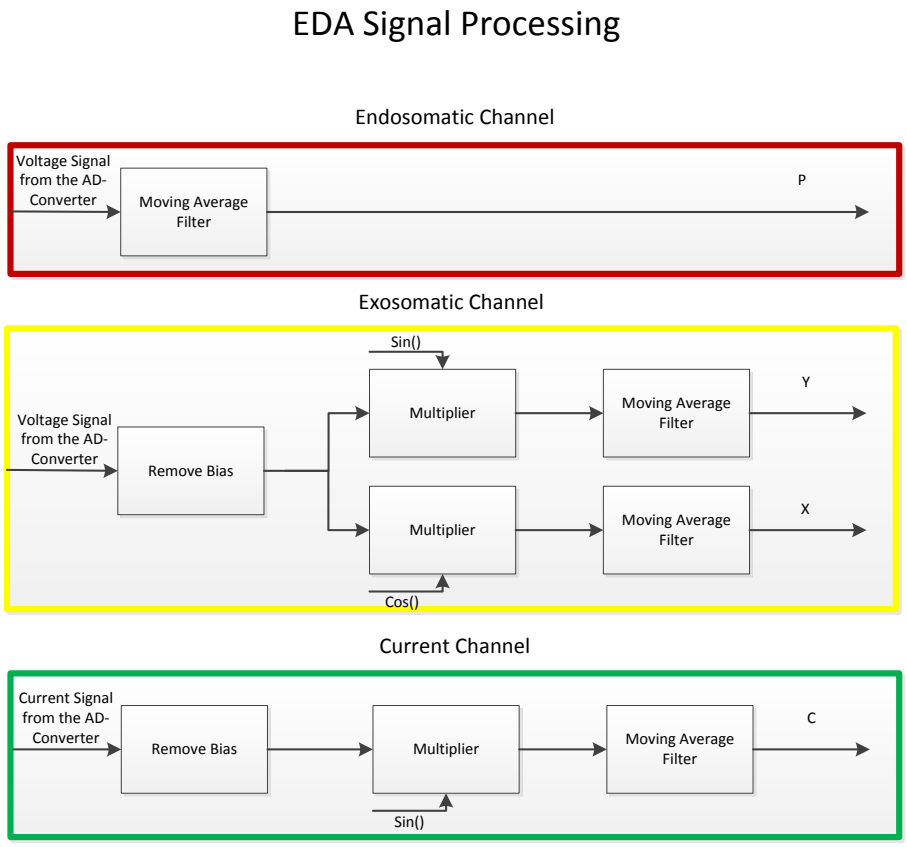


Figure 5.19: The digital signal processing used to separate the endosomatic (red), exosomatic (yellow) and the current (green) measurements from the 25 Hz carrier frequency.

5.6.5 Digital Signal Processing

Figure 5.19 shows the digital signal processing used to separate the endosomatic (red), exosomatic (yellow) and the current (green) measurements from the 25 Hz carrier frequency. The endosomatic channel use a moving average digital low-pass filter to extract the skin potential P measured bye the instrumentation amplifier. The exosomatic channel use the Lock-In Detection technique explained in Section 3.5 to extract the real Y and imaginary part X of the impedance. The current channel uses the same type of Lock-In Detection scheme as the real section of the exosomatic channel. In order for the Lock-In Detection algorithm to work the offset is removed from the voltage and current input signals.

The Multiplier Module

The Multiplier module is implemented using the standard VHDL library package *ieee.numeric_std*. This library provides arithmetic vector functions for the numeric types SIGNED and UNSIGNED. The modules takes the two input signals $DATA\_IN\_A$ and $DATA\_IN\_B$ and returns the product as the signal $DATA\_OUT$.

The Moving Average Filter Module

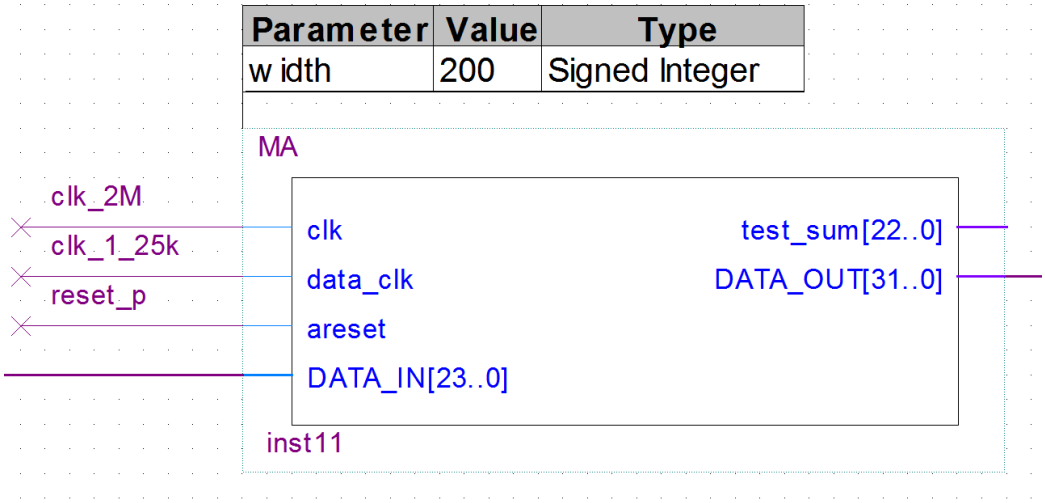


Figure 5.20: The Moving Average Filter Module.

The module in Figure 5.20 performs a moving average filtering on the input data signal $DATA\_IN$. When a new data item is received, the output signal $DATA\_OUT$ is the average of the previous 200 samples. The filter can be implemented as a general finite impulse response (FIR) filter. In a general FIR the samples are multiplied with a coefficient, and the results

are summed for each new input value. Since a moving average filter is a special case of the FIR filter with all its coefficients equal to $\frac{1}{numberofsamples}$ the number of computations can be reduced.

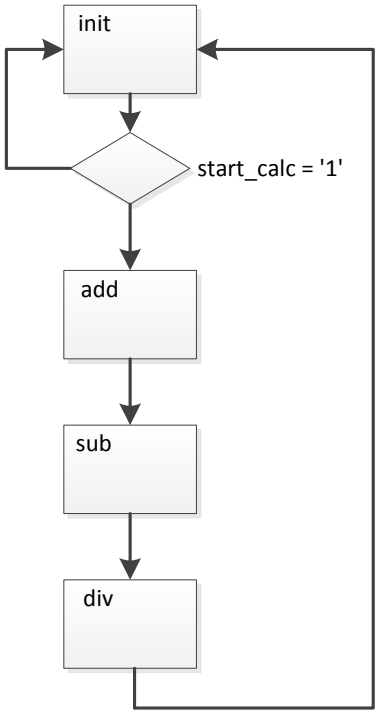


Figure 5.21: Diagram of the FSM used to implement the moving average filter.

To reduce the number of computations the filter is implemented according to the state diagram shown in Figure 5.21. The FSM use a shift registers to hold the 200 values, and keep a running sum that is updated for each new value. In this way the calculations are reduced to one addition, one subtraction and one division for each new sample.

To find the correct length of the shift register , the filter was analyzed in MATLAB. Figure 5.22 shows the frequency and phase response of the moving average filter with an order of 200 and a data rate f_s of 1.25 kHz. From the frequency response one can see that the carrier frequency of 25 Hz and its harmonics are significantly dampened. This has the added benefit of also reducing the 50 Hz and 100 Hz external noise. To prevent the filtering of fast changes in the signal, the step response is set to 200 mS.

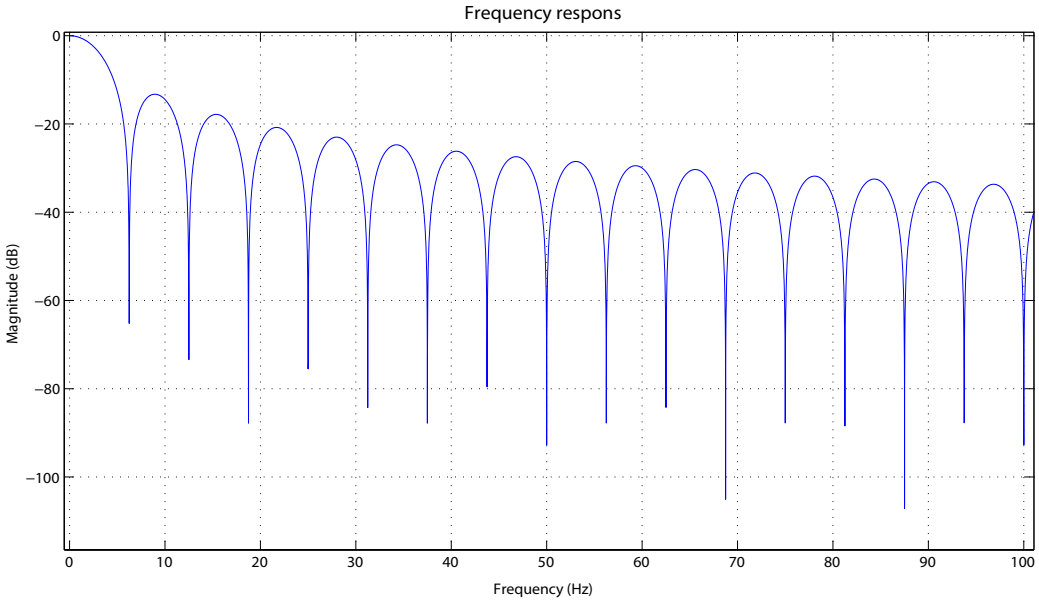


Figure 5.22: The frequency response of a moving average filter with an order of 200 and a f_s of 1.25 kHz (the frequency axis is scaled in order to make it more readable).

The Remove Bias Module

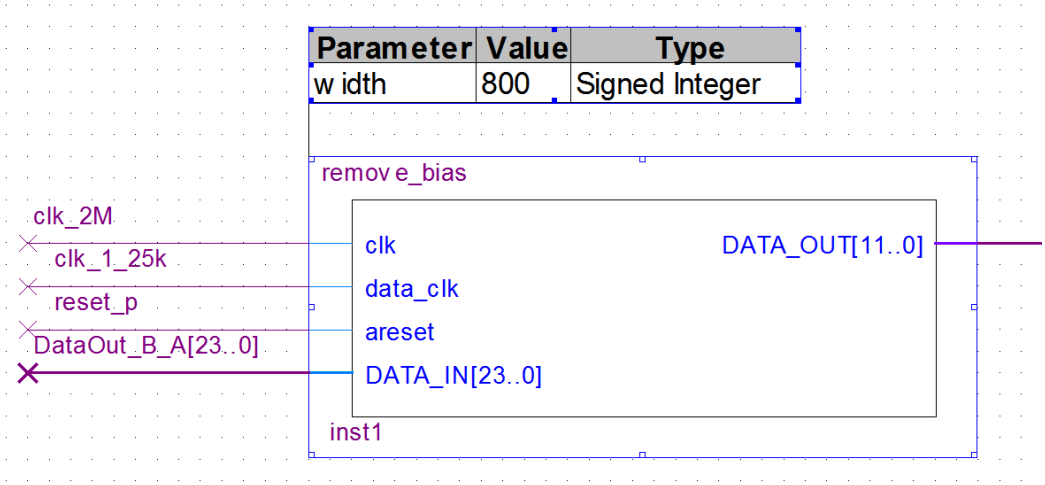


Figure 5.23: The Remove Bias Module

In order for the Lock-In Detection technique to work, the endosomatic bias needs to be removed from the input signals. To achieve this, the remove bias module shown in Figure 5.23 is used. This module is constructed according to the FSM diagram shown in Figure 5.24. As one can see from the FSM diagram, this module is almost identical to the

moving average filter.

The difference lies in the additional state named *sub_mean*. In this state, the mean of the previous samples are subtracted from the input signal resulting in a bias free output signal. In order to make the mean value less dependent on changes in the bias, the shift register is set to a length of 800 samples.

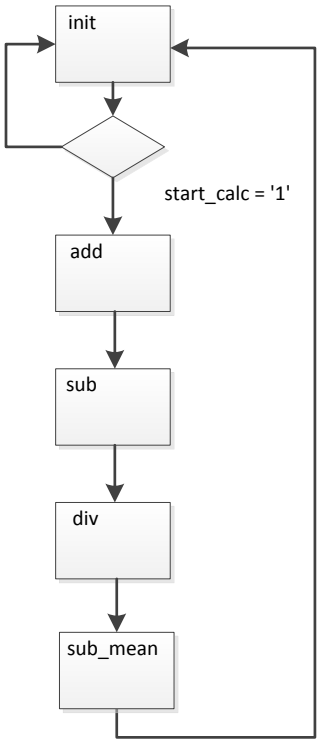


Figure 5.24: Diagram of the FSM used to implement the remove bias module.

The Data Enable Module

The Data enable module shown in Figure 5.25 is used to generate the signal data_enable. This signal is used to enable synchronous triggering of the input data for the Nios II processor. The data_enable is also used to enable the data register module that clocks data in to the PIO ports of the Nios II processor.

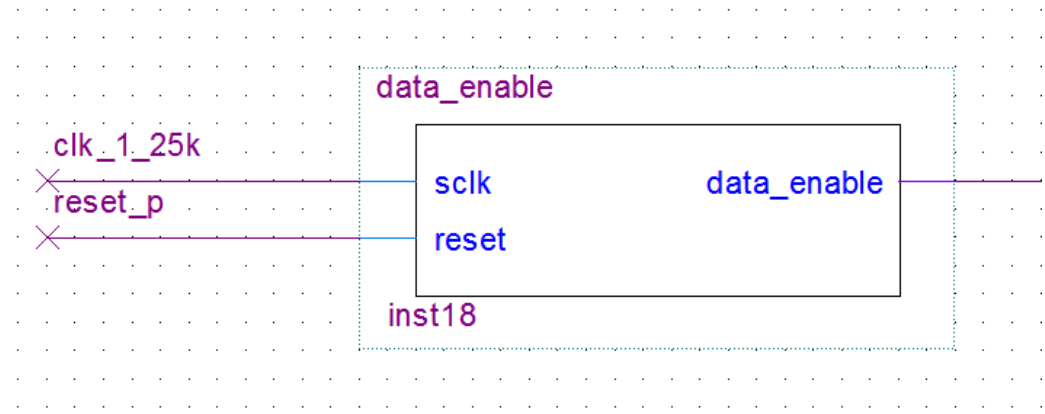


Figure 5.25: The Data Enable Module.

The Data Register Module

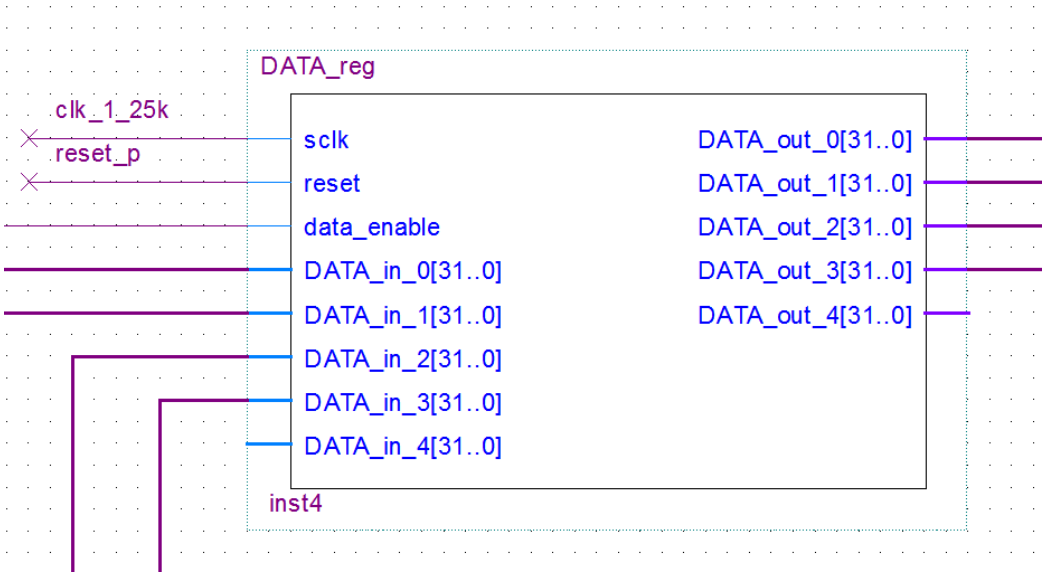


Figure 5.26: The Dataregister Module

The Data Register module shown in Figure 5.26 is a data register that is used to ensure correct timing during a read operation on the Nios II processor. The module use the input signal data_enable to update the PIO input ports of the processor.

5.6.6 Nios II

In order to send data from the FPGA an interface to the external Bluetooth module is needed. This can be done with a custom UART driver written in

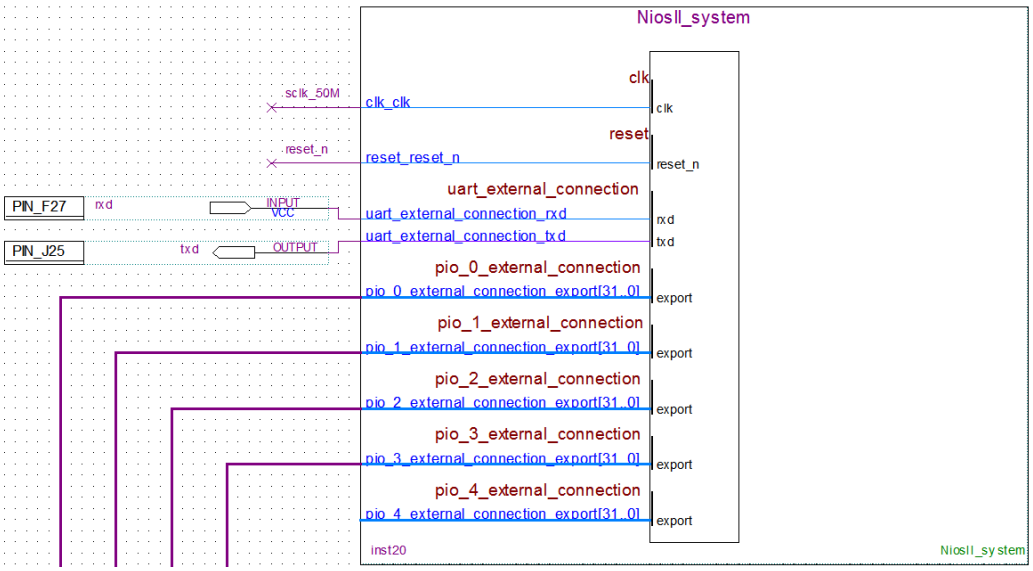


Figure 5.27: The Nios II processor.

VHDL. A more flexible solution is to use a Nios II processor with a UART peripheral. This solution enables the use of C code to control data transfer and formatting. The Nios II processor used in this system is shown in Figure 5.27. The peripherals used are: programmed input/output (PIO) modules for data transfer to the processor, and a UART module to control the external Bluetooth module.

The C used by the Nios II processor is located in Appendix G

The Python development Tool

In order to simplify the prototyping and development of the digital system, a Python PC application was developed. The main advantage with this tool is that the Python programming language enables changes to be implemented more easily then the Android application written in Java. In its present form the application has the following functionality:

- Display multiple data signals in the two plot views.
- Write the incoming data to a text file.
- Monitor the size of the input data queue.
- Adjust the x and y axes of both the plot views.
- Clear the incoming data queue.

A screenshot of the application is shown in Figure 5.28. The source code and its UML class diagram is located in Appendix C.

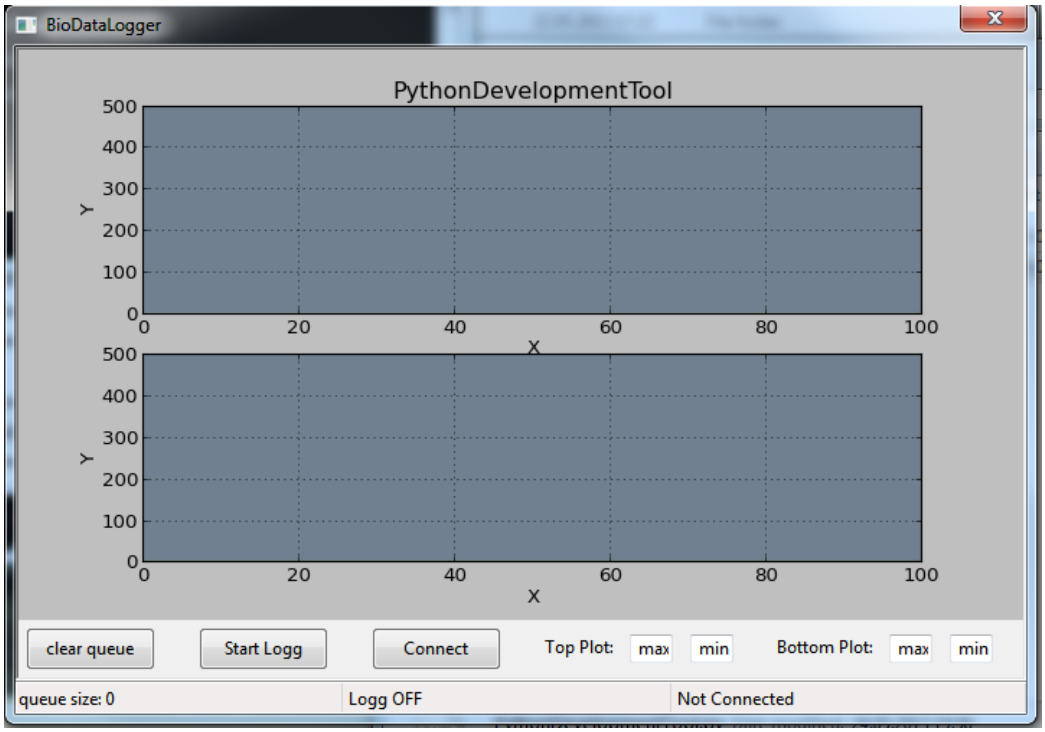


Figure 5.28: Screenshot of the Python development Tool.

5.7 Bluetooth module

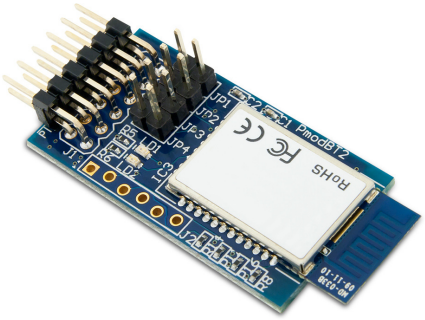


Figure 5.29: The PmodBT2 peripheral module.

In order to transfer data between the FPGA and the software platforms, a communication link is needed. To obtain a flexible interface that supports both the Android OS and computer platforms, the Bluetooth interface module PmodBT2 shown in Figure 5.29 was selected. This module is made by Digilent and based on the RN-42 Bluetooth link from Roving Networks. The PmodBt2 interface has a standard 12-pin

connection and communicates via a UART interface. The RN-42 Bluetooth link supports the Bluetooth standards 2.1/2.0/1.2/1.1. By default, the PmodBT2 is set to a baud rate of 115.2 kbps with a packet format of 8 data bits, no parity bit and 1 stop bit. In order to set the module in auto discovery/pairing the jumper JP2 is set. The selected settings for PmodBt2 are selected in order to simplify the prototyping process. If additional restrictions like added security or custom baud rates are needed, the module can easily be configured by setting it in command mode.

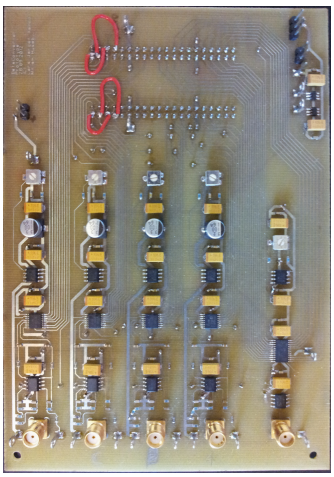
5.8 PCB Design

Two different PCB prototypes have bean developed for this project: The Analog Front-end and the Data Acquisition Card. Both of the PCBs are made from two layer FR4 epoxy substrates. A two layer design was selected in order to produce the PCBs at the Electronic Workshop at the Department of Physics at UiO. The schematic design and PCB layout were done in Zucken CADSTAR. The Data Acquisition Card was designed and produced in collaboration with Miriam Kirstine Huseby.

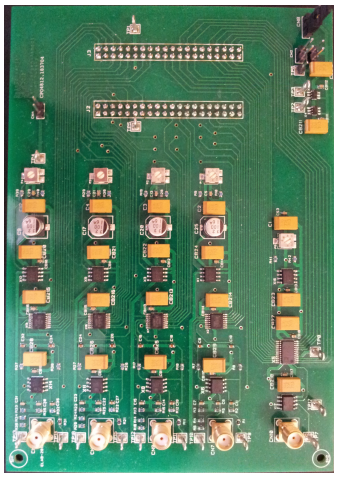
To provide a stable and uniform ground reference throughout the two PCBs, ground planes were used. Fast switching digital circuits serve as noise sources; the sensitive preamplifier on the Analog Front-end is therefore supplied with a separate ground plane. This ground plane is isolated from the rest of the card and the digital system, as discussed in Section 3.4.1.

To ensure minimal power fluctuations and reduce noise, all active components have two bypass capacitors. A 100 nF ceramic capacitor is used to suppress high frequency noise, and a 100 μF tantal capacitor is used to suppress low frequencies noise and serve as a current buffer for the IC. The capacitors are positioned as close to the IC pins as possible. On the Analog Front-end PCB the capacitors are placed on the bottom layer. This was done in order to give space fore the probes used to verify the circuit. On both cards the different power rails are put on the bottom layer.

As one can see from Figure 5.30 and Figure 5.31, two different revisions of the Analog Front-end and the Data Acquisition Card were made. The first revisions were used to debug and test the individual sub circuits. The experience gained from these tests, were then used to make the final revisions. The revision markings follow the different versions of the schematics, and the final versions of the Analog Front-end and the Data Acquisition are therefore marked REV.E and REV.C respectively.

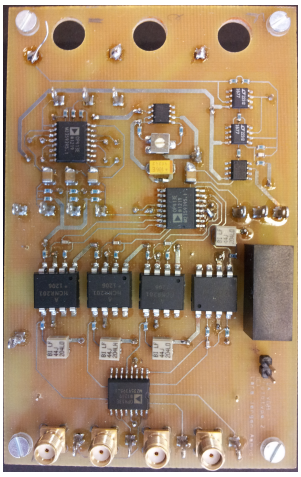


Prototype 1

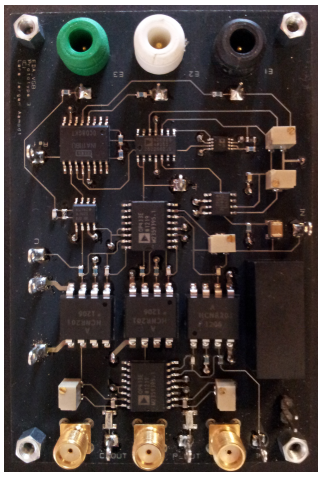


REV.C

Figure 5.30: The two different revisions of the Data Acquisition Card.



Prototype 1



REV.E

Figure 5.31: The two different revisions of the Analog Front-end.

5.9 The Android Application

To display and store the data transmitted by the Bluetooth module, an Android application named BioDataLogger has been developed. The BioDataLogger is developed for Android devices that supports Bluetooth functionality and has a software platform between Android 2.2 and Android 4.1.2. The application is intended to be used as a display and data storage tool. Figure 5.32 shows a screenshot of the BioDataLogger.

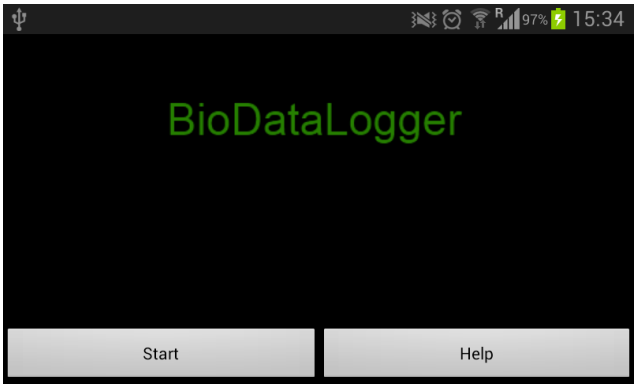


Figure 5.32: The BioDataLogger.

The UML diagram in Figure 5.33 shows the architecture and relations used by the classes of the application. As shown in the UML diagram the application is constructed from different activities and one background service. The activities are used to interact with the user, and they hold the views needed to render information to the screen. The background service contains the data queue and the threads needed to maintain the Bluetooth connection. This producer consumer architecture was chosen in order to preserve the Bluetooth and data storage capabilities in the event of a disturbance from the Android OS. If for instance the phone receives a call, the Android system will stop the activity, but not the background service. The communication between the activities, the service and the threads are managed by message handlers that operate on message queues.

The rest of this section will give a short review of the key components used in the application. The user manual and source code is located in Appendix B.

5.9.1 The Start Activity

The StartActivity serves as the starting point of the BioDataLogger application. When the user pushes the application icon this activity is launched. The activity contains two buttons that point to the MainActivety and the HelpActivity.

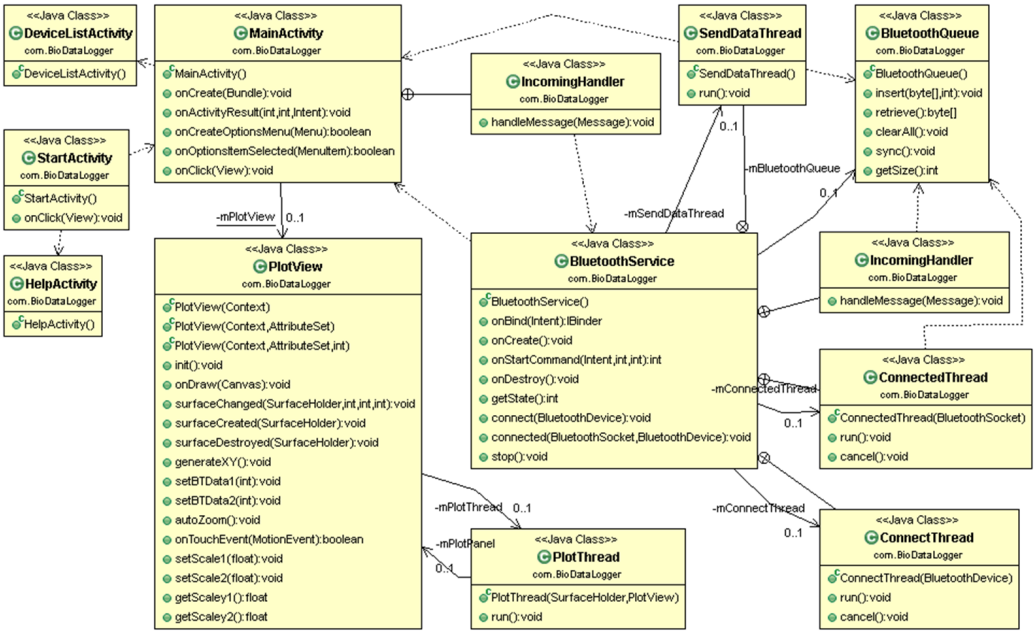


Figure 5.33: UML diagram displaying the architecture of the Android application.

5.9.2 The Help Activity

This activity is used to display information like the intended use of the application and contact information.

5.9.3 Device List Activity

This activity is used to list paired devices and devices that are in range of the unit. When the Bluetooth module is selected from the list, its MAC address is return to the MainActivity.

5.9.4 The Main Activity

This activity serves as the main component of the BioDataLogger and holds the views needed to display information to the screen. The PlotView is designed as a game graphics engine, and extends the integrated SurfaceView class in order to draw graphics to the screen. To make sure that the application remains responsive, the actual rendering is done by a thread named PlotThread. If Bluetooth is disabled when the MainActivity launches, it will ask the user for permission to enable it. This activity also holds the different buttons needed to start the data log, connect/disconnect the Bluetooth link and stop the background service. When the user push the connect button, the DeviceListActivity returns the MAC address of the external device. When the MainActivity

receives the MAC address, it starts the BluetoothService, binds to it, and sends the MAC address. Communication between the MainActivity and the BluetoothService is maintained by the message handler named IncomingHandler. When a data point is sent from the BluetoothService, it is decoded by the MainActivity and sent to the PlotView.

5.9.5 The Bluetooth Service

This component is a service that runs in its own separate process. This means that the Bluetooth connection and data log will remain operational regardless of what happens to the MainActivity. When the BluetoothService starts, it creates a notification that is displayed on the phones notification drawer. This enables the user to restart the MainActivety, and rebind to the Service. In order to initiate and maintain a Bluetooth connection, the three threads ConnectThread, ConnectedThread and SendDataThread are used. The ConnectThread establish an outgoing connection with the external device. This thread runs straight through, and it either succeeds or fails. The ConnectedThread runs during a connection with the external device, and handles all incoming transmissions. When data is received, the thread writes the data to a text file located on the SD card and appends it to the BluetoothQueue. Since the socket used by the ConnectedThread uses blocking calls, a separate thread named SendDataThread is used to transmit the data from the BluetoothQueue to the MainActivity. In order to stop the BluetoothServis an exit button is implemented in the MainActivity. When this button is pushed the service receives a message that activates a destroy routine.

Chapter 6

System Verification and Calibration

This chapter covers verification and analysis of the measurement systems primary modules. Method and results of the system calibration are also included.

6.1 Verification

6.1.1 The Digital-To-Analog Converter

Experimental Setup

A Tektronix TDS 2024B oscilloscope is used to verify the output of the AD5340 DAC. The oscilloscope is used to measure the output signal and to calculate an Fast Fourier Transform (FFT) of the signal.

Results

Presented in Figure 6.1 is the output signal from the DAC. The signal has a peak-to-peak voltage level of 5 V at a frequency of 25 Hz. As expected, the signal is a sine wave.

An FFT of the output signal is presented in Figure 6.2. The signal power P_s is approximately 68 dB and the noise floor P_n is estimated to 10 dB. This gives an SNR of 68 dB - 10 dB = 58 dB.

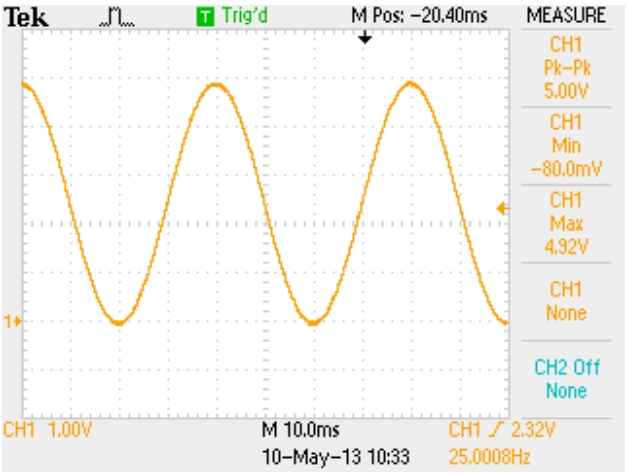


Figure 6.1: The output signal from the DAC.

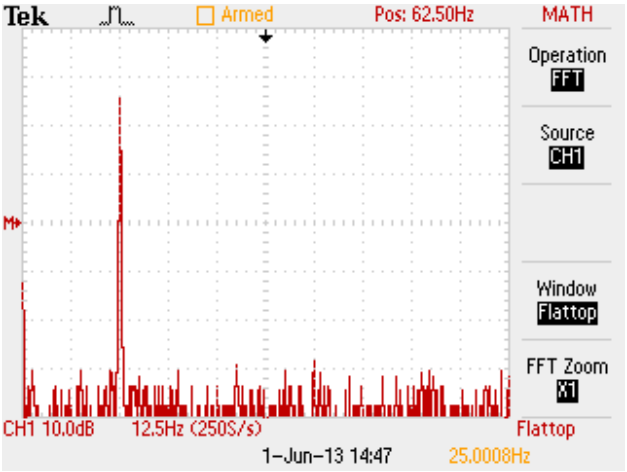


Figure 6.2: An FFT of the output signal from the DAC.

6.1.2 Digital Design Verification

This section contains the timing diagrams used to verify the different modules on the FPGA. The diagrams are recorded in real-time using the SignalTap II Logic Analyzer.

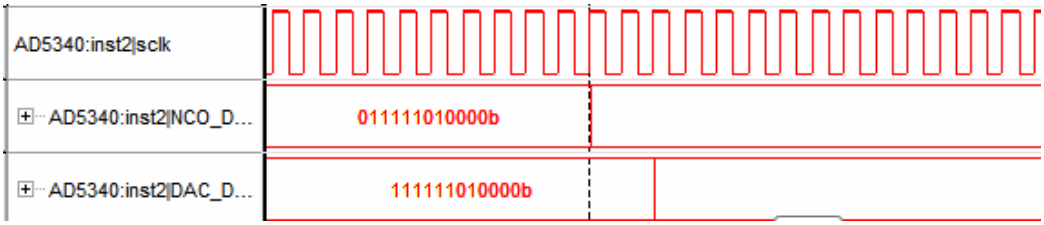


Figure 6.3: Timing diagram of the AD5340 controller. The diagram shows the signal from the NCO being converted to binary offset and shifted out to the DAC (bottom).

Figure 6.4: Timing diagram of the AD7766A controller used for the ECG part of the system. The signal at the bottom of the diagram shows the serial data received from the ADC.

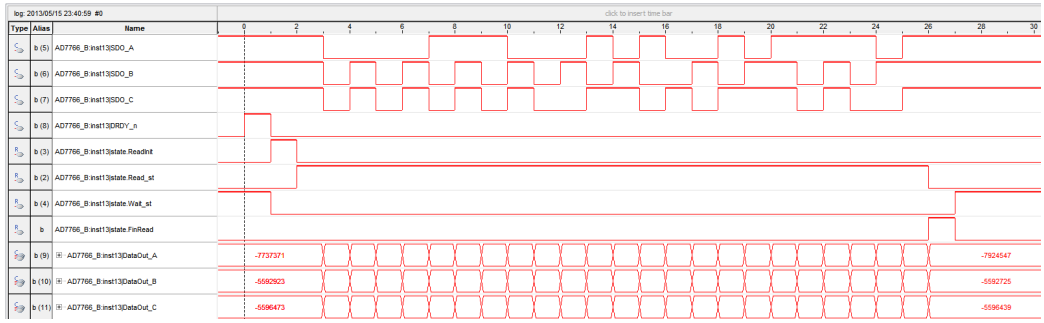


Figure 6.5: Timing diagram used to verify the AD7766B controller used for the EDA part of the system. The three signals at the bottom of the diagram shows the serial data received from the ADCs.

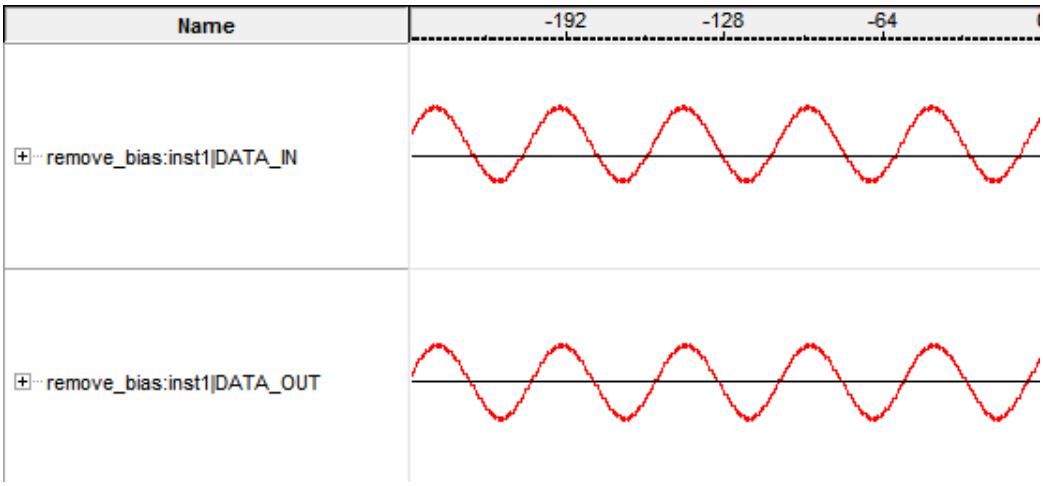


Figure 6.6: Timing diagram displaying the input and output of the Remove Bias Module as a waveform. It is evident that the module removes the offset, as seen in the bottom, output, waveform.

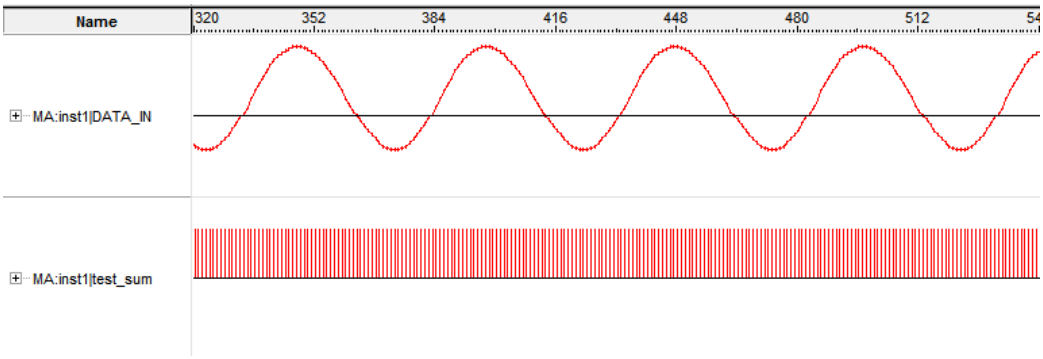


Figure 6.7: Timing diagram displaying the input and output signals of the Moving Average Filter Module. The bit length of the output signal (the bottom line) is displayed as a bar plot and scaled to better display the averaging effect.

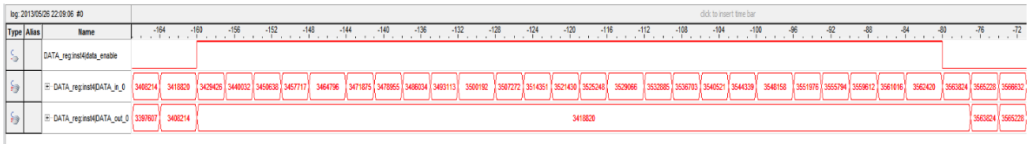


Figure 6.8: Timing diagram of the Data Enable and Data Register Modules used to synchronize communication with the Nios II processor. The bottom line of the diagram shows the duration and value of the sample that is delivered to the processor.

6.1.3 Nios II and Bluetooth Communication

The PmodBT2 Bluetooth module and the Nios II processor is verified with a C script on the Nios II. The script generated test data and transmits them to the PmodBT2 via the UART peripheral. The PmodBT module is wirelessly connected to a PC with an integrated Bluetooth link. To display the transmitted data, the Docklight RS232 terminal tool is used. The Docklight RS232 terminal communicates with a baud rate of 115.2 kbps with a packet format of 8 data bits, no parity bit and 1 stop bit.

Results

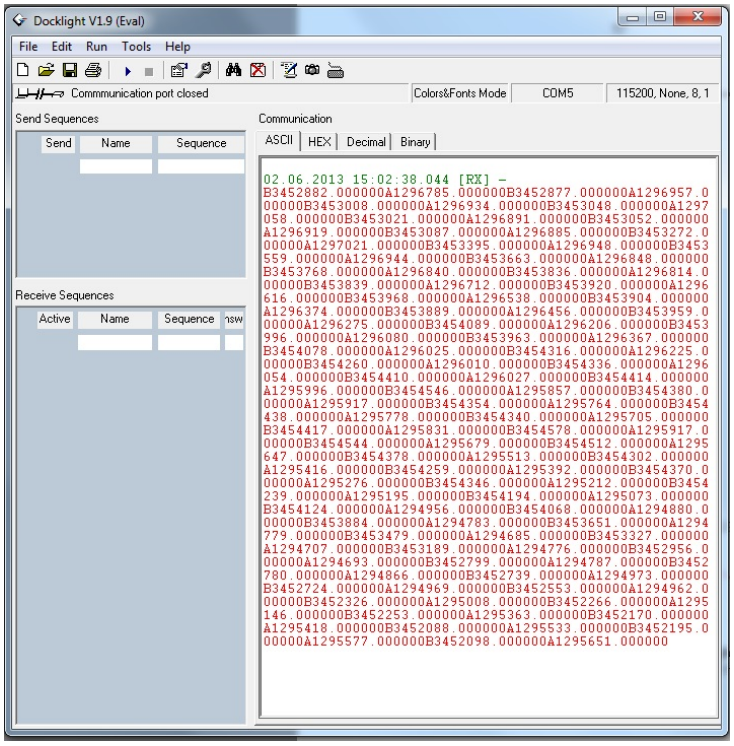


Figure 6.9: The Docklight RS232 terminal tool displaying the data stream received from the PmodBT2 Bluetooth module.

The data stream sent from the PmodBT2 Bluetooth module is displayed in Figure 6.9.

6.1.4 The Android Application

Experimental Setup

Each sub module of the BioDataLogger was tested individually via a debug terminal supplied by the Android SDK. Verification of the

BioDataLogger was performed to check the stability of the application. To test if any bugs are present or if any input combinations could crash the application, a user test was performed. During this test, simulated data was transmitted from the Nios II processor. While the application was plotting the data, the user was trying different button combinations to deliberately trigger an error. In order to profile the software performance, the Android SDK traceview and dmtracdump tools were used. When problems were detected, they were corrected and the profiling process was repeated. This was done until the intended software performance and functionality was obtained.

Results

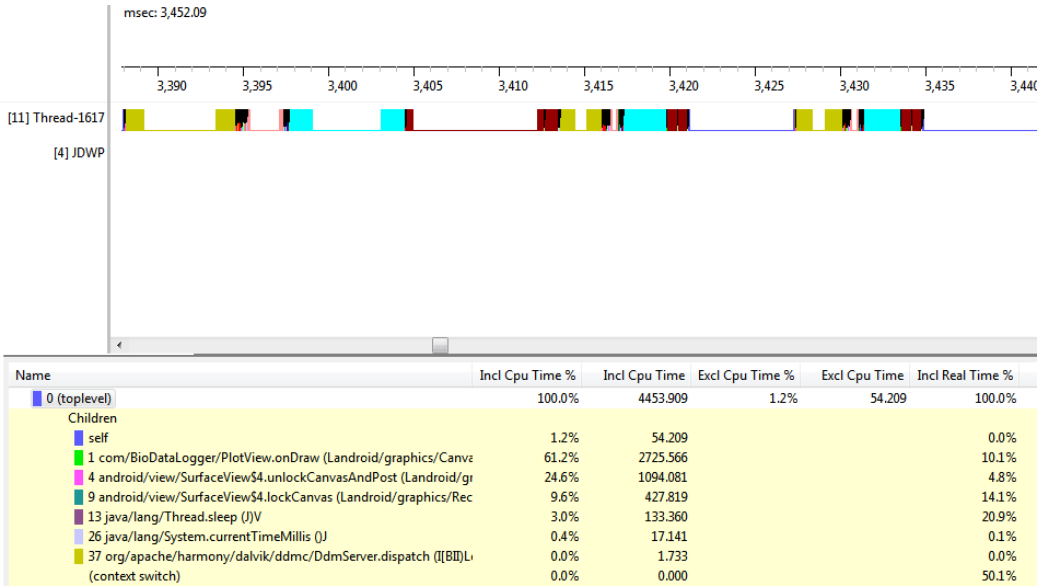


Figure 6.10: The profiling tool displaying the use of resources over time (top) and the time spent in the different functions of the BioDataLogger.

The timeline and profile panel supplied by the traceview and dmtracdump tools are displayed in Figure 6.10. The resource usage is shown in the timeline (top) and the profile panel (bottom) displays the resources used by the different functions of the BioDataLogger. As seen in the profile panel, the most demanding task is the function used by the PlotView to draw the graphs from the incoming data, with a CPU usage of 61.2%. The screenshot in Figure 6.11 on the facing page shows the BioDataLogger drawing data generated by the Nios II processor.

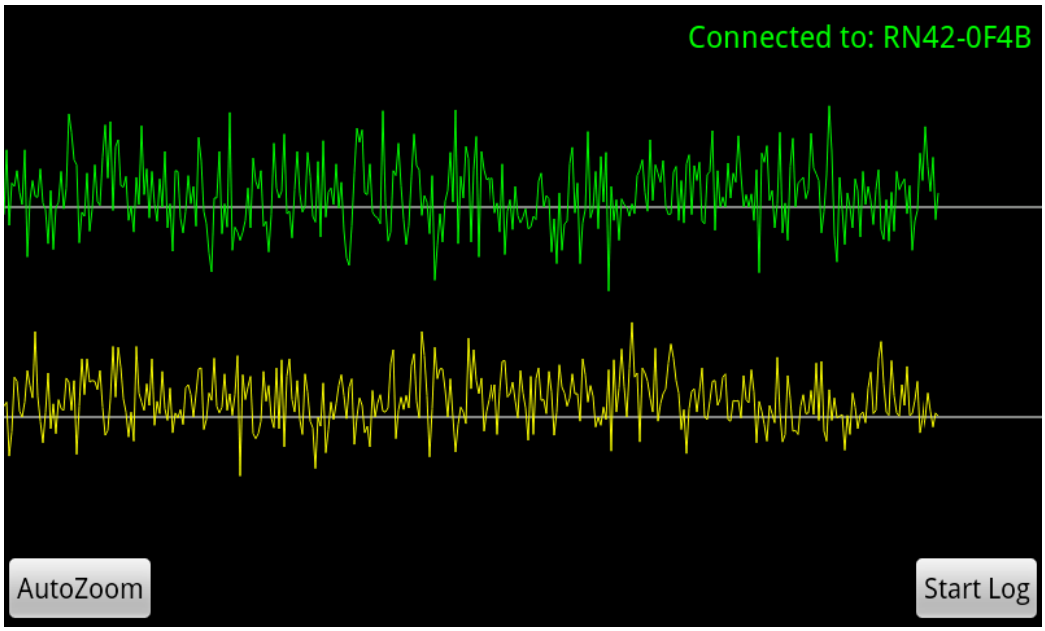


Figure 6.11: The BioDataLogger drawing generated data from the Nios II processor.

6.1.5 The RSO-2412DZ/H3 DC-DC Converter

Experimental Setup

A Tektronix TDS 2024B oscilloscope is used to measure output noise of the RSO-2412DZ/H3. The measurement is conducted on both the positive and negative rails in respect to the neutral connection.

Results

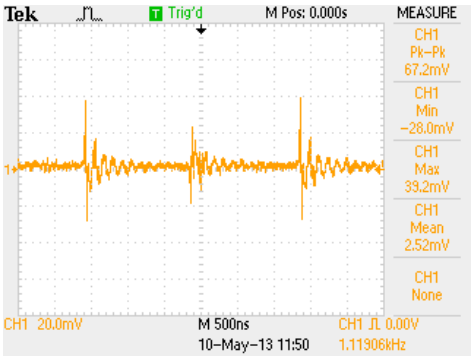


Figure 6.12: Output noise on the positive rail.

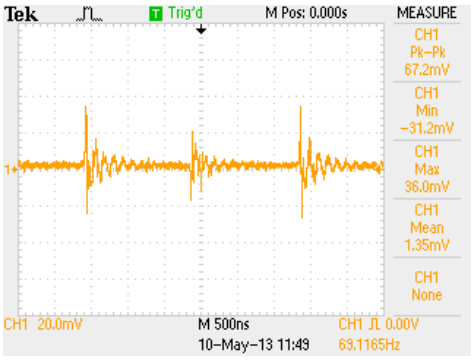


Figure 6.13: Output noise on the negative rail.

Figure 6.12 and 6.13 shows the noise on the positive and negative rails. The ripple and noise occurs at a frequency of approximately 666 kHz. The output ripple and noise of the RSO-2412DZ/H3 is approximately 67 mV_{p-p}. This measured value is higher then the 50 mV_{p-p} value specified by its datasheet.

6.2 Calibration of the Analog Front-end

The Matlab scripts used to generate the plots presented in this section can be found in Appendix D.3.

6.2.1 Howland Current Source

Experimental Setup

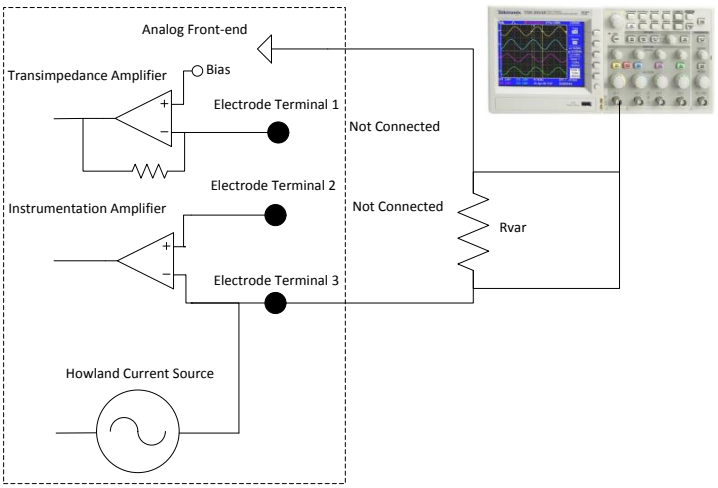


Figure 6.14: The setup used to test the Howland current source.

The setup used to test the Howland current source is shown in Figure 6.14. A Tektronix TDS 2024B oscilloscope is used to measure the voltage drop over resistor R_{var}. The resistance values used for R_{var} range from 65.4 kΩ to 267.4 k Ω. To compensate for uncertainties in the resistor values, they are measured with KEITHLEY 2635 SYSTEM SourceMeter.

Results

As seen in Figure 6.15 on the facing page, there is a linear relation between the different resistors R_{var} and the measured peak-to-peak voltage U_R. According to ohm's law this means that the Howland current source is able to maintain a stable AC current when the loads are changed. The

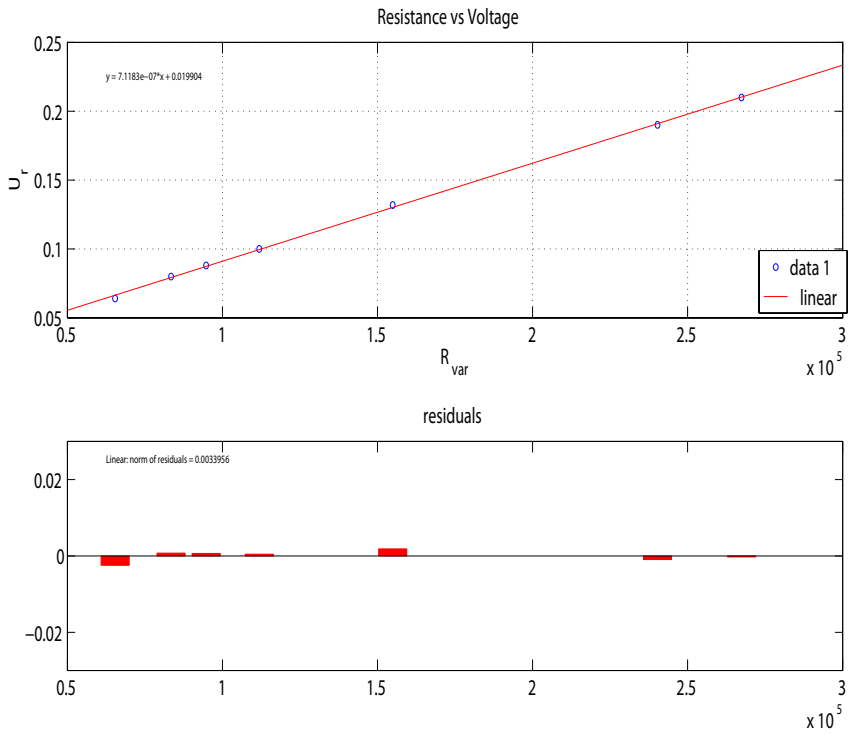


Figure 6.15: The measured peak-to-peak voltage U_R plotted against the different resistors R_{var} (Top pane). Residuals between measurements and the linear fit (Bottom pane).

range of the resistor values used is restricted due to the resolution of the oscilloscope and noise. The linear fit has an R-squared = 1.

6.2.2 The Resistance Measurement

Experimental Setup

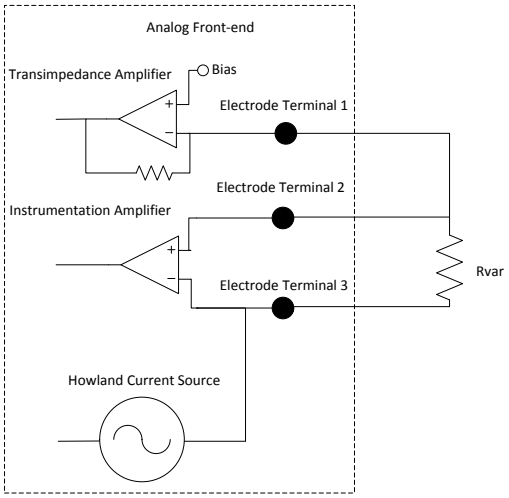


Figure 6.16: The setup used to calibrate the resistance measurement.

The setup used to calibrate the resistance measurements is shown in Figure 6.16. The resistor values used for R_{var} range from 239 Ω to 121.83 k Ω. To compensate for uncertainties in the resistor values, they are measured with the KEITHLEY 2635 SYSTEM SourceMeter. The instrument readout is done in the Nios II debug terminal.

Results

Figure 6.17 on the facing page demonstrates a linear relation between the values read by the instrument, and the different resistors R_{var}.The linear fit has an R-squared = 1.00.

6.2.3 The Reactance Measurement

Experimental Setup

The setup used to calibrate the reactance mesurments is shown in Figure 6.18 on the next page. The resistor R has a fixed value of 10 kΩ. C_x represents different capacitors in a range from 30 nF to 2 μF.

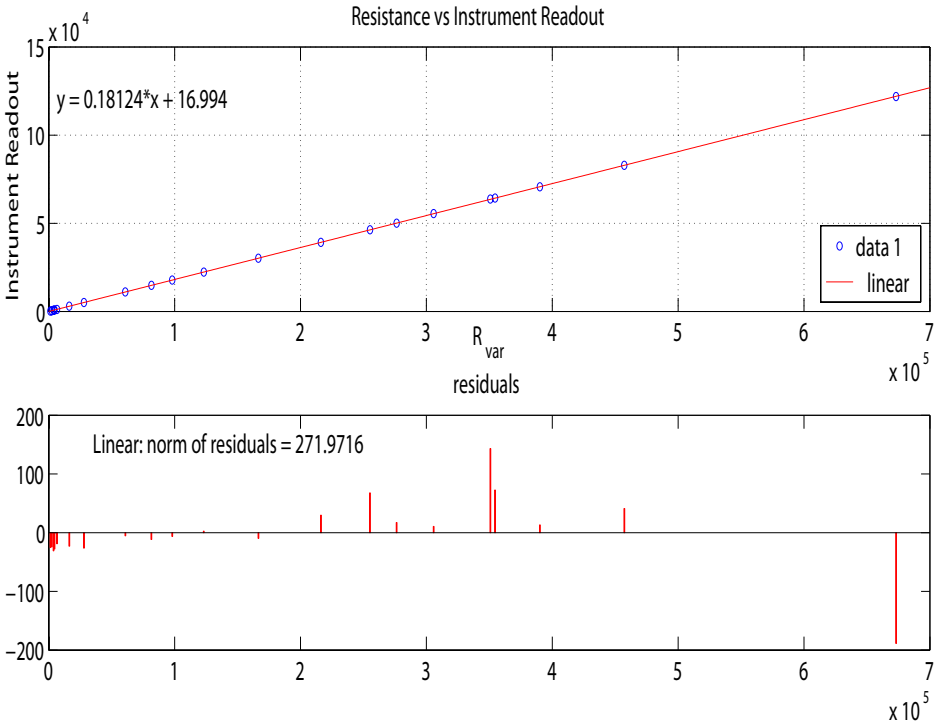


Figure 6.17: The instrument output plotted against the different resistors R_{var} (Top pane). Residuals between measurements and the linear fit (Bottom pane).

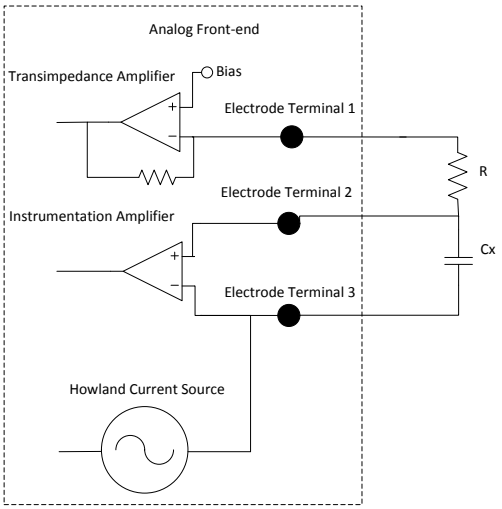


Figure 6.18: The setup used to calibrate the reactance measurement.

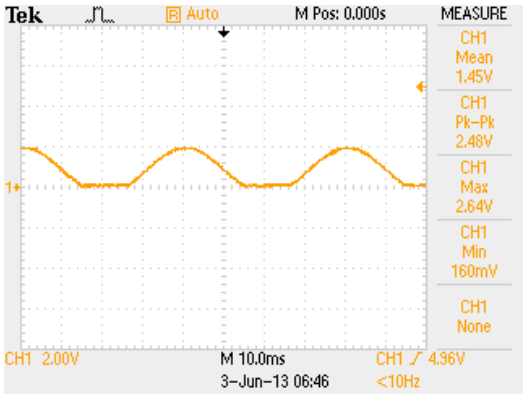


Figure 6.19: The offset on the inputs of the analog front-end.

Results

During this calibration, a small offset was discovered on the inputs of the Analog Front-end. This leads to a charging of the capacitor C_x as shown in Figure 6.19.

6.2.4 The Electric Potential Measurement

Experimental Setup

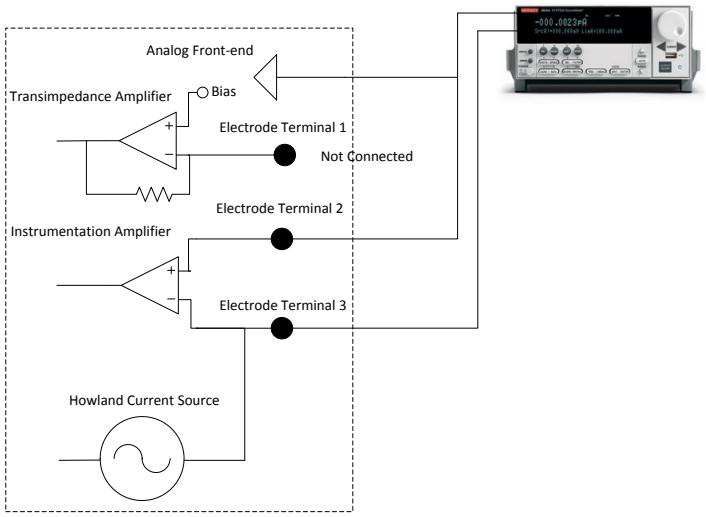


Figure 6.20: The setup used to calibrate the electric potential measured by the instrumentation amplifier.

The setup used to test the electric potential measured by the instrumentation amplifier is shown in Figure 6.20. The KEITHLEY 2635 SYS-

TEM SourceMeter is used to apply an electric potential in the range of -5 $\mu V$to -65 μV in steps of 5 μV. The instrument readout is done in the Nios II debug terminal.

Results

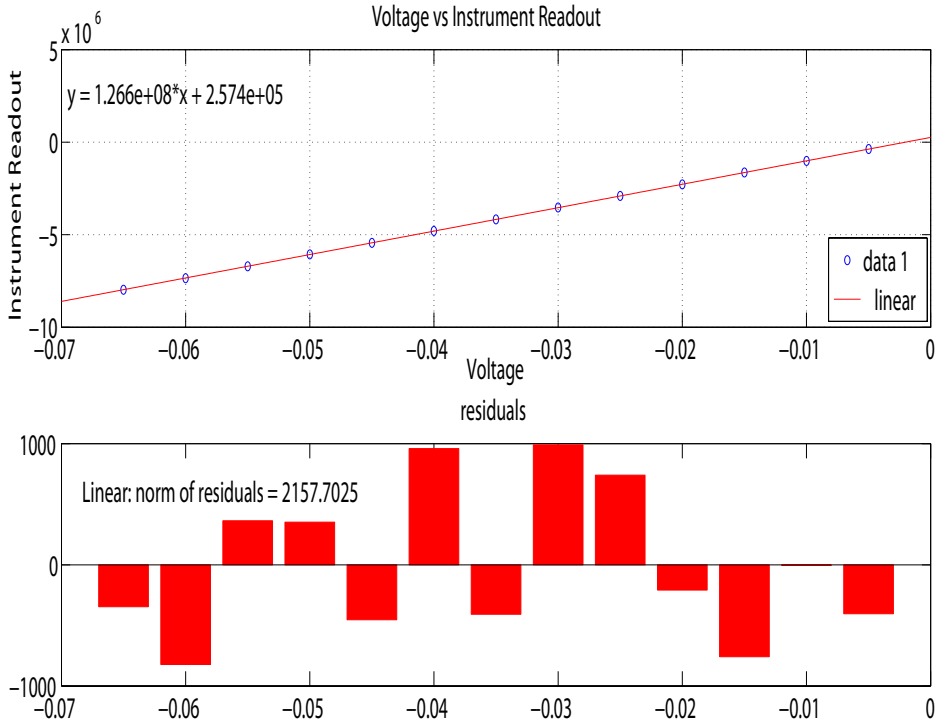


Figure 6.21: The instrument readout plotted against the different electric potentials (Top pane). Residuals between measurements and the linear fit (Bottom pane).

Figure 6.21 shows that there is a linear relation between the values read by the instrument, and the electric potential. The linear fit has an R-squared = 1.00.

6.2.5 The Current Measurement

Experimental Setup

The setup used to calibrate the current measurements is shown in Figure 6.22 on the following page. In order to generate a constant current AC signal, the Howland current source was used. The input voltage was in the range of 1 V to 4 V in steps of 0.3 V. This corresponds to a current range of 400 nA to 1.6 μA in steps of 120 nA.

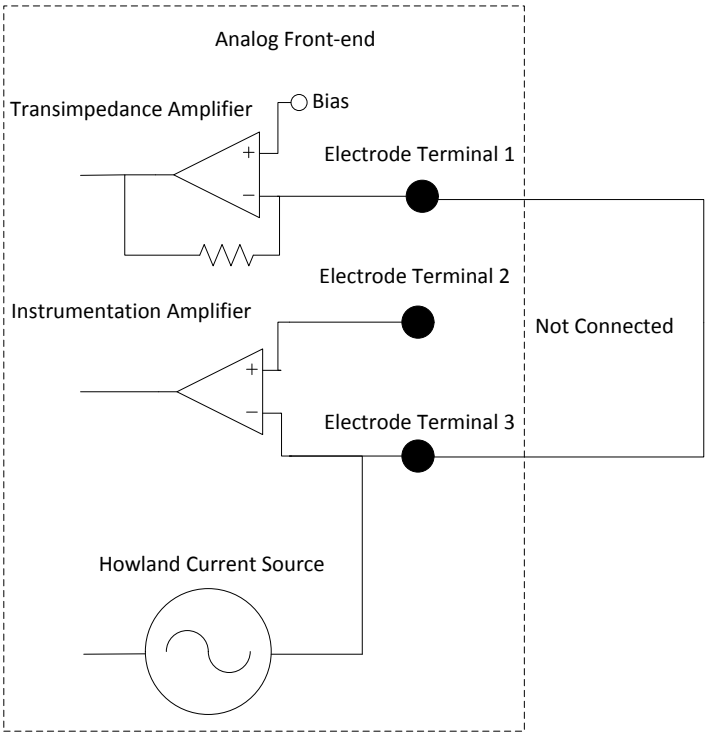


Figure 6.22: The setup used to calibrate the Current measurements.

Results

Figure 6.23 on the next page demonstrates a linear relation between the values read by the instrument, and the applied current. The linear fit has an R-squared = 0.9996.

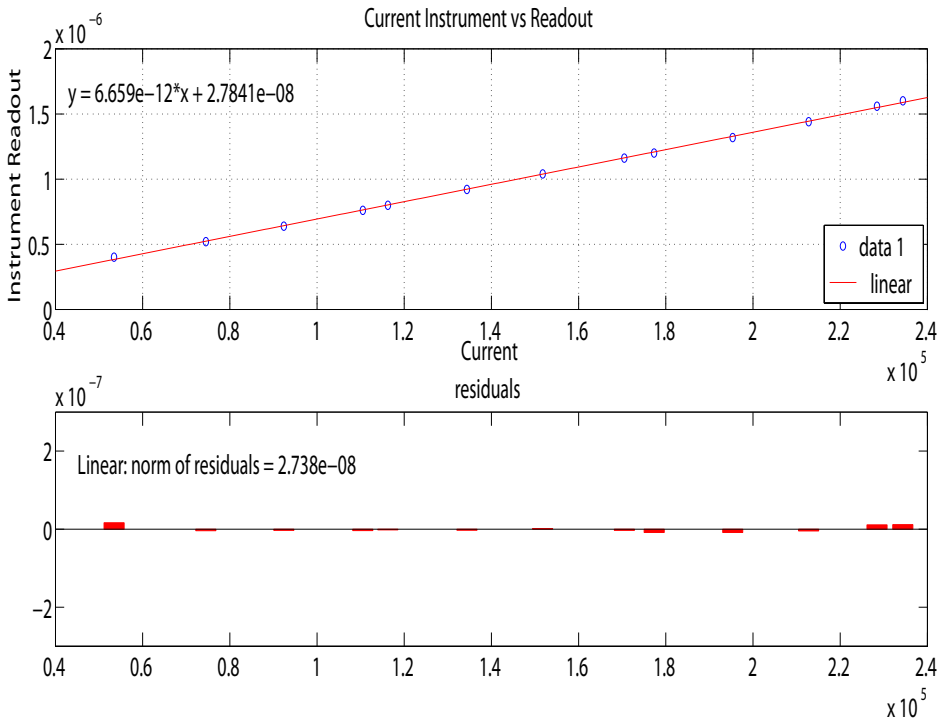


Figure 6.23: The instrument readout plotted against the applied current (Top pane). Residuals between measurements and the linear fit (Bottom pane).

6.3 Summary

The results presented in Section 6.1 demonstrate that the different modules operate according to the given specification in Chapter 5.

In Section 6.2, it has been shown that an offset is present at the input of the preamplifier. This was assumed to originate from the Howland Current Source. A series capacitor was added to the input of the preamplifier to remove this offset. This modification improved the measurements, but there is still an offset on the input. These results suggested that the offset originated from the transimpedance amplifier. To investigate this, the feedback resistor of the transimpedance amplifier was replaced with a lower value, which resulted in an increased offset. This implies that the offset is originating from the transimpedance amplifier or the following analog stages. Because of time restrictions, further investigations could not be conducted.

It has been shown that, even though the reactance measurement chain is inoperable, the potential, current and resistance measurements chains show a high degree of linearity. This verifies that the digital Lock-In Detection technique operate according to its specification. The Android mobile application is able to receive, display and store data produced by the operable measurement chains.

Chapter 7

Summary and Conclusion

This chapter summarizes the work and results presented in this thesis. It reviews the main contributions and results, and gives recommendations for further work.

7.1 Conclusion of the Present Work

The work of this thesis has been related to the development of a FPGA Based Development Platform for Biomedical Measurements. This thesis describes the design, development and verification of the development platform. Based on the accomplishments and contributions, the following conclusions are drawn:

- A prototype of an FPGA Based Development Platform for Biomedical Measurements has been developed.
- The digital signal processing is implemented based on modules, this makes the system easy to scale and expand.
- An Android mobile application that is able to receive data sent over Bluetooth from the development platform has been implemented tested and verified.
- A custom built front-end needed for the measurements of electrodermal activity has bean built and tested. The calibration of the reactance measurement chain was found to be inoperable. The potential, current and resistance measurement chains show a high degree of linearity.

7.2 Future Work and Recommendations

Based on the work presented in this, the following improvements and changes are recommended for future versions of the development Platform:

- The source of the offset present on the input of the preamplifier needs to be identified and corrected.
- Implement LC filtering to remove the output noise of the RSO-2412DZ/H3 DC-DC Converter.
- The single ended to differential circuit based on ADA4941-1 should be replaced with the ADC driver AD8476. This will simplify the input stages of the Data Acquisition Card and make them less reliant on impedance matching.
- Implement and test alternative algorithms like Coordinate Rotation Digital Computer (CORDIC) and FFT to extract the real and imaginary parts of the impedance.
- Make a new multi-layer PCB design that combines the data acquisition channels, FGPA, communication and power conversion. This makes the system able to be operated on battery power, making it portable.

Bibliography

Altera (2012). 'Cyclone III Device Handbook'. In: Volume 1.

— (2013a). *Altera Megafunctions*. http://www.altera.com/products/ip/altera/mega.html. [Online; accessed 8-April-2013].

— (2013b). *Nios II Processor: The World's Most Versatile Embedded Processor*. http://www.altera.com/devices/processor/nios2/ni2-index.html. [Online; accessed 8-April-2013].

Android-System-Architecture. https://en.wikipedia.org/wiki/File:Android-System-Architecture.svg. [Online; accessed 8-April-2013].

BURR-BROWN (1990). 'Implementation and Applications of Current Sources and Current Receivers'. In: pp. 20 –22.

— (1998). 'High Speed FET-Input INSTRUMENTATION AMPLIFIER'. In:

Bluetooth, SIG (2013). *Android*. :http://www.bluetooth.com/Pages/Bluetooth-Home.aspx. [Online; accessed 8-April-2013].

Boucsein, Wolfram (2012). *Electrodermal Activity*. Second Edition. Springer.

Bronzino, J.D. (2000). *The biomedical engineering handbook. 1*. The Biomedical Engineering Handbook. Springer. ISBN: 9783540663515. URL: http://books.google.no/books?id=6bK84ZHFuW4C.

Cacioppo, John T et al. (2007). *HANDBOOK OF PSYCHOPHYSIOLOGY*. Third Edition. Cambrige University Press.

Chen, D. X. et al. (Mar. 2010). 'Comparison of three current sources for single-electrode capacitance measurement'. In: *Review of Scientific Instruments* 81.3, pp. 034704 –034704–3. ISSN: 0034-6748. DOI: 10.1063/1.3367879.

D.C., Fowles (1974). 'Mechanisms of elektrodermal activity'. In: *Methods in Physiological Psychology. Bioelectric Recording Techniques*.

Dittmar, A. et al. (1991). 'Analysis Of Skin Potential Response Using A Novel Feature Code For The Study Of The Emotional Response'. In: pp. 427–428.

Google (2013). *Android*. :http://www.android.com/. [Online; accessed 8-April-2013].

Grimnes, S. (1982). 'Psychogalvanic reflex and changes in electrical parameters of dry skin'. English. In: *Medical and Biological Engineering and Computing* 20 (6), pp. 734–740. ISSN: 0140-0118. DOI: 10.1007/BF02442528. URL: http://dx.doi.org/10.1007/BF02442528.

Grimnes, S. et al. (2005). 'Cole electrical impedance Model-a critique and an alternative'. In: *Biomedical Engineering, IEEE Transactions on* 52.1, pp. 132–135. ISSN: 0018-9294. DOI: 10.1109/TBME.2004.836499.

Grimnes, Sverre (1984). 'Pathways of Ionic Flow through Human Skin in vivo'. In: *Acta Derm Venereol*.

Grimnes, Sverre et al. (2006). 'BIOIMPEDANCE'. In: *Wiley Encyclopedia of Biomedical Engineering*.

— (2008). *BIOIMPEDANCE AND BIOELECTRICITY BASICS*. Second Edition. Elsevier Ltd.

Grimnes, Sverre et al. (2010). 'Electrodermal activity by DC potential and AC conductance measured simultaneously at the same skin site'. In: *Skin Research and Technology* 17.1, pp. 26–34. ISSN: 1600-0846.

Herrmann et al. (1973). *Biochemie der Haut*. Stuttgart: Georg Thieme.

Huseby, Miriam Kirstine (2013). 'FPGA Based Development Platform for Biomedical Measurements'. In:

Jabbari, A et al. (2010). 'Simultaneous measurement of skin potential and conductance in electrodermal response monitoring'. In: *Journal of Physics: Conference Series* 224.1, p. 012091.

Johnsen, Børge (2009). 'PDA-basert instrument for måling av elektrodermal aktivitet'. In:

KEITHLEY (2004). *Low Level Measurements Handbook*. Sixth Edition. KEITHLEY.

Kerassidis, S. (1994). 'Is palmar and plantar sweating thermoregulatory?' In: *Acta Physiologica Scandinavica* 152.3, pp. 259–263. ISSN: 1365-201X. DOI: 10.1111/j.1748-1716.1994.tb09805.x. URL: http://dx.doi.org/10.1111/j.1748-1716.1994.tb09805.x.

Kitchin, Charles et al. (2006). *A DESIGNERâ€™S GUIDE TO INSTRUMENTATION AMPLIFIERS*. Third Edition. Analog Devices.

Kong, WW (2010). *UART â€“ Universal Asynchronous Receiver and Transmitter*. http://tutorial.cytron.com.my/2012/02/16/uart-universal-asynchronous-receiver-and-transmitter/. [Online; accessed 8-April-2013].

Kuno, Y. (1956). *Human perspiration*. American lecture series. Thomas. URL: http://books.google.no/books?id=6oI_AAAAYAAJ.

Malmivuo, Jaakko et al. (1995). *Bioelectromagnetism*. [Online; accessed 8-April-2013]. URL: \url{http://www.bem.fi/book/27/27.htm}.

Martinsen, Ø. G. et al. (1999). 'Measuring depth depends on frequency in electrical skin impedance measurements'. In: *Skin Research and Technology*, pp. 179–181.

Martinsen, Ørjan G et al. (2001). 'Facts and Myths about Electrical Measurement of Stratum corneum Hydration State'. In: *Medical and Biological Engineering and Computing*.

Masciotti, J.M. et al. (Jan. 2008). 'Digital Lock-In Detection for Discriminating Multiple Modulation Frequencies With High Accuracy and Computational Efficiency'. In: *Instrumentation and Measurement, IEEE Trans-*

actions on 57.1, pp. 182 –189. ISSN: 0018-9456. DOI: 10.1109/TIM.2007.908604.

Millington, Philip F. et al. (1983). *Skin (Biological Structure and Function Books)*. Cambrige University Press.

Montagna, William et al. (1974). *THE STRUCTURE AND FUNCTION OF SKIN*. Third Edition. Academic Press, INC, pp. 1–17.

Motchenbacher, C.D. et al. (1993). *Low-Noise Electronic System Design*. JOHN WILEY and SONS, INC.

Osborne, A. (1980). *An Introduction to Microcomputers: Basic concepts*. An Introduction to Microcomputers. Osborne/McGraw-Hill. ISBN: 9780931988349. URL: http://books.google.no/books?id=ScAjAAAAMAAJ.

Oslo Bioimpedance Group (2010). :http://www.bioimpedance.org. [Online; accessed 8-April-2013].

Psychophysiological Research, Society for et al. (2012). 'Publication recommendations for electrodermal measurements'. In: *Psychophysiology*, pp. 1017–1034.

Rabaey, J.M. (2009). *Low Power Design Essentials*. Series on integrated circuits and systems. Springer London, Limited. ISBN: 9780387717135. URL: http://books.google.no/books?id=A-sBy_nmQ8wC.

Sheingold, D. H. (1964). 'Impedance and Admittance Transformations using Operational Amplifiers'. In: *Lightning Empiricist* Volume 12, p. 7.

Skin layers. http://en.wikipedia.org/wiki/File:Skinlayers.png. [Online; accessed 8-April-2013].

TI (2002). *Op Amps For Everyone*. Texas Instruments.

Technologies, Avago (2010). 'HCNR200 and HCNR201 Applications in Motor Drive and Current Loop'. In: *Application Note 5394*.

Terry, Ryan. *The Integumentary System*. http://www.ck12.org/user:dGVycnlyQHZhbGxleTI2Mi5vcmc./section/The-Integumentary-System-\%253A\%253Aof\%253A\%253A-Introduction-to-the-Human-Body\%253A-Bones\%252C-Muscles\%252C-and-Skin/. [Online; accessed 8-April-2013].

Tronstad, Christian (2012). *Developments in biomedical sensors based on electrical impedance : improvements in electrodes, instrumentation and signal processing technology for new and existing biomedical sensors for clinical use*. [Department of Physics], Faculty of Mathematics and Natural Sciences, University of Oslo.

Tronstad, Christian et al. (2008). 'Electrical measurement of sweat activity'. In: *Physiological Measurement* 29.6, S407. URL: http://stacks.iop.org/0967-3334/29/i=6/a=S34.

Tronstad, Christian et al. (2010). 'A study on electrode gels for skin conductance measurements'. In: *Physiological Measurement* 31.10, p. 1395. URL: http://stacks.iop.org/0967-3334/31/i=10/a=008.

Tronstad, Christian et al. (2013). 'Improved Estimation of Sweating Based on Electrical Properties of Skin'. In: *Annals of Biomedical Engineering* 41.5, p. 1074. URL: http://dx.doi.org/10.1007/s10439-013-0743-4.

Vishay, Semiconductors (2008). 'Designing Linear Amplifiers Using the IL300 Optocoupler'. In: *Application Note 50*.

Webster, J.G. (2009). *Medical Instrumentation Application and Design*. Wiley. ISBN: 9780471676003. URL: http://books.google.no/books?id=1Y4lAAAACAAJ.

Xiaoke, Li et al. (May 2012). 'Analysis of constant-current characteristics for current sources'. In: *Control and Decision Conference (CCDC), 2012 24th Chinese*, pp. 2607 –2612. DOI: 10.1109/CCDC.2012.6244414.

Yamamoto, T. et al. (1981). 'Non-linear electrical properties of skin in the low frequency range'. English. In: *Medical and Biological Engineering and Computing* 19.3, pp. 302–310. ISSN: 0140-0118. DOI: 10.1007/BF02442549. URL: http://dx.doi.org/10.1007/BF02442549.

Appendix A

User Manual for the BioDataLogger Android Application

This user manual will go through the steps needed to use the BioDataLogger application. Before you can install and run the application, you need to enter settings - security, and tap the Unknown Sources box. When the

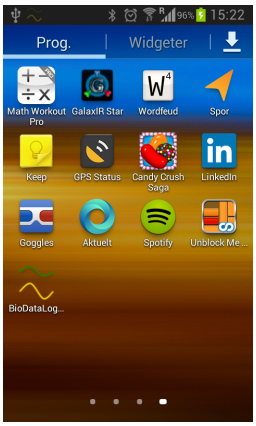


Figure A.1: Start the application.

application is installed, you can locate and run the BioDataLogger logo shown in the bottom right corner of Figure A.1

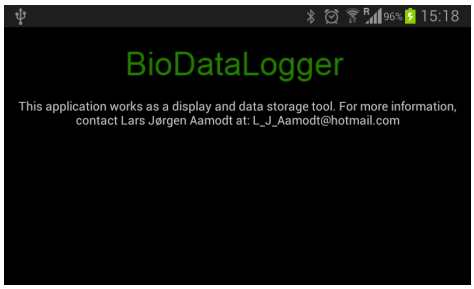


Figure A.2: The information screen.

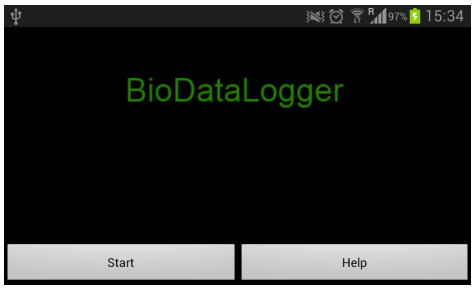


Figure A.3: The start screen.

Figure A.3 shows the start screen of the application. This screen has two buttons located under the BioDataLogger logo. The Help button enters the information screen shown in Figure A.3. To enter the main screen of the application and start a recording, push the start button.

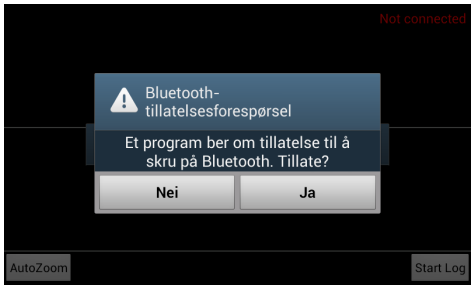


Figure A.4: Activated Bluetooth dialog.

Figure A.5: The main screen.

If Bluetooth is not activated, the application will ask for permission to activate it as shown in FigureA.4. After the Bluetooth is activated, the application will enter the main screen shown in Figure A.5. In order to start a bluetooth connection, push the option button on the lover left corner of the device. This will activate the option menu shown in Figure A.6.

Figure A.6: The option menu.

The Connect Bluetooth and Disconnect Bluetooth buttons are used to connect and disconnect from the development platform. When the Connect Bluetooth button is pushed, the dialog shown in Figure A.7 will appear.

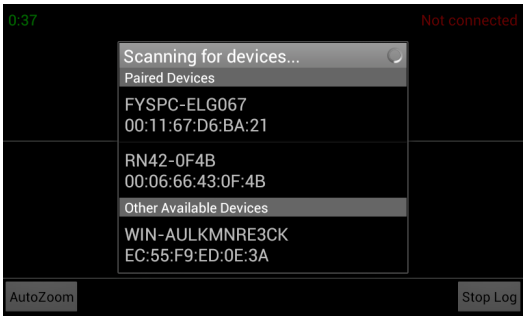


Figure A.7: Select Bluetooth device dialog.

In order to connect to the development platform, select the device named RN42-0F4B with the MAC address 00:06:66:43:0F:4B. After a successful connection is established, the name of the device will appear in the top right corner of the screen. The device will now start to plot the incoming data as shown in Figure A.8. The two buttons Auto Zoom and Start Log are used to scale the incoming data and start the log respectively. When the log is activated a counter will appear in the upper write corner of the screen.

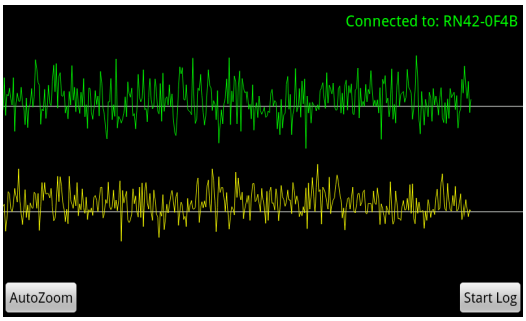


Figure A.8: BioDataLogger

The screen is divided in two sections by a horizontal line. The upper section is dedicated to the EDA related measurements, and the lower section is dedicated to the ECG related measurements. If the recording session is disturbed during a recording, the notification shown in Figure A.9 on the following page can be used to enter the application and continue the recording. Since the data log is managed by a separate serves, there will be no data loss. If a zoom level other then the one provided by the Auto Zoom is desired. The individual graphs can be

scaled by sliding a finger on the screen. The two graphs can also be scaled together by sliding a finger up or down the right side of the screen.

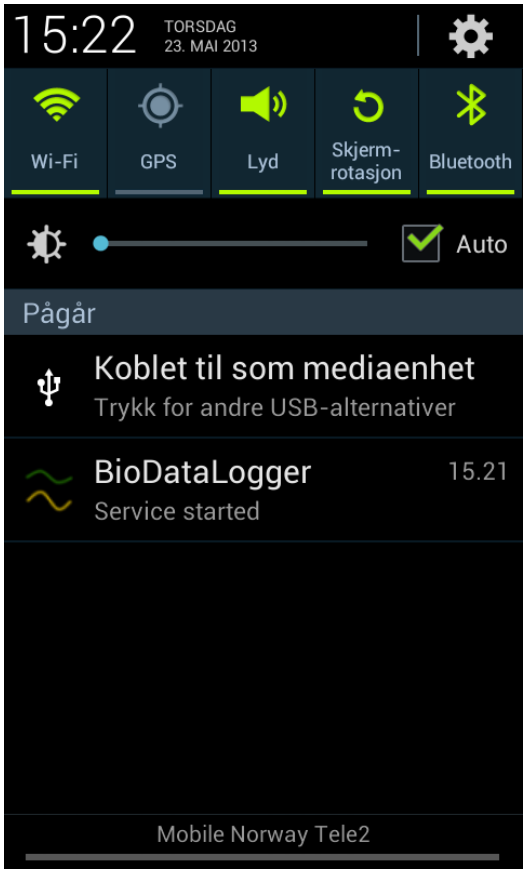


Figure A.9: The notification.

The log file will be stored in a folder named Data on the phones SD-card.

Appendix B

The BioDataLogger UML and Code

B.1 UML Diagram

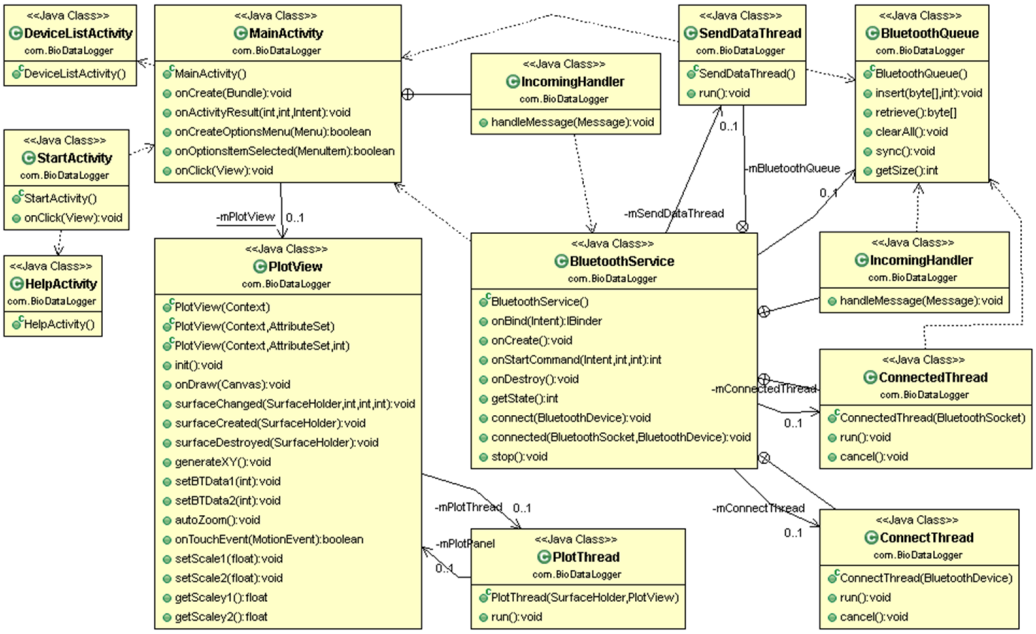


Figure B.1: UML diagram displaying the architecture of the BioDataLogger.

B.2 Code

Listing: StartActivity.java

```
package com.BioDataLogger;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.view.Window;
import android.widget.Button;

/**
 * This Activity serves as a starting point for the
 * BioDataLogger and displays information.
 * @author Lars Jørgen Aamodt
 */
public class StartActivity extends Activity implements View.OnClickListener{



    /** Called when the activity is first created. */
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        requestWindowFeature(Window.FEATURE_NO_TITLE);

        setContentView(R.layout.start);

        Button startPlot = (Button) findViewById(R.id.startPlot);
        startPlot.setOnClickListener(this);

        Button help = (Button) findViewById(R.id.help);
        help.setOnClickListener(this);


    }

    /**Listener for menu */
    public void onClick(View v) {
        Intent in;

        switch (v.getId())  {

        // start plot activity
        case R.id.startPlot :

            in = new Intent(this, MainActivity.class);
            startActivity(in);
            this.finish();
            break;
        //start help activity
        case R.id.help :

            in = new Intent(this, HelpActivity.class);
            startActivity(in);

            break;


        }

    }
```

```
}
```

Listing: PlotView.java

```
package com.BioDataLogger;

import java.util.Arrays;
import java.util.concurrent.atomic.AtomicBoolean;

import android.content.Context;
import android.graphics.Bitmap;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Paint;
import android.util.AttributeSet;
import android.util.Log;
import android.view.MotionEvent;
import android.view.SurfaceHolder;
import android.view.SurfaceView;
/**
 * This class extends SurfeceView in order to draw 2D information to the screen.
 * @author Lars Jørgen Aamodt
 */
public class PlotView extends SurfaceView implements SurfaceHolder.Callback {

    private PlotThread mPlotThread;

    private final static int dataLength = 720;
    private final static int doubleDataLength = dataLength*2;

    private float[] plotData1 = new float[doubleDataLength];
    private float[] plotData2 = new float[doubleDataLength];
    private static final int[] dataX = new int[dataLength];

    private static final int width = 800;
    private static final int height = 422;
    private static final int dy = height /2;
    private float scale1 = 15;
    private float scale2 = 15;
    private float scaleOld1;
    private float scaleOld2;

    private  static final Paint paintPlot1 = new Paint();
    private  static final Paint paintPlot2 = new Paint();
    private  static final Paint paintBackground = new Paint();


    final AtomicBoolean backgroudMade = new AtomicBoolean(false);//TODO trenger
        nok ikke atomic


    private static final boolean D = true;
    private static final String TAG = "PlotView";

    // 2 touch states
    private static final int NONE = 0;
    private static final int DRAG = 1;

    //Variable for touch state
    private int mode = NONE;

    // background Bitmap for drawing center line
    private static Bitmap background;

    private float yOld;
    private float xOld;
```

```
    private float yNew;
    private float yDif;
    private boolean fingerDown = true;

    /**
     * Constructor PlotView
     */
    public PlotView(Context context) {
        super(context);
        init();
    }

    /**
     * Constructor PlotView
     */
    public PlotView(Context context, AttributeSet attrs) {
        super(context,attrs);
        init();
    }

    /**
     * Constructor PlotView
     */
    public PlotView(Context context, AttributeSet attrs, int defStyle) {
        super(context,attrs,defStyle );
        init();
    }

    /**
     * initiation method
     */
    public void init() {

        getHolder().addCallback(this);

        mPlotThread = new PlotThread(getHolder(), this);

        generateXY();

        //enable painting tools
        paintingTools();

        setKeepScreenOn(true);

    }

    /**
     * This method is where the drawing takes place.
     */
    @Override
    public void onDraw(Canvas canvas) {
        //pre-draw background
        drawBackground(canvas);

        // draw the background

        canvas.drawBitmap(background, 0, 0, paintBackground);

        drawPlot(canvas,plotData1,paintPlot1,plotData2,paintPlot2);



    }

    /**
     * This method is called immediately after any structural changes (format or
         size) have been made to the surface.
     * The surface is locked to landscape, so this is not in use.
```

```
     */
    public void surfaceChanged(SurfaceHolder holder, int format, int width,int
        height) {

    }

    /**
     * This method is called immediately after the surface is first created.
     */
    public void surfaceCreated(SurfaceHolder holder) {
        if (D)Log.i(TAG, "surfaceCreated");

        // make new thread
        mPlotThread = new PlotThread(getHolder(), this);
        mPlotThread.setRunning(true);
        // start the thread
        mPlotThread.start();
    }

    /**
     * This method is called immediately before a surface is destroyed.
     */
    public void surfaceDestroyed(SurfaceHolder holder) {
        if (D)Log.i(TAG, "surfaceDestroyed");

        backgroudMade.set(false);

        //tell thread to shut down & wait for it to finish
        //clean shutdown
        boolean retry = true;
        mPlotThread.setRunning(false);
        while (retry) {
            try {
                mPlotThread.join();
                retry = false;
            } catch (InterruptedException e) {
                //try shutting down the thread again and again...
            }
        }

    }

    /**
     * Pre-draw the black background and center line
     */
    private void drawBackground(Canvas canvas){
        if(!backgroudMade.get()){
            background = Bitmap.createBitmap(canvas.getWidth(), canvas.getHeight
                (), Bitmap.Config.RGB_565);
            Canvas c = new Canvas(background);
            c.drawColor(Color.BLACK);
            for(int j=0; j<(width-1); j++){
                c.drawLine(0,dy,width,dy, paintBackground);
                c.drawLine(0,height,width,height, paintBackground);
            }
            backgroudMade.set(true);
        }
    }

    /**
     * This method generates x values for dataX array
     */
    public void generateXY(){

        for(int k = 0; k < doubleDataLength; k++){
            plotData1[k] = height-(dy+2);
            plotData2[k] = height-(2);
        }
```

```
        for ( int j = 1, i = 1; j <= dataLength-2; j+=2 ,i++){
            dataX[j] =  i*2;
            dataX[j+1]= i*2;
        }
        dataX[dataLength-1]= dataLength;
    }

    /**
     * This method draws the two data plots.
     * @param canvas the canvas to draw on.
     */
    private static void drawPlot(Canvas canvas,float[] data1,Paint p1, float[]
        data2,Paint p2){
        // draw plot 1
        canvas.drawLines(data1,p1);
        // draw plot 2
        canvas.drawLines(data2,p2);
    }

    /**
     * This method sets and shift data in to the data array 1.
     * @param d1
     */
    public void setBTData1(int d1){

        //if (D) Log.d(TAG,"setBTData");
        // shift Data
        System.arraycopy(plotData1, 0, plotData1, 4, doubleDataLength-4);
        plotData1[1] = height-((d1*scale1)+(dy+2));

        plotData1[3] = plotData1[5];

        // Replace x values in plot
        for(int i = 0, j= 0; i < doubleDataLength-1; i+=2, j++){
            plotData1[i] = dataX[j];
        }
    }

    /**
     * This method sets and shift data in to the data array 2.
     * @param d2
     */
    public void setBTData2(int d2){


        // shift Data
        System.arraycopy(plotData2, 0, plotData2, 4, doubleDataLength-4);
        plotData2[1] = height-((d2*scale2)+(2));

        plotData2[3] = plotData2[5];

        // Replace x values in plot
        for(int i = 0, j= 0; i < doubleDataLength-1; i+=2, j++){
            plotData2[i] = dataX[j];
        }
    }

    /**
     * This method sets the different drawing parameters.
     */
    private void paintingTools() {

        // paint plot1
        paintPlot1.setColor(Color.GREEN);
        paintPlot1.setAntiAlias(true);
        //paintPlot1.setStrokeWidth(1);
        //paintPlot1.setStyle(Paint.Style.STROKE);
```

```
        // paint plot1
        paintPlot2.setColor(Color.YELLOW);
        paintPlot2.setAntiAlias(true);
        //paintPlot2.setStrokeWidth(1);
        //paintPlot2.setStyle(Paint.Style.STROKE);

        //paint center line
        paintBackground.setColor(Color.GRAY);
        paintBackground.setAntiAlias(true);
        //paintBackground.setStrokeWidth(1);
        paintBackground.setStyle(Paint.Style.STROKE);

    }

    /**
     * This method adjusts the scaling of data array 1 and 2.
     * @return true
     */
    public void autoZoom(){

        //Array for y values of plotData1
        float[] tmpDataY1 = new float[dataLength];
        //Array for y values of plotData2
        float[] tmpDataY2 = new float[dataLength];
        //Get y values from plotData and copy them to tmpDataY
        int n = 0;
        for(int i = 1; i < doubleDataLength; i++){
            //If number is odd (y)
            if (i%2 != 0){
                tmpDataY1[n] = plotData1[i];
                tmpDataY2[n] = plotData1[i];
                n++;
            }
        }

        Arrays.sort(tmpDataY1);
        // diff1 = max - min
        float diff1 = tmpDataY1[dataLength-1]-tmpDataY1[0];

        scaleOld1= scale1;
        scale1= (height/diff1) *5;//(((height/2) / diff1)/2);

        Arrays.sort(tmpDataY1);
        // diff2 = max - min
        float diff2 = tmpDataY1[dataLength-1]-tmpDataY1[0];

        scaleOld2= scale2;
        scale2= (height/diff2)*5;//(((height/2) / diff2)/2);

        // Replace y values in plot
        for(int i1 = 1; i1 < doubleDataLength; i1++){
            //If number is odd (y)
            if (i1%2 != 0){
                //find the original value and calculate the new
                plotData1[i1] =    height-(( (((-plotData1[i1]+height)-dy-2)/
                    scaleOld1) *scale1)+(dy+2));
                plotData2[i1] =    height-(( (((-plotData2[i1]+height)-2)/
                    scaleOld2) *scale2)+(2));
            }
        }
    }

    /**
     * This method handles touch screen motion events.
     */
    public boolean onTouchEvent(MotionEvent event) {
```

```
// Handle touch events

switch (event.getAction() & MotionEvent.ACTION_MASK) {

case MotionEvent.ACTION_DOWN: //finger down
    //if (D) Log.d(TAG, "finger down" );

    yOld= event.getY();
    xOld= event.getX();
    fingerDown = true;

    mode = DRAG;
    break;

case MotionEvent.ACTION_UP: //finger  up
    //if (D) Log.d(TAG, "finger up" );

    fingerDown = false;
    break;

case MotionEvent.ACTION_MOVE:
    if (D) Log.d(TAG, "finger move" );

    if (mode == DRAG){

        if (fingerDown) {
            yNew = event.getY();
        }

        if ((yOld < height*0.5) || (xOld > width*0.9)){
            yDif = yOld-yNew;

            //if (D) Log.d(TAG, "y"+ yOld );
            if (yDif >= 10 || yDif <= -10){

                scaleOld1= scale1;
                scale1 += yDif/100;

                // Replace y values in plot
                for(int i1 = 1; i1 < doubleDataLength; i1++){
                    //If number is odd (y)
                    if (i1%2 != 0){
                        //find the original value and calculate the new
                        plotData1[i1] =    height-(( (((-plotData1[i1]+
                            height)-dy-2)/scaleOld1) *scale1)+(dy+2));
                    }
                }
            }
        }

        if ((yOld > height*0.5) || (xOld > width*0.9)){
            yDif = yOld-yNew;

            //if (D) Log.d(TAG, "y"+ yOld );
            if (yDif >= 10 || yDif <= -10){
                scaleOld2= scale2;
                scale2 += yDif/100;

                // Replace y values in plot
                for(int i1 = 1; i1 < doubleDataLength; i1++){
                    //If number is odd (y)
                    if (i1%2 != 0){
                        //find the original value and calculate the new
                        plotData2[i1] =    height-(( (((-plotData2[i1]+
                            height)-2)/scaleOld2) *scale2)+(2));
                    }
                }
            }
```

```
                }
                if (xOld > width*0.9){
                    yDif = yOld-yNew;

                    //if (D) Log.d(TAG, "y"+ yOld );
                    if (yDif >= 10 || yDif <= -10){
                        scaleOld1= scale1;
                        scaleOld2= scale2;

                        scale1 += yDif/100;
                        scale2 += yDif/100;

                        // Replace y values in plot
                        for(int i1 = 1; i1 < doubleDataLength; i1++){
                            //If number is odd (y)
                            if (i1%2 != 0){
                                //find the original value and calculate the new
                                plotData1[i1] =    height-(( (((-plotData1[i1]+
                                    height)-dy-2)/scaleOld1) *scale1)+(dy+2));
                                plotData2[i1] =    height-(( (((-plotData2[i1]+
                                    height)-2)/scaleOld2) *scale2)+(2));
                            }
                        }
                    }
                }
                break;
            }
        }
        return true;
    }
    /**
     * This method sets scale on restore
     * @param y the scale
     */
    public void setScale1(float y){

        scale1 = y;
    }

    /**
     * This method sets scale on restore
     * @param y the scale
     */
    public void setScale2(float y){

        scale2 = y;
    }

    /**
     * This method gets scale.
     * @return y the current scale
     */
    public float getScaley1(){
        return scale1;
    }

    /**
     * This method gets scale.
     * @return y the current scale
     */
    public float getScaley2(){
        return scale2;
    }



}
```

Listing: PlotThread.java

```
package com.BioDataLogger;

import android.graphics.Canvas;
import android.util.Log;
import android.view.SurfaceHolder;


/**
 * This Thread holds the Canvas and executes drawing.
 * @author Lars Jørgen Aamodt
 */
public class PlotThread extends Thread {

    private  final String TAG = "PlotThread";
    private  final boolean D = false;


    private SurfaceHolder sHolder;
    private PlotView mPlotPanel;
    private boolean running = false;

    // maximum frames per second
    private final static int    MAX_FPS = 100;
    // maximum number of frames to be skipped
    private final static int    MAX_FRAME_SKIPS = 200;
    // the frame period
    private final static int    FRAME_PERIOD = 1000 / MAX_FPS;

    /**
     * Constructor
     * @param surfaceHolder
     * @param panel
     */
    public PlotThread(SurfaceHolder surfaceHolder, PlotView panel) {
        sHolder = surfaceHolder;
        mPlotPanel = panel;
    }

    /**
     * start and stop run()
     * @param run true for start and False for stop
     */
    protected void setRunning(boolean run) {
        running = run;
    }


    @SuppressWarnings("unused")// there is no need to calculate FPS when not
        debugging
    @Override
    public void run() {
        long startTime;      // start time of the cycle
        long timeDiff;           // time used by the cycle
        int sleepTime;           // ms to sleep if FPS > max_FPS
        int framesSkipped;       // number of frames being skipped
        int framesCounter = 0;// number of frames
        long totalTime = 0L;     // the total of n cycles
        double FPS;          // the calculated frames per second

        Canvas canvas;

        while (running) {
            canvas = null;
            try {
```

```
                canvas = sHolder.lockCanvas(null);
                if (canvas != null){
                    synchronized (sHolder) {
                        startTime = System.currentTimeMillis();
                        framesSkipped = 0;
                        if(D)framesCounter++;

                        mPlotPanel.onDraw(canvas); // draw plot

                        timeDiff = System.currentTimeMillis() - startTime;

                        sleepTime = (int)(FRAME_PERIOD - timeDiff);

                        if (D)totalTime+= timeDiff;
                        // is there time to sleep
                        if (sleepTime > 0) {
                            //add the sleep time to the total time
                            totalTime+= timeDiff+sleepTime;
                            try {
                                Thread.sleep(sleepTime);

                            } catch (InterruptedException e) {}
                        }



                        if(framesCounter == 100 && D){
                            FPS = (1000/(totalTime/100));
                            if (D) Log.d(TAG,"FPS: "+ FPS);
                            totalTime=0L;
                            framesCounter= 0;
                        }

                        while (sleepTime < 0 && framesSkipped < MAX_FRAME_SKIPS)
                            {

                            sleepTime += FRAME_PERIOD;

                            framesSkipped++;
                        }
                    }
                }
            } finally {
                // make shore the surface is not left in an inconsistent state
                if (canvas != null) {
                    sHolder.unlockCanvasAndPost(canvas);
                }
            }
        }
    }


}
```

Listing: MainActivity.java

```
package com.BioDataLogger;


import android.app.Activity;
import android.bluetooth.BluetoothAdapter;
import android.content.ComponentName;
import android.content.Context;
import android.content.Intent;
import android.content.ServiceConnection;
import android.graphics.Color;
```

```
import android.os.Bundle;
import android.os.Handler;
import android.os.IBinder;
import android.os.Message;
import android.os.Messenger;
import android.os.RemoteException;
import android.os.SystemClock;
import android.util.Log;
import android.view.Menu;
import android.view.MenuInflater;
import android.view.MenuItem;
import android.view.View;
import android.view.View.OnClickListener;
import android.view.Window;
import android.view.WindowManager;
import android.widget.Button;
import android.widget.TextView;
import android.widget.Toast;
import android.widget.ToggleButton;
/**
 * This Activity serves as the main class of BioDataLogger and holds the
     different Views needed to display information to the screen.
 * It also starts the BluetothService and binds to it in order to receive the
     data sent over the Bluetooth link.
 * @author Lars Jørgen Aamodt
 *
 */
public class MainActivity extends Activity implements  OnClickListener {

    private  final String TAG = "MainActivity";
    private  final boolean D = true;

    // views used by this activity
    private  static PlotView mPlotView;
    private  static TextView mTextViewConnect;
    private  static TextView mTextViewLog;
    private  static TextView mTextViewPlot1;
    private  static TextView mTextViewPlot2;
    private  static TextView mTextViewError;

    // Name of the connected device
    private  String mConnectedDeviceName = null;
    // Local Bluetooth adapter
    private  BluetoothAdapter mBluetoothAdapter = null;
    private  String mBluetoothAddress;
    private  Messenger mService = null;

    // Array for incoming data.
    private int[] incomingData = new int[5];

    private  boolean mIsBound;

    private Handler timeHandler;
    private long startLogTime;

    private final Messenger mMessenger = new Messenger(new IncomingHandler());



    // Message types sent from the BluetoothService Handler
    public  final static int MESSAGE_STATE_CHANGE = 1;
    public  final static int MESSAGE_READ = 2;
    public  final static int MESSAGE_DEVICE_NAME = 3;
    public  final static int MESSAGE_TOAST = 4;

    // Intent request codes
    private final static int REQUEST_ENABLE_BT = 1;
    private final static int REQUEST_CONNECT_DEVICE = 2;
```

```
// Key names received from the BluetoothService Handler
public final static String DEVICE_NAME = "deviceName";
public final static String TOAST = "toast";



/** Called when the activity is first created. */
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    // request to remove title
    requestWindowFeature(Window.FEATURE_NO_TITLE);
    getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
        WindowManager.LayoutParams.FLAG_FULLSCREEN);

    android.os.Process.setThreadPriority(android.os.Process.
        THREAD_PRIORITY_MORE_FAVORABLE);

    setContentView(R.layout.plot);

    mPlotView = (PlotView) findViewById(R.id.plotview);

    mTextViewPlot1 = (TextView) findViewById(R.id.plotText_1);
    mTextViewPlot1.setTextColor(Color.GREEN);

    mTextViewPlot2 = (TextView) findViewById(R.id.plotText_2);
    mTextViewPlot2.setTextColor(Color.YELLOW);

    mTextViewError = (TextView) findViewById(R.id.errorText);
    mTextViewError.setTextColor(Color.RED);
    mTextViewError.setText("Data format ERROR!");


    mTextViewLog = (TextView) findViewById(R.id.logText);
    mTextViewLog.setTextColor(Color.GREEN);

    mTextViewConnect = (TextView) findViewById(R.id.connectText);

    timeHandler = new Handler();

    Button autoZoom = (Button) findViewById(R.id.autozoom);
    autoZoom.setOnClickListener(this);

    ToggleButton startLog = (ToggleButton) findViewById(R.id.startlog);
    startLog.setOnClickListener(this);

    restore(savedInstanceState);
    // Bind to BluetoothService
    try {
        doBindService();
    } catch (Throwable t) {
        Log.e(TAG, "Failed to bind the service", t);
    }
}

/**
 * Called to retrieve per-instance state from an activity before being killed
 *  so that the state can be restored in onCreate(Bundle).
 */
@Override
protected void onSaveInstanceState(Bundle outState) {
    super.onSaveInstanceState(outState);
    outState.putInt("connectColor",mTextViewConnect.getCurrentTextColor());
    outState.putCharSequence("connectText", mTextViewConnect.getText());
    outState.putFloat("scale1", mPlotView.getScaley1());
    outState.putFloat("scale2", mPlotView.getScaley2());
```

```
    }

    /**
     * Restore the bundle made by onSaveInstanceState.
     * @param state Bundle made by onSaveInstanceState.
     */
    private void restore(Bundle state) {
        if (state!=null) {
            mTextViewConnect.setTextColor(state.getInt("connectColor"));
            mTextViewConnect.setText((String) state.getCharSequence("connectText"
                ));
            mPlotView.setScale1(state.getFloat("scale1"));
            mPlotView.setScale2(state.getFloat("scale2"));
        }
    }

    /**
     * Called after onCreate(Bundle) — or after onRestart() when the
     * activity had been stopped, but is now again being displayed to the user.
     */
    @Override
    protected void onStart() {
        super.onStart();
        if (D) Log.d(TAG,"On Start");

        checkBT();

        setErrorText(false);

        // Initialize the BluetoothService
        startService(new Intent(MainActivity.this, BluetoothService.class));

    }
    /**
     * This method gets the BluetoothAdapter, and tries to enable it.
     */
    protected void checkBT(){
        //Local Bluetooth adapter
        mBluetoothAdapter = BluetoothAdapter.getDefaultAdapter();


        if (mBluetoothAdapter == null) {
            Toast.makeText(getBaseContext(), "This device does not support
                Bluetooth!", Toast.LENGTH_SHORT).show();
            finish();
        }

        // if bluetooth is not enabled, request to enable it.
        if (!mBluetoothAdapter.isEnabled()) {
            Intent enableIntent = new Intent(BluetoothAdapter.
                ACTION_REQUEST_ENABLE);
            startActivityForResult(enableIntent, REQUEST_ENABLE_BT);
        }
    }

    /**
     * Called after onRestoreInstanceState(Bundle), onRestart(),
     * or onPause(), for your activity to start interacting with the user.
     */
    @Override
    protected void onResume() {
        super.onResume();
        if (D) Log.d(TAG,"On Resume");
    }

    /**
     * Called after onStop() when the current activity is being re-displayed to
         the user
```

```
     * (the user has navigated back to it). It will be followed by onStart() and
       then onResume().
     */
    @Override
    protected void onRestart() {
        super.onRestart();
        if (D) Log.d(TAG,"On Restart");
    }

    /**
     * Called when you are no longer visible to the user.
     */
    @Override
    protected void onStop() {
        super.onStop();
        if (D) Log.d(TAG,"On Stop");
    }

    /**
     * Perform final cleanup before an activity is destroyed.
     */
    @Override
    protected void onDestroy() {
        super.onDestroy();
        if (D) Log.d(TAG,"On Destroy");
        // Unbind from BluetoothService
        try {
            doUnbindService();
        } catch (Throwable t) {
            if (D) Log.e(TAG, "Failed to unbind from the service", t);
        }

    }

    /**
     * Called as part of the activity lifecycle when an
     * activity is going into the background, but has not (yet) been killed.
     */
    @Override
    protected void onPause() {
        super.onPause();
        if (D) Log.d(TAG,"On Pause");
    }

    /**
     * Initiates Bluetooth by binding to BluetoothService and launching
       DeviceListActivity.
     */
    private void initBT(){
        if (D) Log.d(TAG,"initBT");

        try {
            doBindService();
        } catch (Throwable t) {
            if (D) Log.e(TAG, "Failed to bind to the service", t);
        }

        // Launch the DeviceListActivity to see devices and do scan
        Intent serverIntent = new Intent(this, DeviceListActivity.class);
        startActivityForResult(serverIntent, REQUEST_CONNECT_DEVICE);

    }

    /**
     * Called when an activity you launched exits, giving you the requestCode you
       started it with,
     * the resultCode it returned, and any additional data from it. The
       resultCode will be RESULT_CANCELED
```

```
     * if the activity explicitly returned that, didn't return any result, or
         crashed during its operation.
     */
    public void onActivityResult(int requestCode, int resultCode, Intent data) {
        switch (requestCode) {
        case REQUEST_CONNECT_DEVICE:
            // When DeviceListActivity returns with a device to connect
            if (resultCode == Activity.RESULT_OK) {
                // Get the device MAC address
                mBluetoothAddress = data.getExtras().getString(DeviceListActivity
                    .EXTRA_DEVICE_ADDRESS);
                // send MAC address to service and start connection.
                Bundle b = new Bundle();
                if (mService != null) {
                    try {
                        if (D) Log.d(TAG,"MAC: " + mBluetoothAddress);
                        //Bundle b = new Bundle();
                        b.putString("str",mBluetoothAddress);
                        Message msg = Message.obtain(null, BluetoothService.
                            MSG_DEVICE_ADDRESS);
                        msg.setData(b);
                        mService.send(msg);
                        b.clear();
                    } catch (RemoteException e) {}
                }

            }
            break;
        case REQUEST_ENABLE_BT:
            // When the request to enable Bluetooth returns
            if (resultCode == Activity.RESULT_OK) {
                // Bluetooth is now enabled

            } else {
                // User did not enable Bluetooth or an error occurred
                Toast.makeText(this, "Bluetooth not enabled", Toast.LENGTH_SHORT)
                    .show();
                //finish();
            }
        }
    }


    /**
     * Sets error message to screen.
     * @param b true for error, false for no message.
     */
    private void setErrorText(boolean b){
        if (b){
            mTextViewError.setVisibility(View.VISIBLE);
        }else{
            mTextViewError.setVisibility(View.INVISIBLE);
        }
    }



    /**
     * Decode incoming data and sends it to setPlotText and PlotView to be
         displayed.
     * @param d incoming data
     */
    private void setData(byte[] d){

        if (D) Log.i(TAG, "DATA_length: " + d.length);

        for(int i = 0; i < 5; i++){
            incomingData[i] = (int) (d[i] & 0xFF); // convert unsigned Byte to
```

```
                integer
            if (D) Log.i(TAG, "DATA_in: " + i);
        }

        for(int i = 0; i < incomingData.length; i++){
            if (D) Log.i(TAG, "DATA_testifo: " + i);
        //* If data don't start on 0, display error message and sync queue.
        if ( incomingData[0] != 0 ){

            //setErrorText(true);
            /**
            try {
                Message msg = Message.obtain(null, BluetoothService.
                    MSG_SYNC_QUEUE);
                msg.replyTo = mMessenger;
                mService.send(msg);
            } catch (RemoteException e) {}
            // draw 0 for bad data

            mPlotView.setBTData1(0);

            mPlotView.setBTData2(0);
            return;
        }*/

        //setErrorText(false);

        //data start
        /**
        if ( incomingData[0] == 0 ){
            try{

                mTextViewPlot1.setText(Integer.toString(incomingData[1]));

                mTextViewPlot2.setText(Integer.toString(incomingData[2]));


                mPlotView.setBTData1(incomingData[3]);

                mPlotView.setBTData2(incomingData[4]);

            }catch (NumberFormatException e1){
            }
        }**/
    }
 }
}

    /**
     * Incoming Handler class that receives messages form the BluetoothService.
     * @author ljaamodt
     */
    class IncomingHandler extends Handler {
        @Override
        public void handleMessage(Message msg) {
            switch (msg.what) {
            case MESSAGE_STATE_CHANGE:
                if (D) Log.i(TAG, "MESSAGE_STATE_CHANGE: " + msg.arg1);
                switch (msg.arg1) {
                case BluetoothService.STATE_CONNECTED:
                    mTextViewConnect.setText("Connected to: "+
                        mConnectedDeviceName);
                    mTextViewConnect.setTextColor(Color.GREEN);
                    break;

                case BluetoothService.STATE_CONNECTING:
                    mTextViewConnect.setText("Connecting..");
                    mTextViewConnect.setTextColor(Color.GREEN);
```

```
                    break;

                case BluetoothService.STATE_NONE:
                    mTextViewConnect.setText("Not connected");
                    mTextViewConnect.setTextColor(Color.RED);
                    break;

                case BluetoothService.STATE_DISCONNECTED:
                    mTextViewConnect.setText("Not connected");
                    mTextViewConnect.setTextColor(Color.RED);
                    break;
                }
                break;

            case MESSAGE_READ:
                if (D) Log.i(TAG, "Read message");

                setData(msg.getData().getByteArray("data"));
                break;

            case MESSAGE_DEVICE_NAME:
                // save the connected device's name
                mConnectedDeviceName = msg.getData().getString(DEVICE_NAME);
                Toast.makeText(getApplicationContext(), "Connected to "
                        + mConnectedDeviceName, Toast.LENGTH_SHORT).show();
                break;

            case MESSAGE_TOAST:
                Toast.makeText(getApplicationContext(), msg.getData().getString(
                    TOAST),
                        Toast.LENGTH_SHORT).show();
                break;
            default:
                super.handleMessage(msg);
            }
        }
    }


    private ServiceConnection mConnection = new ServiceConnection() {

        public void onServiceConnected(ComponentName className, IBinder service)
            {
            mService = new Messenger(service);
            if (D) Log.d(TAG,"Attached");
            try {
                Message msg = Message.obtain(null, BluetoothService.
                    MSG_REGISTER_CLIENT);
                msg.replyTo = mMessenger;
                mService.send(msg);
            } catch (RemoteException e) {
                // In this case the service has crashed before we could even do
                    anything with it
            }
        }

        public void onServiceDisconnected(ComponentName className) {
            // This is called when the connection with the service has been
                unexpectedly disconnected - process crashed.
            mService = null;
            if (D) Log.d(TAG,"Disconnected");
        }
    };

    /**
     *Unbind to the BluetoothService.
     */
    private void doBindService() {
```

```
        bindService(new Intent(this, BluetoothService.class), mConnection,
            Context.BIND_AUTO_CREATE);
        mIsBound = true;
        if (D) Log.d(TAG,"Binding");
    }
    /**
     *Unbind form the BluetoothService.
     */
    private void doUnbindService() {
        if (mIsBound) {
            // If we have received the service, and hence registered with it,
                then now is the time to unregister.
            if (mService != null) {
                try {
                    Message msg = Message.obtain(null, BluetoothService.
                        MSG_UNREGISTER_CLIENT);
                    msg.replyTo = mMessenger;
                    mService.send(msg);
                } catch (RemoteException e) {}
            }
            // Detach existing connection.
            unbindService(mConnection);
            mIsBound = false;
            if (D) Log.d(TAG,"Unbinding");
        }
    }

    /**
     * Initialize the contents of the Activity's standard options menu.
     */
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        MenuInflater inflater = getMenuInflater();
        inflater.inflate(R.menu.main_activity_menu,  menu);
        return (super.onPrepareOptionsMenu(menu));
    }
    /**
     * Listener that responds to selections made in the option menu.
     */
    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        // Handle item selection
        switch (item.getItemId()) {

        case R.id.bluetoothconnect:
            initBT();
            return true;


        case R.id.bluetoothdisconnect:
            if (mService != null) {
                try {
                    Message msg = Message.obtain(null, BluetoothService.
                        STATE_DISCONNECTED);
                    mService.send(msg);
                } catch (RemoteException e) {
                    e.printStackTrace();
                }
            }
            try {
                doUnbindService();
            } catch (Throwable t) {
                if (D) Log.e(TAG, "Failed to unbind from the service", t);
            }

            //stop the log timer
            stopTimer();
            return true;
```

```
        case R.id.exit:
            stopService(new Intent(MainActivity.this, BluetoothService.class));
            //stop the log timer
            stopTimer();
            this.finish();

            return true;

        default:
            return super.onOptionsItemSelected(item);
        }
    }

    /**
     * Listener for on display Buttons.
     * */
    public void onClick(View v) {

        switch (v.getId())  {

        case R.id.autozoom :
            mPlotView.autoZoom();
            Toast.makeText(getBaseContext(), "AutoZoom", Toast.LENGTH_SHORT).show
                ();
            break;

        case R.id.startlog :


            if(((ToggleButton)v).isChecked()){
                //Start timer and tell the BluetoothService to start logging
                    incoming data.
                Toast.makeText(getBaseContext(), "Start Log" , Toast.LENGTH_SHORT
                    ).show();
                startTimer();
                if (mService != null) {
                    try {
                        Message msg = Message.obtain(null, BluetoothService.
                            MSG_LOG_ON);
                        msg.replyTo = mMessenger;
                        mService.send(msg);
                    } catch (RemoteException e) {}
                }
            }else{
                // Stop timer and tell the BluetoothService to stop logging
                    incoming data.
                Toast.makeText(getBaseContext(), "Stop log", Toast.LENGTH_SHORT).
                    show();
                stopTimer();
                if (mService != null) {
                    try {
                        Message msg = Message.obtain(null, BluetoothService.
                            MSG_LOG_OFF);
                        msg.replyTo = mMessenger;
                        mService.send(msg);
                    } catch (RemoteException e) {}
                }
            }
            break;

        }

    }
    /**
     * Creates a Runnable that operates the log timer in a separate thread.
     */
```

```
    private Runnable mUpdateTimeTask = new Runnable(){
        public void run() {

            final long start = startLogTime;
            long millis = SystemClock.uptimeMillis() - start;
            int seconds = (int) (millis / 1000);
            int minutes = seconds / 60;
            seconds     = seconds % 60;
            if (seconds < 10) {
                mTextViewLog.setText("" + minutes + ":0" + seconds);
                Log.d("PlotActivity","" + minutes + ":0" + seconds);
            } else {
                mTextViewLog.setText("" + minutes + ":" + seconds);
                Log.d("PlotActivity","" + minutes + ":" + seconds);
            }
            timeHandler.postAtTime(this,start + (((minutes * 60) + seconds + 1) *
                 1000));

        }
    };

    /**
     * Starts the Runnable timer.
     */
    private void startTimer(){
        if (startLogTime == 0L) {
            startLogTime =  SystemClock.uptimeMillis();
            timeHandler.removeCallbacks(mUpdateTimeTask);
            timeHandler.postDelayed(mUpdateTimeTask, 100);

        }
    }
    /**
     * Stops the runnable timer.
     */
    private void stopTimer(){
        startLogTime = 0L;
        timeHandler.removeCallbacks(mUpdateTimeTask);
        mTextViewLog.setText("");
    }


}
```

Listing: HelpActivity.java

```
package com.BioDataLogger;

import android.app.Activity;
import android.os.Bundle;
import android.view.Window;
/**
 * This Activity displays help information.
 * @author Lars Jørgen Aamodt
 */
public class HelpActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        requestWindowFeature(Window.FEATURE_NO_TITLE);
        setContentView(R.layout.help);
    }
}
```

Listing: DeviceListActivity.java

```
package com.BioDataLogger;
/*
 * Copyright (C) 2009 The Android Open Source Project
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */



import java.util.Set;

import android.app.Activity;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.view.View.OnClickListener;
import android.view.Window;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.ListView;
import android.widget.TextView;

/**
 * This Activity appears as a dialog. It lists any paired devices and
 * devices detected in the area after discovery. When a device is chosen
 * by the user, the MAC address of the device is sent back to the parent
 * Activity in the result Intent.
 */
public class DeviceListActivity extends Activity {
    // Debugging
    private static final String TAG = "DeviceListActivity";
    private static final boolean D = false;

    // Return Intent extra
    public static String EXTRA_DEVICE_ADDRESS = "device_address";

    // Member fields
    private BluetoothAdapter mBtAdapter;
    private ArrayAdapter<String> mPairedDevicesArrayAdapter;
    private ArrayAdapter<String> mNewDevicesArrayAdapter;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        // Setup the window
        requestWindowFeature(Window.FEATURE_INDETERMINATE_PROGRESS);
        setContentView(R.layout.device_list);
```

```
        // Set result CANCELED in case the user backs out
        setResult(Activity.RESULT_CANCELED);

        // Initialize the button to perform device discovery
        Button scanButton = (Button) findViewById(R.id.button_scan);
        scanButton.setOnClickListener(new OnClickListener() {
            public void onClick(View v) {
                doDiscovery();
                v.setVisibility(View.GONE);
            }
        });

        // Initialize array adapters. One for already paired devices and
        // one for newly discovered devices
        mPairedDevicesArrayAdapter = new ArrayAdapter<String>(this, R.layout.
            device_name);
        mNewDevicesArrayAdapter = new ArrayAdapter<String>(this, R.layout.
            device_name);

        // Find and set up the ListView for paired devices
        ListView pairedListView = (ListView) findViewById(R.id.paired_devices);
        pairedListView.setAdapter(mPairedDevicesArrayAdapter);
        pairedListView.setOnItemClickListener(mDeviceClickListener);

        // Find and set up the ListView for newly discovered devices
        ListView newDevicesListView = (ListView) findViewById(R.id.new_devices);
        newDevicesListView.setAdapter(mNewDevicesArrayAdapter);
        newDevicesListView.setOnItemClickListener(mDeviceClickListener);

        // Register for broadcasts when a device is discovered
        IntentFilter filter = new IntentFilter(BluetoothDevice.ACTION_FOUND);
        this.registerReceiver(mReceiver, filter);

        // Register for broadcasts when discovery has finished
        filter = new IntentFilter(BluetoothAdapter.ACTION_DISCOVERY_FINISHED);
        this.registerReceiver(mReceiver, filter);

        // Get the local Bluetooth adapter
        mBtAdapter = BluetoothAdapter.getDefaultAdapter();

        // Get a set of currently paired devices
        Set<BluetoothDevice> pairedDevices = mBtAdapter.getBondedDevices();

        // If there are paired devices, add each one to the ArrayAdapter
        if (pairedDevices.size() > 0) {
            findViewById(R.id.title_paired_devices).setVisibility(View.VISIBLE);
            for (BluetoothDevice device : pairedDevices) {
                mPairedDevicesArrayAdapter.add(device.getName() + "\n" + device.
                    getAddress());
            }
        } else {
            String noDevices = getResources().getText(R.string.none_paired).
                toString();
            mPairedDevicesArrayAdapter.add(noDevices);
        }
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();

        // Make sure we're not doing discovery anymore
        if (mBtAdapter != null) {
            mBtAdapter.cancelDiscovery();
        }

        // Unregister broadcast listeners
        this.unregisterReceiver(mReceiver);
```

```
    }

    /**
     * Start device discover with the BluetoothAdapter
     */
    private void doDiscovery() {
        if (D) Log.d(TAG, "doDiscovery()");

        // Indicate scanning in the title
        setProgressBarIndeterminateVisibility(true);
        setTitle(R.string.scanning);

        // Turn on sub-title for new devices
        findViewById(R.id.title_new_devices).setVisibility(View.VISIBLE);

        // If we're already discovering, stop it
        if (mBtAdapter.isDiscovering()) {
            mBtAdapter.cancelDiscovery();
        }

        // Request discover from BluetoothAdapter
        mBtAdapter.startDiscovery();
    }

    // The on-click listener for all devices in the ListViews
    private OnItemClickListener mDeviceClickListener = new OnItemClickListener()
        {
        public void onItemClick(AdapterView<?> av, View v, int arg2, long arg3) {
            // Cancel discovery because it's costly and we're about to connect
            mBtAdapter.cancelDiscovery();

            // Get the device MAC address, which is the last 17 chars in the View
            String info = ((TextView) v).getText().toString();

            // prevent the return of "No devices have been paired" and "No
                devices found" bug
            if (!info.equals("No devices have been paired") ){
                if (!info.equals("No devices found") ){
                    String address = info.substring(info.length() - 17);
                    // Create the result Intent and include the MAC address
                    Intent intent = new Intent();
                    intent.putExtra(EXTRA_DEVICE_ADDRESS, address);

                    // Set result and finish this Activity
                    setResult(Activity.RESULT_OK, intent);
                }
            }
            finish();
        }
    };

    // The BroadcastReceiver that listens for discovered devices and
    // changes the title when discovery is finished
    private final BroadcastReceiver mReceiver = new BroadcastReceiver() {
        @Override
        public void onReceive(Context context, Intent intent) {
            String action = intent.getAction();

            // When discovery finds a device
            if (BluetoothDevice.ACTION_FOUND.equals(action)) {
                // Get the BluetoothDevice object from the Intent
                BluetoothDevice device = intent.getParcelableExtra(
                    BluetoothDevice.EXTRA_DEVICE);
                // If it's already paired, skip it, because it's been listed
                    already
                if (device.getBondState() != BluetoothDevice.BOND_BONDED) {
                    mNewDevicesArrayAdapter.add(device.getName() + "\n" + device.
                        getAddress());
```

```
                }
                // When discovery is finished, change the Activity title
            } else if (BluetoothAdapter.ACTION_DISCOVERY_FINISHED.equals(action))
                {
                setProgressBarIndeterminateVisibility(false);
                setTitle(R.string.select_device);
                if (mNewDevicesArrayAdapter.getCount() == 0) {
                    String noDevices = getResources().getText(R.string.none_found
                        ).toString();
                    mNewDevicesArrayAdapter.add(noDevices);
                }
            }
        }
    };

}
```

Listing: BluetoothService.java

```
package com.BioDataLogger;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.util.UUID;

import android.app.Notification;
import android.app.NotificationManager;
import android.app.PendingIntent;
import android.app.Service;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.bluetooth.BluetoothSocket;
import android.content.Intent;
import android.os.Bundle;
import android.os.Environment;
import android.os.Handler;
import android.os.IBinder;
import android.os.Message;
import android.os.Messenger;
import android.os.RemoteException;
import android.util.Log;
/**
 * This Service initiate and controls  the Bluetooth connection.
 * It receives data over Bluetooth, stores it in a queue, and sends it to the
     MainActivity.
 * @author Lars Jørgen Aamodt
 */
public class BluetoothService extends Service {

    private static final boolean D = true;
    private static final String TAG = "BluetoothService";

    // the bluetooth address from client
    private String mAddress;

    private BluetoothQueue mBluetoothQueue;

    //For showing and hiding notification.
    private NotificationManager mNM;

    //currently registered client
    private Messenger mClient = null;
    private BluetoothDevice mDevice = null;
    private BluetoothAdapter mBluetoothAdapter = null;
```

```
// Unique UUID for this application
private static final UUID MY_UUID = UUID.fromString("
    00001101-0000-1000-8000-00805F9B34FB");

// Member fields
private ConnectThread mConnectThread;
private ConnectedThread mConnectedThread;
private SendDataThread mSendDataThread;
private int mState;

private boolean logState = false;

// Constants that indicate the current service state
public static final int MSG_REGISTER_CLIENT = 1;
public static final int MSG_UNREGISTER_CLIENT = 2;
public static final int MSG_DEVICE_ADDRESS = 3;



// Constants that indicate the current connection state
public static final int STATE_NONE = 1;
public static final int STATE_LISTEN = 2;
public static final int STATE_CONNECTING = 3;
public static final int STATE_CONNECTED = 4;
public static final int STATE_DISCONNECTED = 5;
public static final int MESSAGE_READ = 6;

// Constants that indicate the current log state
public static final int MSG_LOG_ON = 7;
public static final int MSG_LOG_OFF = 8;

// Constant that indicate queue sync
public static final int MSG_SYNC_QUEUE =9;

//Target we publish for clients to send messages to IncomingHandler.
private final Messenger mMessenger = new Messenger(new IncomingHandler());

/**
 *Return the communication channel to the service.
 *May return null if clients can not bind to the service.
 */
@Override
public IBinder onBind(Intent arg0) {
    return mMessenger.getBinder();
}

@Override
public void onCreate() {
    super.onCreate();
    if (D) Log.i(TAG, "Service create");

    android.os.Process.setThreadPriority(android.os.Process.
        THREAD_PRIORITY_URGENT_AUDIO);

    mNM = (NotificationManager)getSystemService(NOTIFICATION_SERVICE);
    // Display notification
    showNotification();

    mBluetoothQueue = new BluetoothQueue();
}

/**
 * Called by the system every time a client explicitly starts the service by
     calling startService(Intent)
 */
@Override
public int onStartCommand(Intent intent, int flags, int startId) {
```

```
        if (D) Log.i(TAG, "onStartCommand");
        return START_NOT_STICKY;
    }

    /**
     * Called by the system to notify a Service that it is no longer used and is
         being removed.
     */
    @Override
    public void onDestroy() {
        super.onDestroy();
        if (D) Log.i(TAG, "Service destroy");
        stop();
        mBluetoothQueue.clearAll();
        // Cancel the persistent notification.
        mNM.cancel(R.string.remote_service_started);
    }

    /**
     *  Set the current state of the bluetooth connection.
     */
    private synchronized void setState(int state) {
        if (D) Log.i(TAG, "setState() " + mState + " -> " + state);
        mState = state;

        // Give the new state to the Handler so the UI Activity can update
        if (mClient != null){
            try {
                mClient.send(Message.obtain(null, MainActivity.
                    MESSAGE_STATE_CHANGE, state, -1));

            } catch (RemoteException e) {
                // The client is dead.
                mClient=null;
            }
        }
    }

    /**
     * Return the current connection state.
     */
    public synchronized int getState() {
        return mState;
    }

    /**
     * Start the ConnectThread to initiate a connection to a remote device.
     * @param device  The BluetoothDevice to connect
     */
    public synchronized void connect(BluetoothDevice device) {
        if (D) Log.i(TAG, "Connect to: " + device);

        // Cancel any thread attempting to make a connection
        if (mState == STATE_CONNECTING) {
            if (mConnectThread != null) {
                mConnectThread.cancel();
                mConnectThread = null;
            }
        }

        // Cancel any thread currently running a connection
        if (mConnectedThread != null) {
            //mConnectedThread.cancel();
            mConnectedThread.interrupt();
            mConnectedThread = null;
        }

        // Start the thread to connect with the given device
```

```
        mConnectThread = new ConnectThread(device);
        mConnectThread.start();
        //clear queue
        mBluetoothQueue.clearAll();
        setState(STATE_CONNECTING);
    }

    /**
     * Start the ConnectedThread to begin managing a Bluetooth connection
     * @param socket  The BluetoothSocket on which the connection was made
     * @param device  The BluetoothDevice that has been connected
     */
    public synchronized void connected(BluetoothSocket socket, BluetoothDevice
        device) {
        if (D) Log.i(TAG, "Connected");

        // Cancel the thread that completed the connection
        if (mConnectThread != null) {
            mConnectThread.cancel();
            mConnectThread = null;
        }

        // Cancel any thread currently running a connection
        if (mConnectedThread != null) {
            mConnectedThread.cancel();
            mConnectedThread.interrupt();
            mConnectedThread = null;
        }
        // Cancel any thread currently sending data to UI Activity
        if (mSendDataThread != null) {
            mSendDataThread.interrupt();
            mSendDataThread = null;
        }


        // Start the thread to manage the connection and perform transmissions
        mConnectedThread = new ConnectedThread(socket);
        mConnectedThread.start();

        // Start the thread to manage data transmissions to UI Activity
        mSendDataThread = new SendDataThread();
        mSendDataThread.start();

        Bundle b = new Bundle();

        // Send the name of the connected device back to the UI Activity
        if (mClient != null){
            try {
                b.putString(MainActivity.DEVICE_NAME, device.getName());
                Message msg = Message.obtain(null, MainActivity.
                    MESSAGE_DEVICE_NAME);
                msg.setData(b);
                mClient.send(msg);
                b.clear();
            } catch (RemoteException e) {}
            setState(STATE_CONNECTED);
        }
    }

    /**
     * Stop all threads
     */
    public synchronized void stop() {
        if (D) Log.i(TAG, "Stoping alle threads!");

        if (mConnectThread != null) {
            mConnectThread.cancel();
            mConnectThread = null;
```

```
        }

        if (mConnectedThread != null) {
            mConnectedThread.cancel();
            mConnectedThread.interrupt();
            mConnectedThread = null;
        }

        if (mSendDataThread != null) {
            mSendDataThread.interrupt();
            mSendDataThread = null;
        }
        setState(STATE_NONE);
    }

    /**
     * Indicate that the connection attempt failed and notify the UI Activity.
     */
    private void connectionFailed() {
        if (D) Log.e(TAG,"Connection Faild");
        Bundle b = new Bundle();
        if (mClient != null){
            try {
                // Send a failure message back to the Activity
                b.putString(MainActivity.TOAST, "Unable to connect device");
                Message msg = Message.obtain(null, MainActivity.MESSAGE_TOAST);
                msg.setData(b);
                mClient.send(msg);
                b.clear();
            } catch (RemoteException e) {}

            setState(STATE_DISCONNECTED);
        }
    }

    /**
     * Indicate that the connection was lost and notify the UI Activity.
     */
    private void connectionLost() {
        if (D) Log.e(TAG,"Connection Lost");
        Bundle b = new Bundle();
        try {
            if (mClient != null){
                // Send a failure message back to the UI Activity
                b.putString(MainActivity.TOAST, "Device connection was lost");
                Message msg = Message.obtain(null, MainActivity.MESSAGE_TOAST);
                msg.setData(b);
                mClient.send(msg);
                b.clear();
            }
        } catch (RemoteException e) {}
        // stop all threads
        stop();
        setState(STATE_DISCONNECTED);
    }

    /**
     * This thread runs while attempting to make an outgoing connection
     * with a device. It runs straight through; the connection either
     * succeeds or fails.
     */
    private class ConnectThread extends Thread {
        private BluetoothSocket mmSocket = null;
        private BluetoothDevice mmDevice = null;


        public ConnectThread(BluetoothDevice device) {
            mmDevice = device;
```

```
            BluetoothSocket tmp = null;


            // Get a BluetoothSocket for a connection with the given
                BluetoothDevice
            try {
                tmp = device.createRfcommSocketToServiceRecord(MY_UUID);

            } catch (IOException e) {
                if (D) Log.e(TAG,  "create() failed", e);
            }
            mmSocket = tmp;
        }

        public void run() {
            if (D) Log.i(TAG, "BEGIN mConnectThread" );
            setName("ConnectThread" );

            // Cancel discovery because it will slow down a connection
            mBluetoothAdapter.cancelDiscovery();

            // Make a connection to the BluetoothSocket
            try {
                // This is a blocking call and will only return on a
                // successful connection or an exception
                mmSocket.connect();
            } catch (IOException e) {
                // Close the socket
                try {
                    mmSocket.close();
                } catch (IOException e2) {
                    if (D) Log.e(TAG, "unable to close() socket during connection
                        failure", e2);
                }
                connectionFailed();
                return;
            }

            // Reset the ConnectThread because we're done
            synchronized (BluetoothService.this) {
                mConnectThread = null;
            }

            // Start the connected thread
            connected(mmSocket, mmDevice);
        }

        public void cancel() {
            try {
                mmSocket.close();
            } catch (IOException e) {
                if (D) Log.e(TAG, "close() of connect socket failed", e);
            }
        }
    }

    /**
     * This thread runs during a connection with a remote device.
     * It handles all incoming transmissions and data logging to SDcard.
     */
    private class ConnectedThread extends Thread {
        private BluetoothSocket mmSocket = null;
        private InputStream mmInStream = null;
        private FileOutputStream fOut = null;
```

```
public ConnectedThread(BluetoothSocket socket) {
    if (D) Log.i(TAG, "Create ConnectedThread: " );
    mmSocket = socket;
    InputStream tmpIn = null;

    this.setPriority(Thread.MAX_PRIORITY);

    // Get the BluetoothSocket input stream
    try {
        tmpIn = socket.getInputStream();

    } catch (IOException e) {
        e.printStackTrace();
        if (D) Log.e(TAG, "temp sockets not created", e);
    }
    mmInStream = tmpIn;

    // make data file on SDcard
    File sdCard = Environment.getExternalStorageDirectory();
    File dir = new File (sdCard.getAbsolutePath() + "/Data");
    dir.mkdirs();
    File file = new File(dir, "/data.txt");


    // Get the fileOutputStream
    try {
        fOut = new FileOutputStream(file);
    } catch (FileNotFoundException e1) {
        e1.printStackTrace();
        if (D) Log.e(TAG, "FileOutputStream not created", e1);
    }
}

public void run() {
    if (D) Log.i(TAG, "BEGIN mConnectedThread");
    // Temporary buffer for InputStream
    byte[] buffer = new byte[3072];
    int bytes= 0;
    byte[] bufferClone;


    // Keep listening to the InputStream while connected
    while (!isInterrupted()) {
        try {

            // Read from the InputStream
            bytes = mmInStream.read(buffer);

            bufferClone = buffer.clone();

            // insert data in queue
            mBluetoothQueue.insert(bufferClone, bytes);

            // if logging activated, write to file on SDcard
            if (logState && fOut != null){
                fOut.write(bufferClone,0,bytes);
            }


        } catch (IOException e) {
            //if (D) Log.e(TAG, "disconnected", e);
            //e.printStackTrace();
            connectionLost();

        }
    }
```

```
            if(isInterrupted()){
                //clear the queue
                mBluetoothQueue.clearAll();
                //indicate process ended successfully
                if (D) Log.i(TAG, "mConnectedThread ended successfully");

            }//else{}
        }

        /**
         * Close stream and socket.
         */
        public void cancel() {
            if (D) Log.d(TAG, "mConnectedThread.cancel()");

            if (mmSocket != null){
                try {
                    mmSocket.close();
                    mmSocket = null;
                } catch (IOException e) {
                    e.printStackTrace();
                    if (D) Log.e(TAG, "close() of connect socket failed", e);
                }
            }

            if (fOut!= null){
                try {
                    //fOut.flush();
                    fOut.close();
                    fOut = null;
                } catch (IOException e) {
                    e.printStackTrace();
                    if (D) Log.e(TAG, "close() of file failed", e);
                }
            }
        }
    }

    /**
     * This thread runs during a connection with a remote device.
     * It handles transfer of data to the UI Activity.
     */
    private class SendDataThread extends Thread {



        // Constructor
        public SendDataThread() {
            if (D) Log.i(TAG, "Create SendDataThread: " );
            this.setPriority(Thread.MAX_PRIORITY);
        }

        public void run() {
            if (D) Log.i(TAG, "BEGIN  mSendDataThread");
            byte[] data;
            Bundle b = new Bundle();
            Message msg;
            //while data ready
            while (!isInterrupted()) {

                //send data to UI Activity
                try {
                    if((data = mBluetoothQueue.retrieve()) != null){

                        if (mClient != null){
                            try {
                                b.putByteArray("data", data);
                                msg = Message.obtain(null, MainActivity.
```

```
                                    MESSAGE_READ);
                            msg.setData(b);
                            mClient.send(msg);
                            b.clear();
                        } catch (RemoteException e) {
                            e.printStackTrace();
                        }
                    }
                }
            } catch (InterruptedException e) {
                // Restore the interrupted status
                Thread.currentThread().interrupt();
            }
        }
        if(isInterrupted()){
            //cleanup
            //indicate process ended successfully
            if (D) Log.i(TAG, "mSendDataThread ended successfully");

        }//else{}

    }

}

/**
 * Handler of incoming messages from client.
 */
private class IncomingHandler extends Handler {
    @Override
    public void handleMessage(Message msg) {
        switch (msg.what) {
        case MSG_REGISTER_CLIENT:
            mClient=msg.replyTo;
            break;

        case MSG_UNREGISTER_CLIENT:
            mClient=null;
            break;

        case MSG_DEVICE_ADDRESS:
            mAddress = msg.getData().getString("str");
            // Get the BluetoothAdapter
            mBluetoothAdapter = BluetoothAdapter.getDefaultAdapter();
            // Get the BluetoothDevice object
            mDevice = mBluetoothAdapter.getRemoteDevice(mAddress);
            if (D) Log.d(TAG, "Device: " + mDevice);
            connect(mDevice);
            break;

        case STATE_DISCONNECTED:
            if (D) Log.d(TAG, "DISCONNECTED");
            // stop all threads
            stop();
            break;

        case MSG_LOG_ON:
            if (D) Log.d(TAG, "LOG ON");
            // start logging to file
            logState = true;

            break;
        case MSG_LOG_OFF:
            if (D) Log.d(TAG, "LOG OFF");
            // stop logging to file
            logState = false;
            break;
```

```
            case MSG_SYNC_QUEUE:
                //if (D) Log.d(TAG, "SYNC QUEUE");
                mBluetoothQueue.sync();
                break;

            default:
                super.handleMessage(msg);
            }
        }
    }

    /**
     * Show a notification while this service is running.
     */
    private void showNotification() {
        // Notification text
        CharSequence text = getText(R.string.remote_service_started);

        // Set the icon, scrolling text and timestamp
        Notification notification = new Notification(R.drawable.
            single_logo_launcher, text, System.currentTimeMillis());

        notification.flags |= Notification.FLAG_ONGOING_EVENT;
        notification.flags |= Notification.FLAG_NO_CLEAR;

        // The PendingIntent to launch PlotActivity if the user selects this
            notification
        PendingIntent contentIntent = PendingIntent.getActivity(this, 0, new
            Intent(this, MainActivity.class), 0);

        // Set the info for the views that show in the notification panel.
        notification.setLatestEventInfo(this, getText(R.string.service_label),
            text, contentIntent);

        // Send the notification.
        // We use a string id because it is a unique number.  We use it later to
            cancel.
        mNM.notify(R.string.remote_service_started, notification);
    }
}
```

Listing: BluetoothQueue.java

```
package com.BioDataLogger;

import java.util.concurrent.ArrayBlockingQueue;
import java.util.concurrent.atomic.AtomicBoolean;

import android.util.Log;
/**
 * This Class creates a FIFO queue for incoming data from BluetoothService.
 * @author Lars Jørgen Aamodt
 */
public class BluetoothQueue {

    // test variables
    private static final String TAG = "BluetoothQueue";
    private static final boolean D = false;

    //Data queue object
    private static  ArrayBlockingQueue<Byte> queue;
    private byte[] data;
    private Byte tmp = 0;
    final AtomicBoolean syncON = new AtomicBoolean(false);
    /**
     * Constructor
     */
```

```
public BluetoothQueue(){
    if (D) Log.i(TAG, "create BluetoothQueue" );
    queue = new ArrayBlockingQueue<Byte>(3072);
    data = new byte[5];

}

/**
 * Inserts the specified element at the tail of this queue if it is possible.
 * @param b Byte array of data.
 * @param i valid data in data array.
 */
public void insert(byte[] b,int i ) {

    for (int n = 0; n<i ; n++ ){
        queue.offer(b[n]);
    }
    //if (D) Log.d(TAG, "Queue size : " + queue.size());
}


/**
 *  Gets data form the queue and returns the correct amount.
 * @return byte Array of data or null if data not ready.
 * @throws InterruptedException
 */
public byte[] retrieve() throws InterruptedException {
    //if not in sync mode
    if (!syncON.get()){
        //try to take correct amount of data from queue
        for(int i=0; i < 5; i++){

            tmp = queue.take();

            data[i]= tmp;

        }

        return data;
    }else
        return null;

}

/**
 * Removes all of the elements form the queue
 */
public void clearAll(){
    //if (D) Log.i(TAG, "BluetoothQueue clear!");
    queue.clear();

}

/**
 * Stops retrieve and removes element form the queue in order to synchronize
     data flow.
 */
public void sync(){
    // Stop retrieve
    syncON.set(true);

    // while retrieve not running
    while( syncON.get() ){
        // if elements in queue
        if (queue.size() > 10){
            // if not on new data marker (.)
            if ( (queue.peek().intValue())  != 0){
                // remove element and try again
```

```
                    queue.poll();

                }else {
                    // if on new data marker, remove marker and set sycnON to
                        false so retrieve can start working
                    queue.poll();
                    syncON.set(false);

                }
            }
        }
    }


    /**
     * @return The size of the queue.
     */
    public int getSize() {
        return queue.size();
    }

    //if (D) Log.e(TAG, "syncOFF");
    //if (D) Log.e(TAG, "Queue data : " + queue.toString());
    //if (D) Log.e(TAG, "Queue data : " + queue.toString());
    //if (D) Log.e(TAG, "Peek: " + (char) (queue.peek().byteValue()));
    //if (D) Log.e(TAG, "syncON");
    //if (D) Log.i(TAG, "BluetoothQueue sync on! "+ queue.size()+" syncON = "+
        syncON);
    //if (D) Log.d(TAG, "Queue data : " + queue.toString());
    //if (D) Log.d(TAG, "Inserting");
    //if (D) Log.d(TAG, "Insert data : " +"("+new String(b, 0,i)+")"+i);
}
```

Listing: AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.BioDataLogger"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="8" />

    <uses-permission android:name="android.permission.BLUETOOTH" />
    <uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <uses-permission id="android.permission.RAISED_THREAD_PRIORITY"/>


    <application
        android:debuggable="true"
        android:icon="@drawable/logo_launcher"
        android:label="@string/app_name" >
        <activity
            android:label="@string/app_name"
            android:name=".StartActivity"
            android:screenOrientation="landscape" >
            <intent-filter >
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity
            android:theme="@style/Theme.NoBackground"
            android:launchMode="singleTask"
            android:name=".MainActivity"
```

```
            android:screenOrientation="landscape" >
        </activity>
        <activity
            android:name=".HelpActivity"
            android:screenOrientation="landscape" >
        </activity>
        <activity
            android:name=".DeviceListActivity"
            android:configChanges="orientation|keyboardHidden"
            android:label="@string/select_device"
            android:theme="@android:style/Theme.Dialog" >
        </activity>

        <service
            android:name="com.BioDataLogger.BluetoothService"
            android:process=":remote" >
        </service>
    </application>

</manifest>
```

Listing: Adevice_list.xml

```
<?xml version="1.0" encoding="utf-8"?>
<!-- Copyright (C) 2009 The Android Open Source Project

     Licensed under the Apache License, Version 2.0 (the "License");
     you may not use this file except in compliance with the License.
     You may obtain a copy of the License at

          http://www.apache.org/licenses/LICENSE-2.0

     Unless required by applicable law or agreed to in writing, software
     distributed under the License is distributed on an "AS IS" BASIS,
     WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
     See the License for the specific language governing permissions and
     limitations under the License.
-->
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    >
    <TextView android:id="@+id/title_paired_devices"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/title_paired_devices"
        android:visibility="gone"
        android:background="#666"
        android:textColor="#fff"
        android:paddingLeft="5dp"
    />
    <ListView android:id="@+id/paired_devices"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:stackFromBottom="true"
        android:layout_weight="1"
    />
    <TextView android:id="@+id/title_new_devices"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/title_other_devices"
        android:visibility="gone"
        android:background="#666"
        android:textColor="#fff"
        android:paddingLeft="5dp"
    />
```

```
    <ListView android:id="@+id/new_devices"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:stackFromBottom="true"
        android:layout_weight="2"
    />
    <Button android:id="@+id/button_scan"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/button_scan"
    />
</LinearLayout>
```

Listing: device_name.xml

```
<?xml version="1.0" encoding="utf-8"?>
<!-- Copyright (C) 2009 The Android Open Source Project

     Licensed under the Apache License, Version 2.0 (the "License");
     you may not use this file except in compliance with the License.
     You may obtain a copy of the License at

          http://www.apache.org/licenses/LICENSE-2.0

     Unless required by applicable law or agreed to in writing, software
     distributed under the License is distributed on an "AS IS" BASIS,
     WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
     See the License for the specific language governing permissions and
     limitations under the License.
-->
<TextView xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:textSize="18sp"
    android:padding="5dp"
/>
```

Listing: help.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:background="@drawable/background"
    android:orientation="vertical" >

    <ImageView
        android:id="@+id/logo"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal|top"
        android:src="@drawable/logo" />


    <TextView
        xmlns:android="http://schemas.android.com/apk/res/android"
        android:id="@+id/textview"
        android:layout_width="match_parent"
        android:layout_height="117dp"
        android:layout_alignParentBottom="true"
        android:gravity="center_horizontal"
        android:text="@string/info2" />

</LinearLayout>
```

Listing: main_activity_menu.xml

```
    <menu xmlns:android="http://schemas.android.com/apk/res/android">

     <item android:id="@+id/bluetoothconnect"
          android:title="Connect Bluetooth"/>

     <item android:id="@+id/bluetoothdisconnect"
          android:title="Disconnect Bluetooth"/>


     <item android:id="@+id/exit"
          android:title="Exit">

     </item>
</menu>
```

Listing: plot.xml

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
     android:id="@+id/layout_main"
     android:layout_width="fill_parent"
     android:layout_height="fill_parent" >

     <com.BioDataLogger.PlotView
          android:id="@+id/plotview"
          android:layout_width="wrap_content"
          android:layout_height="wrap_content" />

     <TextView
          android:id="@+id/connectText"
          android:text="Not connected"
          android:textColor="#ff0000"
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:layout_gravity="right|top"
          android:gravity="center_horizontal"
          android:padding="5dp"
          android:textSize="15dp" />

     <TextView
          android:id="@+id/logText"
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:layout_gravity="left|top"
          android:gravity="center_horizontal"
          android:padding="5dp"
          android:textSize="15dp" />

     <TextView
          android:id="@+id/plotText_1"
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:paddingLeft="490dp"
          android:paddingTop="53dp"
          android:textSize="30dp" />

     <TextView
          android:id="@+id/plotText_2"
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:paddingLeft="490dp"
          android:paddingTop="200dp"
          android:textSize="30dp" />

     <TextView
```

```
        android:id="@+id/errorText"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:gravity="center_vertical|center_horizontal"
        android:textSize="50dp"
        android:textStyle="bold" />

    <Button
        android:id="@+id/autozoom"
        style="?android:attr/buttonStyleSmall"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="left|bottom"
        android:text="AutoZoom" />

    <ToggleButton
        android:id="@+id/startlog"
        style="?android:attr/buttonStyleSmall"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="right|bottom"
        android:textOff="Start Log"
        android:textOn="Stop Log" />

</FrameLayout>
```

Listing: start.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:background="@drawable/background"
    android:orientation="vertical" >

    <ImageView
        android:id="@+id/logo"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal|top"
        android:paddingTop="30dp"
        android:src="@drawable/logo" />

    <TextView
        xmlns:android="http://schemas.android.com/apk/res/android"
        android:id="@+id/textview"
        android:layout_width="match_parent"
        android:paddingTop="20dp"
        android:layout_height="100dp"
        android:layout_alignParentBottom="true"
        android:gravity="center_horizontal"
        android:text="@string/info1" />

    <LinearLayout
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:gravity="bottom"
        android:orientation="horizontal" >

        <Button
            android:id="@+id/startPlot"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="Start" />

        <Button
```

```
            android:id="@+id/help"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="Help" />
    </LinearLayout>

</LinearLayout>
```

Listing: strings.xml

```
<?xml version="1.0" encoding="utf−8"?>
<resources>

    <string name="app_name">BioDataLogger</string>

    <!−− StartActivity −−>
    <string name="info1"></string>

    <!−− HelpActivity −−>
    <string name="info2">This application works as a display and data storage tool. For more information,
        contact Lars JÃ¸rgen Aamodt at: L_J_Aamodt@hotmail.com</string>

    <!−− BluetoothService −−>
    <string name="remote_service_started">Service started</string>
    <string name="service_label">BioDataLogger</string>

    <!−− BluetoothService −−>
    <string name="connected_to">Connected to: </string>


    <!−− DeviceListActivity −−>
    <string name="scanning">Scanning for devices...</string>
    <string name="select_device">Select a device to connect</string>
    <string name="none_paired">No devices have been paired</string>
    <string name="none_found">No devices found</string>
    <string name="title_paired_devices">Paired Devices</string>
    <string name="title_other_devices">Other Available Devices</string>
    <string name="button_scan">Scan for devices</string>

</resources>
```

Listing: theme.xml

```
<resources>
    <style name="Theme.NoBackground" parent="android:Theme">
        <item name="android:windowBackground">@null</item>
    </style>
</resources>
```

Appendix C

PythonDevelopmentTool UML and Code

C.1 UML

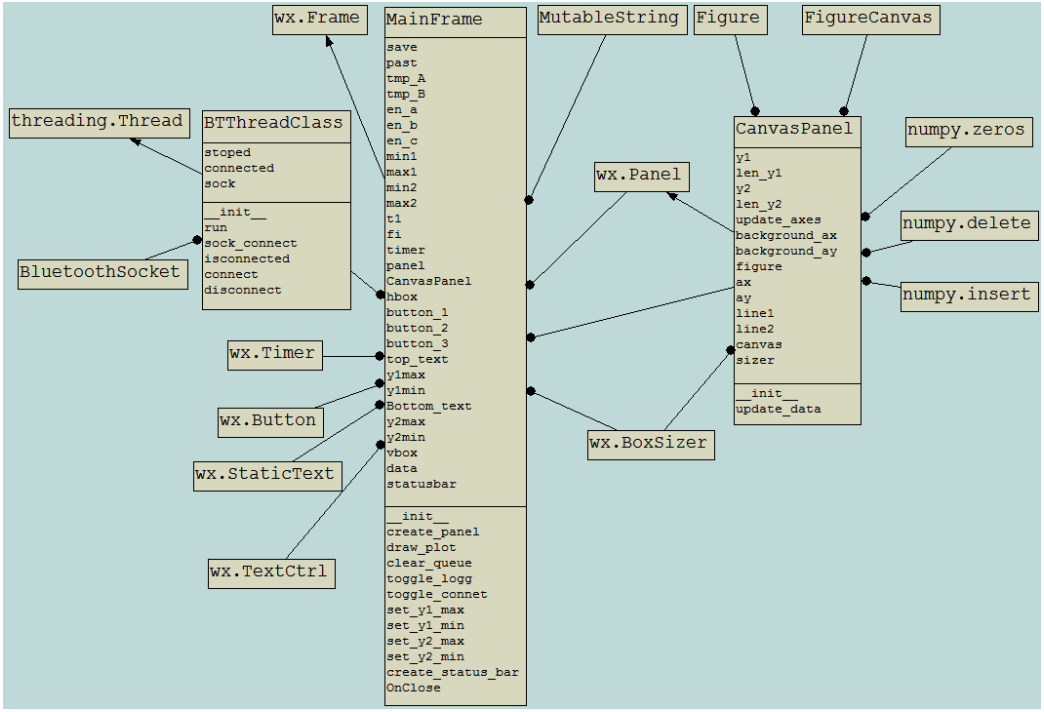


Figure C.1: UML diagram displaying the architecture of the Python application.

C.2 Code

Listing: PythonDevelopmentTool.py

```
#############################################################################
#File: PythonDevelopmentTool.py
#Project: FPGA Based Development Platform for Biomedical Measurements
#Author: Lars JÃ¸rgen Aamodt
#############################################################################
#!/usr/bin/python
import threading
import Queue
import os
import time
from bluetooth import * #PyBluez
# The recommended way to use wx with mpl is with the WXAgg
# backend.
import matplotlib
matplotlib.use('WXAgg')
matplotlib.interactive(True)
from matplotlib.figure import Figure
from matplotlib.backends.backend_wxagg import \
    FigureCanvasWxAgg as FigureCanvas
import numpy
import wx
from UserString import MutableString
q = Queue.Queue()

class BTThreadClass(threading.Thread):

    def __init__(self):
        threading.Thread.__init__(self)
        self.stoped = True
        self.connected = 0

    def run(self):


        while True:
            if not self.stoped:
                try:
                    data = self.sock.recv(512)
                    for i in data:
                        q.put(i,block = False)
                except IOError:
                    print "pafsdaf"
                    raise

    def sock_connect(self):
        addr = "00:06:66:43:0F:4B"

        #nearby_devices = discover_devices(lookup_names = True, flush_cache = ↩
            True, duration = 20)
        #print "############################"
        #print "found %d devices" % len(nearby_devices)
        #print "____________________________________"
        #for addr,name in nearby_devices:
        #    print " %s - %s" % (addr, name)



        # search for the SPP service
        uuid = "00001101-0000-1000-8000-00805F9B34FB"
        #service_matches = find_service( uuid = uuid, address = addr )
        service_matches = find_service( uuid = uuid)
```

```
        #print service_matches
        if len(service_matches) == 0:
            print "couldn't find the service"
            self.connected = 0
            return False

        first_match = service_matches[0]
        port = first_match["port"]
        name = first_match["name"]
        host = first_match["host"]

        print "connecting to \"%s\" on %s" % (name, host)
        self.connected = name

        # Create the client socket
        self.sock=BluetoothSocket( RFCOMM )
        self.sock.connect((host, port))
        return True

    def isconnected(self):
            return self.connected

    def connect(self):
        if self.sock_connect():
            self.stoped = False

    def disconnect(self):
        self.sock.close()
        self.connected = 0
        self.stoped = True
        self.join()



class CanvasPanel(wx.Panel):
    def __init__(self, parent):


        self.y1 = numpy.zeros(100)# y-array
        self.len_y1 = len(self.y1)

        self.y2 = numpy.zeros(100)# y-array
        self.len_y2 = len(self.y2)

        self.update_axes = False

        self.background_ax = ''
        self.background_ay = ''

        wx.Panel.__init__(self, parent)

        self.figure = Figure(figsize=(6, 5))#Figure(dpi = 100)


        self.ax = self.figure.add_subplot(211)
        self.ax.set_xlabel("X")
        self.ax.set_ylabel("Y")
        self.ax.set_title("PythonDevelopmentTool")
        self.ax.grid(True,color='black')
        #ax.set_autoscaley_on(True)
        self.ax.set_axis_bgcolor('slategray')


        self.ay = self.figure.add_subplot(212)
        self.ay.set_xlabel("X")
        self.ay.set_ylabel("Y")
        #ay.set_title("test")
        self.ay.grid(True,color='black')
```

```
        self.ay.set_axis_bgcolor('slategray')


        self.line1, = self.ax.plot(range(self.len_y1),self.y1,'yellow')
        self.line2, = self.ay.plot(range(self.len_y2),self.y2,'yellow')

        self.line1.axes.set_ylim(0,500)
        self.line1.axes.set_xlim(0,self.len_y1)
        self.line2.axes.set_ylim(0,500)
        self.line2.axes.set_xlim(0,self.len_y2)

        self.ax.set_autoscale_on(False)
        self.ay.set_autoscale_on(False)


        self.canvas = FigureCanvas(self, -1, self.figure)
        self.sizer = wx.BoxSizer(wx.VERTICAL)
        self.sizer.Add(self.canvas, 1, wx.LEFT | wx.TOP | wx.GROW)
        self.SetSizer(self.sizer)
        self.Fit()



    def update_data(self,data,min1,max1,min2,max2):
        if self.background_ax == '' or self.background_ay == '':
            # save a clean background and save it as a pixel buffer
            self.background_ax = self.canvas.copy_from_bbox(self.ax.bbox)#11111
            self.background_ay = self.canvas.copy_from_bbox(self.ay.bbox)#11111

        # restore the clean background
        self.canvas.restore_region(self.background_ax) #11111
        self.canvas.restore_region(self.background_ay) #11111

        # update the data
        self.y1 = numpy.insert(self.y1,0,data)
        self.y1 = numpy.delete(self.y1,(self.len_y1-1))
        self.y2 = numpy.insert(self.y2,0,data)
        self.y2 = numpy.delete(self.y2,(self.len_y2-1))

        print data

        #self.line1.set_data(self.x,self.y)  # update the data
        #self.line2.set_data(self.x,self.y)  # update the data
        self.line1.set_ydata(self.y1)
        self.line2.set_ydata(self.y2)

        if self.update_axes == True:
            self.line1.axes.set_ylim(min1,max1)
            self.line2.axes.set_ylim(min2,max2)
            self.canvas.draw()
            self.update_axes = False

        # just draw the animated artist
        self.ax.draw_artist(self.line1)
        self.ax.draw_artist(self.line2)

        # just redraw the axes rectangle
        self.canvas.blit(self.ax.bbox)
        self.canvas.blit(self.ay.bbox)



        #self.canvas.draw()

class MainFrame(wx.Frame):

    def __init__(self, parent, id):
```

```
        wx.Frame.__init__(self,parent,id,'PythonDevelopmentTool',style= wx.↩
            SYSTEM_MENU | wx.CAPTION | wx.CLOSE_BOX)#size=(500,800))

        self.save = ""
        self.past = ""

        self.tmp_A = MutableString()
        self.tmp_B = MutableString()
        #self.tmp_C = MutableString()

        self.en_a = False
        self.en_b = False
        self.en_c = False

        self.min1 = 0
        self.max1 = 51
        self.min2 = 0
        self.max2 = 51

        self.t1 = BTThreadClass()
        self.t1.setDaemon(True)
        self.t1.start()

        self.create_status_bar()
        self.create_panel()

        self.fi = open("PythonDevelopmentTool.txt", "w") # Open a file
        self.statusbar.SetStatusText("Logg OFF",1)
        self.statusbar.SetStatusText("Not Connected",2)

        self.timer = wx.Timer(self)
        self.Bind(wx.EVT_TIMER, self.draw_plot, self.timer)
        self.timer.Start(wx.TIMER_CONTINUOUS)


        self.Bind(wx.EVT_CLOSE, self.OnClose)

    def create_panel(self):

        self.panel = wx.Panel(self, style= wx.SUNKEN_BORDER)

        self.CanvasPanel = CanvasPanel(self.panel)

        self.hbox = wx.BoxSizer(wx.HORIZONTAL)

        self.button_1 = wx.Button(self.panel,label= "clear queue",size=(90,30))
        self.Bind(wx.EVT_BUTTON,self.clear_queue,self.button_1)
        self.hbox.Add(self.button_1, 0, border=5, flag=wx.ALL | wx.↩
            ALIGN_CENTER_VERTICAL)
        self.hbox.AddSpacer(20)

        self.button_2 = wx.Button(self.panel,label= "Start Logg",size=(90,30))
        self.Bind(wx.EVT_BUTTON,self.toggle_logg,self.button_2)
        self.hbox.Add(self.button_2, 0, border=5, flag=wx.ALL | wx.↩
            ALIGN_CENTER_VERTICAL)
        self.hbox.AddSpacer(20)

        self.button_3 = wx.Button(self.panel,label= "Connect",size=(90,30))
        self.Bind(wx.EVT_BUTTON,self.toggle_connet,self.button_3)
        self.hbox.Add(self.button_3, 0, border=5, flag=wx.ALL | wx.↩
            ALIGN_CENTER_VERTICAL)
        self.hbox.AddSpacer(20)

        self.top_text = wx.StaticText(self.panel, -1, "Top Plot:")
        self.hbox.Add(self.top_text, 0, border=5, flag=wx.ALL | wx.↩
            ALIGN_CENTER_VERTICAL)
        self.hbox.AddSpacer(1)
```

```
        self.y1max = wx.TextCtrl(self.panel, -1, "max", size=(30,20))
        self.hbox.Add(self.y1max, 0, border=5, flag=wx.ALL | wx.↩
            ALIGN_CENTER_VERTICAL)
        self.Bind(wx.EVT_TEXT_ENTER,self.set_y1_max,self.y1max)
        self.hbox.AddSpacer(2)

        self.y1min = wx.TextCtrl(self.panel, -1, "min", size=(30,20))
        self.hbox.Add(self.y1min, 0, border=5, flag=wx.ALL | wx.↩
            ALIGN_CENTER_VERTICAL)
        self.Bind(wx.EVT_TEXT_ENTER,self.set_y1_min,self.y1min)
        self.hbox.AddSpacer(20)

        self.Bottom_text = wx.StaticText(self.panel, -1, "Bottom Plot:")
        self.hbox.Add(self.Bottom_text, 0, border=5, flag=wx.ALL | wx.↩
            ALIGN_CENTER_VERTICAL)
        self.hbox.AddSpacer(1)

        self.y2max = wx.TextCtrl(self.panel, -1, "max", size=(30,20))
        self.hbox.Add(self.y2max, 0, border=5, flag=wx.ALL | wx.↩
            ALIGN_CENTER_VERTICAL)
        self.Bind(wx.EVT_TEXT_ENTER,self.set_y2_max,self.y2max)
        self.hbox.AddSpacer(2)

        self.y2min = wx.TextCtrl(self.panel, -1, "min", size=(30,20))
        self.hbox.Add(self.y2min, 0, border=5, flag=wx.ALL | wx.↩
            ALIGN_CENTER_VERTICAL)
        self.Bind(wx.EVT_TEXT_ENTER,self.set_y2_min,self.y2min)
        self.hbox.AddSpacer(20)

        self.vbox = wx.BoxSizer(wx.VERTICAL)
        self.vbox.Add(self.CanvasPanel, 1, flag=wx.LEFT | wx.TOP | wx.GROW)
        self.vbox.Add(self.hbox, 0, flag=wx.ALIGN_LEFT | wx.TOP)

        self.panel.SetSizer(self.vbox)
        self.vbox.Fit(self)


    def draw_plot(self,event):

        self.statusbar.SetStatusText("queue size: %s" % str(q.qsize()),0)

        try:
            self.data = q.get(block = False)

            #print "past: %s" % self.past

            if self.past == 'A':
                self.en_a = True
                self.en_b = False
                self.en_c = False
                self.tmp_B = MutableString()
                #self.tmp_C = MutableString()


            if self.past == 'B':
                self.en_a = False
                self.en_b = True
                self.en_c = False
                self.tmp_A = MutableString()
                #self.tmp_C = MutableString()

            #if self.past == 'C':
            #    self.en_a = False
            #    self.en_b = False
            #    self.en_c = True
            #    self.tmp_A = MutableString()
            #    self.tmp_B = MutableString()
```

```
            if self.en_a == True and self.data != 'A' and self.data != 'B' and ↩
                self.data != 'C':
                self.tmp_A += self.data

            if self.en_b == True and self.data != 'A' and self.data != 'B' and ↩
                self.data != 'C':
                self.tmp_B += self.data

            #if self.en_c == True and self.data != 'A' and self.data != 'B' and↩
                 self.data != 'C':
            #    self.tmp_C += self.data

            self.past = self.data

            if self.data == 'B':
                if len(self.tmp_A) > 0:
                    self.CanvasPanel.update_data( float(self.tmp_A) ,self.min1,↩
                        self.max1,self.min2,self.max2) #jon list to string an ↩
                        cast to float
                    if self.save == True:
                        self.fi.write("%s\n" % self.tmp_A)

            #else:
            #    if len(tmp_0) > 0:
            #        #self.CanvasPanel.update_data(float(''.join(self.data)),↩
                self.min1,self.max1,self.min2,self.max2) #jon list to string an↩
                 cast to float
            #
            #        if self.save == True:
            #            self.fi.write("%s\n" % (''.join(self.data)))
            #
            #        print float(''.join(tmp_0))
            #        tmp_0 = []

        except Queue.Empty:
            return

    def clear_queue(self, event):
        with q.mutex: q.queue.clear()
        self.statusbar.SetStatusText(str(q.qsize()),0)

    def toggle_logg(self, event):
        btnLabel = self.button_2.GetLabel()
        if btnLabel == "Start Logg":
            self.statusbar.SetStatusText("Logg ON",1)
            self.save = True
            self.button_2.SetLabel("Stop Logg")
        else:
            self.statusbar.SetStatusText("Logg OFF",1)
            self.save = False
            self.button_2.SetLabel("Start Logg")

    def toggle_connet(self,event):
        btnLabel = self.button_3.GetLabel()
        if btnLabel == "Connect":
            self.statusbar.SetStatusText("Connecting",2)
            self.t1.connect()
            if self.t1.isconnected() != 0:
                self.statusbar.SetStatusText("Connected: %s" % self.t1.↩
                    isconnected(),2)
                self.button_3.SetLabel("Disconnect")
            else:
                self.statusbar.SetStatusText("Not Connected",2)
                self.button_3.SetLabel("Connect")
        else:
            self.statusbar.SetStatusText("Not Connected",2)
            self.t1.disconnect()
            self.button_3.SetLabel("Connect")
```

```
    def set_y1_max(self, event):
        try:
            self.max1 = int(self.y1max.GetValue())
            self.CanvasPanel.update_axes=True
        except ValueError:
            return

    def set_y1_min(self, event):
        try:
            self.min1 = int(self.y1min.GetValue())
            self.CanvasPanel.update_axes=True
        except ValueError:
            return

    def set_y2_max(self, event):
        try:
            self.max2 = int(self.y2max.GetValue())
            self.CanvasPanel.update_axes=True
        except ValueError:
            return

    def set_y2_min(self, event):
        try:
            self.min2 = int(self.y2min.GetValue())
            self.CanvasPanel.update_axes=True
        except ValueError:
            return

    def create_status_bar(self):
        self.statusbar = self.CreateStatusBar(3)

    def OnClose(self,event):
        dlg = wx.MessageDialog(self,"Do you really want to close this ↩
            application?",
            "Confirm Exit", wx.OK|wx.CANCEL|wx.ICON_QUESTION)
        result = dlg.ShowModal()
        dlg.Destroy()
        if result == wx.ID_OK:
            self.timer.Stop()
            self.fi.close()
            if self.t1.isconnected() != 0:
                self.t1.disconnect()
            self.Destroy()

if __name__=='__main__':

    app=wx.App(0)
    frame = MainFrame(parent = None, id=-1)
    frame.Show(True)
    app.MainLoop()
```

Appendix D

Matlab and Simulink Code

D.1 Moving Average Filter Analysis Code

```
clear 'all'

N = 200; %Number of points in filter

%Time
Fs = 1250;  % Sampling Frequency
dt = 1/Fs;  % Seconds per sample
StopTime = 3; %seconds

t = (0:dt:StopTime-dt);

%sin A
Fc_A = 25; %Frequency in Hz
x_A = sin(2*pi*Fc_A*t);% Generate Sine Wave A

%MA filter
a=[1 zeros(1,N-1)];%coefficent array of MA filter
b=(1/N)*ones(1,N);%coefficent array of MA filter

out=filter(b,a,x_A);

figure(1)
plot(t,x_A);
title('IN')
grid 'on'
axis([0 1 -1.5 1.5])

figure(2)
plot(t,out);
title('OUT')
grid 'on'

figure(3)
stepz(b,a,N,Fs);
grid 'on'

figure(4)
impz(b,a,N);
grid 'on'

figure(5)
freqz(b,a,2^16,Fs);% Compute frequency response H(w)
```

D.2 Lock-in Simulation

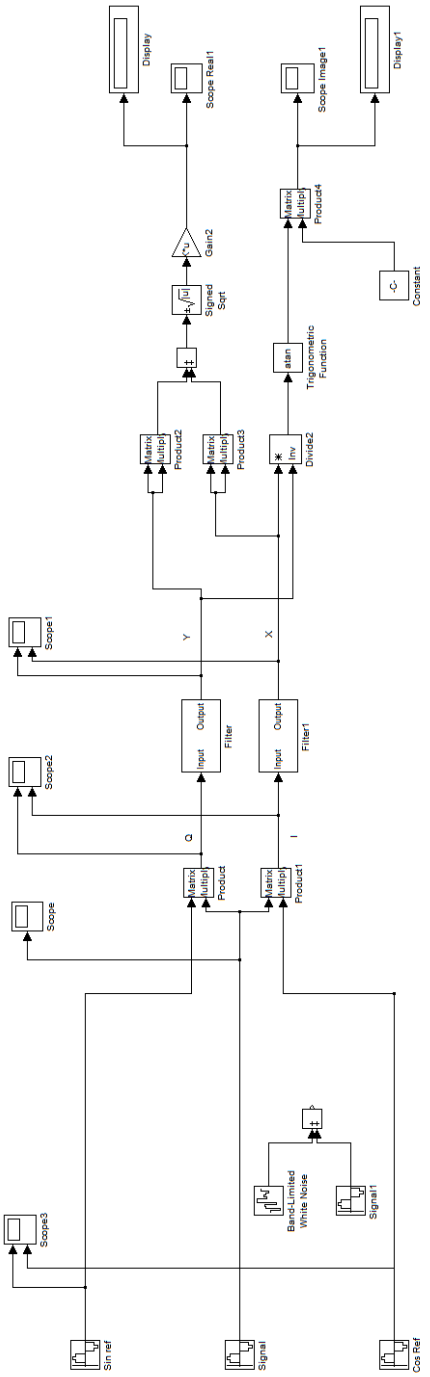


Figure D.1: Simulink simulation used to test the lock-in algorithm.

D.3 Calibration Code

Listing: Howland.m

```
r = [65.4e3,94.74e3,83.42e3,111.88e3,154.9e3,240.3e3,267.4e3];
v=[64e-3   ,88e-3  ,80e-3  ,100e-3  ,132e-3 ,190e-3 ,210e-3];

figure;
plot(r,v,'o')
% Turn on the grid
grid on;

% Add title and axis labels
title('Resistance vs Voltage','fontsize', 10);
xlabel('R_{var}');
ylabel('U_r');
```

Listing: Resistance.m

```
r_ = [239,426.9,674,812.7,1184,2.95e3,5.06e3,11.04e3,14.78e3,17.79e3,22.33e3
    ,30.2e3,39.24e3,46.33e3,50.12e3,55.44e3,63.75e3,64.35e3,70.75e3,82.93e3↩
        ,121.83e3];
r_m=[ 1365,2395,3796,4544,6543,16310,27970,60850,81520,98099,123099,166590,
    216250,255160,276350,305740,350860,354560,390200,457250,673150];

plot(r_m,r_,'o')
% Turn on the grid
grid on;

% Add title and axis labels
title('Resistance vs Instrument Readout','fontsize', 10);
xlabel('R_{var}');
ylabel('Instrument Readout');
```

Listing: Current.m

```
v_in_howland=[1,1.3,1.6,1.9,2,2.3,2.6,2.9,3,3.3,3.6,3.9,4];
i_m=[53450,74499,92386,110490,116240,134390,151744,170495,
    177330,195350,212760,228450, 234360];

i_c = v_in_howland./2.5e6;

plot(i_m,i_c,'o')
% Turn on the grid
grid on;

% Add title and axis labels
title('Instrument Readout vs Current','fontsize', 20);
xlabel('Current','fontsize', 20);
ylabel('Instrument Readout','fontsize', 20);
```

Listing: Potential.m

```
v_ = [-5e-3 ,-10e-3  ,-15e-3   ,-20e-3  ,-25e-3   ,-30e-3  ,-35e-3  ,-40e-3 ↩
    ,-45e-3  ,-50e-3  ,-55e-3  ,-60e-3  ,-65e-3];
v_m=[-375984,-1008562,-1642293 ,-2274720,-2906748 ↩
    ,-3539475,-4173850,-4805460,-5439850,-6072021,-6704987,-7339150,-7971650 ];
plot(v_,v_m,'o')
% Turn on the grid
grid on;

% Add title and axis labels
title('Voltage vs Instrument Readout ','fontsize', 20);
xlabel('Voltage','fontsize', 20);
ylabel('Instrument Readout','fontsize', 20);
```

Appendix E

LTspice Simulation

E.1 LTspice Simulation used to verify the values used for the ADA4941

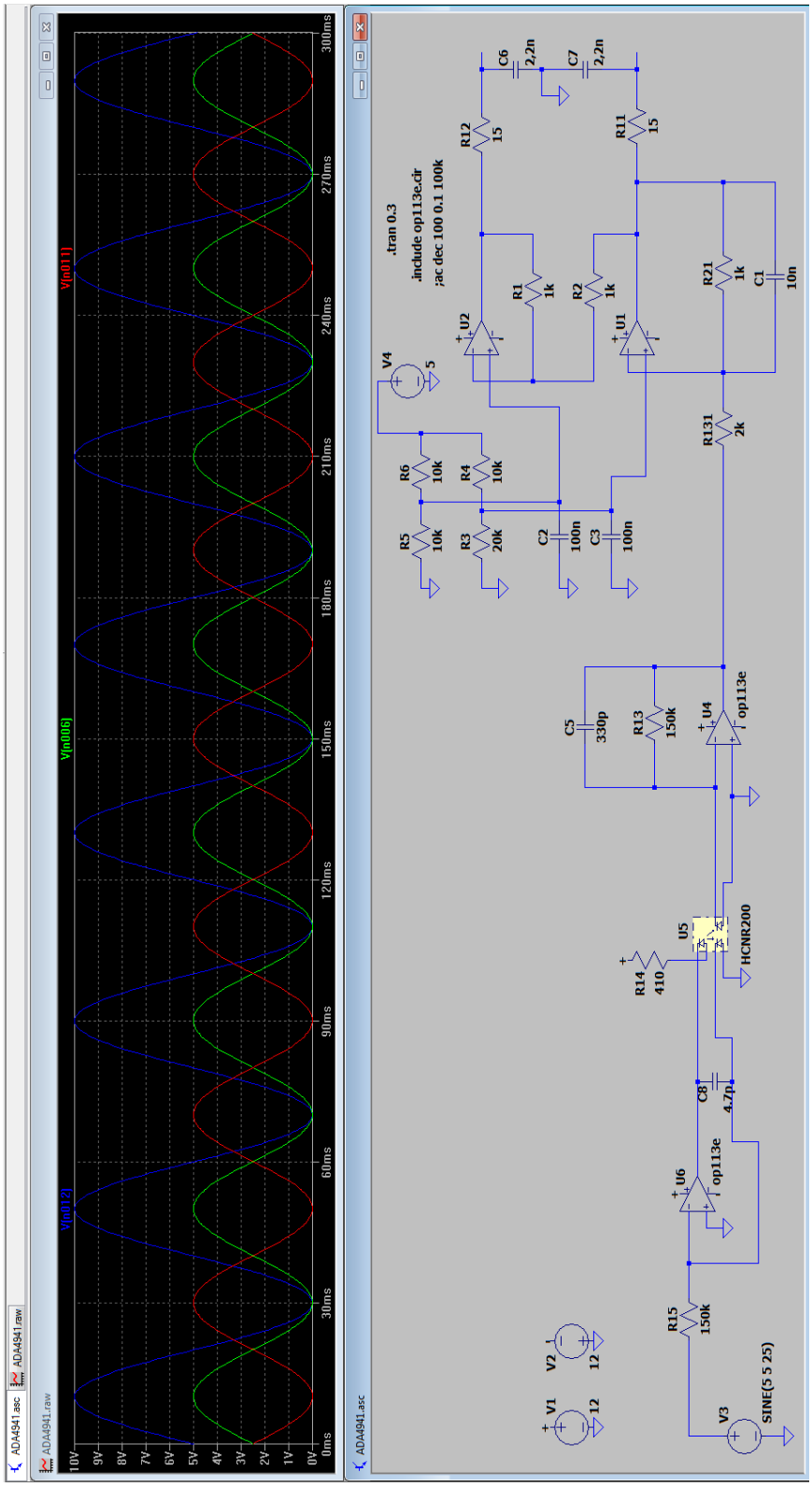
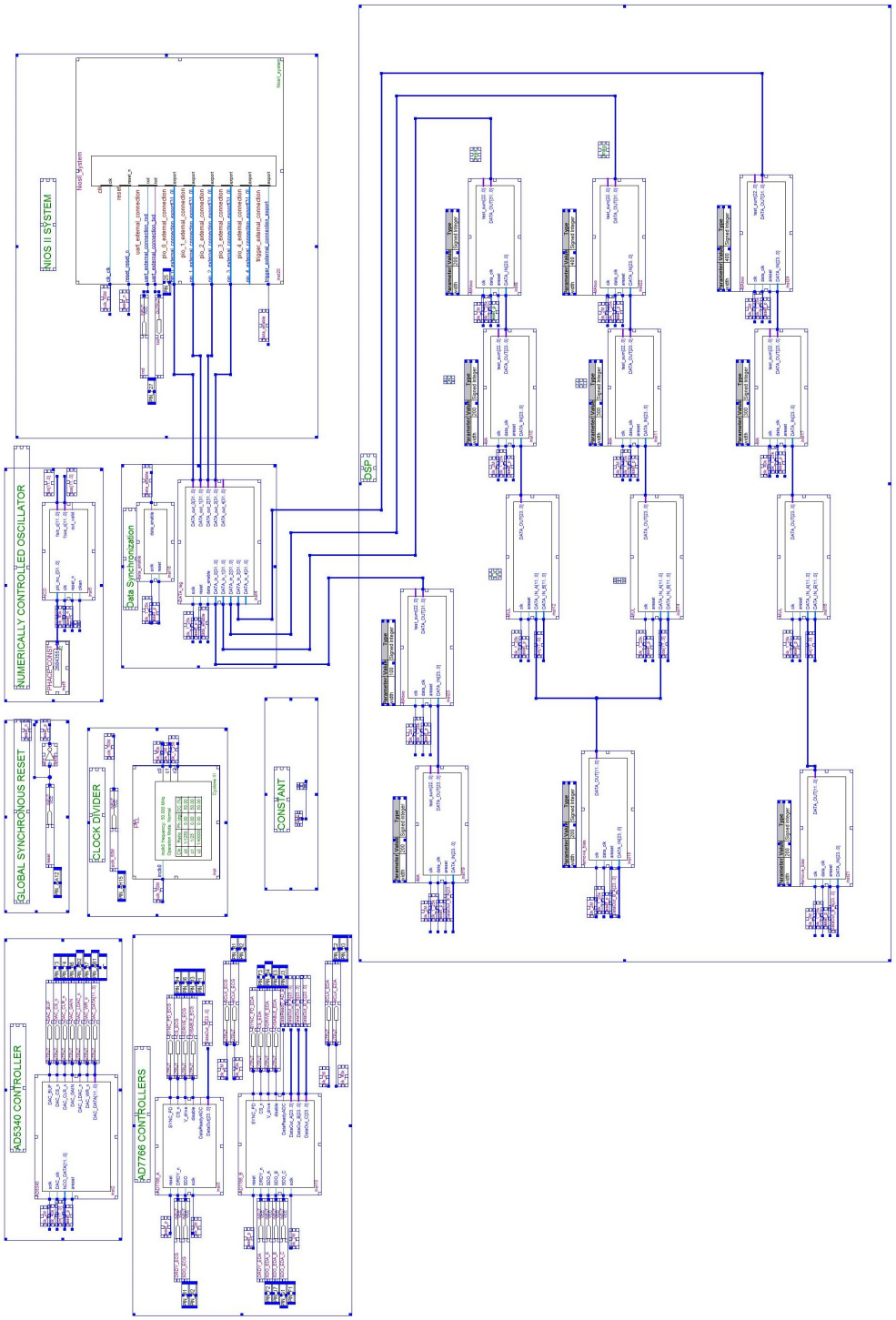


Figure E.1: LTspice Simulation used to verify the values used for the ADA4941.

Appendix F

VHDL Code

F.1 Top File


NIOS II SYSTEM
Nios II System
NUMERICALLY CONTROLLED OSCILLATOR
Data Synchronization
DSP
GLOBAL SYNCHRONOUS RESET
CLOCK DIVIDER
CONSTANT
AD5340 CONTROLLER
AD7766 CONTROLLERS

Listing: AD5340.vhd

```
-- Author        : Lars Jørgen Aamodt
-- Company       : University of Oslo
-- File name     : AD5340.vhd
-- Project       : Master project
-- Function      : Control circuit for the AD5340

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

entity AD5340 is
    PORT
    (
    sclk            : in            std_logic;  ↩
                                        -- System clock
    DAC_clk         : in            std_logic;    ↩
                                      -- DAC clock
    NCO_DATA        : in            std_logic_vector(11 downto 0); ↩
                    -- Data from NCO to DAC ([11:0])
    areset          : in            std_logic;


    DAC_BUF         : out               std_logic;
    DAC_CS_n        : out               std_logic;
    DAC_CLR_n       : out               std_logic;
    DAC_GAIN        : out               std_logic;
    DAC_LDAC_n      : out               std_logic;
    DAC_WR_n        : out               std_logic;
    DAC_DATA        : out            std_logic_vector(11 downto 0)  ↩
                    -- Data in to DAC (DA[11:0])
    );

end entity AD5340;

architecture Behavior of AD5340 is

signal DATA_ff1         : std_logic_vector(11 downto 0);
signal DATA_ff2         : std_logic_vector(11 downto 0);
signal DATA_ff1_scaled  : std_logic_vector(11 downto 0);




begin

-- convert to offset binary
DATA_ff2 <= DATA_ff1 xor "100000000000";


 DAC_WR_n       <= not DAC_clk;

 DAC_BUF        <= '0';
 DAC_CS_n       <= '0';
 DAC_CLR_n      <= '1';
 DAC_GAIN       <= '0';
 DAC_LDAC_n     <= '0';


ff1: process(sclk)
    begin
        if rising_edge(sclk) then
        DATA_ff1 <= NCO_DATA;
        DAC_DATA  <= DATA_ff2;
        end if;
end process ff1;
```

```
end architecture Behavior;
```

Listing: AD7766_A.vhd

```
-- Author        : Lars Jørgen Aamodt
-- Company       : University of Oslo
-- File name     : AD7766_A.vhd
-- Project       : Master project
-- Function      : Control circuit for the AD7766

library ieee;
use ieee.std_logic_1164.all;
--use ieee.std_logic_arith.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↩
         ready, active low
    SDO                     : in            std_logic;                  -- Data↩
         output, MSB first

    sclk                    : in            std_logic;                  -- ↩
        System clock

    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)-- ↩
        Output data

    );

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 ADC
            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: AD7766_B.vhd

```
-- Author        : Lars Jørgen Aamodt
-- Company       : University of Oslo
-- File name     : AD7766_B.vhd
-- 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_B is

  port
    (
    reset                   : in            std_logic;
    DRDY_n                  : in            std_logic                   -- ↩
        Data ready, active low
    SDO_A                   : in            std_logic;                  -- ↩
        Data output, MSB first
    SDO_B                   : in            std_logic;                  -- ↩
        Data output, MSB first
    SDO_C                   : in            std_logic;                  -- ↩
        Data output, MSB first


    sclk                    : in            std_logic;                  -- ↩
        System clock

    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_A               : out           std_logic_vector(23 downto 0);↩
        -- Output data
    DataOut_B               : out           std_logic_vector(23 downto 0);↩
        -- Output data
    DataOut_C               : out           std_logic_vector(23 downto 0) ↩
        -- Output data

    );

end AD7766_B;

architecture AD7766_arch of AD7766_B 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_regs
SRin_Pout_reg_A: entity work.SRin_Pout_reg
    port map (
            clk     => sclk,
          DataIn  => SDO_A,
          shift_en=> shift_en,
          DataOut =>    DataOut_A
            );

SRin_Pout_reg_B: entity work.SRin_Pout_reg
    port map (
            clk     => sclk,
          DataIn  => SDO_B,
          shift_en=> shift_en,
          DataOut =>    DataOut_B
            );

SRin_Pout_reg_C: entity work.SRin_Pout_reg
    port map (
            clk     => sclk,
          DataIn  => SDO_C,
          shift_en=> shift_en,
          DataOut =>    DataOut_C
            );

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 ADC
            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 =>

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

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

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

    end case;
  end if;
end  process FSM_CONF_READ;

end AD7766_arch;
```

Listing: AD7766_B.vhd

```
-- Author       : Lars Jørgen Aamodt
-- Company      : University of Oslo
-- File name    : AD7766_B.vhd
-- 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_B is

  port
    (
```

```
    reset                   : in                std_logic;
    DRDY_n                  : in                std_logic                   -- ↩
        Data ready, active low
    SDO_A                   : in                std_logic;                  -- ↩
        Data output, MSB first
    SDO_B                   : in                std_logic;                  -- ↩
        Data output, MSB first
    SDO_C                   : in                std_logic;                  -- ↩
        Data output, MSB first


    sclk                    : in                std_logic;                  -- ↩
        System clock

    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_A               : out               std_logic_vector(23 downto 0);↩
        -- Output data
    DataOut_B               : out               std_logic_vector(23 downto 0);↩
        -- Output data
    DataOut_C               : out               std_logic_vector(23 downto 0) ↩
        -- Output data

    );

end AD7766_B;

architecture AD7766_arch of AD7766_B 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_regs
SRin_Pout_reg_A: entity work.SRin_Pout_reg
    port map (
            clk     => sclk,
          DataIn  => SDO_A,
          shift_en=> shift_en,
          DataOut =>    DataOut_A
            );

SRin_Pout_reg_B: entity work.SRin_Pout_reg
    port map (
            clk     => sclk,
          DataIn  => SDO_B,
          shift_en=> shift_en,
          DataOut =>    DataOut_B
            );

SRin_Pout_reg_C: entity work.SRin_Pout_reg
    port map (
            clk     => sclk,
          DataIn  => SDO_C,
          shift_en=> shift_en,
          DataOut =>    DataOut_C
```

```
            );

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 ADC
            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 =>

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

        when  Wait_st =>
            if drdy_n = '1' then
```

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

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

    end case;
  end if;
end  process FSM_CONF_READ;

end AD7766_arch;
```

Listing: delay_shiftreg.vhd

```
-- Author        : Lars Jørgen Aamodt
-- Company       : University of Oslo
-- File name     : DAC5672.vhd
-- Date          : 25.04.2012
-- Project       : Master project
-- Function      :

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

entity delay_shiftreg is

    generic (
    width : integer := 12);


    PORT
    (
    clk             : in           std_logic;  ↩
                                        -- System clock 125 MHz

     DATA_IN        : in            std_logic_vector(11 downto 0);
     DATA_OUT       : out          std_logic_vector(11 downto 0)
    );

end entity delay_shiftreg;

architecture Behavior of delay_shiftreg is
type reg_shft is array(width-1 downto 0) of std_logic_vector(11 downto 0);
signal shft_reg : reg_shft :=(others => x"000");


begin

ff1: process(clk)

    begin

        if rising_edge(clk) then

        shft_reg(0) <= DATA_IN;
        for i in width-2 downto 0 loop
            shft_reg(i+1) <= shft_reg(i);
        end loop;
        DATA_OUT <= shft_reg(width-1);
        end if;

    end process ff1;
```

```
end architecture Behavior;
```

Listing: MA.vhd

```
-- Author        : Lars Jørgen Aamodt
-- Company       : University of Oslo
-- File name     : MA.vhd
-- Project       : Master project
-- Function      : Perform moving average filtering
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;


entity MA is

    generic (
    width : integer := 1);


    PORT
    (
    clk             : in            std_logic;  ↩
                                      -- System clock
    data_clk        : in            std_logic;    ↩
                                      -- Data clock
    areset          : in            std_logic;
    test_sum        : out           std_logic_vector(22 downto 0); ↩
                    -- Output for testing purposes
    DATA_IN         : in            std_logic_vector(23 downto 0); ↩
                    -- Input data
    DATA_OUT        : out           std_logic_vector(23 downto 0) ↩
                     -- Output data
    );

end entity MA;

architecture Behavior of MA is
--typedef.
type statetype is ( init,add,sub,div);
type reg_shft is array(width-1 downto 0) of std_logic_vector(23 downto 0);


signal state : statetype;
signal shft_reg : reg_shft :=(others => (others => '0'));
signal sum : signed(63 downto 0):=(others => '0');



signal start_calc       : std_logic:= '0';
signal last_clk     : std_logic:= '0';
signal old_value            : signed(23 downto 0):=(others => '0');

begin

start_calc <= '1' when last_clk = '0' and data_clk = '1' else '0';

-- Register used to hold the data values
s_reg:
process(data_clk, areset)
begin
```

```
    if (areset = '1') then
        shft_reg <= (others => (others => '0'));

   elsif rising_edge(data_clk) then
        shft_reg(0) <= DATA_IN;

        for i in width-2 downto 0 loop
            shft_reg(i+1) <= shft_reg(i);
        end loop;

    end if;
end  process s_reg;

last_clk_reg:
process(clk, areset)
begin

    if (areset = '1') then
        last_clk <= '0';

    elsif rising_edge(clk) then
        last_clk <= data_clk;

    end if;
end  process last_clk_reg;

-- Register used to hold the sub data value
old_value_reg:
process(data_clk, areset)
begin
    if (areset = '1') then
        old_value <= (others => '0');
    elsif rising_edge(data_clk) then
        old_value <= signed(shft_reg(width-1));
    end if;
end  process old_value_reg;

FSM_MA:
process(clk, areset)
begin

    if (areset = '1') then

        DATA_OUT <= (others => '0');
        state <= init;

   elsif rising_edge(clk) then

     case state is

        when init =>
            if (start_calc = '1') then
                state <= add;
            end if;

        when add =>
                sum <= sum + signed(shft_reg(0));
                state <= sub;

        when sub =>
            sum <= sum - signed(old_value);
            state <= div;

        when div =>
                test_sum <= std_logic_vector(sum / width)(22 downto 0);
                DATA_OUT <= std_logic_vector(sum / width)(23 downto 0);
                state <= init;
```

```
        when others =>
            state <= init;              -- Fault tolerance

    end case;
  end if;
 end  process FSM_MA;
end architecture Behavior;
```

Listing: MAtwo.vhd

```
-- Author        : Lars Jørgen Aamodt
-- Company       : University of Oslo
-- File name     : MA
-- Project       : Master project
-- Function      : Perform moving average filtering

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


entity MAtwo is

    generic (
    width : integer := 1);


    PORT
    (
    clk           : in          std_logic;                                  -- ↩
        System clock
     data_clk      : in             std_logic;                               -- ↩
         Data clock
     areset        : in             std_logic;
     test_sum      : out            std_logic_vector(22 downto 0);           -- ↩
         Output for testing purposes
     DATA_IN       : in             std_logic_vector(23 downto 0);           -- ↩
         Input data
     DATA_OUT      : out            std_logic_vector(31 downto 0)            -- ↩
         Output data
    );

end entity MAtwo;

architecture Behavior of MAtwo is
--typedef.
type statetype is ( init,add,sub,div);
type reg_shft is array(width-1 downto 0) of std_logic_vector(23 downto 0);


signal state : statetype;
signal shft_reg : reg_shft :=(others => (others => '0'));
signal sum : signed(63 downto 0):=(others => '0');



signal start_calc       : std_logic:= '0';
signal last_clk     : std_logic:= '0';
signal old_value            : signed(23 downto 0):=(others => '0');

begin

start_calc <= '1' when last_clk = '0' and data_clk = '1' else '0';

-- Register used to hold the data values
```

```
s_reg:
process(data_clk, areset)
begin

    if (areset = '1') then
        shft_reg <= (others => (others => '0'));

   elsif rising_edge(data_clk) then
        shft_reg(0) <= DATA_IN;

        for i in width-2 downto 0 loop
            shft_reg(i+1) <= shft_reg(i);
        end loop;

    end if;
end  process s_reg;

last_clk_reg:
process(clk, areset)
begin

    if (areset = '1') then
        last_clk <= '0';

    elsif rising_edge(clk) then
        last_clk <= data_clk;

    end if;
end  process last_clk_reg;

-- Register used to hold the sub data value
old_value_reg:
process(data_clk, areset)
begin
    if (areset = '1') then
        old_value <= (others => '0');
    elsif rising_edge(data_clk) then
        old_value <= signed(shft_reg(width-1));
    end if;
end  process old_value_reg;

FSM_MA:
process(clk, areset)
begin

    if (areset = '1') then

        DATA_OUT <= (others => '0');
        state <= init;

   elsif rising_edge(clk) then

     case state is

        when init =>
            if (start_calc = '1') then
                state <= add;
            end if;

        when add =>
                sum <= sum + signed(shft_reg(0));
                state <= sub;

        when sub =>
            sum <= sum - signed(old_value);
            state <= div;

        when div =>
```

```
                test_sum <= std_logic_vector(sum / width)(22 downto 0);
                DATA_OUT <= std_logic_vector(sum / width)(31 downto 0);
                state <= init;

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

    end case;
  end if;
 end  process FSM_MA;
end architecture Behavior;
```

Listing: MUL.vhd

```
-- Author        : Lars Jørgen Aamodt
-- Company       : University of Oslo
-- File name     : MUL.vhd
-- Project       : Master project
-- Function      : Multiply

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


entity MUL is


    PORT
    (
    clk             : in           std_logic;  ↩
                                        -- System clock
    areset          : in                std_logic;
    DATA_IN_A       : in             std_logic_vector(11 downto 0);
    DATA_IN_B       : in             std_logic_vector(11 downto 0);
    DATA_OUT        : out           std_logic_vector(23 downto 0)
    );

end entity MUL;

architecture Behavior of MUL is

begin

ff1: process(clk)
    begin
        if rising_edge(clk) then
        DATA_OUT <= std_logic_vector( signed(DATA_IN_A) * signed(DATA_IN_B) );

        end if;

    end process ff1;

end architecture Behavior;
```

Listing: remove_bias.vhd

```
-- Author        : Lars Jørgen Aamodt
-- Company       : University of Oslo
-- File name     : remove_bias.vhd
-- Project       : Master project
-- Function      : Remove Bias
```

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


entity remove_bias is

    generic (
    width : integer := 1);


    PORT
    (
    clk             : in            std_logic;  ↩
                                        -- System clock
    data_clk        : in            std_logic;
    areset          : in            std_logic;
    DATA_IN         : in            std_logic_vector(23 downto 0);
    DATA_OUT        : out           std_logic_vector(11 downto 0)
    );

end entity remove_bias;

architecture Behavior of remove_bias is
--typedef.
type statetype is ( Init,add,sub,div,sub_mean);
type reg_shft is array(width-1 downto 0) of std_logic_vector(23 downto 0);


signal state : statetype;
signal shft_reg : reg_shft :=(others => (others => '0'));
signal sum : signed(63 downto 0):=(others => '0');



signal start_calc       : std_logic:= '0';
signal last_clk     : std_logic:= '0';
signal old_value            : signed(23 downto 0):=(others => '0');
signal mean                 : std_logic_vector(23 downto 0):=(others => '0');--↩
    64

begin


start_calc <= '1' when last_clk = '0' and data_clk = '1' else '0';

s_reg:
process(data_clk, areset)
begin

    if (areset = '1') then
        shft_reg <= (others => (others => '0'));

   elsif rising_edge(data_clk) then
        shft_reg(0) <= DATA_IN;

        for i in width-2 downto 0 loop
            shft_reg(i+1) <= shft_reg(i);
        end loop;

    end if;
end  process s_reg;

last_clk_reg:
process(clk, areset)
begin

    if (areset = '1') then
```

```
        last_clk <= '0';

    elsif rising_edge(clk) then
        last_clk <= data_clk;

    end if;
end  process last_clk_reg;

old_value_reg:
process(data_clk, areset)
begin
    if (areset = '1') then
        old_value <= (others => '0');
    elsif rising_edge(data_clk) then
        old_value <= signed(shft_reg(width-1));
    end if;
end  process old_value_reg;

FSM_RB:
process(clk, areset)
begin

    if (areset = '1') then

        DATA_OUT <= (others => '0');
        state <= Init;

   elsif rising_edge(clk) then

     case state is

        when Init =>
            if (start_calc = '1') then
                state <= add;
            end if;

        when add =>
                sum <= sum + signed(shft_reg(0));
                state <= sub;

        when sub =>
            sum <= sum - signed(old_value);
            state <= div;

        when div =>

                mean    <= std_logic_vector(sum / width)(23 downto 0);--sum / ↩
                    width
                state <= sub_mean;

        when sub_mean =>
                DATA_OUT <= std_logic_vector(signed(DATA_IN) - signed(mean))(23↩
                     downto 12);
                state <= Init;

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

     end case;
  end if;
 end  process FSM_RB;
end architecture Behavior;
```

Listing: SRin_Pout_reg.vhd

```
-- Author        : Lars Jørgen Aamodt
-- Company       : University of Oslo
-- File name     : SR_SerIn_redge.vhd
-- Project       : Master project
-- Function      : Serial in parallel out shift register

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;
```

Appendix G

Nios II Firmware

G.1 Code

Listing: NiosII_Data_Acquisition.c

```
/*************************************************************
*File: NiosII_Data_Acquisition.c
*Project: FPGA Based Development Platform for Biomedical Measurements
*Author: Lars JÃ¸rgen Aamodt
*************************************************************/

#include "alt_types.h"
#include "sys/alt_irq.h"
#include "altera_avalon_pio_regs.h"
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include "system.h"

/* A variable to hold the value of the trigger pio edge capture register. */
volatile int edge_capture;

/* A pointer to the interface UART. */
volatile FILE* fp;

/* Variables to hold the PIOs. */
alt_32 pio_0;
alt_32 pio_1;
alt_32 pio_2;
alt_32 pio_3;
alt_32 pio_4;


/* Handle interrupts */
static void handle_trigger_interrupts(void* context,  alt_u32 id)
{

    /* cast the context pointer to an integer pointer. */
    volatile int* edge_capture_ptr = (volatile int*) context;

    /*
    * Read the edge capture register on the trigger PIO.
    * Store value.
    */
    *edge_capture_ptr = IORD_ALTERA_AVALON_PIO_EDGE_CAP(TRIGGER_BASE);
```

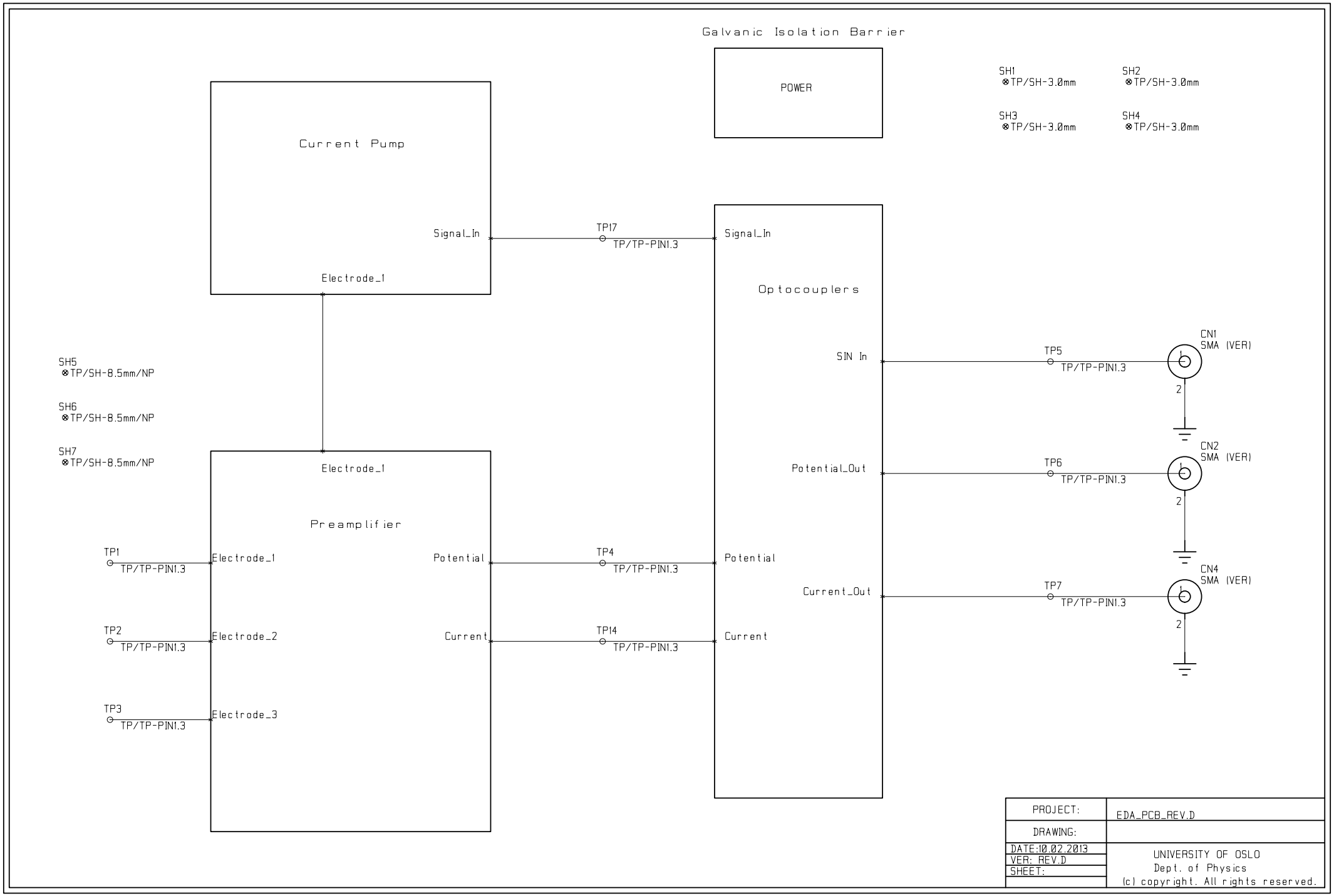
```
    /* Write to the edge capture register to reset it. */
    IOWR_ALTERA_AVALON_PIO_EDGE_CAP(TRIGGER_BASE, 0);

    /* reset interrupt capability for the Button PIO. */
    IOWR_ALTERA_AVALON_PIO_IRQ_MASK(0, 0xf);

}

/*Initiate the trigger PIO*/
static void init_trigger_pio()
{
    /* Recast the edge_capture pointer to match the alt_irq_register() function
         */
    void* edge_capture_ptr = (void*) &edge_capture;
    /* Enable trigger interrupt. */
    IOWR_ALTERA_AVALON_PIO_IRQ_MASK(TRIGGER_BASE, 0xf);
    /* Reset the edge capture register. */
    IOWR_ALTERA_AVALON_PIO_EDGE_CAP(TRIGGER_BASE, 0x0);
    /* Register the interrupt handler. */
    alt_ic_isr_register(TRIGGER_IRQ_INTERRUPT_CONTROLLER_ID, TRIGGER_IRQ,
        handle_trigger_interrupts, edge_capture_ptr, 0x0);

}

/*Initiate the UART*/
static void init_UART()
{
    //Open file for reading and writing "r+"
    fp = fopen ("/dev/uart", "r+");
    //setbuf(fp, NULL);

    if (fp!=NULL){
        printf("UART OK \n");
    }
    else{
        printf("UART ERROR \n");
    }
}

/*Read data*/
static void pio_data(){
    pio_0 = IORD_ALTERA_AVALON_PIO_DATA(PIO_0_BASE); //alt_32

    pio_1 = IORD_ALTERA_AVALON_PIO_DATA(PIO_1_BASE); //alt_32

    pio_2 = IORD_ALTERA_AVALON_PIO_DATA(PIO_2_BASE); //alt_32

    pio_3 = IORD_ALTERA_AVALON_PIO_DATA(PIO_3_BASE); //alt_32

    pio_4 = IORD_ALTERA_AVALON_PIO_DATA(PIO_4_BASE); //alt_32


}

static void initial_message()
{
    printf("\n\n**********************\n");
    printf("* Nios II Controller *\n");
    printf("**********************\n");
}


//***********  MAIN  **********//
int main(void){


    initial_message();
```

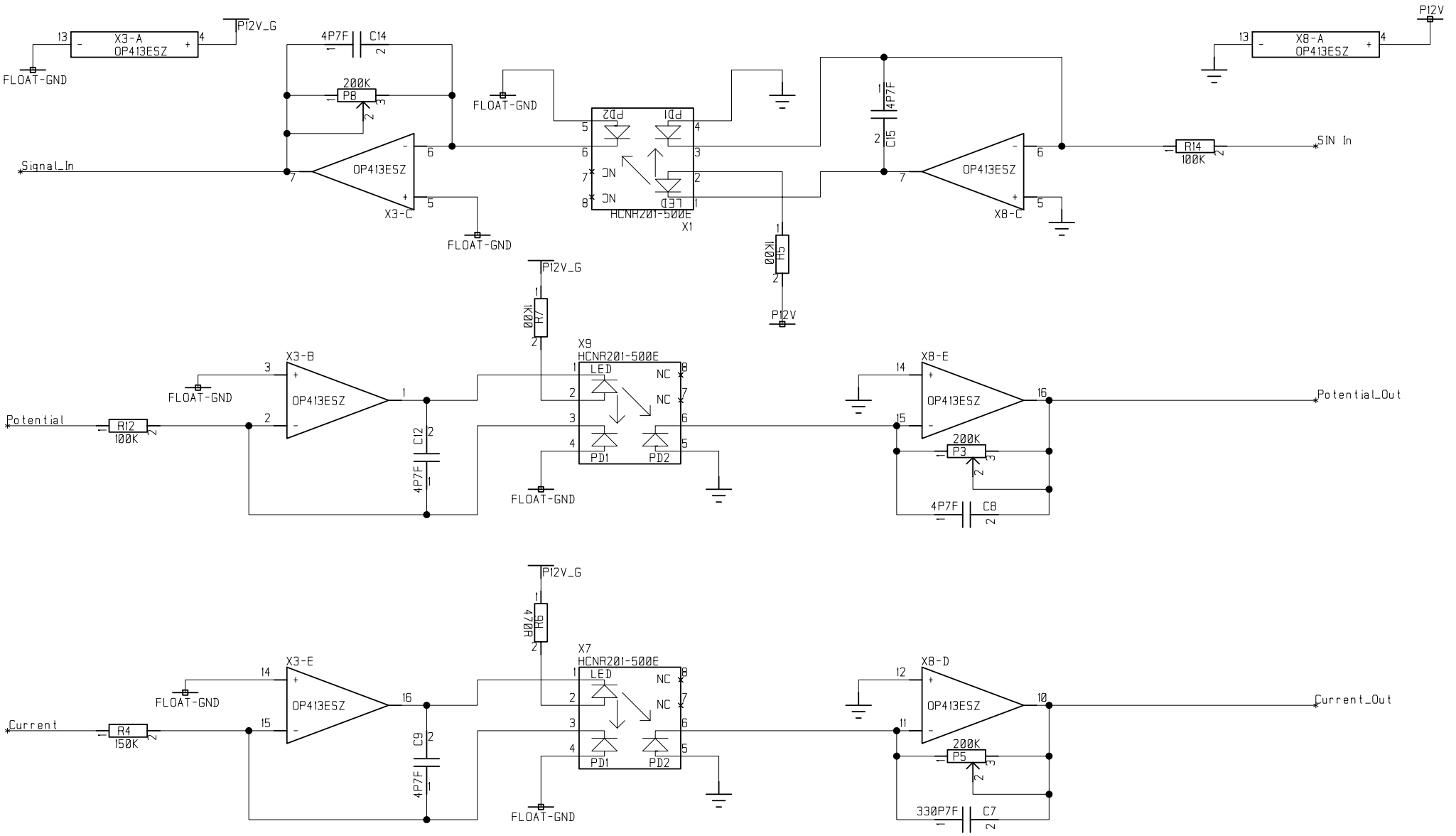
```
    init_trigger_pio();

    init_UART();


    while( 1 )
    {

        if (edge_capture == 0x1){

            pio_data();

            float x = pio_1; //res
            float y = pio_2;//rec

            float z = 2*sqrt((y*y) + (x*x));

            float phase = atan(x/y);


            //float xc = z*sin(phase);
            //printf("pio_0: %f \n",abs((float) pio_0));
            //printf("pio_1: %f \n",(float) pio_1);
            //printf("pio_2: %f \n",(float) pio_2);
            //printf("pio_3: %f \n",(float) pio_3);
            //printf("pio_4: %f \n",(float) pio_4);
            //printf("Z: %f \n",z);
            //printf("Phase: %f \n",phase);
            printf("Reactance:  %f \n",y);
            printf("Resistance: %f \n",x);
            printf("Potensial:  %f \n",(float) pio_0));

            char* msg_0[50];
            char* msg_1[50];

            /*Convert data to character array.*/
            snprintf(msg_0, sizeof(msg_0), "%f", x);
            snprintf(msg_1, sizeof(msg_1), "%f", y );



        /*Send data over UART*/
         if (fp!=NULL)
        {
            fprintf(fp, "%s","A");
            fprintf(fp, "%s",msg_0);
            fflush(fp);
            fprintf(fp, "%s","B");
            fprintf(fp, "%s",msg_1);

        }else{
            printf("UART ERROR \n");
        }

            edge_capture = 0;
    }//end if

    }// end while
    fclose (fp);
    return 0;
}// ene main
```

Appendix H

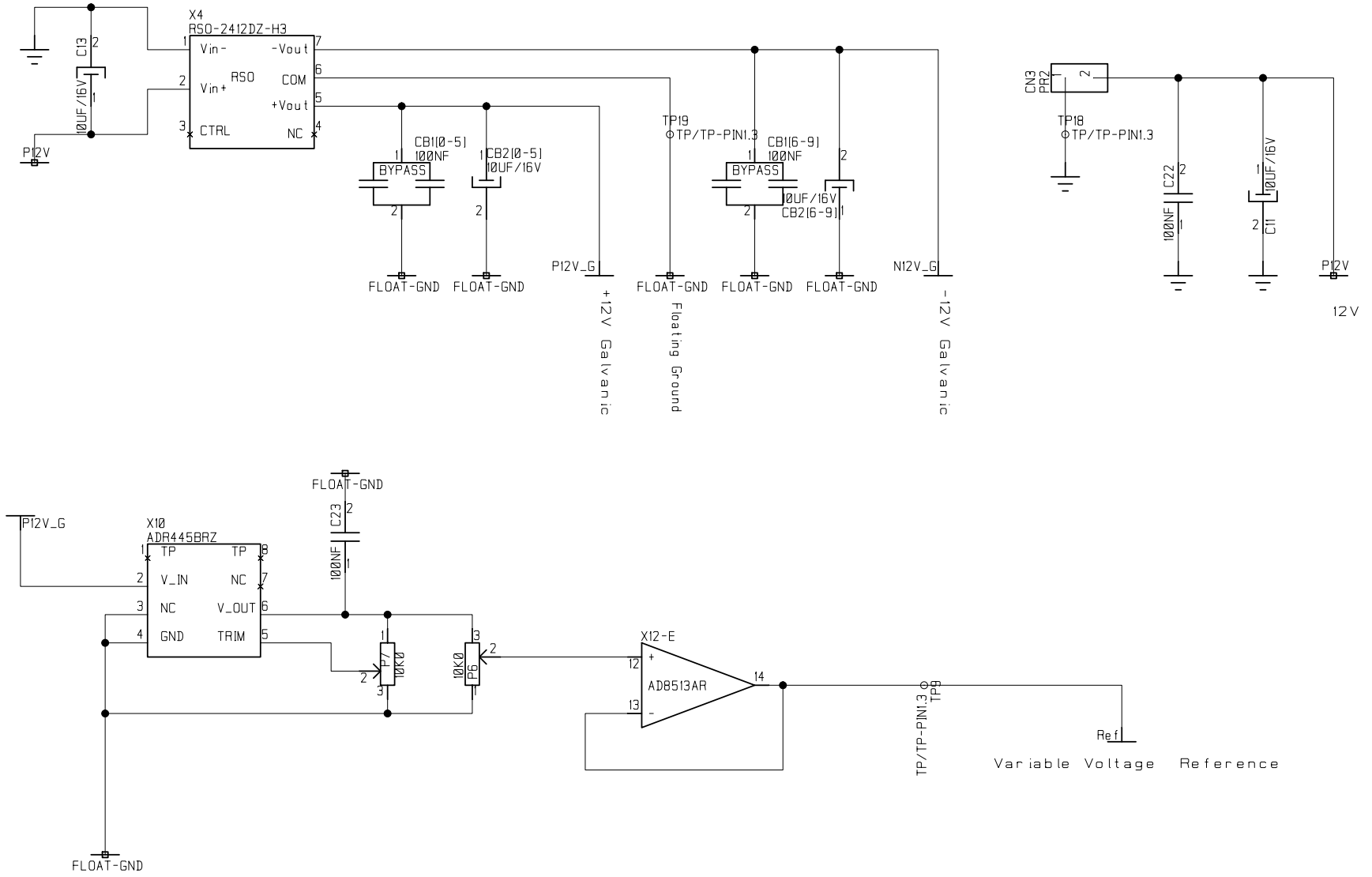
Analog Front-end Files

H.1 Schemetics Analog Front-end Files

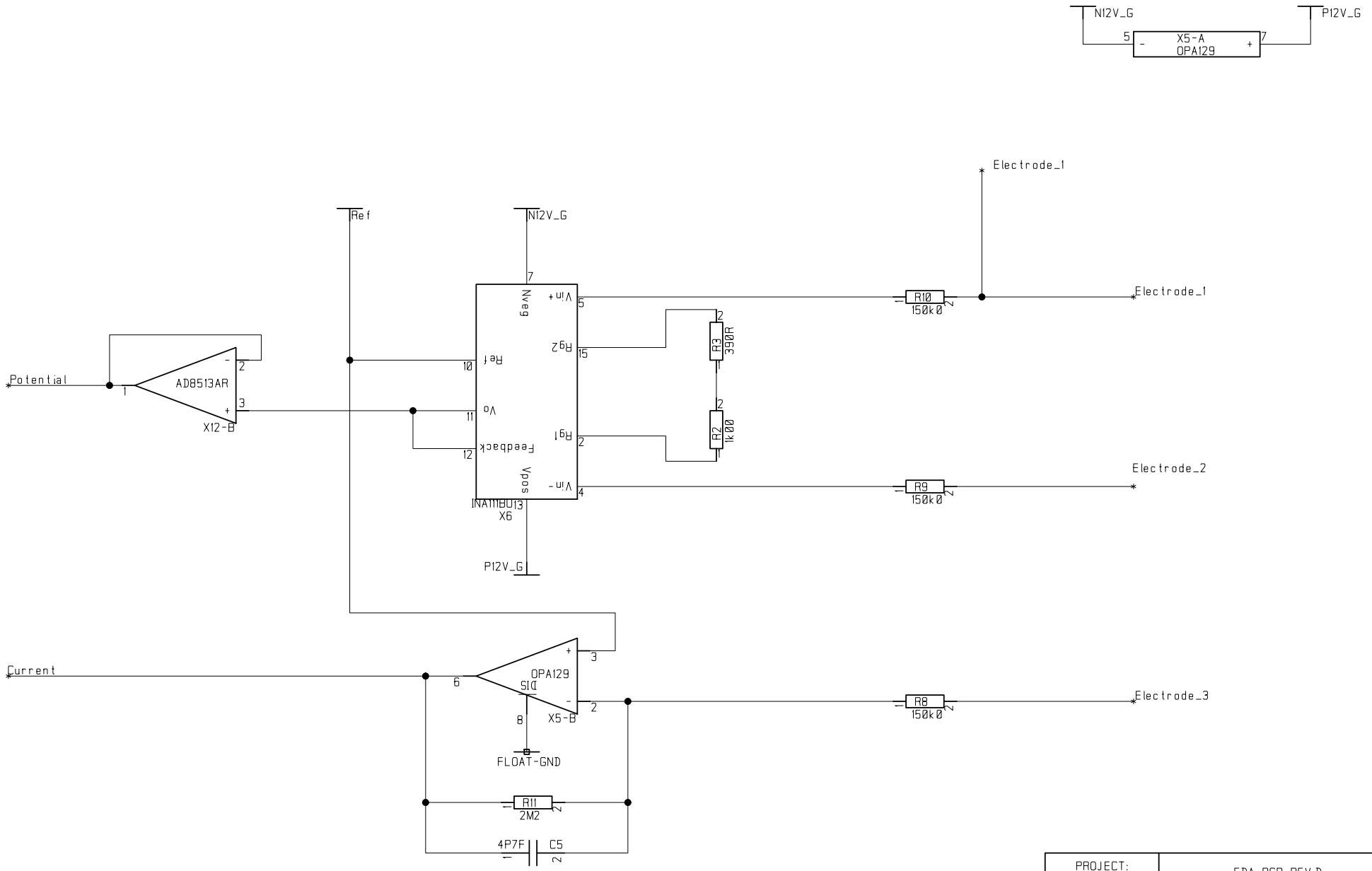

Galvanic Isolation Barrier
POWER
SH1
⊗TP/SH-3.0mm
SH2
⊗TP/SH-3.0mm
SH3
⊗TP/SH-3.0mm
SH4
⊗TP/SH-3.0mm
Current Pump
Signal_In
Electrode_1
TP17
TP/TP-PIN1.3
Signal_In
Optocouplers
SIN In
Potential_Out
Current_Out
CN1
SMA (VER)
CN2
SMA (VER)
CN4
SMA (VER)
TP5
TP/TP-PIN1.3
TP6
TP/TP-PIN1.3
TP7
TP/TP-PIN1.3
2
2
2
SH5
⊗TP/SH-8.5mm/NP
SH6
⊗TP/SH-8.5mm/NP
SH7
⊗TP/SH-8.5mm/NP
Electrode_1
Preamplifier
TP1
TP/TP-PIN1.3
TP2
TP/TP-PIN1.3
TP3
TP/TP-PIN1.3
Electrode_1
Electrode_2
Electrode_3
Potential
Current
TP4
TP/TP-PIN1.3
TP14
TP/TP-PIN1.3
Potential
Current
PROJECT:
EDA_PCB_REV.D
DRAWING:
DATE:10.02.2013
VER: REV.D
SHEET:
UNIVERSITY OF OSLO
Dept. of Physics
(c) copyright. All rights reserved.



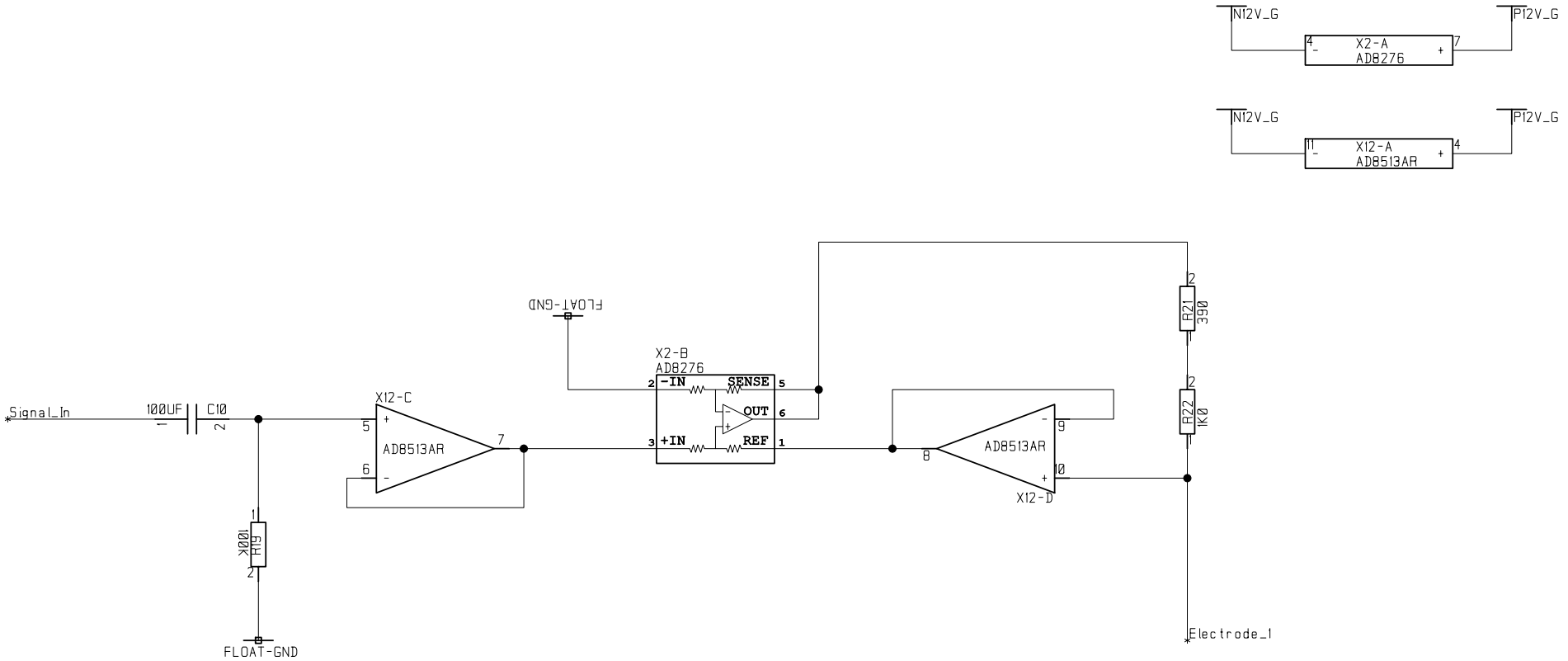
PROJECT:	EDA_PCB_REV.D
DRAWING:	
DATE: 10.02.2013	UNIVERSITY OF OSLO
VER: REV.D	Dept. of Physics
SHEET:	(c) copyright. All rights reserved.



PROJECT:	EDA_PCB_REV.D
DRAWING:	
DATE: 10.02.2013	UNIVERSITY OF OSLO
VER: REV.D	Dept. of Physics
SHEET:	



PROJECT:	EDA_PCB_REV.D
DRAWING:	
DATE: 10.02.2013	UNIVERSITY OF OSLO
VER: REV.D	Dept. of Physics
SHEET:	(c) copyright. All rights reserved.

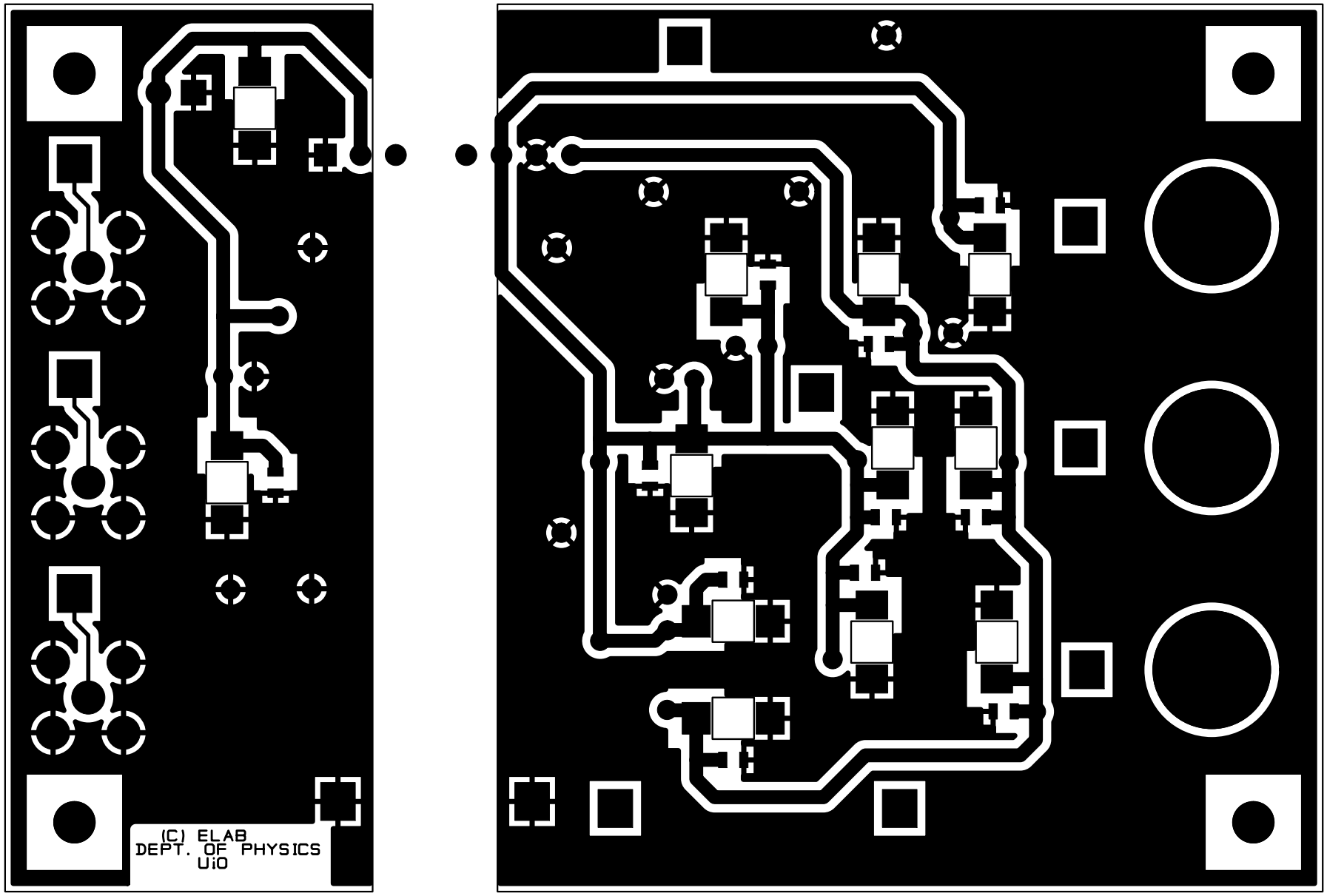


PROJECT:	EDA_PCB_REV.D
DRAWING:	
DATE: 10.02.2013	UNIVERSITY OF OSLO
VER: REV.D	Dept. of Physics
SHEET:	(c) copyright. All rights reserved.

H.2 PCB Analog Front-end

IN
TUO_B
TUO_C
IN
B
E1
E2
E3
C
Ref
EDA_VGB
Prototype 3
(C)
Lars Jørgen Aamodt

TopElec


(C) ELAB
DEPT. OF PHYSICS
UiO
Bo tE le c

Parts List

CADSTAR Design Editor Version 13.0.0.1

Design: M:\Skole\Master\EDA_PCB\EDA_PCB_REV.E\EDA_PCB_REV.E.scm

Design Title:
ELAB-2012

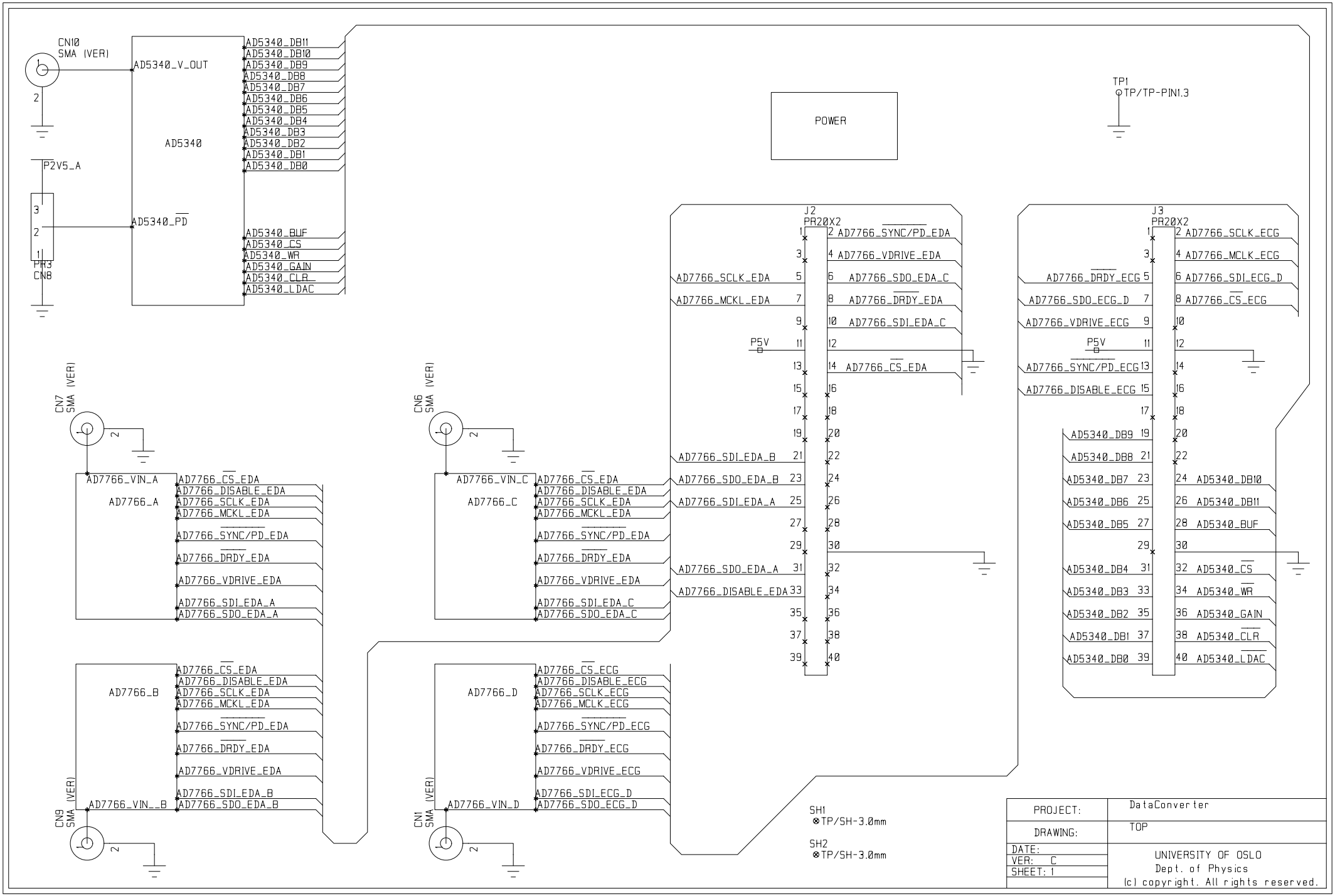
Date: 4. juni 2013
Time: 20:46

Part Name	Part Number	Description	Qty.	Comps.
AD-SSM/ADR445BRZ	D-ADR445BRZ-ND	Ultralow Noise, LDO XFET Volta	1	X10
BB/INA111AP/SMD	D-INA111BU-ND	BURR BROWN FAST FET. INSTR. OP	1	X6
CAP/100NF/0603R	E-65-759-63	10% 16V 0603 X7R	2	C22-23
CAP/100UF/SMD1210R	F-1759457	20% 1210 X7R	1	C10
CAP/4P7F/0603R	E-65-749-09	+/-0.25pF 50V 0603 NP0	7	C5
				C7-9
				C12
				C14-15
CAP/BYPASS/0603R	E-65-759-63	10% 16V 0603 X7R	10	CB10-19
CON/PR2	E-43-702-19	2 SCOTT ELEC. PINROW	1	CN3
CON/SMA_VER/	E-46-123-13	RADIALL SMA-CONNECTOR, PCB	3	CN1-2
				CN4
OPAMP/AD8276ARMZ/MSOP-08		Low power, Wide supply range,	1	X2
** New part - not verified **				
OPAMP/AD8513AR/SMD-S	F-8397970	QUAD JFET-OPAMP 3Mhz. 13V/uS.	1	X12
OPAMP/OP413ESZ	D-OP413ESZ-ND	QUAD IC OPAMP GP 3.4MHZ LN 40M	2	X3
				X8
OPAMP/OPA129/SMD	D-OPA129U-ND	TI OPA129	1	X5
OPTO/HCNR201-500E	D-516-1773-1-ND	High-Linearity Analog Optocoup	3	X1
				X7
				X9
POT/10K0/44W/SMD	E-64-324-62	BI-TECH VERT.MULTI-T.TR.POT	2	P6-7
POT/200K/44W/SMD	E-64-325-04	BI-TECH VERT.MULTI-T.TR.POT	3	P3
				P5
				P8
POW/RSO-2412DZ-H3	D-945-1332-ND	DC/DC-CONVERTER Isoolation of	1	X4
RES/0R00/0603R	E-60-440-02	RESISTOR KOA 0603 1% 0.1W	4	R2
				R8-10
RES/100K/0603R	E-60-452-64	RESISTOR KOA 0603 1% 0.1W	4	R4
				R12
				R14
				R19
RES/10K0/0603R	E-60-450-25	RESISTOR KOA 0603 1% 0.1W	1	R22
RES/1K00/0603R	E-60-447-88	RESISTOR KOA 0603 1% 0.1W	3	R5-7
RES/220K/0603R	E-60-453-48	RESISTOR KOA 0603 1% 0.1W	1	R11
RES/240K/0603R	E-60-453-55	RESISTOR KOA 0603 1% 0.1W	1	R21
RES/51K0/0603R	E-60-451-99	RESISTOR KOA 0603 1% 0.1W	1	R3
TANT/10UF/16V/SMD	E-67-725-52	TANTAL ELECTROLYTIC CAP	12	C11
				C13
				CB20-29

Appendix I

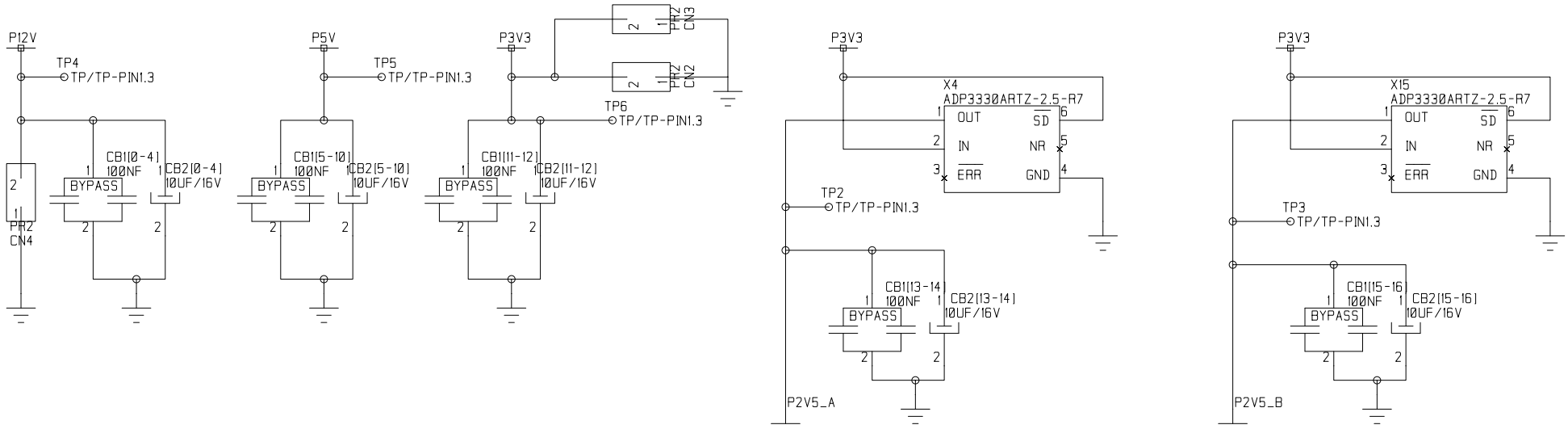
Data Acquisition Card Production Files

I.1 Schemetics

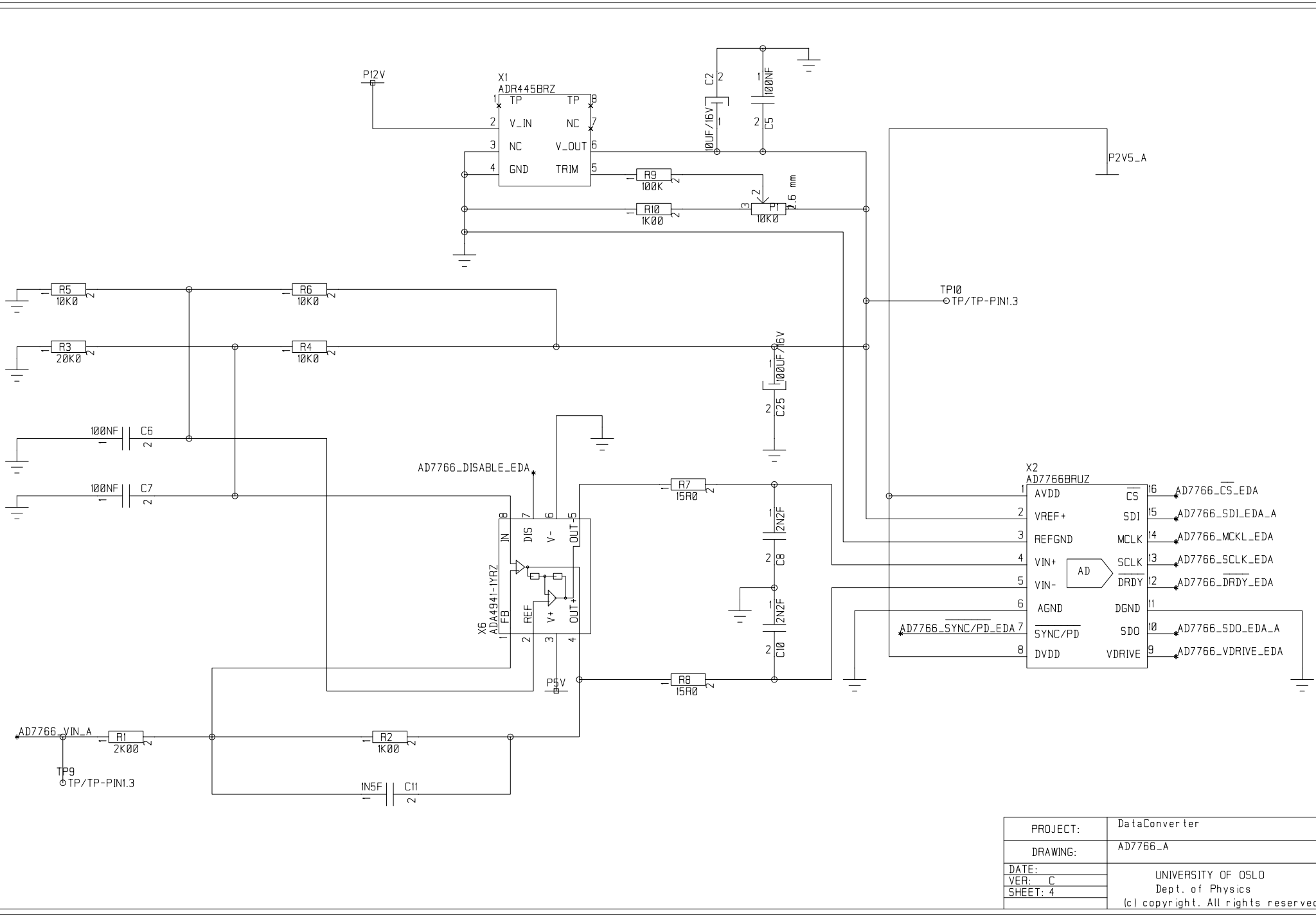


PROJECT:	DataConverter
DRAWING:	TOP
DATE:	
VER: C	
SHEET: 1	

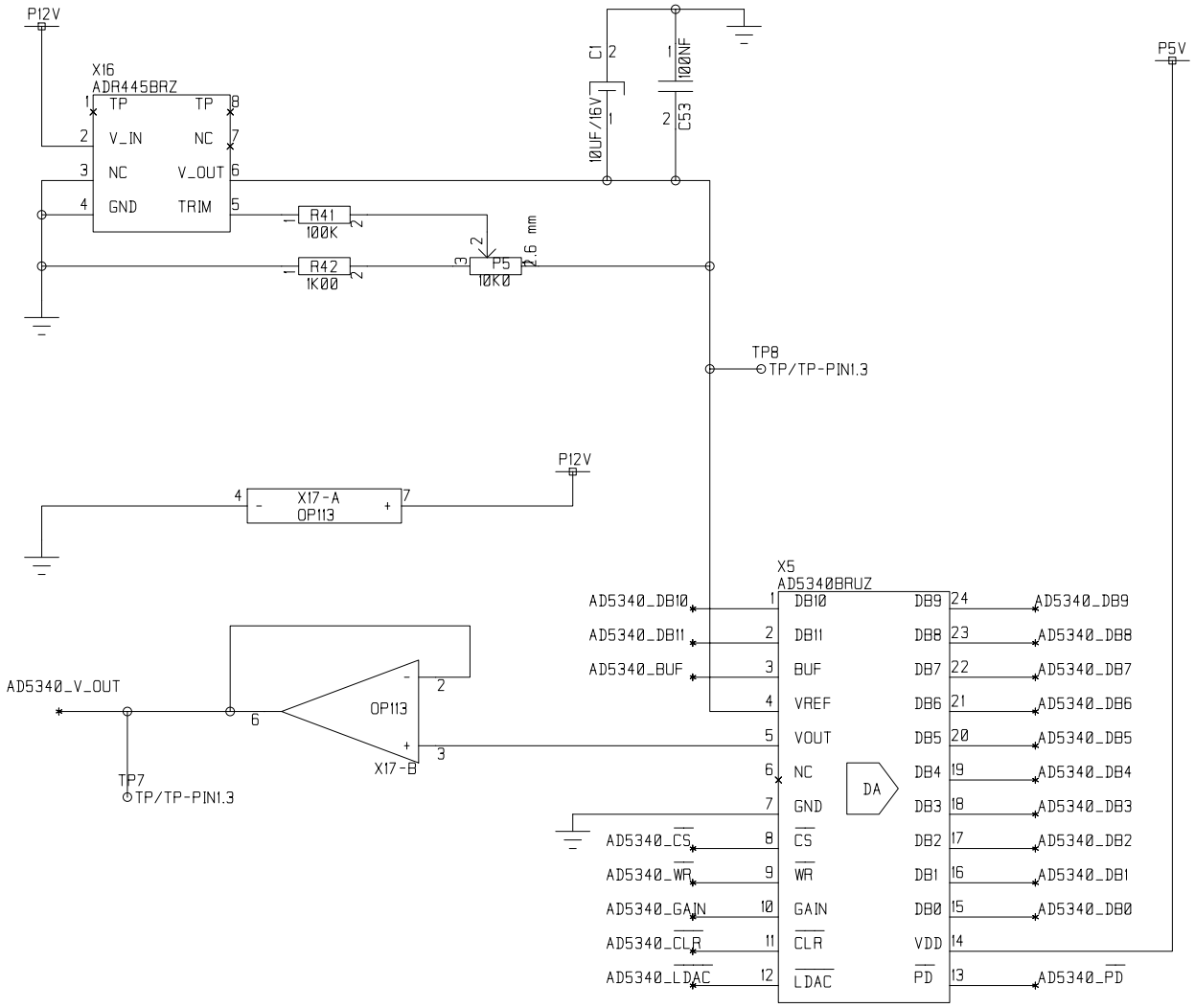
UNIVERSITY OF OSLO
Dept. of Physics



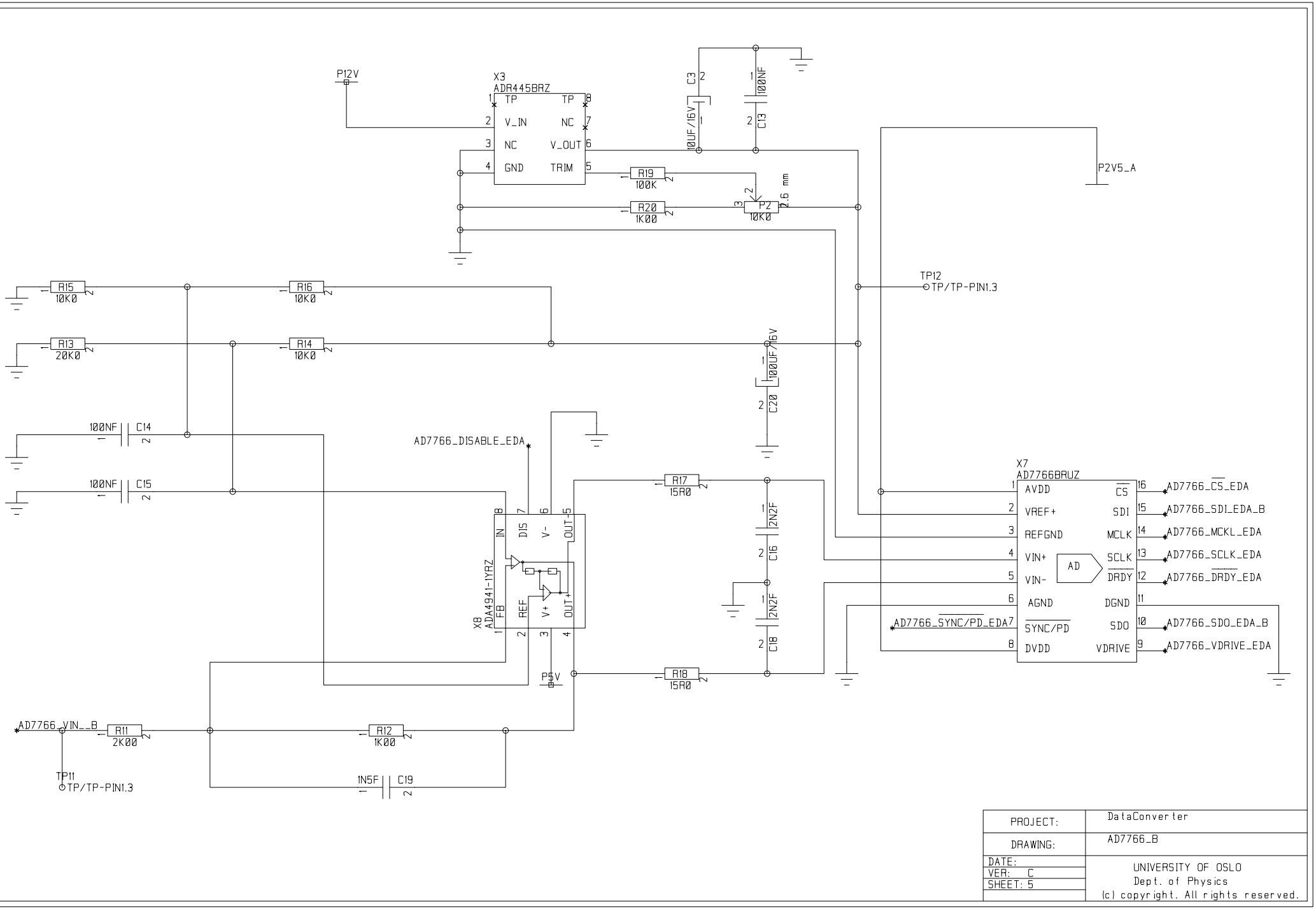
PROJECT:	DataConverter
DRAWING:	POWER
DATE:	
VER: C	UNIVERSITY OF OSLO
SHEET: 2	Dept. of Physics

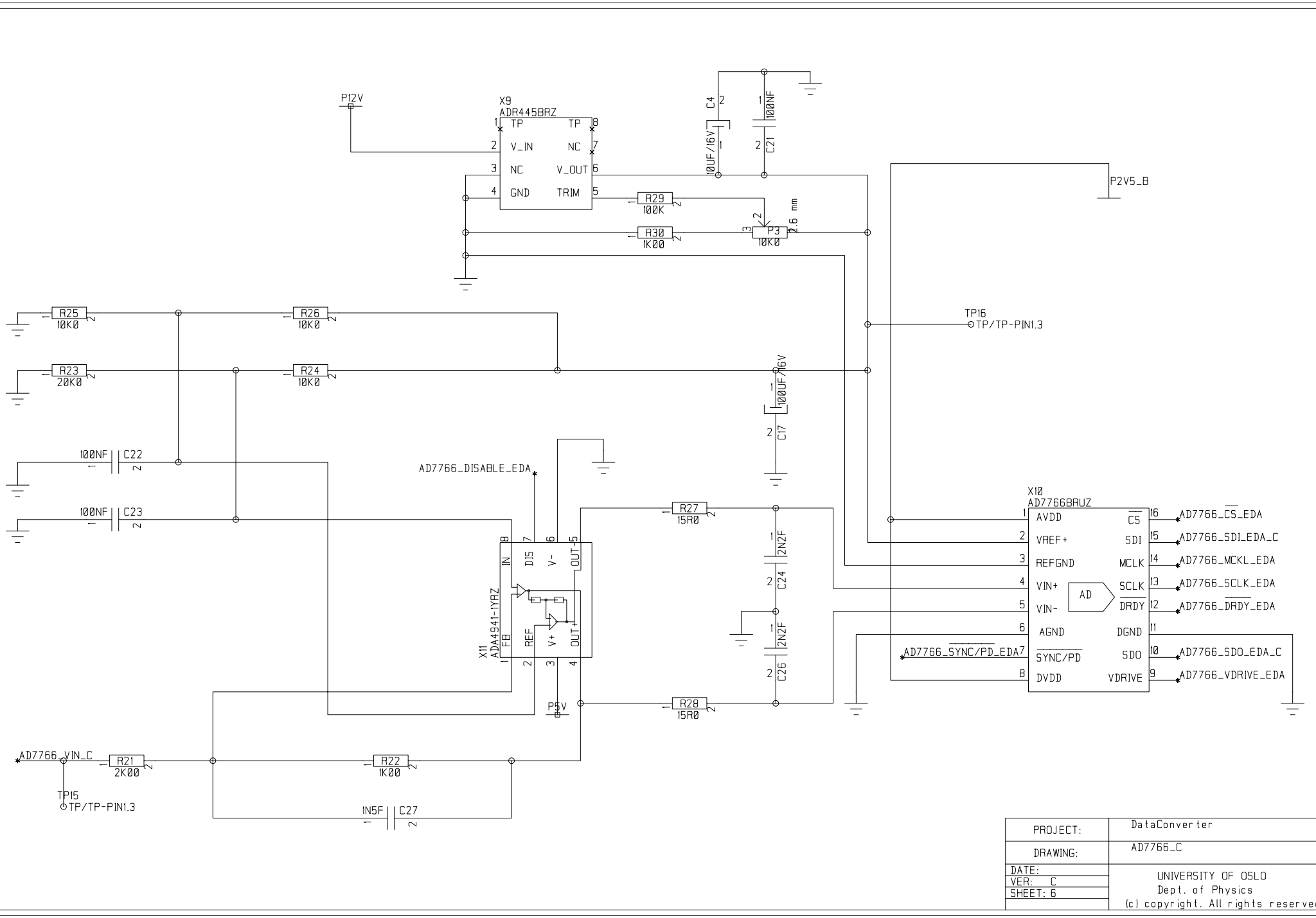


PROJECT:	DataConverter
DRAWING:	AD7766_A
DATE:	UNIVERSITY OF OSLO
VER: C	Dept. of Physics
SHEET: 4	(c) copyright. All rights reserved.

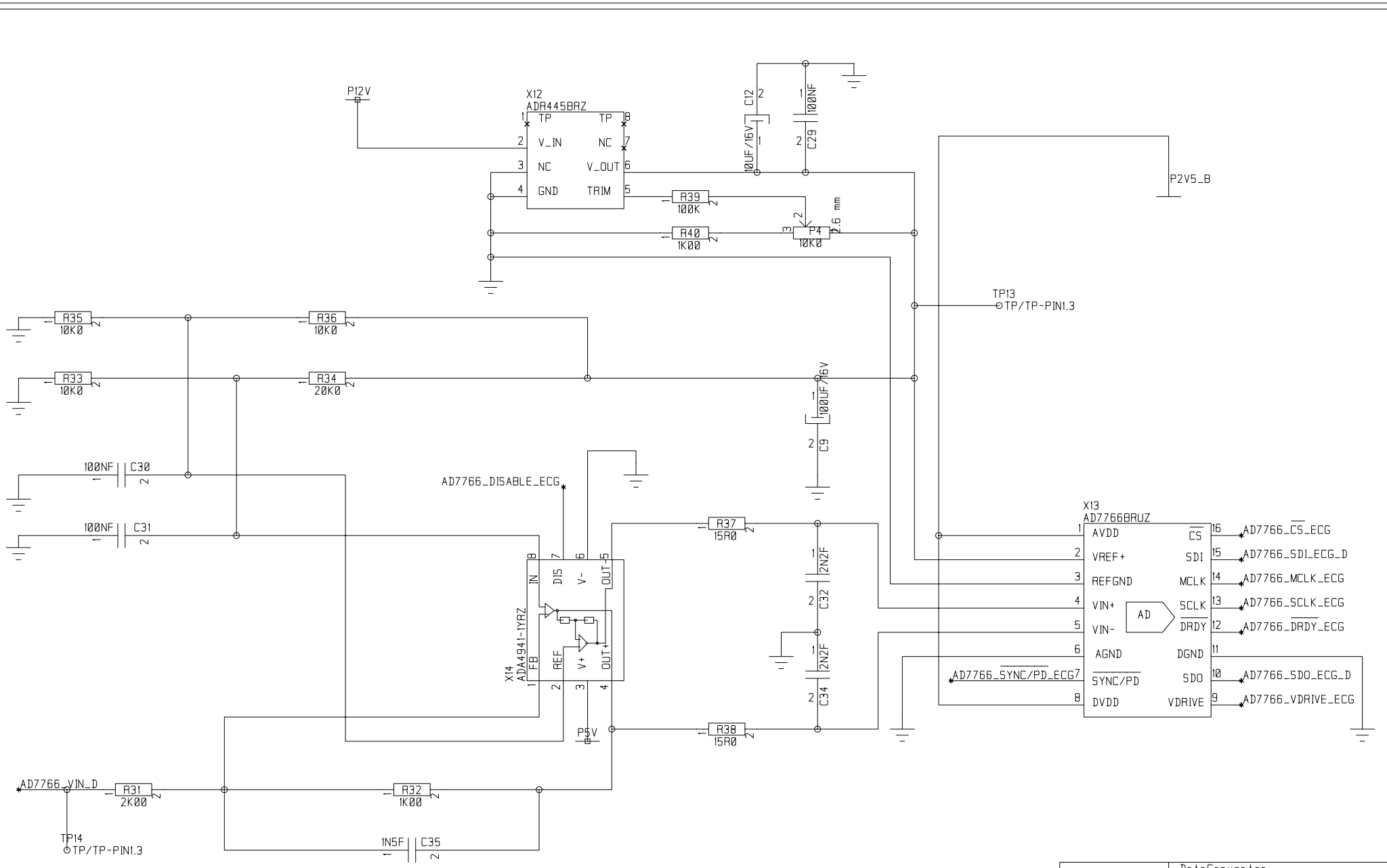


PROJECT:	DataConverter
DRAWING:	AD5340
DATE:	UNIVERSITY OF OSLO
VER: C	Dept. of Physics
SHEET: 3	(c) copyright. All rights reserved.


P12V
X3
ADR445BRZ
TP
V_IN
NC
GND
V_OUT
TRIM
C3
10UF/16V
100NF
C13
R19
100K
R20
1K00
P2
10K0
2.6 mm
P2V5_A
TP12
TP/TP-PIN1.3
R15
10K0
R16
10K0
R13
20K0
R14
10K0
100UF/16V
C20
100NF
C14
100NF
C15
AD7766_DISABLE_EDA
X8
ADA4941-1YRZ
IN
DIS
V-
OUT
FB
REF
V+
OUT+
R17
15R0
R18
15R0
2N2F
C16
2N2F
C18
P5V
X7
AD7766BRUZ
AVDD
VREF+
REFGND
VIN+
VIN-
AGND
SYNC/PD
DVDD
CS
SDI
MCLK
SCLK
DRDY
DGND
SDO
VDRIVE
AD
AD7766_CS_EDA
AD7766_SDI_EDA_B
AD7766_MCKL_EDA
AD7766_SCLK_EDA
AD7766_DRDY_EDA
AD7766_SDO_EDA_B
AD7766_VDRIVE_EDA
AD7766_SYNC/PD_EDA
AD7766_VIN__B
R11
2K00
R12
1K00
TP11
TP/TP-PIN1.3
1N5F
C19
PROJECT:
DataConverter
DRAWING:
AD7766_B
DATE:
VER: C
SHEET: 5
UNIVERSITY OF OSLO
Dept. of Physics
(c) copyright. All rights reserved.



PROJECT:	DataConverter
DRAWING:	AD7766_C
DATE:	
VER: C	UNIVERSITY OF OSLO
SHEET: 6	Dept. of Physics

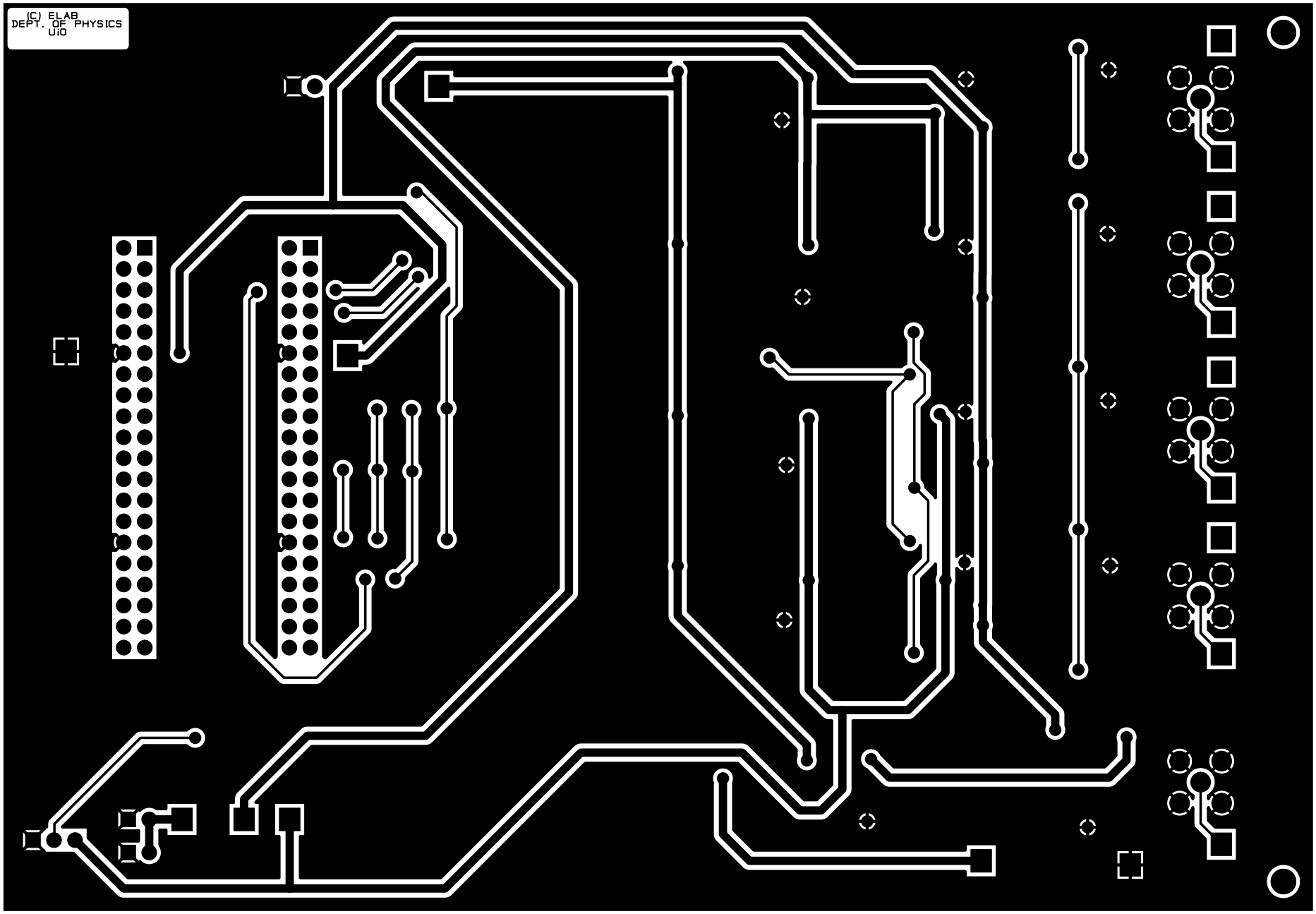


PROJECT:	DataConverter
DRAWING:	AD7766_D
DATE:	UNIVERSITY OF OSLO
VER: C	Dept. of Physics
SHEET: 7	(c) copyright. All rights reserved.

I.2 PCB

DataConverter
REV.C
26.11.2012
(C)
Lars Jørgen Aamodt
Mirjam Husebø

TopElec


(C) ELAB
DEPT. OF PHYSICS
UiO
Bot Elec

I.3 Part List

Parts List

CADSTAR Design Editor Version 13.0.0.1

Design: C:\Users\ljaamodt\DataConverter_REV.C.scm

Design Title:
EIAB-2012

Date: 4. juni 2013
Time: 20:51

Part Name	Part Number	Description	Qty.	Comps.
AD-SSM/AD5340BRUZ	D-AD5340BRUZ-ND	DAC 12BIT Parallel interface.	1	X5
AD-SSM/AD7766BRUZ	D-AD7766BRUZ-2-ND	ADC 24BIT 32kSPS SAR	4	X2 X7 X10 X13
AD-SSM/ADA4941-1YRZ	D-ADA4941-1YRZ-ND	Single-Supply, Differential 18	4	X6 X8 X11 X14
AD-SSM/ADP3330ART2	D-ADP3330ART2-2.5-R7CT-ND	High Accuracy Ultralow IQ, 200	2	X4 X15
AD-SSM/ADR445BRZ	D-ADR445BRZ-ND	Ultralow Noise, LDO XFET Volta	5	X1 X3 X9 X12 X16
CAP/100NF/0603R	E-65-759-63	10% 16V 0603 X7R	13	C5-7 C13-15 C21-23 C29-31 C53
CAP/1N5F/0603R	E-65-757-40	10% 50V 0603 X7R	4	C11 C19 C27 C35
CAP/2N2F/0603R	E-65-757-65	10% 50V 0603 X7R	8	C8 C10 C16 C18 C24 C26 C32 C34
CAP/BYPASS/0603R	E-65-759-63	10% 16V 0603 X7R	17	CB10-19 CB110-116
CON/PR2	E-43-702-19	2 SCOTT ELEC. PINROW	3	CN2-4
CON/PR20X2	E-43-701-10	20X2 SCOTT ELEC. PINROW	2	J2-3
CON/PR3	E-43-702-19	3 SCOTT ELEC. PINROW	1	CN8
CON/SMA_VER/	E-46-123-13	RADIALL SMA-CONNECTOR, PCB	5	CN1 CN6-7 CN9-10
OPAMP/OP113/SMD	X-XX-XXX-XX	SINGLE LOWNOISE OPAMP	1	X17
POT/10K0/23B/SMD	E-64-320-90	BI-TECH CERMET TR.POT	5	P1-5
RES/100R/0603R	E-60-452-64	RESISTOR KOA 0603 1% 0.1W	5	R9 R19 R29 R39 R41
RES/10K0/0603R	E-60-450-25	RESISTOR KOA 0603 1% 0.1W	12	R3 R5-6 R13 R15-16 R23 R25-26 R33 R35-36
RES/15R0/0603R	E-60-443-41	RESISTOR KOA 0603 1% 0.1W	8	R7-8 R17-18 R27-28 R37-38
RES/1K00/0603R	E-60-447-88	RESISTOR KOA 0603 1% 0.1W	9	R2 R10 R12 R20 R22 R30 R32 R40 R42
RES/20K0/0603R	E-60-450-90	RESISTOR KOA 0603 1% 0.1W	4	R4 R14 R24 R34
RES/2K00/0603R	E-60-448-53	RESISTOR KOA 0603 1% 0.1W	4	R1 R11 R21 R31
TANT/100UF/16V/SMD	E-67-133-25	SANYO CV-GX SMD ELYT	4	C9 C17 C20 C25
TANT/10UF/16V/SMD	E-67-725-52	TANTAL ELECTROLYTIC CAP	22	C1-4 C12 CB20-29 CB210-216