Analog Integrated Circuit Applications

by

Long Nguyen, Robert Perry, Lou Seneres, Neda Seyedmahmoud

Submitted to the Faculty of the

WORCESTER POLYTECHNIC INSTITUTE

In partial fulfillment of the requirements for the

Degree of Bachelor of Science in

Electrical and Computer Engineering

Long Nguyen	Robert Perry
Lou Seneres	Neda Seyedmahmoud

Professor John McNeill
MQP Advisor

Abstract

The goal of this project is to design an ultra-low power and ultra-portable wireless biomedical signal chain. This project provides an adaptable electrocardiogram (ECG) reference design that can be modified to accept other bio-potential signals. The designed ECG monitoring system utilizes the AD8232 from Analog Devices featuring low noise and adaptable filtering as well as the nRF51422 for Bluetooth Low Energy (BLE) signal transmission. The resulting design addresses the lack of commercial products that can transmit a conditioned signal over-the-air by providing an adaptable reference design that others can utilize to expedite research and development.

Acknowledgements

The completion of our project would not be accomplished without the sponsorship of the New England Center for Analog and Mixed Signal Integrated Circuit Design (NECAMSID) lab.

We would like to thank the following individuals for their contributions:

- Michael Coln from Analog Devices Inc
- David Plourde from Analog Devices Inc
- Ina Duka from Analog Devices Inc
- Bob Boisse at WPI ECE Shop

Finally, we would like to extend our sincerest thanks to our Major Qualifying Project advisor, Professor John McNeill for his patience, constant support, and guidance throughout this project.

Contents

List of Figures

List of Tables

1 Introduction

The initial purpose of this project was to investigate and define an application for an integrated circuit from Analog Devices Incorporated (ADI). Analog Devices provided information regarding a capacitive application [21], but it was concluded that the depth of that particular application would be more indicative of a graduate thesis than an undergraduate Major Qualifying Project (MQP). Therefore, the project transformed from a single application into a broad range of applications with the use of Analog Devices chip in the final design. At the direction of our advisor Professor McNeill, the team began to generate ideas related to biomedical applications. These preliminary ideas can be seen in Table 1.1.

Table 1.1: Decision Matrix for Possible Project Ideas

Idea	Feasibility	Uniqueness	Presentability	Complexity	Cost	Average
Portable ECG Sensor	8	8	8	6	7	7.4
Biovisualization	5	5	1	7	10	5.6
Pulse or Blood Flow Sensor	8	2	8	3	3	4.8
Traffic Flow Sensor	2	3	0	6	10	4.2

The ideas in Table 1.1 were rated in several categories such as feasibility, uniqueness, presentability, complexity, and cost on a scale of 1 to 10 with 1 being favorable and 0 being most unfavorable. For example, a 0 in the feasibility category would mean that the design is not feasible. Complexity is rated from 0 to 10 with 10 being the most favorable for design complexity.

1.1 Problem Statement

Based on the 2008 data by American Heart Association (AHA), an average of more than 2150 Americans die of Cardiovascular Disease (CVD) each day, or average of 1 death every 40 seconds. The cost of CVD is estimated to be 320 billion USD in 2011 [22]. According to this data the CVD requires extra attention and monitoring. This project addresses this problem by designing an electrocardiogram (ECG) monitoring device.

The widespread usage of health monitoring devices in the United States has increased the demand for smart portable health monitoring systems. Depending on the application of the systems, portable biomedical monitoring devices for temperature, blood pressure and glucose level measurements are commonly used and are widely available to the users. Commercially available wireless portable biomedical acquisition and transmission devices for measuring ECG signal are limited. This project attempts to create a reference design for untethered and portable ECG device and user interface that can be modified for other signals.

1.2 Project Definition

The goal of this project is to design an ultra low power, and ultra portable system. The project started with a desired system that includes five blocks: Power, Analog Front End (AFE), Microcontroller, Communications, and User Interface (UI) as shown in Figure 1.1. The project topic and specifications were chosen after a research and brainstorming phase. After this stage, the group decided upon designing a wireless biomedical application system that matches the design requirements. The design is mainly concerned with ECG measurements. However, it can be expanded to measure other small biopotential signals. The ECG was chosen for the purpose of this project because it is feasible and easy to measure with simple snap electrodes on hands or the chest.

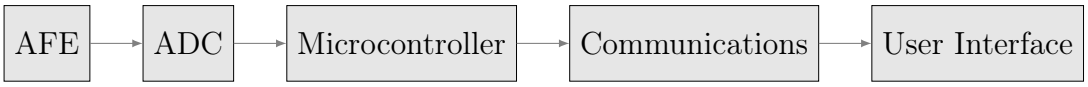


Figure 1.1: System Level Block Diagram

Figure 1.1 illustrates the system level, or rather very general, potential modules for the project. These were group according to functionality into four main modules such as Analog Front End, Communications (which is composed of the ADC, Microcontroller, and Wireless Communications Protocol), Power, and User Interface. Each section will be briefly described and analyzed in the next subsections.

1.2.1 Analog Front End

When it comes to biomedical signals, signal integrity and precise measurements are necessary components to ensure the accuracy of monitoring and diagnosis. The purpose of this block is to amplify and filter ECG signal to eliminate the motion artifacts. The amplitude of ECG ranges from $0.1mV$ to $1mV$ and its bandwidth is from 0.2Hz to 200Hz, therefore the AFE must have the capability to reject the common-mode noise [23]. Given the small signal for the ECG, an amplifier can be utilized to amplify the low signal without adding appreciable noise. Also, appropriate low pass filters can be used to remove the unwanted high frequency noise signals if necessary.

1.2.2 Communications

The communication block of the design is responsible for transmitting the acquired ECG raw signal to the user interface for monitoring and diagnosis. Other functionality of this block includes data conversion and processing. The ECG amplitude is small therefore, the analog to digital conversion before transferring can be done using a Successive Approximation (SAR) ADC of 10 to 12 bits precision. The communication method chosen for the desired portable system is untethered, in order to provide a user friendly experience for the end users. The

ECG monitoring signal needs to remain close to the body with wireless communication that would make it more feasible to connect to different user interfaces without changing the hardware configuration of the system.

1.2.3 Power

Due to the low power and portability restraints of the system design, quantifying the system power is of particular importance. The system should maximize battery life, while attempting to minimize size of the storage medium. In addition to maintaining a low power ECG monitoring system, the system power block must fulfill the basic safety, lifetime, portability and the size consideration of the desired system. It is desirable for the runtime to be sufficiently long such that the user does not need to worry about changing the battery except throughout the week. However, if this is not possible, an indicator to charge or replace the battery should be in place to give the user ample time to change the battery. This project explores the usage of either a primary or secondary cell depending upon the current draw of the design as a lifetime of more than a few days. These properties will be investigated further in Chapter 4.

1.2.4 User Interface

The user interface need to be easy-to-use. It shall also allow for direct and continuous observation of ECG results collected from the other blocks. For the purposes of this project, a desktop or smartphone application can be utilized to demonstrate the ECG signal due to its increasing popularity and availability. Different mobile platforms and other alternatives will be considered for user interface.

2 Analog Front End

Bio-potentials are very weak signals compared to environmental noise sources and Analog-to-Digital Converters (ADC) are not able to process such small signals. Therefore, for proper signal acquisition, a system generally requires an Analog Front-End (AFE) to condition the acquired signal. This project focuses on ECG signal acquisition, so the AFE block is an important requirement for the designed system. The AFE block must be easy to integrate within the system and specifically with the system's ADC. The ECG signal is also susceptible to noise interference and motion artifacts. As a result, the signal conditioning must include the extraction, gain and filtering stages. The purpose of this chapter is to provide information regarding AFE block of the system and its specifications.

2.1 Background Research

2.1.1 Monitoring ECG - Electrode Location and Waveform Quality

In order to have a better understanding of the ECG monitoring systems, a background research was done on the electrode location and ECG waveform quality. ECG system has two or more electrodes used to monitor voltage across one or more leads. Clinical standard uses a 12-lead ECG that would each provide a view of the heart's electrical activity from different views. Each lead corresponds to a vector of ECG electrical potential. The electrodes are typically placed in a setup known as the Einthoven's triangle, which is formed by the right arm (RA), left arm (LA) and left leg (LL) sensing electrodes as shown in Figure 2.1 below. These are the basis for the frontal axis.

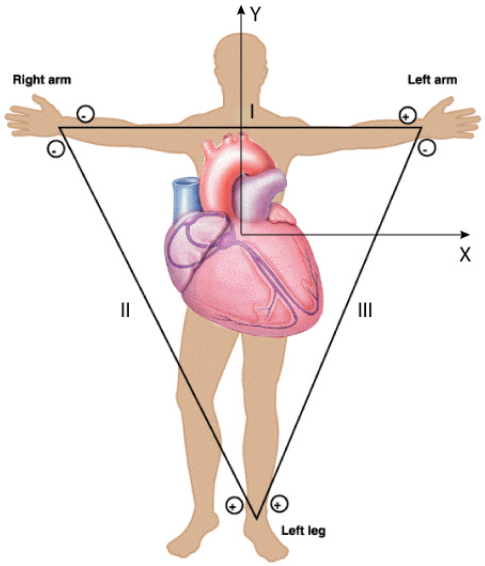


Figure 2.1: Einthoven's Triangle [1]

Different lead location corresponds to different ECG signal as shown in Figure 2.2 below. Lead II is commonly used as it provides largest positive R-wave that is important to measure heart rate.

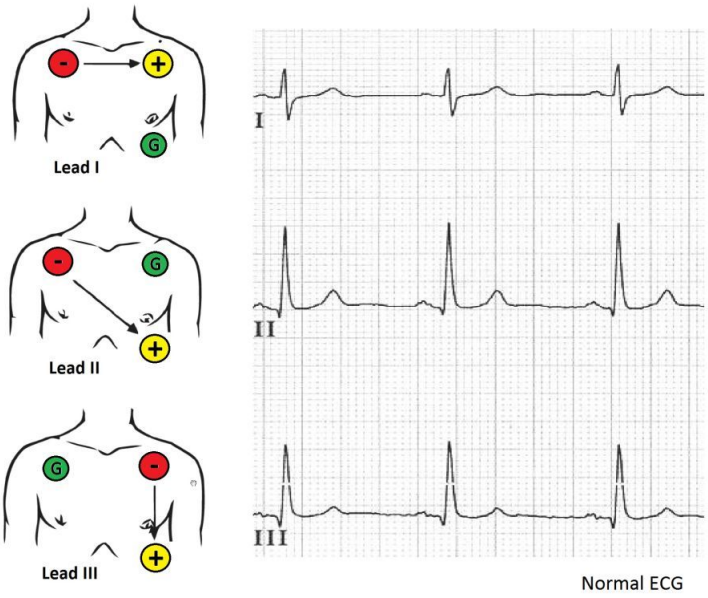
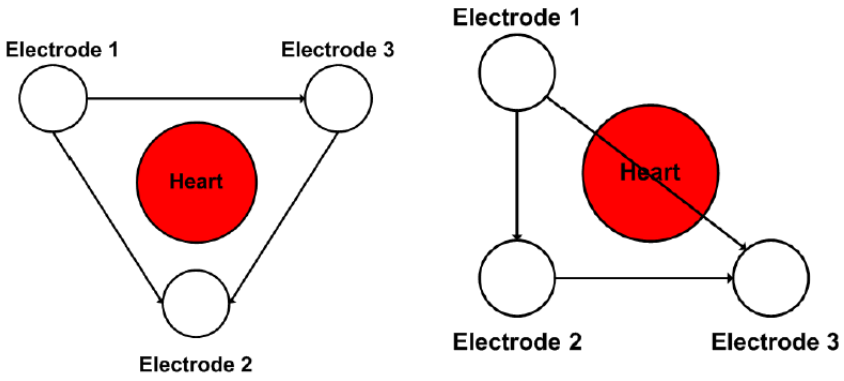


Figure 2.2: Lead Location and ECG Output [2]

Three-electrode Configuration

This configuration would allow a Lead I, Lead II or Lead III to be selected. The two electrodes are used to form a vector while the third electrode is the reference. For three electrodes, it is possible to have up to three unique vectors. This implementation has the advantage having less noise and more lead selection options.

The first option would require three electrodes located around the heart. It has views of Lead I, II and III. One disadvantage is the ease of use as it might have to cross over pectoral muscles and breast tissue. Another alternative aims to provide better comfort for the patient. The trade-off would limited lead view as it only has a good view of Lead II. The two options are shown in Figure 2.5.



(a) Option One (b) Option Two

Figure 2.3: 2-electrode Location Options

In this case, a desired setup is shown in Figure 2.4. This would provide better view of Lead II. The right-leg drive (RLD) electrode (usually a third electrode) is used to bias the patient to a set DC operating point in order to ensure that the input at the same potential as the monitoring system.

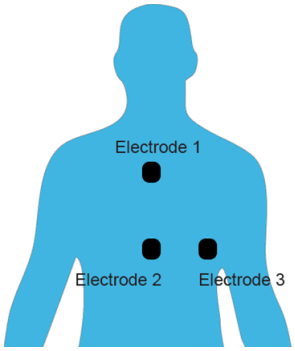
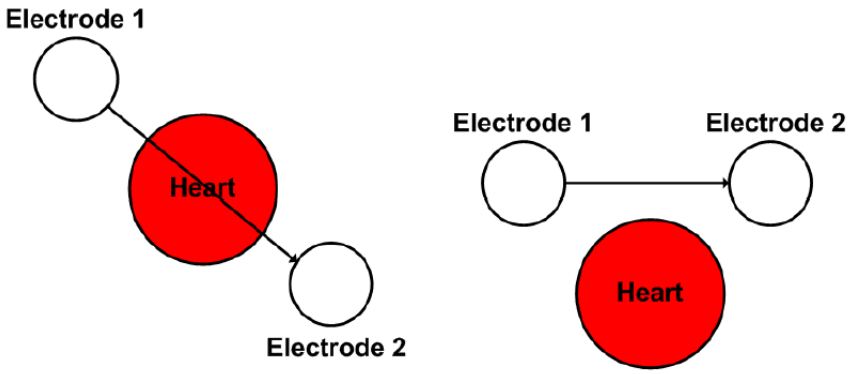


Figure 2.4: 3-electrode Best Setup

Two-electrode Configuration

In this configuration, only one lead is possible. To obtain a Lead II vector, the configuration is shown in Figure 2.5 for option one. Another alternative setup, option two, is also possible but it only has a good view of Lead I. This setup is desired since it is easier for patient.



(a) Option One (b) Option Two

Figure 2.5: 2-electrode Location Options

In this case, an ideal setup is shown in Figure 2.6. This would provide better view of Lead I. One disadvantage is a smaller amplitude of the signal compared to Lead II in three-electrode configuration.

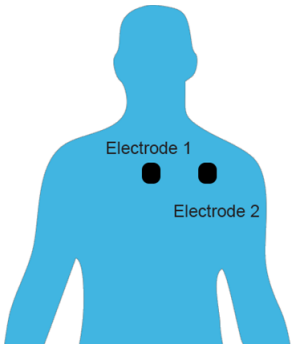


Figure 2.6: 2-electrode Best Setup

2.2 AFE Requirements

The AFE block of an ECG monitoring device must extract low voltage signals ranging from $0.1 - 1mV$ with frequency range of $0.2 - 200Hz$ [23]. The signal extraction is critical in the presence of noisy conditions and various interferences. The designed system is preferred to have the capability to use both two and three-electrode configurations. In ECG measurements, skin electrode contact counts as a source of interference producing an offset of $\pm 200 - 300mV$.

Two type of signals are available at the input of the AFE block, the Common-Mode (same potential) and Differential-Mode (different potential) signals. For ECG signal monitoring applications, the Common-Mode signal is highly undesirable. The potential between the electrodes and ground can create a Common-Mode component of up to $1.5V$ [3]. One requirement of the AFE block is to be able to cancel out common signals and amplify the

differential signal at the input pins. The Common-Mode gain is defined as the ratio of change in output voltage to change in Common-Mode input voltage. The Common-Mode Rejection Ratio (CMRR) is the ratio of the differential gain to the Common-Mode gain. The desired AFE must have a CMRR of $80dB$ to $120dB$ over the input frequencies that need to be rejected [3].

In order to achieve gain as well as Common-Mode rejection, instrumentation amplifier (in-amp) can be employed. An instrumentation amplifier has a differential input and single-ended output with respect to a reference voltage. The in-amp gain is determined by external resistors that are connected between inverting input and the output [3]. Figure 2.7 shows the internal structure of a classic in-amp with 3 operational amplifiers (op-amps).

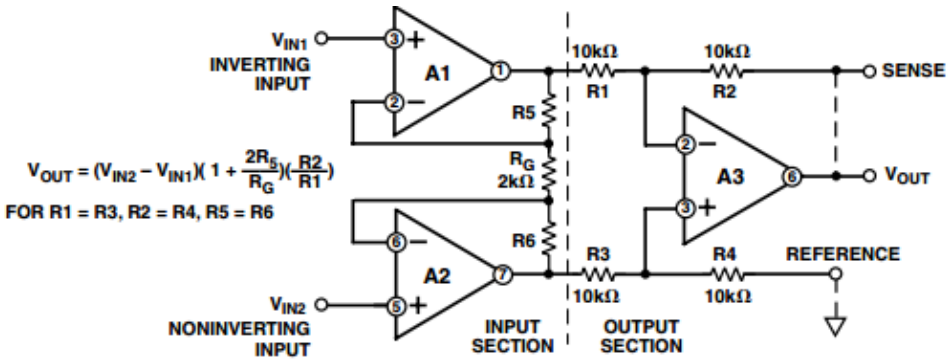


Figure 2.7: The Classic 3 Op-amp In-amp Circuit [3]

2.3 Design Options

This project stipulates that products from Analog Devices are preferred over their competitors' offerings. Analog Devices offers different solutions ranging from general to application-specific chips. After evaluating Analog Device's portfolio of integrated circuits, three design options were considered. These options include two application-specific ICs and a lower power general purpose instrumentation amplifier.

2.3.1 Instrumentation Amplifier

AD8236

Among the instrumentation amplifiers available in the market, the AD8236 has the lowest power consumption [4]. It can operate on a single supply of $1.8V$ or $3V$. Other features of AD8236 are high input impedance, minimum gain of 5, low input bias of $1pA$ and high CMRR of $110dB$. The AD8236 is a monolithic (on a single base semiconductor, referred to as a die) amplifier. In addition to above features, its rail-to-rail input and output provides a wide dynamic range, which means that its maximum input or output swing is equal to

the power supply voltage [3]. The AD8236 REF pin shown in Figure 2.8 allows for shifting the output according to the application. This shifting makes it easier for the AFE block to interface with other blocks of the system such as the ADC. According to its specification, the AD8236 has configurable gain with default value of 5; by placing a resistor across the RG pin, additional gain can be set with respect to the resistor value. The resistor value can be calculated according to Equation 2.1.

$$R_G = \frac{420k\Omega}{G-5} \tag{2.1}$$

All of the pins of the AD8236 are protected against Electrostatic Discharge (ESD) as shown in the simplified schematic Figure 2.8.

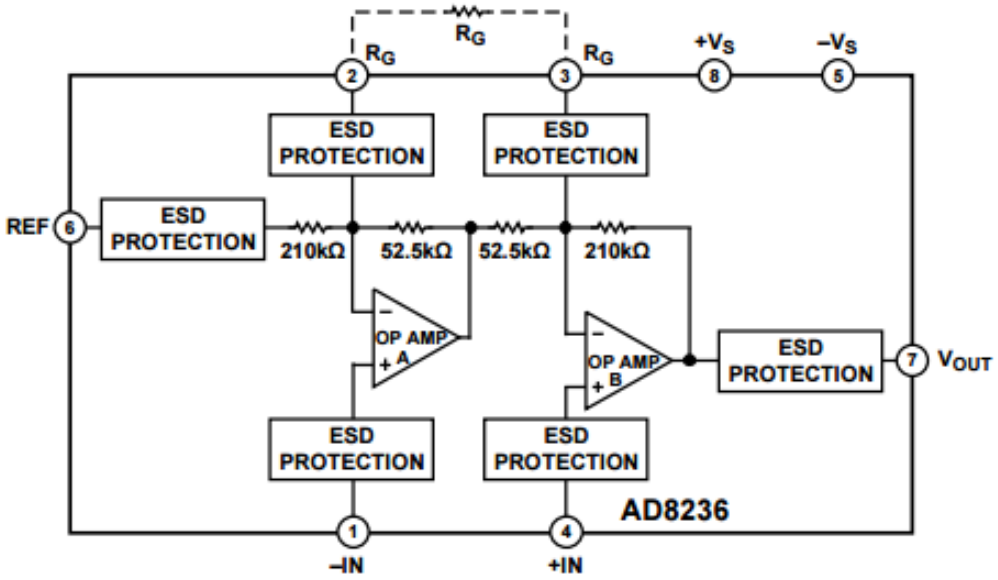


Figure 2.8: Simplified Schematic of AD8236 [4]

The AD8236 has DC overload protection, which allows a diode drop (approximately $0.7V$) above the positive supply and a diode drop below the negative supply. It also can handle a continuous $6mA$ current. The AD8236 has relatively high CMRR ($110dB$) and the capability to remove any differential DC offsets resulted from electrode half-cell potential. These features make AD8236 a good candidate for ECG monitoring applications. The suggested configuration of AD8236 is shown in Figure 2.9.

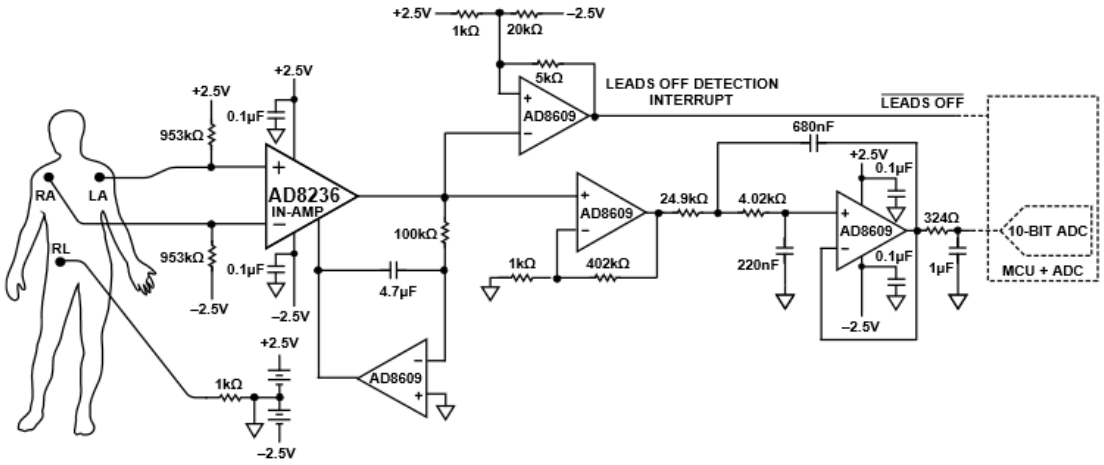


Figure 2.9: AD8236 Heart Rate Monitor Reference Configuration [4]

2.3.2 Specialty Chips

ADAS1000

Purpose

The ADAS1000 is a high-performance, low power analog front end (AFE) IC. This chip ensures the quality of the signal and allows for monitor and diagnostics of ECG signal. It follows the various standards for diagnostic electrocardiograph devices. In addition to measuring the ECG, this chip can measure thoracic impedance and pacing artifacts[5]. The capability of measuring pace artifacts allows for detection of presence of a pacemaker and its effects [24]. Thoracic impedance measurements or the respiration measurement can also be used for monitoring patient breathing. The functional block diagram of the ADAS1000 is shown in Figure 2.10.

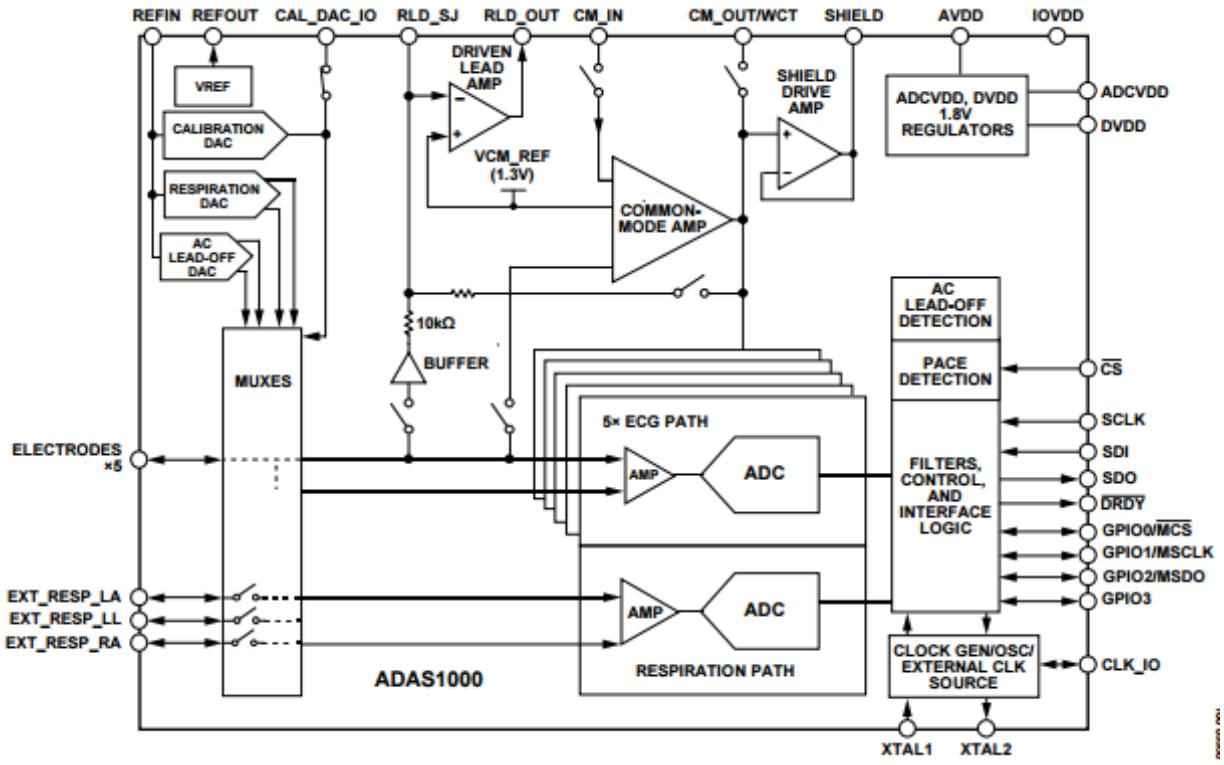


Figure 2.10: ADAS1000 Functional Block Diagram [5]

Input

The ADAS1000 has five input channels and a reference drive, Right Leg Drive (RLD) for ECG signal acquisition. The channels include a differential preamplifier with a programmable gain, a fixed gain 2-pole anti-aliasing filter, a 14 bit 2MHz SAR ADC, and buffers. The implemented anti-aliasing filter is aimed for pace detection. Leads can be configured as either digital and analog.

The RLD amplifier is used to reject the noise and interference and adjust internal reference level and provide maximum input dynamic range[5].

Power Consumption

The ADAS1000 is designed to run off a battery or a standard power line. The chip required supply voltage of $3.3V$ to $5V$. The power consumption ranges from $11mW$ for one-lead to $15mW$ for three-leads, and $21mW$ if all the electrodes are in use. The ADAS1000 has a feature for scaling between noise and power, meaning that the noise can be reduced by increasing the power consumption whenever necessary. It also features power-down mode where the signal acquisition channel can be shut down and the data rates can be reduced to minimize power consumption[5].

Additional Features

The ADAS1000 offers the capability of AC and DC lead off detection. The reference lead can be selected according to the application. It includes externally supplied crystal of ($8.192MHz$) and internal regulators.

A self-test mode is implemented to ensure proper operation of the integrated circuit. The power up testing of the ADAS1000 performs registers reading, Cyclic Redundancy Check (CRC) and calibration of the Digital to Analog Converter (DAC).

Post processing of the aquired data can be performed on a DSP or microcontroller through Serial interface SPI as well as QSPI and DSP. The ADAS1000 transmits the data at programmable data rates either in the form of a data frame supplying either lead or electrode data[5].

AD8232

Purpose

The AD8232 is designed for ultra-low power applications and has the capability to acquire, amplify and filter bio-potential signals. This chip is capable of eliminating the motion artifacts and the electrode half-cell potential. It also features leads off detection circuitry and fast restore ability to prevent a long recovery time resulted from input disconnection. The functional block diagram of the AD8232 is shown in Figure 2.11 indicating that the chip includes four amplifier an in-amp (IA), an operational amplifier (op-amp) A1, a Right Leg Drive (RLD) A2, and a buffer amplifier A3.

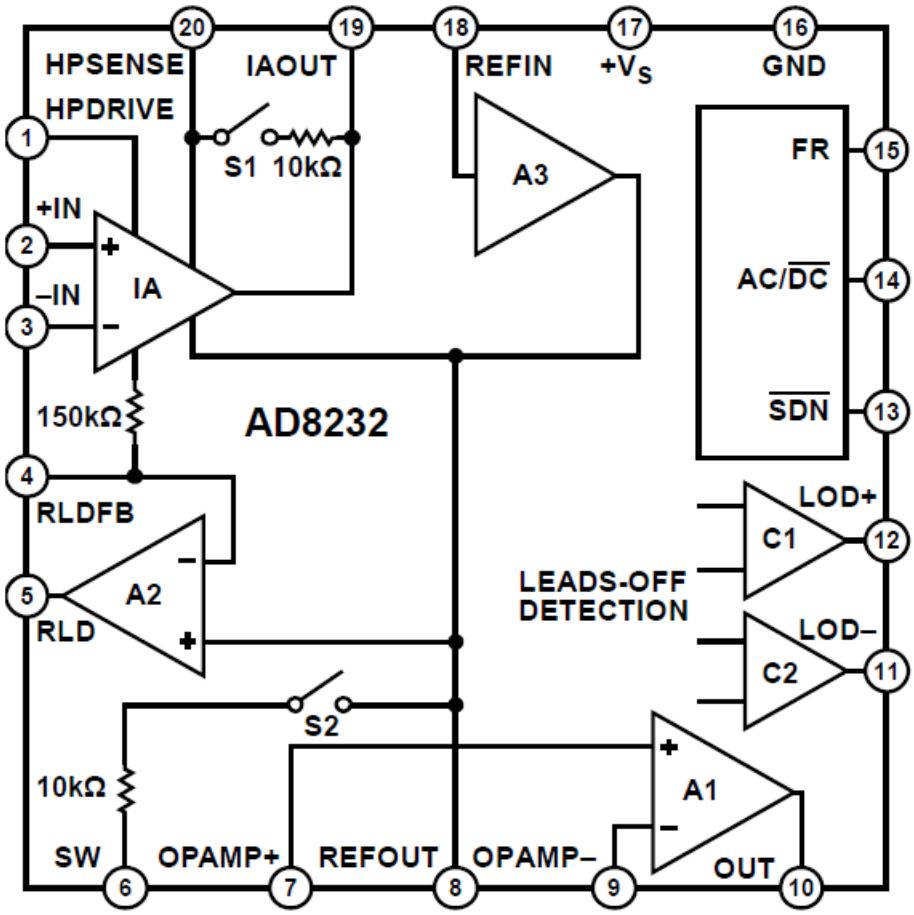


Figure 2.11: Functional Block Diagram[6]

Input

Two-electrode and three-electrode configurations are possible using the AD8232. In the three-electrode configuration, the Right Leg Drive (RLD) can be used as the reference lead. The Right Leg Drive amplifier is employed for rejecting the Common-Mode voltage of the system at the input of the instrumentation amplifier (CMRR = $86dB$). The terminals of the AD8232 are protected against electrostatic discharge (ESD). External resistors can be utilized to avoid overload conditions at the input[6].

Gain and Signal Conditioning

The AD8232 includes an in-amp with a fixed gain of 100. It also features an op-amp for creating low pass filter with selectable cutoff frequency for additional noise removal. To avoid Radio Frequency Interference (RFI) and as a result DC offset at the output, a low pass filter with cutoff frequency of $1.1MHz$ is implemented at each input pins of the AD8232 [6].

Additional Features

The reference buffer is used to create a virtual ground (reference voltage) between the actual ground and the supply. This reference voltage can be driven from REFIN pin using a voltage divider and an external voltage source. The virtual ground is available at REFOUT for use in the circuitry.

The AD8232 provides AC and DC leads off detection. The three electrode configuration uses DC lead off detection in which each input terminal is checked for connection. In this case if the IN+ electrode is disconnected the LOD+ pin is set and if the IN- is disconnected the LOD- pin set.

AC lead off detection is used for two-electrode configuration. This is done by injecting a small current into the input terminals. This feature informs the user with disconnection of either electrodes but it does not specify which electrode is disconnected[6].

Power Consumption

The AD8232 operates on a single supply such as CR2032 cell batteries or rechargeable lithium ion batteries. To reduce the power consumption, the chip provides a shutdown mode with the current draw of less than $200nA$[6].

2.4 Design Decision

Comparing the three AFE blocks from the analog devices, all options were rated base on the complexity, portability and adaptability. The ADAS1000 offers more features but is more complex and requires more power. The AD8236 is a general purpose instrumentation amplifier and requires additional signal conditioning components (at least four op-amps Figure 2.9) to meet the project's needs. This may increase the AFE block footprint. The AD8232, however, has less complexity and power consumption compared to ADAS1000. Conversely, it offers more features than AD8236. Therefore, AD8232 was evaluated to be the best fit for the purposes of this project. Table 2.1 shows a quantitative decision matrix, indicating that ADAS1000 has the highest power consumption and AD8236 does not include the desired features.

Table 2.1: AFE Decision Matrix

IC	CMMR* [dB]	Supply Voltage [V]	Supply Current [μA]	Number of Electrodes	ADC Precision [bit]
ADAS1000	110	3.15-5.5	785	5	14
AD8232	**86**	**2-3.5**	**170**	**3**	**N/A**
AD8236	110	1.8-5.5	33	N/A	N/A

*CMRR reported on this tabel are typical CMRR values for gain of 100

2.5 Design

AD8232 was chosen for AFE block in the designed system since it has the capability to accommodate both the two-electrode and three-electrode configurations. The values of resistances in the device are all chosen to have large values in order to minimize the power consumption[7]. Adaptability is the key feature of the design. As shown in the schematic In Appendix 10.1, several components have the value of Do Not Install (DNI). This is to allow for additional filtering or amplification. The user can chose the value of DNI capacitors and resistors to match their desired configuration for different applications.

The two electrode configuration use LA and RA pins as an input, whereas the three-electrode configuration uses right leg drive (RLD) pin in addition to RL and LA. Fast restore functionality is enabled by setting the corresponding pin (FR) to high to achieve a better signal quality. The default leads off detection features is set to AC for two-electrode configuration but can be adapted for three-electrode configuration (DC) as well. The pins corresponding to the AC and DC leads off detection LO+ and LO- interface to the communication block for further processing and informing the user. The $10M\Omega$ resistors R7, R8 are input bias resistors. These input resistances must be biased for the in-amp to function properly for two-electrode configuration. Three solder jumpers (SJs) available on the design make it possible to bias these resistors with correct voltage values. For two-electrode configuration, connecting SJ3 pads biases the resistors with RLD voltage value while SJ2 uses the REFOUT voltage for biasing. Resistors R9 and R10 are chosen as $182k\Omega$.

The output of the AFE block is fed to the input of communication block ADC for data conversion and transition. To reduce the power consumption, SDN mode pin interfaces to communication block of the system and can be controlled digitally. This allows reducing the current drawn as necessary to reduce the power consumption[7].

As it is shown in the schematic in Figure 10.1 R11 and R12 are forming a voltage divider system connected to supply voltage +VS. This divider creates a midsupply voltage level at REFIN. According to the architecture of the device using a buffer amplifier, the same voltage level appears at the pin REFOUT. C17 is used for filtering the power line noise[7]. C13 and C12 are DC coupling capacitors. The output signal is filtered and amplified by a two-pole Sallen Key low-pass filter. R17, R18, C21 and C25 are responsible for this filtering. The Equations 2.2 and 2.3 are used to calculate the cut-off frequency and gain of the two-pole Sallen Key low-pass filter, and are taken from AD8232 data sheet[6].

$$f_c = \frac{1}{2\pi \cdot \sqrt{R17 \cdot R18 \cdot C21 \cdot C25}} = \frac{1}{2\pi \cdot \sqrt{200k\Omega \cdot 1M\Omega \cdot 10nF \cdot 22nF}} = 23.99Hz \quad (2.2)$$

$$Gain = 1 + \frac{R20}{R23} = 1 + \frac{1M\Omega}{100k\Omega} = 11 \quad (2.3)$$

C13 and R15 creates a high-pass filter of the internal in-amp that has a gain of 100. Therefore,

the cutoff frequency of the high-pass filter is given by Equation 2.4.

$$f_c = \frac{100}{2\pi \cdot R5 \cdot C13} = \frac{100}{2\pi \cdot 10M\Omega \cdot 0.22\mu F} = 7.23Hz \tag{2.4}$$

The gain of 11 multiplies to the fix gain of 100 from in-amp which brings the total gain of the AD8232 to 1100.

A $7.23Hz$ cutoff frequency allows for the in-amp to reject the DC components of the inputs[7]. C22 and R22 create an AC coupling network between the output of the in-amp output (IAOUT) and the REFOUT pin. This allows for negative signal swing as well as creates a second pole at the location of $7.23Hz$ for the highpass filter. The highpass filter at the output of in-amp has total of $40dB$ decrease in magnitude response per decade[7].

$$f_c = \frac{1}{2\pi \cdot R22 \cdot C22} = \frac{1}{2\pi \cdot 100K\Omega \cdot 0.22\mu F} = 7.23Hz \tag{2.5}$$

The capacitor C20 located between the RLD and RLDFB pins is chosen to be $1nF$. The placement of C20 is to improve the CMRR at the inputs of the in-amp when three electrode configuration is used. The C20 and $150k\Omega$ resistor of the Right Leg Drive Amplifier form an integrator and gain of 20 at $50Hz$ to $60Hz$. The output current of the Right Leg Drive opposes the variations of the Common-Mode voltage. To maintain a stable system, R16 is chosen to be $499k\Omega$. R16 also limits the current flowing to the RLD pin of AD8232 to be less than $10\mu A$ for safety reasons[7].

2.6 Testing Capabilities

As shown in the design schematic, a total number of 10 test points are implemented. The test points allow for observing the input or output of each stage or pin. TP1 and TP2 are intended for current drawn measurements across the R2. R2 with the resistance of 10Ω is installed between the battery and the +VS of the AD8232. This provides testing capability and power calculation by measuring the current drawn by the AD8232 in various modes. TP3, TP4 and TP5 correspond to AD8232 input channels (electrodes) to which the input signal can be connected. TP6 is connected to the OUT pin of the AD8232. TP6 and ground can be used to monitor the output of the AFE block. If low pass filters are applied to the output of AD8232, TP7 (one-pole low pass filter), TP8 (two-pole low pass filter) and TP10 (three-pole low pass filter) can be used to test the output of the AFE block. TP10 is for measuring the REFOUT voltage. The REFOUT voltage in the current configuration of the design must equal to half of the voltage supply.

2.7 Adaptability

This project requires the design to be flexible and adaptable to other monitoring situations. Consequently, pads for extra passive components are included in the design to allow for future options. Installing the resistors R19 and R21 results in additional low pass filtering of the output. The RC systems to be used for this purpose are (R19 and C26) and (R21 and C24). C24 and C26 has a value of $1uF$. The reference voltage REFOUT can also be filtered using R24 and C27 system. C27 is installed as $1uF$ while the R24 is not populated. R15 can be used to limit the loop gain of the RLD amplifier and shift its dominant pole[7].

The capacitors C14, C15 and C16 are not populated but they can be installed for additional Radio Frequency Interference (RFI) filtering. RFI filters can be used for improving the CMRR when C15 and C16 are the same value. This yields common-mode signal cutoff frequency given by Equation 2.6.

$$f_c = \frac{1}{2\pi \cdot R14 \cdot C15} \tag{2.6}$$

The next step was to perform primary testing of AFE block which will be described in the next subsection.

2.8 Primary testing and evaluation

For evaluation and reference purposes of the AD8232, the evaluation board (AD8232-EVALZ) as shown in Figure 2.12 was used. In order to have a better understanding of the AD8232 and be able to evaluate its functionality, some of the AD8232-EVALZ datasheet analysis was replicated.

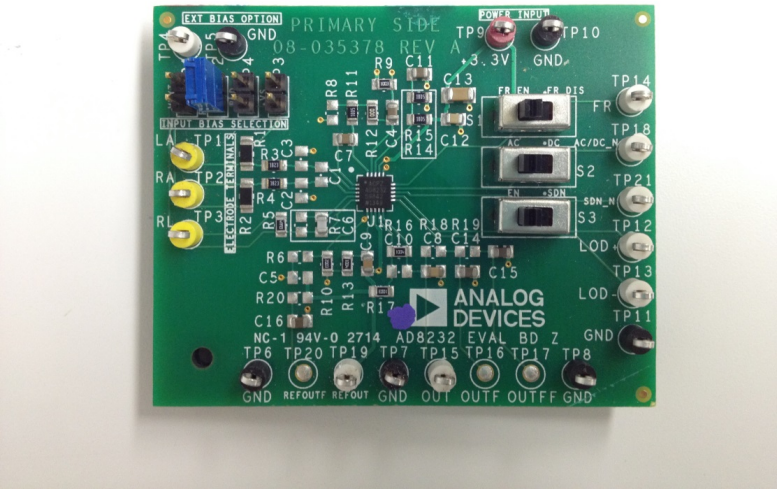


Figure 2.12: AD8232-EVALZ Board[7]

2.8.1 ECG Waveform

For reliable heart signal from the AD8232 evaluation board, electrode connections were soldered properly. The setup used is shown in Figure 2.13. Scope was connected to OUT and GND with input bias set by power supply P3. In the 3-electrode configuration, the output signal is show in the next few figures. Different electrode locations were used to test which location has best good view of the heart signal and provides most amplitude.

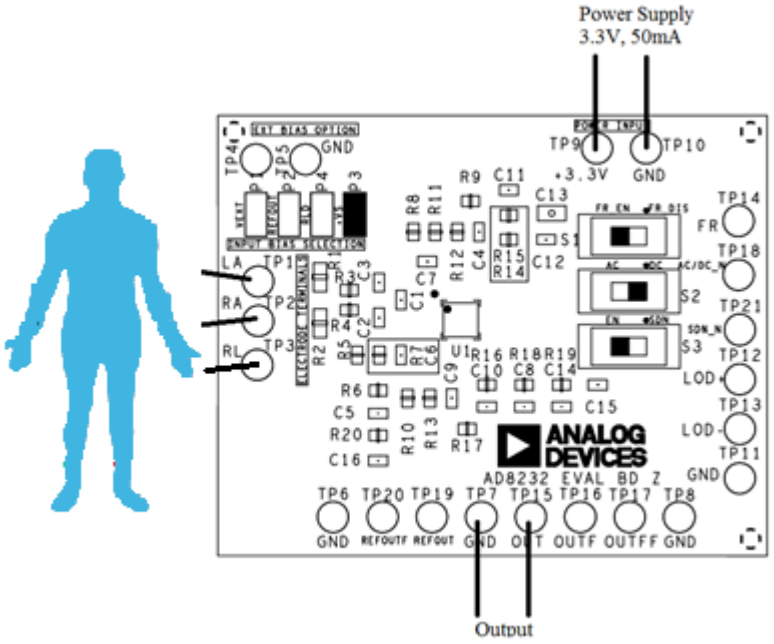


Figure 2.13: Evaluation Board Setup

The output in the Figure 2.14 shows the ECG signal with Lead I configuration. The output signal is about $1V$ peak to peak.

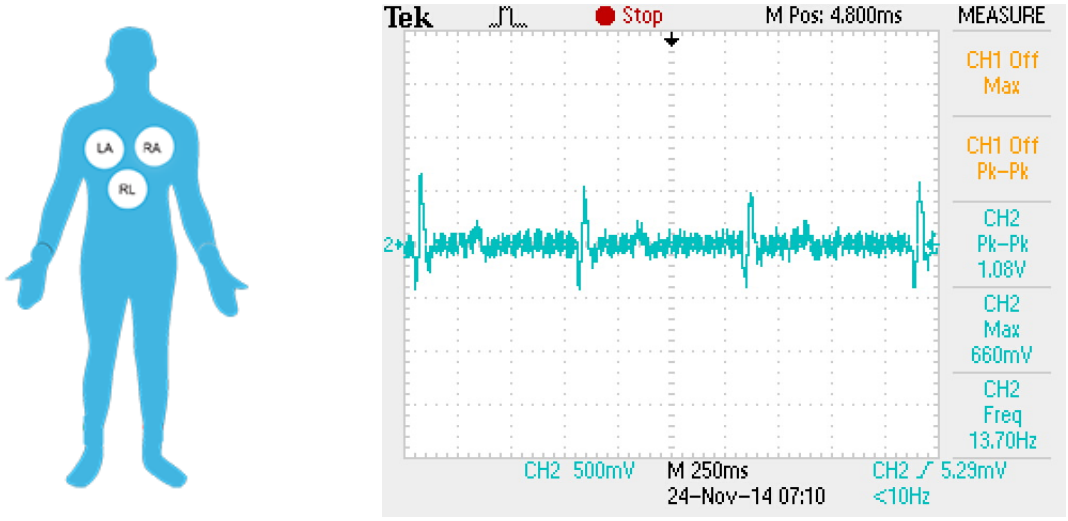


Figure 2.14: Three Electrodes Configuration Lead I

When the electrodes are placed so as to provide good view of Lead II, the output signal shown in the Figure 2.15 appears to be more amplified with a peak to peak of almost $1.5V$.

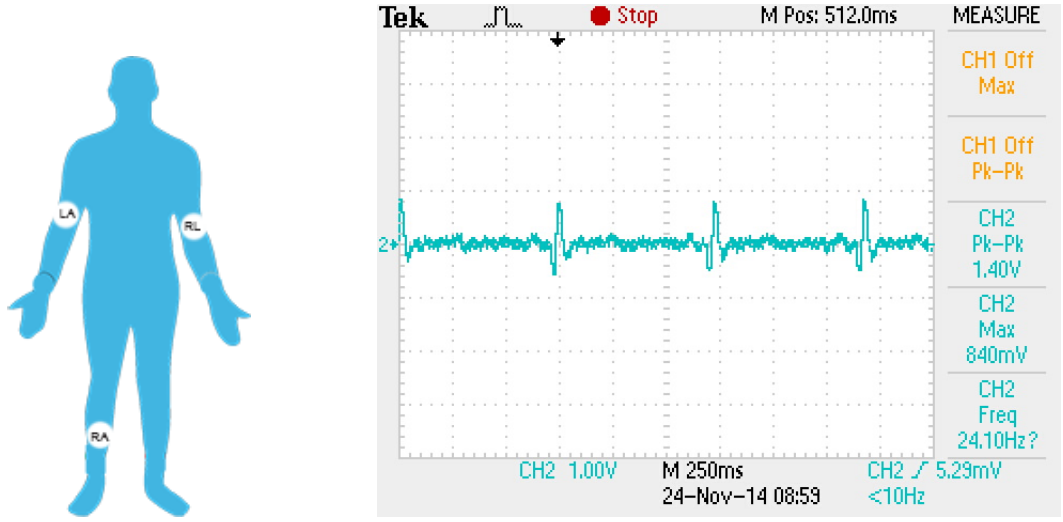


Figure 2.15: Three Electrodes Configuration Lead II

Two electrode configuration provided the ECG output in Figure 2.16. The peak to peak value is $500mV$ which is generally small.

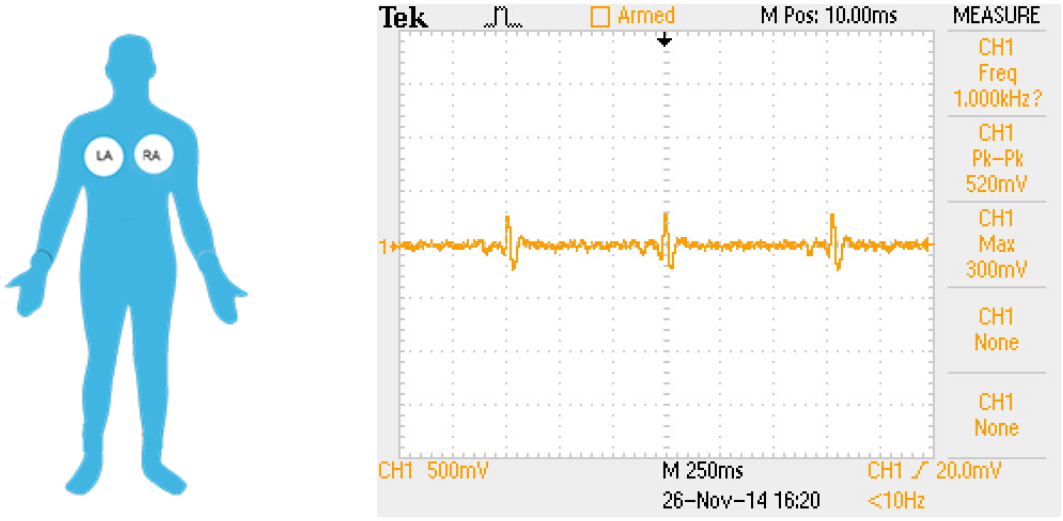


Figure 2.16: Two Electrode Configuration - Oscilloscope Capture

2.8.2 Transfer Function

Based on the frequency response, the transfer function was derived to have of a two-pole low pass filter (constitutes as zeros) and a two-pole high-pass filter (constitutes as poles). Based on the filter configuration there are four poles having two real and two complex poles and two zeros with complex zeros with cutoff frequency at 15Hz or 95 rad/sec. The transfer function was determined to be:

$$H(s) = \frac{(s^2 + s + 20)}{\left(\frac{s}{95} + 1\right)^2 \left(\left(\frac{s}{95}\right)^2 + 2 * 0.6 * \left(\frac{s}{95}\right) + 1\right)}$$

2.8.3 Frequency Response

The next step was to use the evaluation board and be able to achieve correct frequency response using the data sheet as a reference. The AD8232-evalz evaluation board was setup by connecting the power supply common to GND terminal and a 3.3V supply voltage at the +3.3V terminal. The left arm (LA) terminal and the right arm (RA) terminal are connected to the signal source. For this evaluation, an arbitrary ECG signal from function generator was used. The output signal is available on the OUT terminal.

The set-up used to perform measurements is shown in Figure 2.17. The function generator was set to "High Z" as it has factory impedance set to 50Ω since high impedance gives less distortion on waveform.

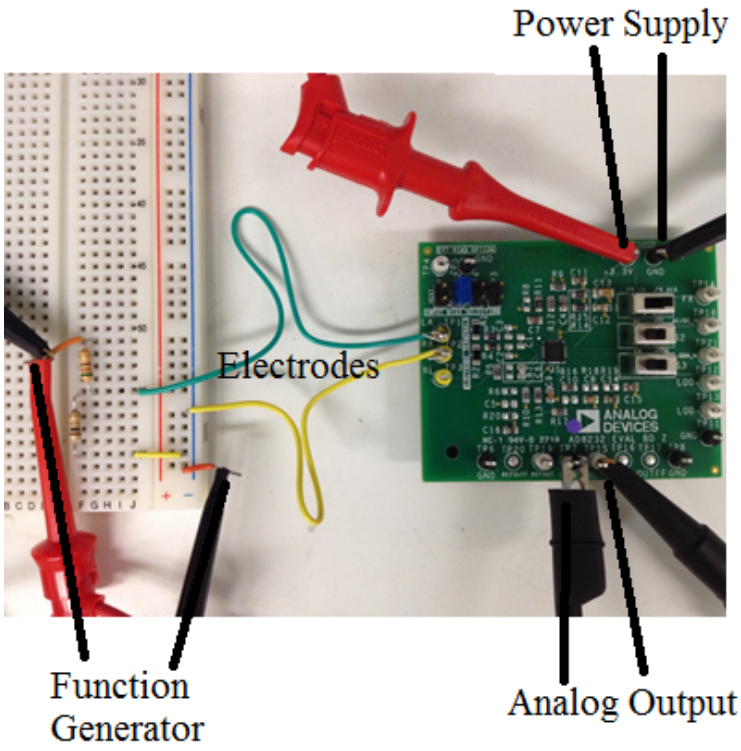


Figure 2.17: Frequency Response Test Setup

For testing, the frequency range gathered was from $1Hz$ to $1000Hz$. A voltage divider was used to provide a $1mVpp$ input to accommodate a low ECG signal from the function generator. Input bias REFOUT at P2 was used.

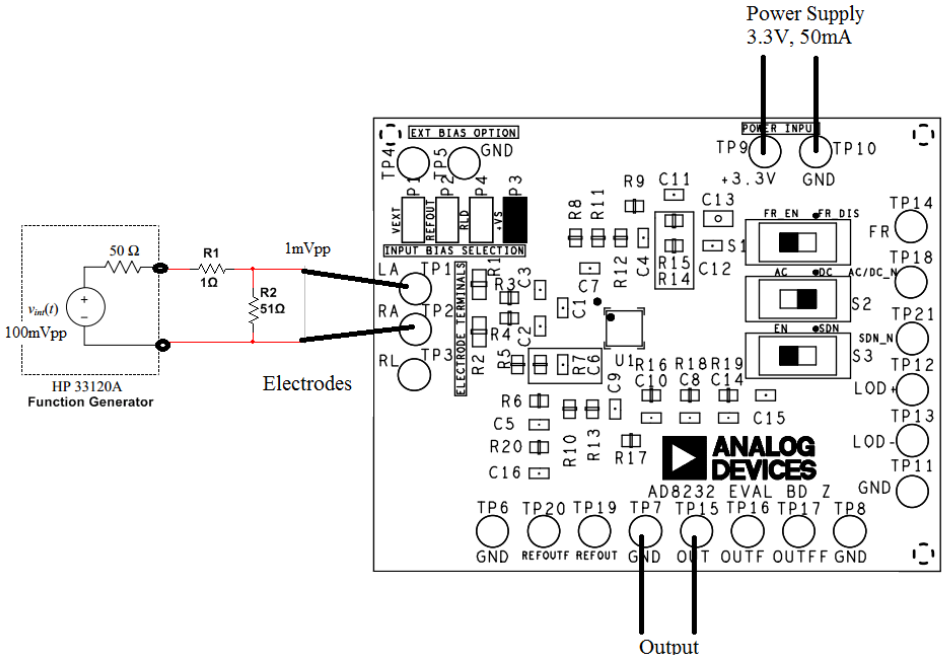


Figure 2.18: Frequency Response measurement circuit diagram

The transfer function of the signal from the differential input of the in-amp to OUT with the default filter configuration is shown in Figure 2.19.

Table 2.2: Frequency Response Derivation

Frequency	Amplitude [Vpp]	Gain	Gain [db]
[Hz]	Vin [V]	Vout [V]	Vout/Vin
1	0.0100	0.1680	17
2	0.0100	0.7440	74
3	0.0100	1.6600	166
4	0.0100	2.8600	286
5	0.0050	2.1000	420
6	0.0050	2.8600	572
7	0.0040	2.8800	720
10	0.0020	2.4600	1230
15	0.0020	3.3600	1680
20	0.0020	2.9600	1480
30	0.0020	1.5200	760
40	0.0020	0.8800	440
50	0.0020	0.5680	284
70	0.0050	0.7280	146
80	0.0050	0.5560	111
90	0.0050	0.4400	88
100	0.0050	0.3680	74
150	0.0050	0.1650	33
200	0.0100	0.1760	18
300	0.0100	0.0816	8
400	0.0100	0.0450	5
500	0.0100	0.0280	3
600	0.0150	0.0304	2
750	0.0150	0.0199	1
850	0.0150	0.0168	1
900	0.0150	0.0152	1
1000	0.0150	0.0150	1

In Figure 2.19, a comparison between the theoretical frequency responses based from the evaluation board data sheet with the interpolation, the measured data, and the derived transfer function are provided.

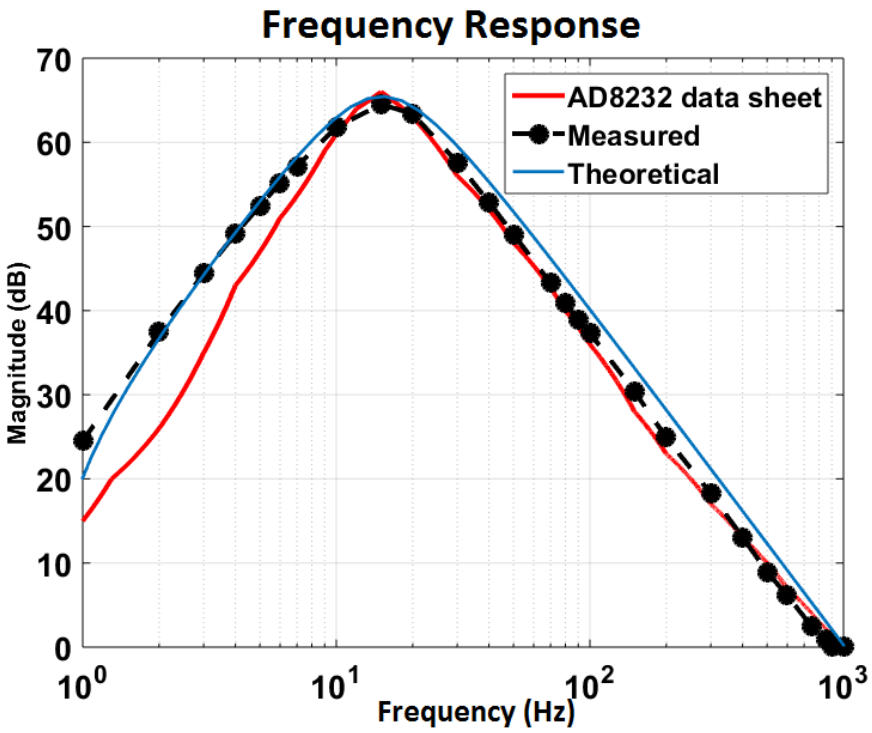


Figure 2.19: Frequency Response Comparison

2.8.4 Step Response

For the step response, the transfer function was scaled to have an amplitude value similar to the measured data. This was found by calculating the ratio between the derived transfer function amplitude and the measured data, which is found to be 0.0019. As shown in Figure 2.20, the derived transfer function is comparable to the measured data.

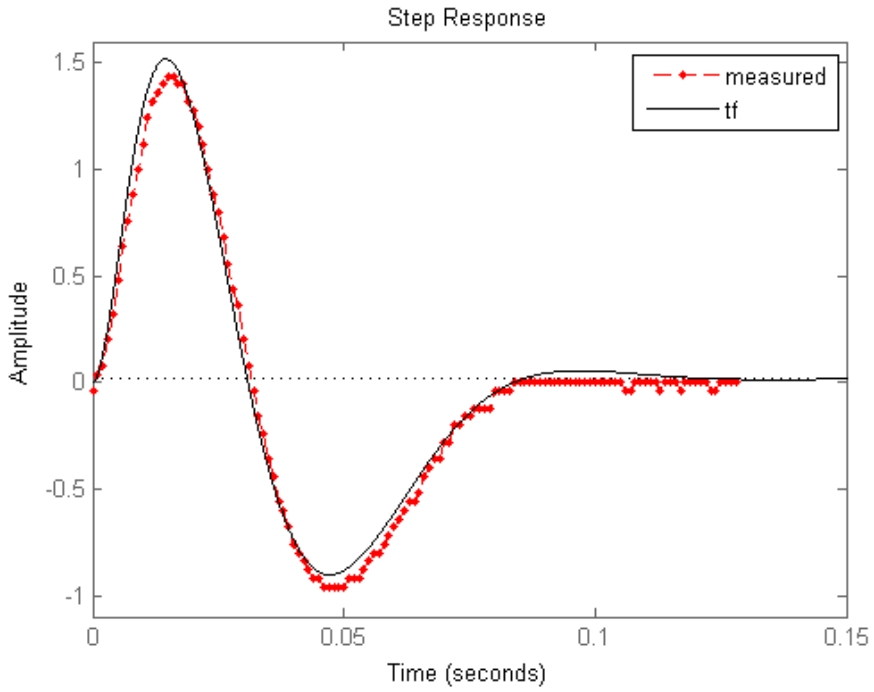


Figure 2.20: Step Response

Zero-pole analysis is shown in Figure 2.21. The imaginary part is on the x-axis while real part is the y-axis. There are two complex zeros and two finite pole located at z= -95. There are two complex poles located at about $\pm\frac{j}{95}$ where j is the imaginary unit.

Output of step response produces the above figure. The overshoot is due to the zeros of low pass filter and the poles of high-pass filter. Second order filter causes resonance especially for complex poles. The zero-pole map diagram is shown in Figure 2.21.

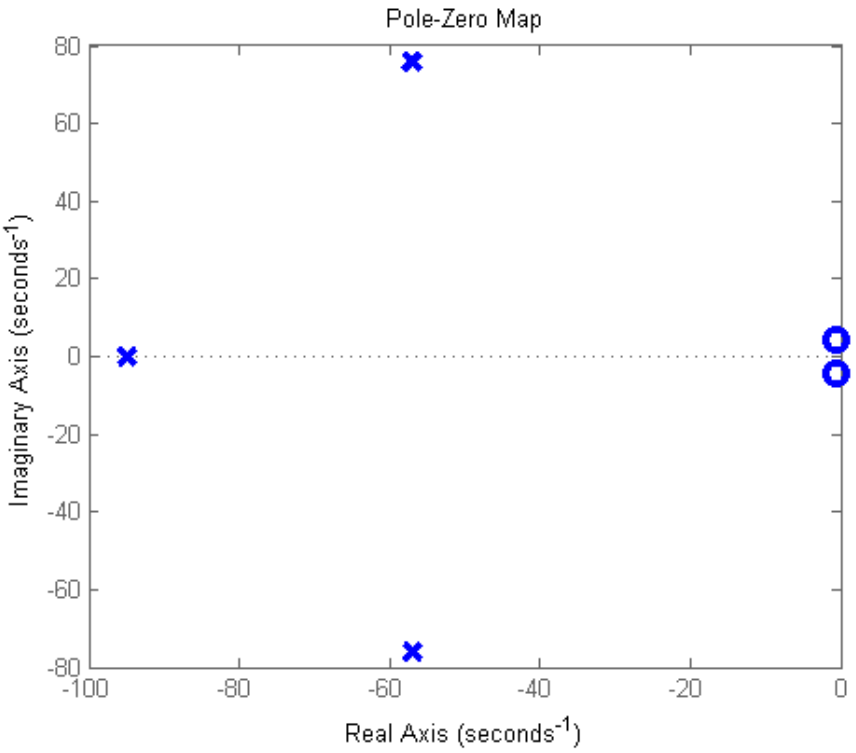


Figure 2.21: Zero Pole Plot

2.8.5 Switch and Jumper Settings

Another feature for AD8232 is the switch settings for that could be configurable for either a 2-electrode or 3-electrode capability. Different modes of AD8323 were investigated as shown in Table 2.4.

Table 2.4: Frequency Response Derivation

Label	Options	Setting
S1	FR_EN / FR_DIS	Fast restore enabled
S2	AC / DC	Leads off detection
S3	EN / SDN	Operation enabled/ shutdown

Fast Restore

This mode reduces the duration of long settling tails of the high-pass filters that allows the AD8232 to recover quickly and therefor to take valid measurements soon after reconnecting

the electrodes. To enable, switch S1 is set to FR_EN position.

The configuration used was a frequency input of 1Hz with $1mVpp$, source $3.3V$ and max current limit to $0.5A$. When FR is disabled, the restore time is t=1.980s. When FR is enabled, the restore time is significantly faster with t=0.240s. Moreover, as shown in Table 2.22, when two leads are off, the restore time if enabled is slower than when only one lead is off.

Table 2.6: Frequency Response Derivation

Setting	FR_DIS Restore time [s]	FR_EN Restore time [s]
LA lead off (one electrode)	1.980	0.240
Both leads off (two electrodes)	1.840	1.020
RA lead off (one electrode)	1.980	0.380

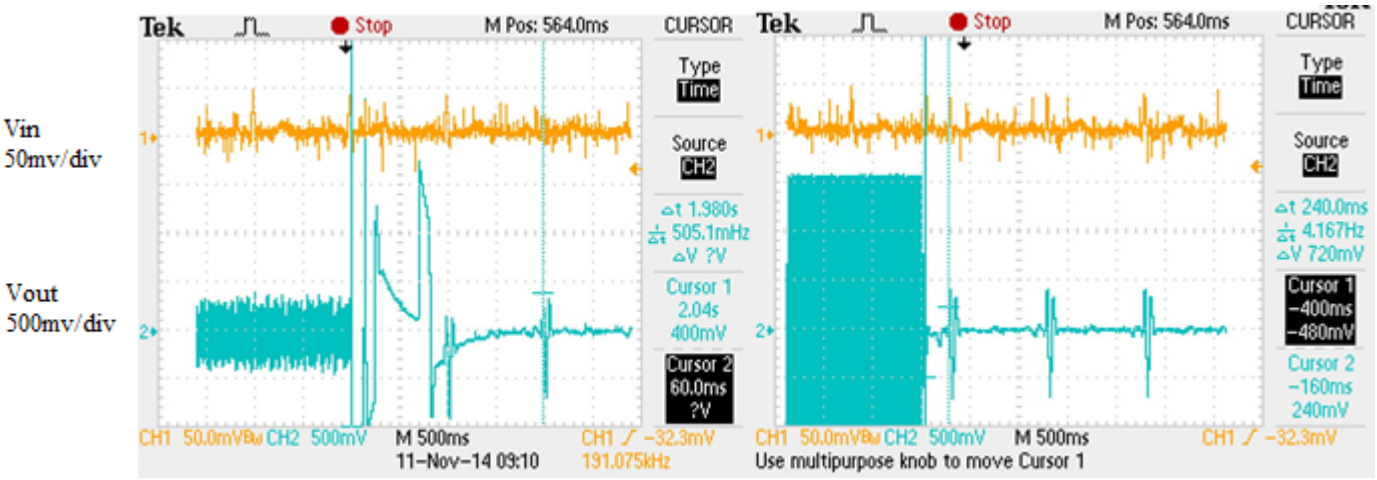


Figure 2.22: FR disabled (left) and FR enabled (right)

These values can be compared to AD8232 plots as shown in Figure 2.23. When only one electrode is disconnected, it has an average recovery time of 0.25s when FR is enabled. The recovery time when it is disabled is approximately 2 seconds.

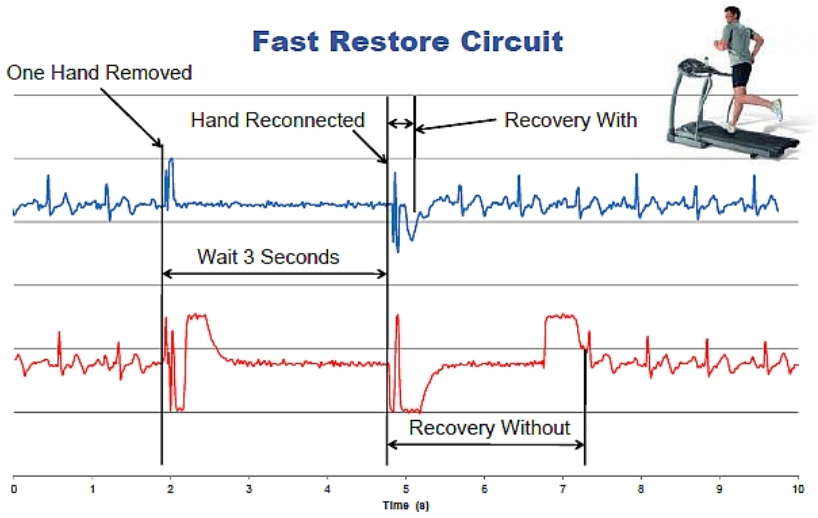


Figure 2.23: Faster Restore Circuit AD8232 [Analog Devices]

Leads off detection

The leads off detection feature provides modes optimized for either two-electrode or three-electrode configuration of the device. The AC leads off detection is used for two-electrode setup. To maintain the inputs inside common-mode range of the amplifier, the bias level for the inputs can be set to REFOUT or RLD by placing the jumper at P2 or P4. The AD8232 detects when an electrode is disconnected by sourcing a small 100kHz current into the electrodes.

The DC leads off detection is used for three-electrode setup that works by sensing when either input goes high. The RLD output terminal must be connected to a driven electrode. It can detect which electrode is disconnected through pins LOD+ and LOD-.

When the leads off detection S2 is set to AC, the output at LOD+ pin produces a high digital value that would indicate that an electrode is disconnected. For testing the AC-lead off detection feature, a sample ECG signal was used from the function generator with 1Hz and $1mVpp$. Output is shown in Figure 2.24 where the middle graph is the LOD+ pin output.

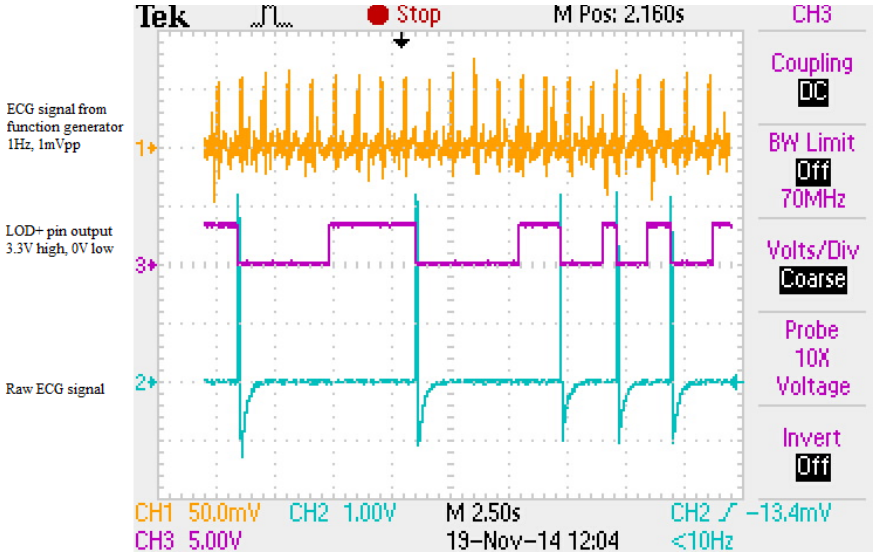


Figure 2.24: AC Leads off Detection

For two-lead electrode configuration, the LOD+ pin has a high or low output. When one electrode is disconnected, the LOD+ pin signals a high value ($3.3V$) and when the electrode is connected again, it goes to a low value (0V). Initially, the pin has a low output signal.

Operation Enabled/Shutdown

According to AD8232 system specifications shown in Figure 2.25, shutdown current has typical values depending on temperature and the voltage supply.

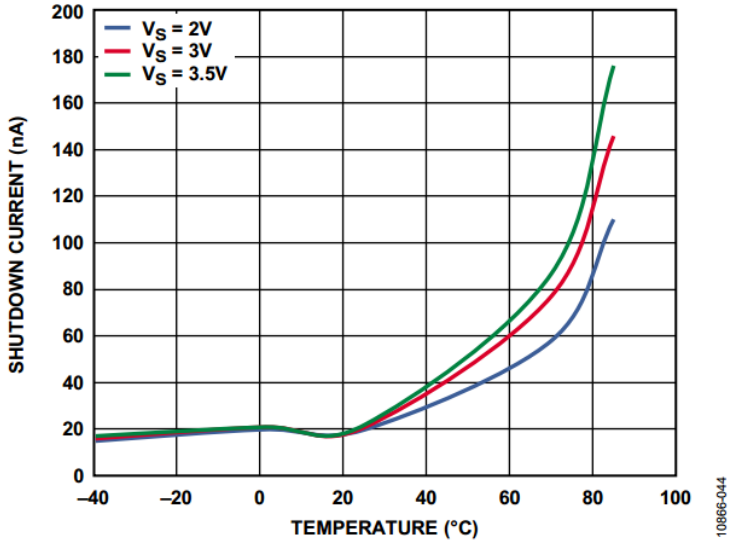


Figure 2.25: Shutdown Current vs. Temperature (System Performance)

To verify this operation, current was measured during shutdown mode by finding the voltage drop at output when connected to a series resistor of $1M\Omega$. It was measured with the following values for input from the function generator: 1Hz, 1mVpp. The resistor value was measured for accuracy and found to be $R = 0.976k\Omega$. Equation 2.7 was used to find shutdown current usage and was verified from Figure 2.25.

$$\mathrm{I} = \frac{V_{\mathrm{DROP}}}{\mathrm{R}} = \frac{72\ \mathrm{mV}}{0.976\ \mathrm{k\Omega}} = 73.77\ \mathrm{nA} \tag{2.7}$$

When shutdown pin is high, the output signal produces a small sinusoidal voltage as shown in Figure 2.26.

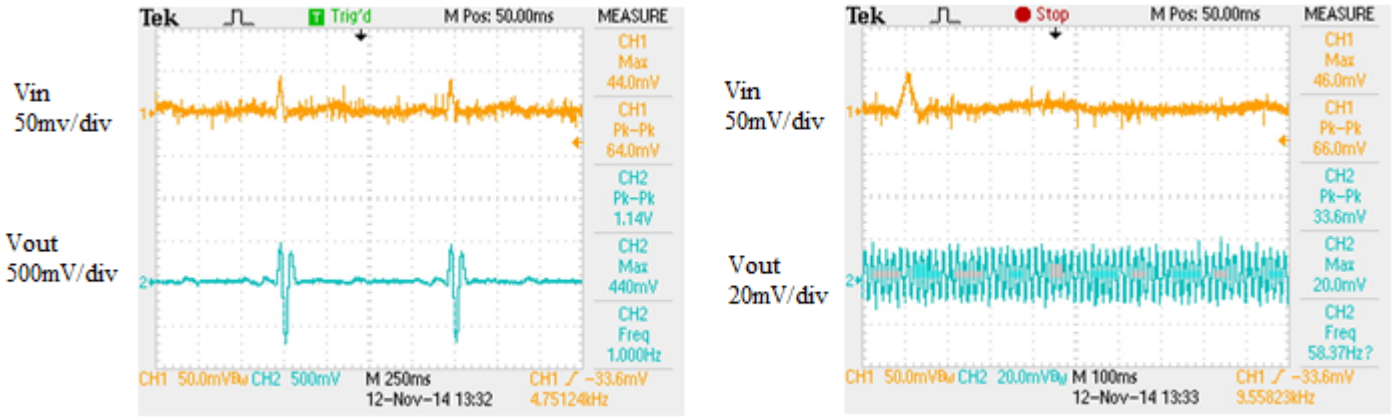


Figure 2.26: Operation enabled (left) and Shutdown mode (right)

2.8.6 Supply Current

According to AD8232 Data sheet, current value is typically $170\mu A$. The current supply usage was measured and found to be $166\mu A$ with a power supply set to 3.3V and 50mA current limit. Current was found by measuring voltage drop across resistor then performing Ohm's law as shown in Equation 2.8. Current measurement setup is shown in Figure 2.27.

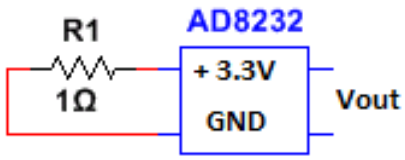


Figure 2.27: Measuring Current Supply with Series Resistor

$$\mathrm{I} = \frac{V_{\mathrm{DROP}}}{R} = \frac{0.183\ \mathrm{mV}}{1.10\ \Omega} = 166\ \mu A \tag{2.8}$$

This value is comparable to the Supply Current vs. Temperature plot from the AD8232 data sheet shown in Figure 2.28. At room temperature, the expected supply current is approximately $170\mu A$.

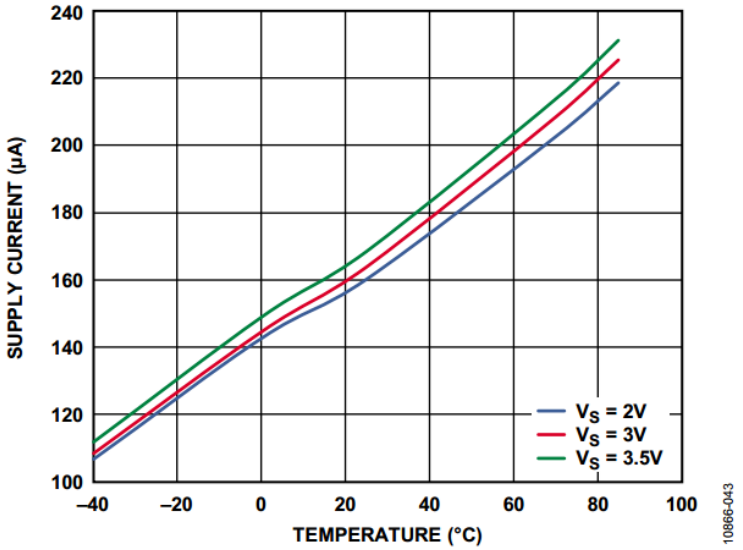


Figure 2.28: Supply Current vs. Temperature (System Performance)

2.8.7 ECG Signal 2-electrode Issues

After conducting preliminary testing, the functionality and features of AD8232 were verified. Results show that two-electrode configuration is more susceptible to AC noise. It was also found that the ideal ECG location would use Lead I which is typically a smaller signal. That is, the device is designed so that it will easily be snapped onto a person's chest as shown in Figure 2.29 which also shows a typical 2-electrode configuration with less noise and laptop and other devices in vicinity. To find which sources would produce the most AC noise, different test setups were used such as touching materials in the vicinity. It was found that when touching a laptop battery source or the metal underneath the desk, 2-electrode signal is vulnerable to noise as shown in Figure 2.30. Otherwise, when the metal above desk or on the desk was touched, it was able to produce results as shown in Figure 2.31.

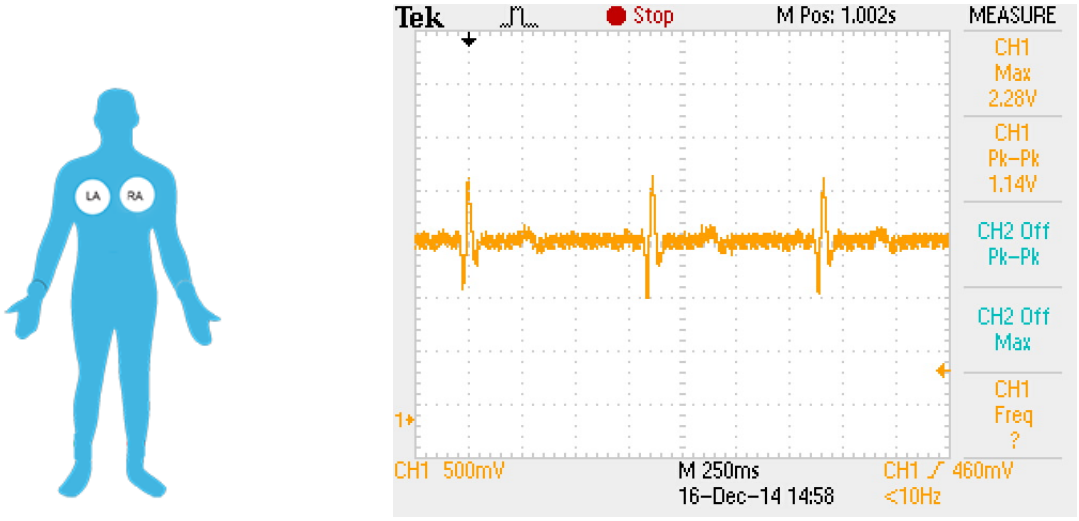


Figure 2.29: Normal 2-electrode configuration with less noise

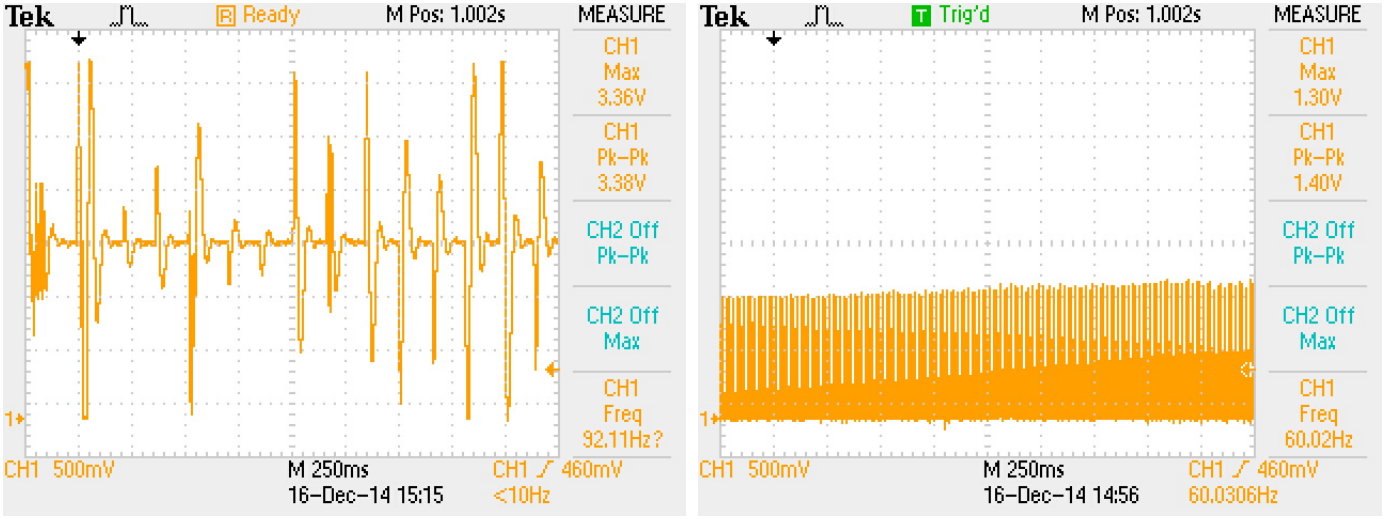


Figure 2.30: When touching a laptop battery source When touching metal underneath desk

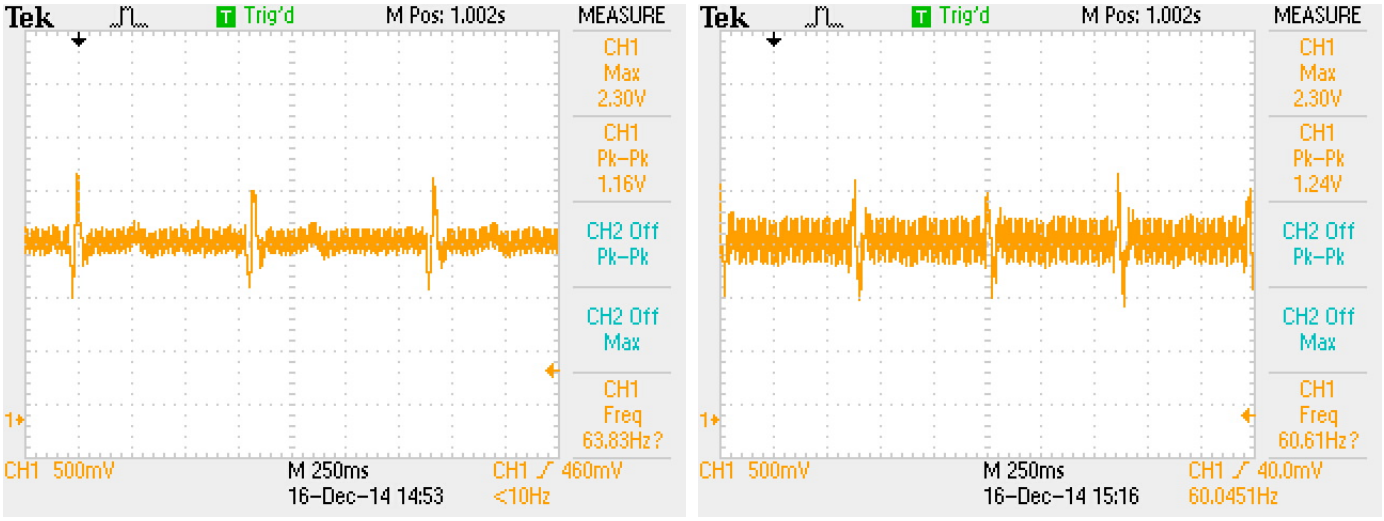


Figure 2.31: When touching metal above desk When touching on the desk

3 Communications

3.1 Background Research

The communications block has several requirements that must be met in order for total system integration.

Requirements

- Ability to run off a 2-3V supply
- Less than 20 mA peak current draw
- Minimize Complexity
- Easy integration with mobile devices

Based on the requirement for low power operation, the Bluetooth Protocol was chosen for further evaluation as it allows for potential integration with mobile devices by using the built-in Bluetooth radio that most mobile devices and laptops contain. Moreover, it is lower current consumption than standard Bluetooth or Bluetooth Enhanced Data Rate (EDR), which typically consumes a excess of $100mA$, which would result in a very low battery discharge time.

Table 3.1: Communications Protocol

Protocol	Frequency	Power	Device Integration	ADC
XBee/ZigBee	950 MHz/2.4 GHz	Medium	No	No
RF Mesh Networks	No standard	Low	No	No
BLE/ANT	2.4 GHz	Low	Yes	Yes
Wi-Fi	2.4 GHz	High	Yes	No

Untethered communication is a fundamental requirement for this project. Therefore, the communication block of the design is responsible for wireless transmission of data to the user interface. Since this project is targeting a low power application, communication block with low power consumption is preferred. Another requirement would be the capability to easily integrate with the smartphones. Among the existing protocols, Wi-Fi and Bluetooth, capability are integrated within the smartphones. This will eliminate the need of an intermediate transceiver module. The communication block must also be battery operated with supply voltage of $2V$ to $3V$.

Both Wi-Fi (IEEE 802.11 standards) and Bluetooth operate in $2.45GHz$ band. Conversely, these two protocol have completely different data transmission rates. Higher data rate would

require more power consumption. This project is concerned with monitoring application. Bio-potential monitoring applications do not require high data transmission rate compared to data streaming applications such as playing music or video. Therefore, high data transmission rate is not a requirement for the desired design. Among the available wireless communication protocols, Bluetooth v4.0 or Bluetooth Low Energy (BLE) (also called Bluetooth Smart) has the lowest data transmission rate of $0.3Mbps$.

3.1.1 Bluetooth Low Energy Protocol

Bluetooth devices are divided into three categories depending on their mode of operation such as dual mode, single mode and Bluetooth classic-only devices. Dual mode devices support both Bluetooth classic and Bluetooth low energy (BLE). Single mode (Smart Devices) devices only support the BLE while classic-only devices are intended for classic Bluetooth protocol. BLE uses a different standard than classic Bluetooth that is mainly designed for monitoring cell battery operated applications[25]. BLE transmits data at a lower data rate with respect to classic Bluetooth. Therefore, it is more energy efficient.

Devices with BLE capabilities are restricted to transmission of packets of data with maximum of 20 bytes. This is to reduce the energy cost of recalibration for each transmission. Bluetooth architecture includes three main layers such as controller, host, and application. The Controller is the Bluetooth module. The host contains protocols and procedures. The Characteristic is a Universal Unique ID (UUID) which is a machine-readable specification. The Application includes characteristic, service, and profile. The Service is the human-readable specification of set of characteristics. The Profile describes devices with services on them and specifies how they need to be connected or discovered[25].

Evaluating the BLE solutions available in the market, this project only considers the system-on-chip (SoC) modules. This decision was made to eliminate the need of an external microcontroller and external ADC. This was done to reduce the complexity of the project while benefiting from an integrated design options. This section investigates the capabilities of Texas Instruments (TI) CC2540 and Nordic Semiconductors nRF51422.

Connection

BLE operate as shown in the flow diagram in Figure 3.1.

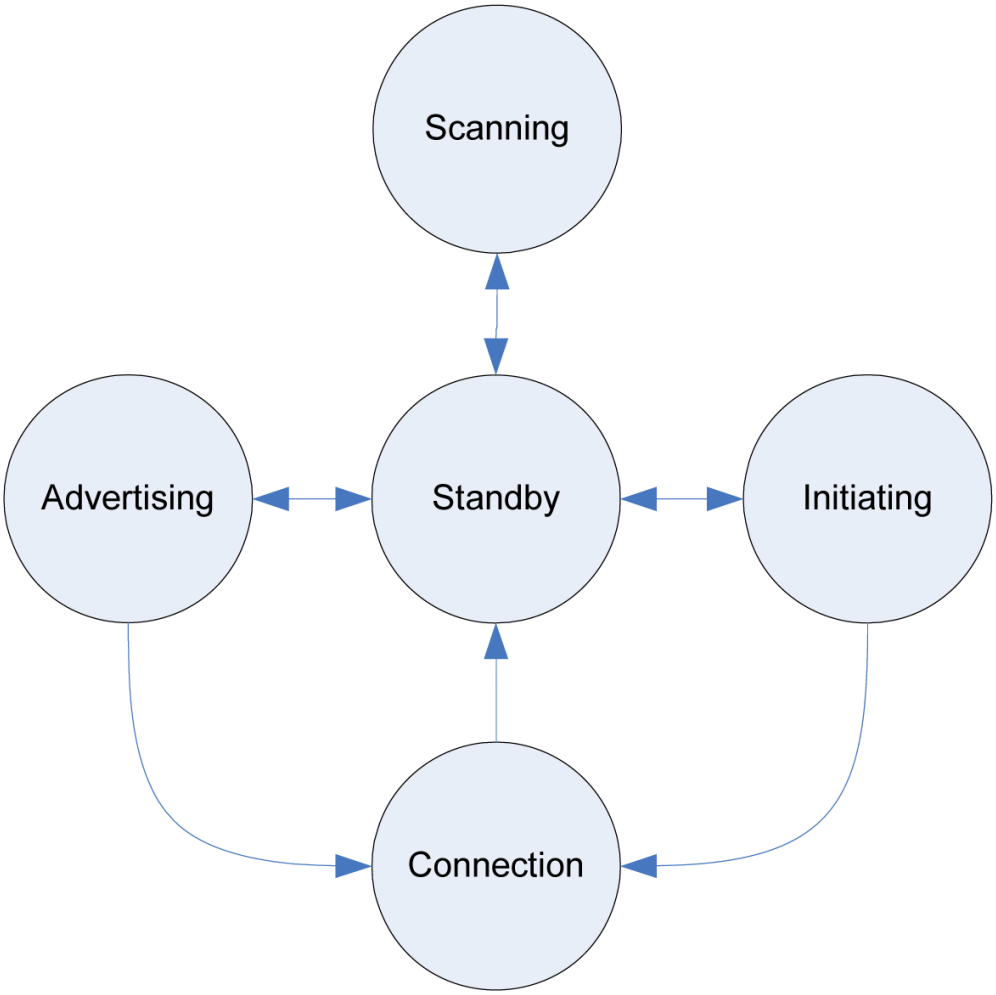


Figure 3.1: Connection Diagram

Initially, it is in the advertising or initializing state and then it moves into the connected state in both circumstances. After the timeout period is reached or a specified sleep time, the device enters standby mode. The scanning mode is entirely separate from the rest of the operations as it is just data and not necessary for connection if a BLE device address is known.

Error Detection and Correction

According to the Bluetooth Core Specification v4.0, the stack must be able to detect and mitigate errors. Therefore, the stack implements cyclic redundancy checking (CRC) on both the transmitter and the receiver radios. Although this takes processor time and adds to current consumption, it also adds to reliability. The TX and RX CRC checks are shown in the flow diagram Figure 3.2.

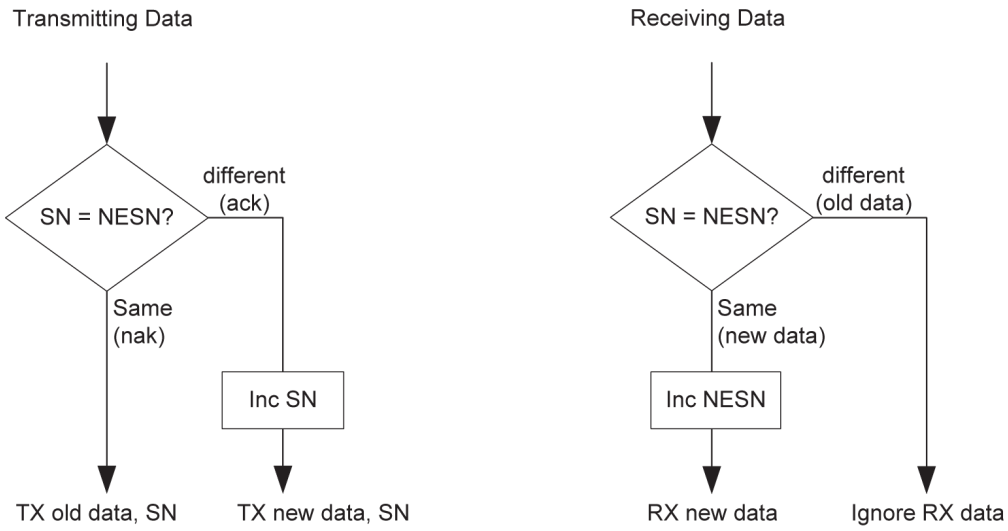


Figure 3.2: BLE Error Checking on TX and RX

3.1.2 TI CC2540 Keyfob

Based on the requirements, the Texas Instrument CC2541-DK Keyfob Figure 3.3 for the communication block was chosen. This board comes with a supporting dongle CC2540 Figure 3.4. These boards are programmable by IAR Embedded Workbench Software for 8051 processors. Moreover, the CC2541 utilizes Hardware Abstraction Layer (HAL) that eliminates the need for low level programming.

Figure 3.3: CC2541 Keyfob [8]

Figure 3.4: CC2540 Dongle [9]

The ADC of CC2541 supported up to 14-bit analog-to-digital conversion with 12 bits Effective Number of Bits (ENOB). It included eight individually configurable channels (differential or single ended) and a reference voltage generator. Single-ended inputs from the ADC are accessible through virtual channels 0 to 7 corresponding to physical pins 0 to 7 of port 0 on the chip. In this configuration, channels 6 and 7 are chosen as the ADC input. The pin configuration is shown in Figure 3.5.

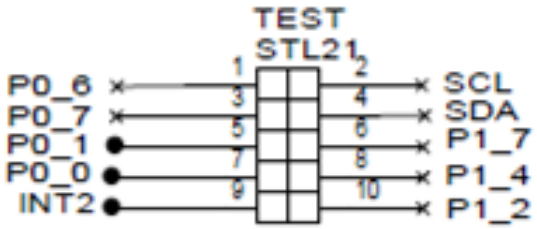


Figure 3.5: Test Pins Schematic

Performing the initial testing of the CC2541, the value read by the ADC did not change with the varied input voltage. Same test was performed on the CC2540 however, for testing purposes such as power measurements the CC2540 is not favorable. Therefore alternative solutions were considered to proceed with the design. These solutions include using an external ADC or use of the ADC on CC2540 Dongle and then send the data over to the CC2541. Both of these options defeated the requirements set in the beginning. Another constraints regrading using the CC2541 was the cost of the SDK. The SDK required to program the CC2541 provides a free license for 30 days. The team negotiated with the company and got an extended free license for 60 more days however, it was decided that the licence restriction would not allow enough time to modify the firmware in future. Therefore another design options for communication block was considered. The alternative BLE solution considered was nRF551422 [10]. This chip is an ARM Cortex M0 based so it has a free SDK. Moreover, it allows over-the-air updates and has more examples for the BLE communication.

3.2 nRF51-DK

3.2.1 Hardware

The nRF51422 integrated circuit has an included 32-bit ARM Cortex-M0, which is significantly more powerful than the 16-bit 8051 core included in the previous chip TI CC2541.

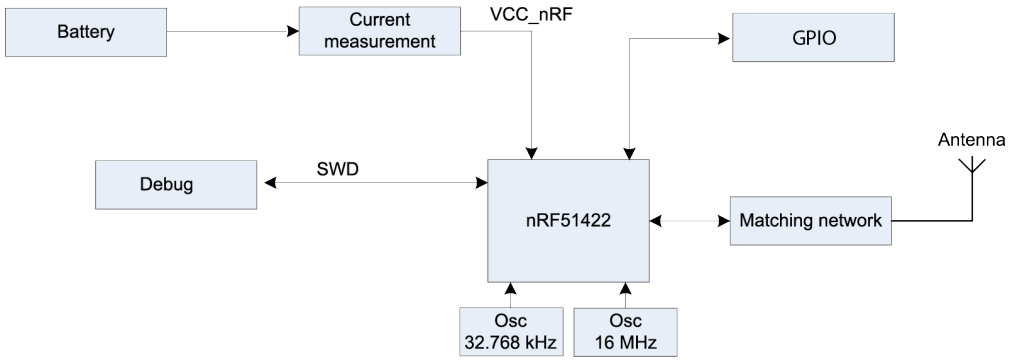


Figure 3.6: nRF51422 Block Diagram

Analog to Digital Converter

This project's design is a wireless signal acquisition system so being able to convert the ECG signal into something that a transceiver can understand and transmit is crucial for the design's operation.

Equation 3.1 shows the generic equation to find the voltage that originates on the pin of an ADC [10].

$$V_{INPUT} = \frac{ADC_{CODE} \cdot V_{REF}}{2^{N-1}} \tag{3.1}$$

$$ADC_{CODE} = \frac{V_{INPUT}}{V_{REF} \cdot 2^{N-1}} \tag{3.2}$$

The ADC in the nRF51422 can support up to a maximum of 72 ksps [10]. Figure 3.7 shows the internal configuration of the analog-to-digital converter built in to the nRF51422 SoC. It can be configured to use the internal $1.2V$ band-gap internal reference or it can be supplied with a reference on AIN0 or AIN1. However, this reference cannot exceed $VS + 0.3V$ otherwise, the system will be damaged [10]. The ADC also offers the ability to use up to 10 bits and 8 channels. However, only two channels are utilized for this project. The signal for the ADC can be supplied on AIN1 to AIN7 and can either be single-ended or differential. This ADC is extremely adaptable, but the challenge is overcoming the timing of the chip and the blocked execution times. The time to convert a single sample in 10-bit resolution mode is

$64\mu s$, which is adequate time for the stack to process messages and continue to sample.

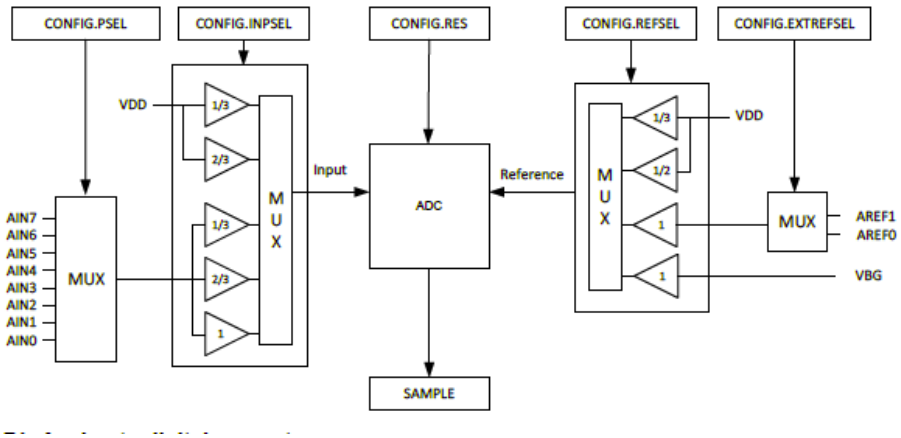


Figure 3.7: nRF51422 Built-in ADC [10]

Several strategies were evaluated for sampling schemes as core execution blocking from the S110 softdevice seems to cause aliasing on the output codes of the ADC [11]. The samples can be aggregated and transmitted in bulk packets of 20 bytes and 6 packets per connection event or they may be transmitted at a minimum of 1 byte per connection interval. Additionally, the data could be stored in the nRF51422's 256kb flash memory for persistent storage and retrieval of the samples. However, the storage module requires a factor of four more cycles to complete than merely transmitting the data.

3.2.2 Software

Due to the proprietary nature of the Bluetooth Low Energy standard, a proprietary library must be utilized to properly implement the BLE stack. Therefore, Nordic Semiconductor must provide an application programming interface (API) to their proprietary implementation of the BLE stack. To do this, two compiled hex files are utilized that occupy two different code spaces thereby not overwriting each other and allowing them to be flashed to the device separately. This allows the user to develop their embedded code independently of the softdevice, meaning that no restrictions or dependencies are enforced except that it must be able to compile for the ARM Cortex-M0 architecture.

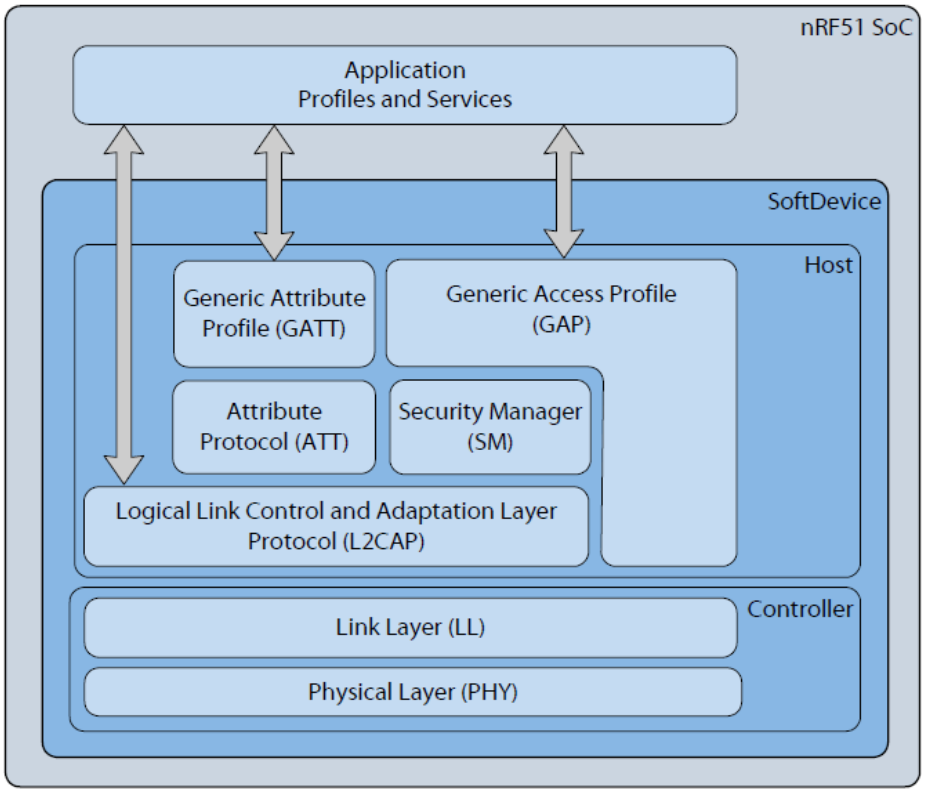


Figure 3.8: nRF51422 Embedded Softdevice [10]

The softdevice block diagram is shown in Figure 3.8. The softdevice abstracts the BLE core specification that calls for three levels of interaction. In this project, encryption is not considered as it is beyond the scope of this project as the goal is not to seek Food and Drug Administration and Federal Communications Commission compliance or certification.

For BLE, there is no continuous uninterrupted transfer mode for data transfer. Instead, the transmitting end transmits an alert notification telling any connected devices that there is data and should receive it. It is important to note that these events happen regardless of whether the initiator requests it or not.

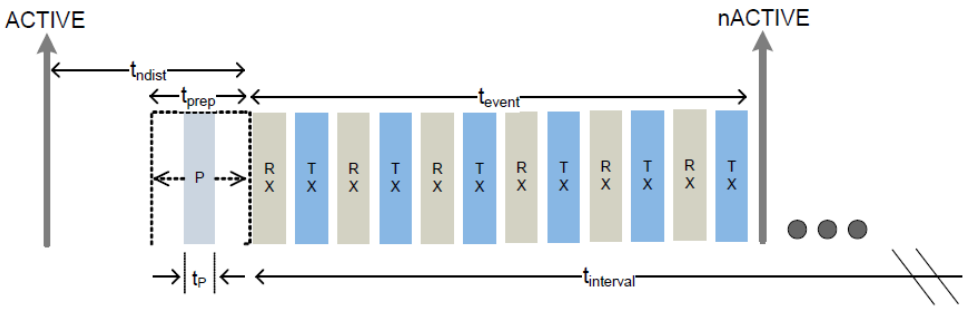


Figure 3.9: nRF51422 Alert Notification Timing while using the S110 Softdevice [11]

Figure 3.9 shows the timing diagram when sending N number of alert notifications to the master.

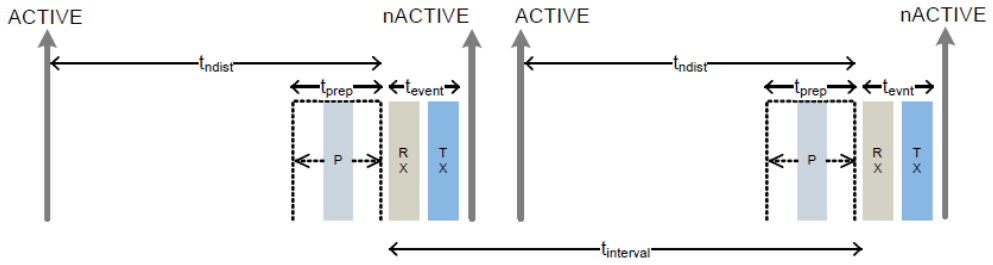


Figure 3.10: nRF51422 Alert Notification Timing while using the S110 Softdevice [11]

Figure 3.10 shows the alert notifications for the S110 Softdevice.

This project's custom profile called ble_ecg to transmit the ECG signal through BLE alert notifications utilizes an address of 0xFF00 with a characteristic address of 0xFF03. The characteristic is periodically updated such that it reaches a desired sampling rate for a specific application. In this project's case, that sampling rate was $3ms$ or approximately $120Hz$. Unfortunately, the softdevice blocks execution of the microprocessor during transmission events and BLE stack events have a higher interrupt priority than application timers set by the user. Therefore, due to the lower execution priority of the interrupt timers than the rest of the stack and the proprietary nature of the library to interface with the stack, there is no certain way to achieve an exact sampling rate without using an external ADC. However, for this project design, data can be sampled at a rate fast enough to accomplish $120Hz \pm 5Hz$.

4 Power

To maintain a low power ECG monitoring system, options for the power block were considered such as the safety, simplicity, portability, and the size. It is desirable for the run-time to be sufficiently long such that the user does not need to worry about changing the battery except throughout the week. In this section, different options for energy storage devices were considered as well as options for energy harvesting. Moreover, power analysis was conducted to determine an estimate of the device's power supply lifetime.

4.1 Background Research

4.1.1 Storage Elements

In general, there are several different methods of energy storage such as inductors, capacitors and batteries which will be discussed in the next few sections. The inductor was eliminated because it was not feasible for this project since inductors are generally unreliable energy storage devices relying on the electric field.

Capacitors

Due to the recent advances in "super" or "ultra" capacitors, these devices were considered as possible storage devices for the MQP design. However, it was discovered that large valued capacitors, still cannot replace the battery in terms of sustaining charge over time as a capacitor will attempt to discharge as fast as possible when conducted, whereas a battery will only discharge under certain conditions. This idea is based on a standard decaying RC circuit and the time constant τ. It is found that capacitors could work well as a ripple smoothing device or as a part of a larger power supply (e.g. regulator with output capacitor). Moreover, it as also found that capacitors and super capacitors work well as momentary backups for lower power circuits that may receive spikes in power supplied to a circuit. However, this requires a supervisory circuit or some comparators to implement this function[26, 27].

Batteries

In portable electronics, batteries are the main source of power as one cannot stay connected to wires all the time while attempting to be portable. Different battery options were considered as shown in Table 4.1 where the highlighted values, meaning the "best one" was considered for the desired application. There is a trade-off of safety for charge density and thus weight. Since this is a portable application, one of the lithium-ion variants were considered as a preliminary design.

In Figure 4.1, it is shown that lithium-ions have a much more desirable (e.g. less than 10% discharge) and larger voltage range than Nickel-cadmium. Thus, they are able to power most

Table 4.1: Secondary Battery Cell Characteristics

System	Cell voltage (V)	Equivalent weight (kg)	Theoretical specific energy (Wh/kg)	Specific energy (Wh/kg)	Specific power (W/kg)	Cost (USD/kWh)
LiS	2.1	0.0216	2600	300	200	<100
$LiSRS$	3	0.08	1000	200 (est.)	400 (est.)	N.A.
LiV_6O_13	2.4	0.0723	890	150	200	>200
$LiFePO_4$	3.5	0.151	621	120	100 (est.)	>200
$LiMn_2O_4$	4	0.1808	593	150	200	>200
$LiCoO_2$	3.6	0.1693	570	125	4200	>100
$LiTiS_2$	2.15	0.1201	480	125	65	>200
$ZnAir$	1.6	1200	65–120	<100	300	100 (est.)
$MHNiOOH$	1.2	200	60–85	200+	500–600	>400
$ZnNiOOH$	1.74	326	55–80	200–300	500+	150–250 (est.)
$FeNiOOH$	1.3	267	40–62	70–150	500–2000	>100
H_2NiOOH	1.3	380	60	160	1000–2000	>400
$CdNiOOH$	1.2	209	35–55	400	2000	>300
$PbPbO_2$	2.1	175	30–45	50–100	4700	60–125

of the circuitry without the need of a boost converter regulator. However, the downside to the higher specific energy density is safety. Therefore, due to the trade-off of safety to weight, the use of a supervisory or specialty charging circuit will be needed. Moreover, the difference between the lithium-ion and lithium-polymer is considered, which as far as research can tell, is related to the electrolyte used in the battery and not the specific electrochemical makeup of a battery. Therefore, a lithium polymer can be a lithium-ion cobalt cell just with the polymer electrolyte. Also, it is noted that the usage of polymer terminology has become a commonplace for mere cosmetic packaging difference (flexible soft shell polymer vs. normal hard shell).

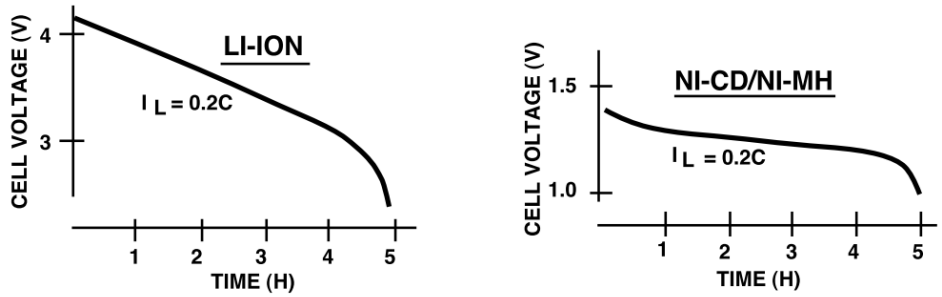


Figure 4.1: Lithium-ion and Nickel-cadmium Voltage over time [12]

As shown in Figure 4.2, lithium-ion batteries needs to be charged using some sort of monitoring mechanism. Because of this, options of using integrated circuits were considered. Although with a single lithium-ion cell, there is no need for the thermocouple, there is still an option to include it for high reliability or high current charging.

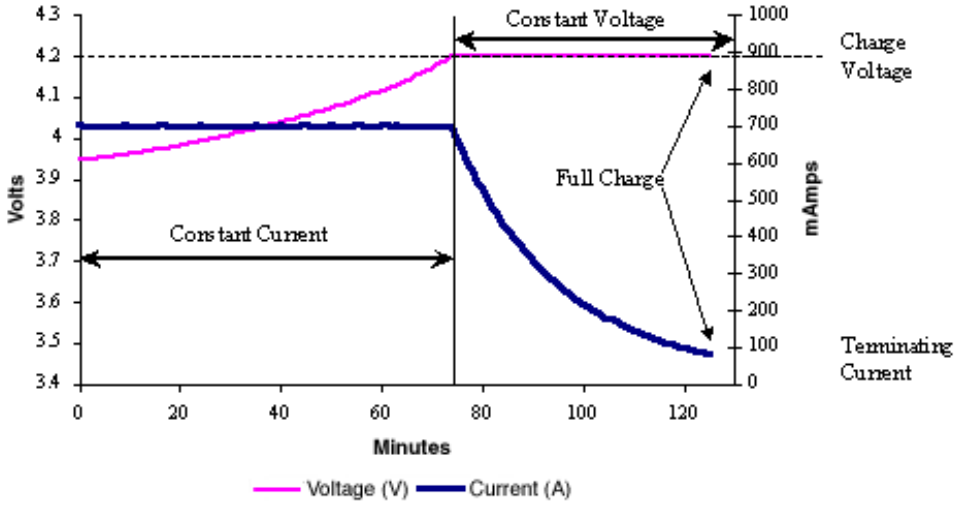


Figure 4.2: Method of Constant Voltage then Constant Voltage Charging [13]

There are more complex digital monitoring systems available such as the AD7280A. However,

it was found that this is far more than needed since it would require to have two comparator banks sitting empty as it expects a minimum of three cells. Figure 4.3 is the circuit that provides all of the features needed for this project. This demonstrates a possible design option.

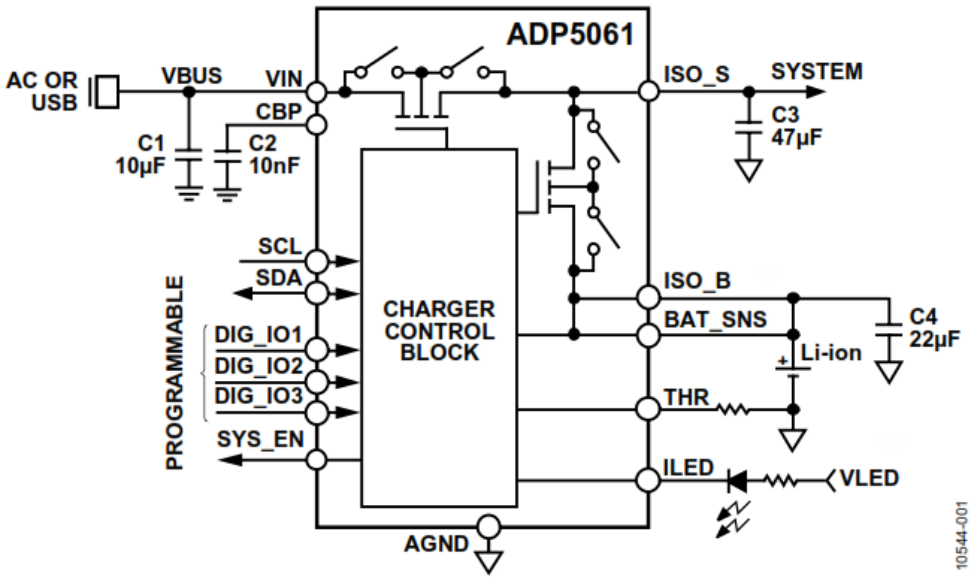


Figure 4.3: Potential Circuit from ADI [28]

Harvesting techniques were also considered as possible power source as shown in Table 4.2. After examining each options such as the feasibility, practicality, and availability, the mentioned energy harvesting techniques were not implemented for this MQP project due to time constraints. Instead, this could be implemented in future projects. In the other hand, an alternate option was found such as the use of battery.

Table 4.2: Energy Harvesting Techniques [19]

Technique	Advantage	Disadvantage
Kinetic	classical spring and mass system	requires changing force with time
Piezoelectric	electric potential accumulates in crystals	not feasible
Thermoelectric and Pyroelectric	produce electric potential	requires constant excitation
Radio Frequency	many radio waves	plausible for future
Photovoltaic	feasible	low efficiency

4.2 Preliminary Design Options

The initial design of this project is shown in the following block diagram on Figure 4.4. In this project, two primary modules consume power: the AFE and the wireless communications.

The AFE runs on a $3V$ voltage supply with an average of $170\mu A$ current consumption while the Communications runs for $3V$ at average of $20mA$.

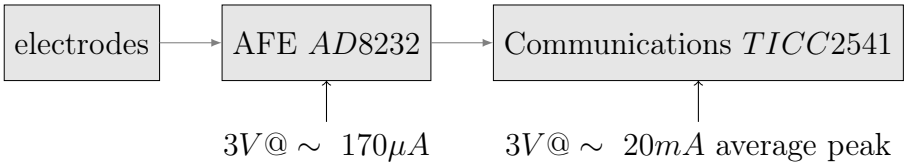


Figure 4.4: Preliminary Block Diagram Power Analysis

Since the AFE has a continuous power discharge, the focus for power block is the periodic current consumption of communications. This is because during normal operation, the BLE protocol maintains synchronous link with the other peer device where data packets are exchanged regularly during connection event. This can be set from tens of milliseconds to several seconds. Packets are sent to the other peer device to stay synchronized. Its current consumption is portrayed through pulse loads as shown Figure 4.5. That is, it consumes most current with an average peak of $20mA$ during a connection event, otherwise it consumes low current of less than $1mA$ during idle mode. Thus, causing a pulsed drain pattern for current consumption.

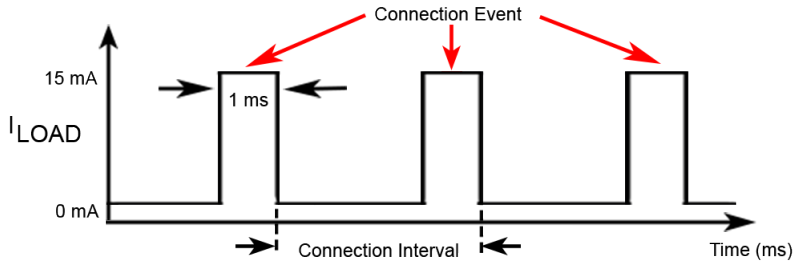


Figure 4.5: Timing Diagram

4.2.1 TI CC2541 Measurements

The preliminary design for this project opted on using the TI CC2541 chip. To find current measurement, the following setup as shown in Figure 4.6 was used. A 10Ω resistor was connected in series to the VDD pin to measure current consumption.

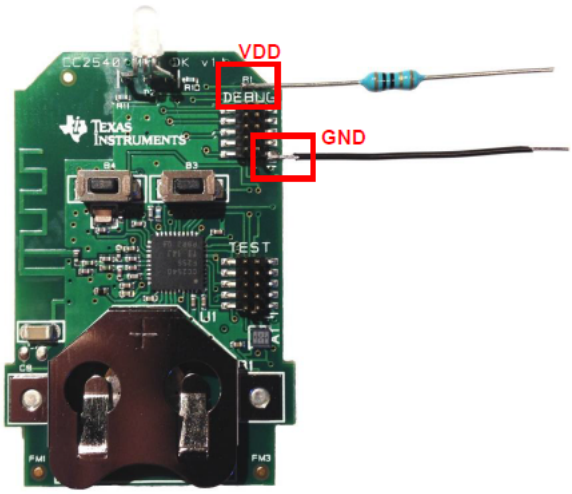


Figure 4.6: TI CC2541 Current Testing Setup [14]

As shown in Figure 4.7, a typical connection event has a few states: sleep, wake-up, pre-processing, Rx, Tx, and post-processing. Sleep mode consumes minimal current of less than 1mA. Upon waking up, current level produces a spike which then drops to a lower current consumption. In the pre-processing state, the BLE protocol prepares the radio for data transmission and receiving. The Rx and Tx usually consumes most current since the radio receiver waits and listens for a packet from the other device while in the Tx, the radio transmits the data packet to the other device. The number of Rx and Tx depends on number of packets sent for each connection interval. One pair corresponds to a single data packet. The last state is post-processing where data is being processed by the BLE protocol stack which then sets up the timer for the next connection event and goes back to sleep mode[14].

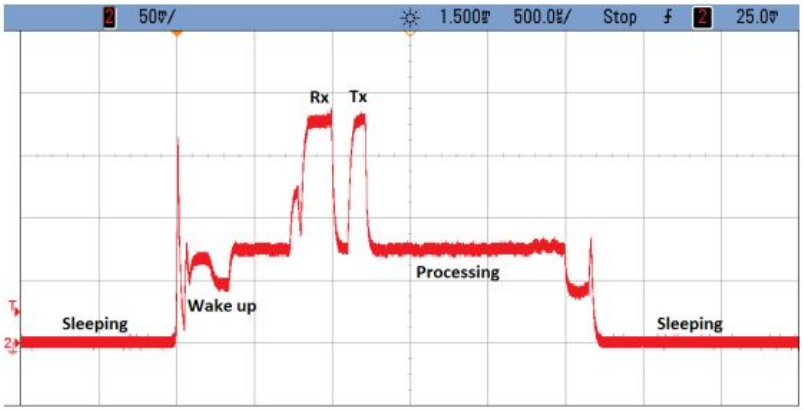


Figure 4.7: Current Consumption vs Time during a single Connection Event [14]

The following Figure 4.8 shows a typical current connection event plot of TI CC2541. Here, the device was configured to send three data packets with maximum of 20 bytes in each outgoing data packet.

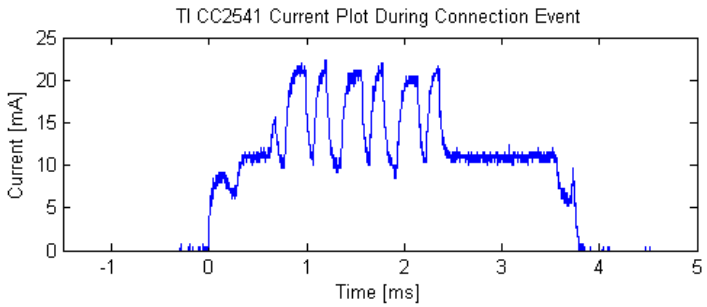


Figure 4.8: Current Measurement for TI CC2541 Connection Event

While the device is in regular operating mode and transmitting at a regular interval equal to that of the packet length (approximately $3ms$), it consumes an average peak of $20mA$ constantly without the negligible current 170μA from the AFE. Using the same setup as the measurement in the previous section, the new measurement code was changed to disable all peripherals except the ADC, the required Bluetooth low energy architecture to minimize and measure current consumption.

Current consumption for TI CC2541 chip was found to have an average peak of $20mA$. Due to implementation issues as discussed in Chapter 3 Communications, it was decided to use the Nordic nRF514822.

4.3 Testing Evaluation

The updated block diagram using nRF515822 is shown in Figure 4.9. Here, it has an average current consumption of 15mA which is less than the initial device TI CC2541.

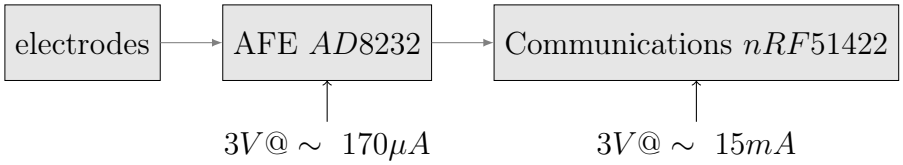


Figure 4.9: Revised Block Diagram Power Analysis

4.3.1 Using Nordic nRF514822

The following setting shown in Figure 4.10 was used for power measurement test. For the board setup, a 10Ω was added and connection on $SB9$ was cut off.

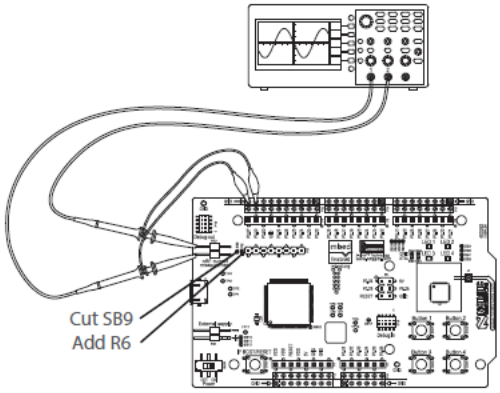


Figure 4.10: Current Measurement Setup for nRF51 [15]

4.3.2 Power Consumption Analysis

One important aspect on BLE communications is that information is sent through aggregated data packets with a set number of bits per connection interval. The nRF51422 SoC used in this project can transmit up to 6 data packets per connection interval where each outgoing data packet can contain up to 20bytes of data. An energy consumption analysis was conducted to decide the most appropriate option to be used for device data packet configuration. The following equations were used for power analysis. Current was found using the Ohm's Law were $R = 10\Omega$.

$$I = \frac{V}{R} \tag{4.1}$$

Energy consumed was found using the following power and charge Equations 4.2 and 4.3, respectively, where V is the supply voltage (in this case, a $3V$ is used)

$$P = V \cdot I \tag{4.2}$$

$$Q = I \cdot t \tag{4.3}$$

$$E = V \cdot Q = V \cdot I \cdot t = P \cdot t \tag{4.4}$$

The expected battery life is found using Equation 4.5.

$$\text{Battery life} = \frac{\text{Corrected battery capacity}}{I_{\text{average max}}}[hours] \tag{4.5}$$

Energy per bit was calculated by measuring the energy per connection interval and divided

it by number of bits sent. A few factors are involved that affects the energy consumption per bit for low power wireless applications. These are the number of bits sent per packet, the number of packets sent per interval, and the ADC sampling rate. A range of connection interval can be set to the slave device however it is the master (i.e User Interface device) that decides a connection event. With the use of smartphones, most Android devices can support at least $7.5ms$ while iOS devices can support at least $30ms$.

Effect of Number of Bits

One factor that affects energy consumption is the number of bits sent per connection interval. An energy consumption analysis was conducted using two sets of data: using a 1-packet and 6-packet interval with varying bit size. The maximum possible energy used for each number of bytes transmitted was measured by finding the total charge per connection interval and divided by the time interval set by the device. The following parameters were used:

Number of Bytes	1 to 20
ADC Sampling	16 msec
Min Connection	10 msec
Max Connection	10000 msec
Slave Latency	0
Supervisory Timeout	4000 msec

Different range of bits were set (from 8 to 160) to measure the energy consumption. Results show that transmitting 1 packet of data requires at least a connection interval of $10ms$ while transmitting with at most 6 packets of data requires at least a connection interval of 100ms. Longer connection interval is needed for multi-packet as it requires more transmission and processing time.

Table 4.3 and Table 4.4 shows the energy consumption using 1-packet and 6-packet per connection interval, respectively. It is shown in the "Average Time on" column that the 6 packet requires almost 3 more times than transmitting only 1 packet. This is because more data is being sent per interval. It is also shown that the energy per bit significantly decreases as more bytes are sent for each connection interval.

For this analysis, battery life was estimated using the coin cell CR2032 with capacity of $175mAh$ which was found by calculating the average current used for each connection interval. In this case, the connection interval is 10ms. It is assumed that the device is continuously transmitting data the whole time which is the worst-case scenario. Results show that for 1-packet per interval, the device could last almost a week. For a 6-packet per interval, the device could last for 2 weeks.

Table 4.3: Energy Consumption with 1-packet at $\tilde{1}$0ms connection interval

Bits	Average Time On ($msec$)	Energy interval (μJ)	Energy per Bit ($\mu J/s$)	Average Power (mW)	Charge (μC)	Ave Current per Interval (mA)	Estimate Battery life w/ 175mAh ($hours$)
8	1.040	26.920	3.365	2.692	8.973	0.897	195.021
16	1.040	27.378	1.711	2.738	9.126	0.913	191.760
24	1.040	26.198	1.092	2.620	8.733	0.873	200.394
40	1.100	28.135	0.703	2.813	9.378	0.938	186.603
80	1.140	29.423	0.368	2.942	9.808	0.981	178.429
120	1.180	29.765	0.248	2.977	9.922	0.992	176.379
160	1.220	32.495	0.203	3.249	10.832	1.083	161.564

Table 4.4: Energy Consumption with 6-packet at 100ms connection interval

Bits ($msec$)	Average Time On (μJ)	Energy interval ($\mu J/s$)	Energy per Bit (mW)	Average Power	Charge (μC) (mA)	Ave Current per Interval ($hours$)	Estimate Battery life w/ 175mAh
8	3.800	115.366	2.403	1.154	38.455	0.385	455.075
16	3.840	118.661	1.236	1.187	39.554	0.396	442.438
32	4.360	129.259	0.673	1.293	43.086	0.431	406.161
80	4.800	139.433	0.290	1.394	46.478	0.465	376.525
120	4.920	148.703	0.207	1.487	49.568	0.496	353.053
160	5.160	157.066	0.164	1.571	52.355	0.524	334.255

The energy usage per bit was plotted comparing the 1-packet and 6-packet as shown in Figure 4.11. Results show that energy per bit decreases as more number of bits is sent per interval. Moreover, when more number of packets is transmitted per interval, the energy consumption also decreases. The red mark highlights what was used for the project: a 1-packet with maximum 20bytes was selected since it uses less energy of less than 0.5μ J/bit. The range for connection interval must be chosen appropriately considering the trade-off between responsiveness for short interval and battery life for longer interval.

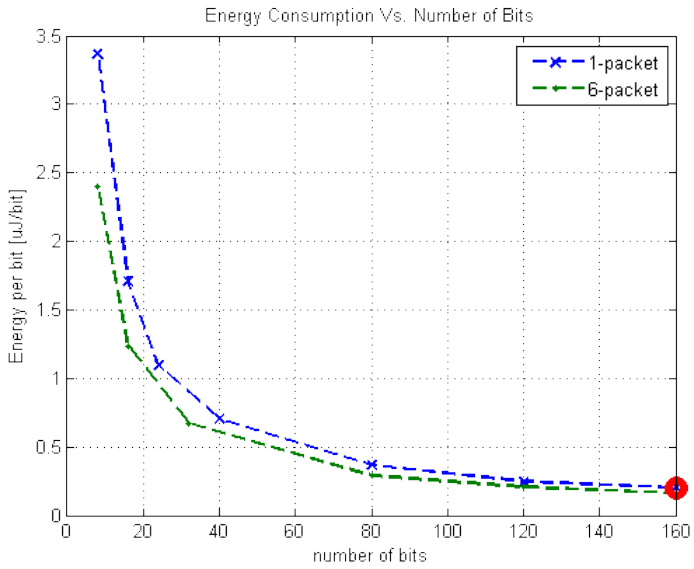


Figure 4.11: Calculated data for Energy Consumption vs. Number of Bits

Effect of Varying Packet Size

As presented in the previous section, the number of packets sent for each connection interval has significant effect on energy consumption. Different scope measurements were also made to capture the power consumption depending on the number of packets. The Nordic nRF51822 can support 1 to 6 packets. However, the size of the packets cannot be increased since it is limited by the SoftDevice and the Core Specification. The number of packets that can be sent in each connection interval depends on what central is used. Most Android devices support up to 4 packets per connection interval while iOS devices supports 6 packets[29].

The connection interval is modified to accommodate a particular number of packets. This is controlled using the Master Control Panel software available for nRF51 devices. The following settings were used for packet size analysis:

Number of Bytes	1
ADC Sampling	16 msec
Min Connection	10 msec
Max Connection	10000 msec
Slave Latency	0
Supervisory Timeout	4000 msec

Table 4.5 shows the results of energy consumption depending on the packet size. Setting a longer connection interval also indicates longer battery life.

Table 4.5: Energy Consumption with Varying Packet Size at ADC sampling rate of 60Hz

Packet Size	Connection Interval (ms)	Average Time On ($msec$)	Energy interval (μJ)	Energy per Bit ($\mu J/s$)	Charge (μC)	Ave Current per Interval (mA)	Estimate Battery life w/ 175mAh ($hours$)
1	10	1.000	31.384	3.923	10.461	1.046	167.285
2	25	1.600	45.746	2.859	15.249	0.610	286.908
3	45	2.080	63.018	2.626	21.006	0.467	374.893
4	70	2.600	85.294	2.665	28.431	0.406	430.865
5	90	3.160	101.684	2.542	33.895	0.377	464.673
6	200	3.800	115.366	2.403	38.455	0.192	910.150

The energy consumption per bit was measured and the energy per bit was calculated for each number of packets. Figure 4.12 shows a plot of energy consumption per bit depending on the number of packets transmitted for each connection interval. It is shown that as more packets are transmitted, less energy per bit is consumed.

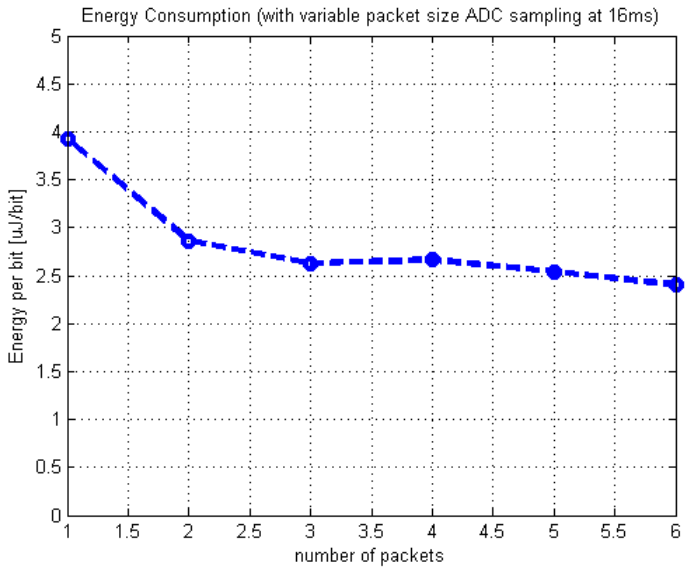


Figure 4.12: Measured Data for Energy Consumption vs. Number of Packets

When the ADC sampling rate was changed to $64msec$ ($15Hz$), the following measurements were gathered as shown in Table 4.6

Table 4.6: Average time on with varying Packet size at ADC rate of $\sim$ 15Hz

Number of Packets	Connection Interval ($msec$)	Average Time On ($msec$)
1	100	1.160
2	$\sim$ 100-200	1.440
3	$\sim$ 200-350	2.440
4	$\sim$ 350	2.800
5	$\sim$ 350-500	3.400
6	500*	3.800

*max connection interval set on Master Control Panel Software

It is shown that increasing the ADC sampling rate would also require more current consumption.

Effect of ADC sampling rate

Energy consumption was measured depending on the ADC sampling rate as shown in the Figure 4.13 and Table 4.7. The connection interval was set to a range of 10ms with 1 byte data sent per packet. When the sampling frequency is increased, there are more packets transmitted per connection interval. In addition, when packets being received were checked through the Nordic Packet Sniffer using Wireshark, it is shown that there are less empty packets transmitted when the sampling rate is higher.

Table 4.7: Energy Consumption Depending on the ADC sampling rate

Packet Size	ADC Sampling Rate (Hz)	Energy interval (μJ)	Energy per Bit ($\mu J/s$)	Charge (μC)	Ave Current per Interval (mA)	Estimate Battery life w/ 175mAh ($hours$)
1	60	27.402	3.425	9.134	0.913	191.592
1	66	33.714	4.214	11.238	1.124	155.722
1	83	31.134	3.892	10.378	1.038	168.626
1	100	23.950	2.994	7.983	0.798	219.210
2	142	25.376	1.586	8.459	0.846	206.885
2	250	26.876	1.680	8.959	0.896	195.343
5	500	24.561	0.614	8.187	0.819	213.752
6	1000	23.061	0.480	7.687	0.769	227.655

Figure 4.13 shows the measured data with varying ADC sampling rate and the expected theoretical prediction. Results show that increasing sampling rate would use less energy.

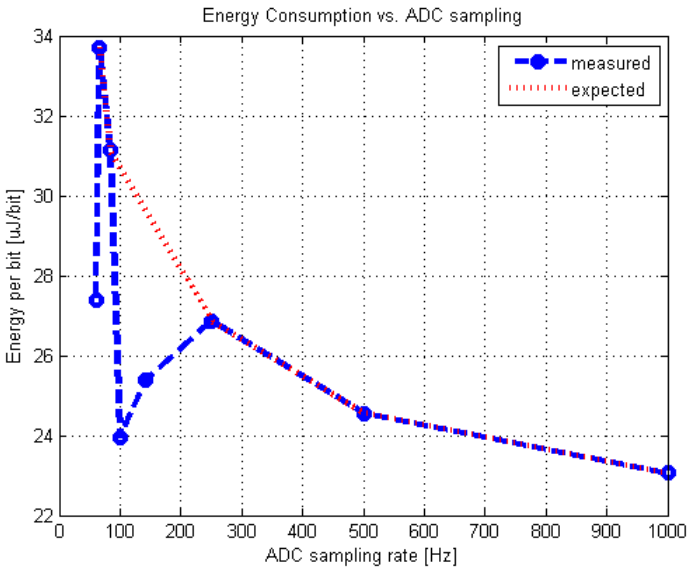


Figure 4.13: Energy Consumption vs. ADC sampling rate

Scope Plots

Current plots for nRF51 low power application has mainly three sections, the advertising event (where the central device advertises data to the Android device), the connection event (where no data is being sent yet), and connection event when data is transmitted and services are enabled. These events are important to be considered to estimate an expected battery life. Figure 4.14 shows a typical nRF51 series current plot of advertising event and connection event, respectively.

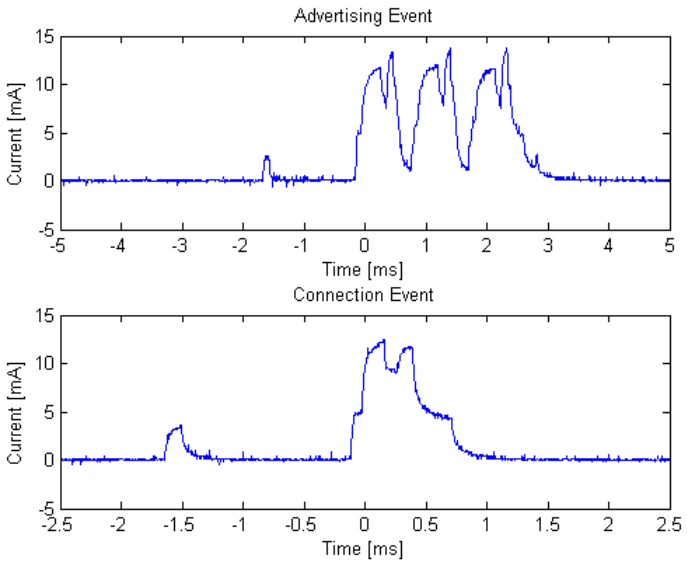


Figure 4.14: Nordic Current Waveforms (Advertising and Connection Event)

The following Figure 4.15 shows a current scope plot during connection event with different packet length of 1, 3, and 6 respectively.

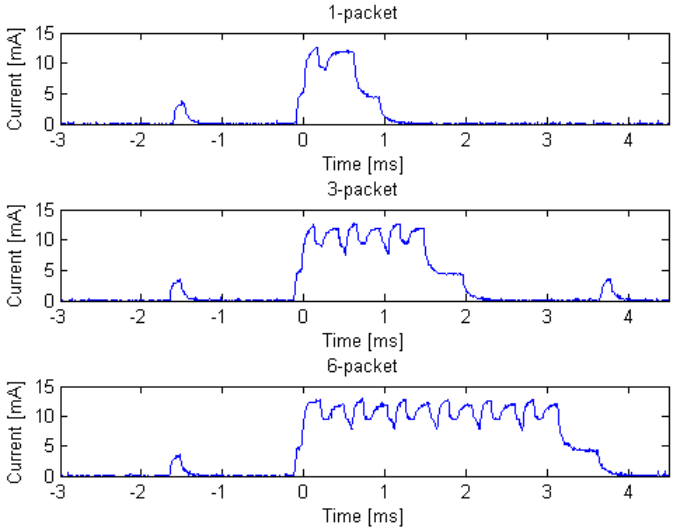


Figure 4.15: Nordic Current Waveforms Multi-packet

Table 4.3.2.4 shows measured energy consumption for each event. Based on the results, the advertising consumes more energy that during connection event. This is because the device would require to send data for advertising. Once a connection is established, energy consumption decreases to about half.

Event	Energy Consumption (μJ)
Advertising	73.39
Connection (empty, no data sent)	23.32

4.4 Design Decisions

Based on the requirements for AFE and Communications block, the CR2032 battery was chosen since it could provide a $3V$ power supply. Table 4.8 shows the coin cell specifications. One important factor to consider for a lithium coin battery is the effect of discharge rate on the available capacity. When battery is put under high load, there could be de-rating. This is caused by the internal resistance (for CR2032, 30Ω) of the battery which causes a voltage drop that is proportional to the current being drained. Moreover, the battery voltage will continue to go down because of the polarization from electrochemical process that is slower than the drain rate applied as the battery is made up of lithium metal and manganese dioxide [30].

Table 4.8: CR2032 Specifications [20]

Characteristic	Value
Nominal Voltage	3V
Nominal Capacity	240mAh
Dimension	20.0mm x 3.2mm
Standard load current	0.2mA
Internal Resistance	10-40Ω

The battery capacity of CR2032 during continuous discharge is shown in Figure 4.16 below. In this case, the battery is drained with continuous currents at different range from $0.5mA$ to $3.0mA$. It is shown that the capacity decreases as the rate of discharge increases.

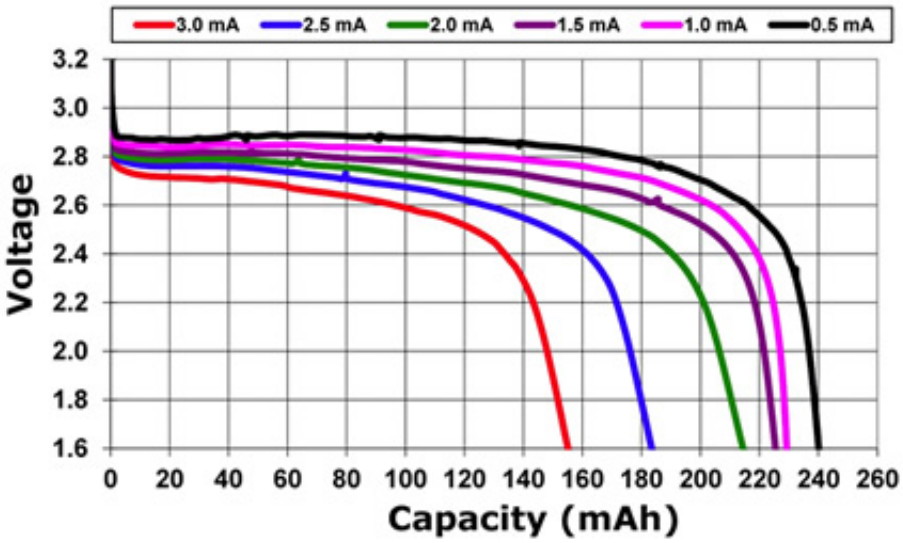


Figure 4.16: CR2032 Coin Cell Continuous Discharge [16]

Another important factor is the Functional End Point (FEP) which is the minimum voltage level for the device to operate. The AD8232 has a supply operation of $2.0V$ to $3.5V$ while the nRF51 operates from 1.8V to $3.6V$. In this application, the FEP would be $2.0V$. Based on the graph, the coin cell battery can deliver a full capacity of $240mAh$ if it has a discharged rate of $0.5mA$. When it drains at $3.0mA$, the capacity decreases to 150mAh[16, 30].

However, the Bluetooth low energy low power wireless application is characterized by pulse loads during connection events. Instead of continuous discharge, it is periodic, has a set period of time it is awake and then back to sleep mode. These pulses vary depending on the pulse period or connection interval. Figure 4.17 below shows different peak current magnitude and its impact on the battery capacity of CR2032 with a pulse cycle of $25ms$. With a FEP of $2.0V$, the capacity with $10mA$ peak current results to a lower capacity less than $200mAh$. For the MQP device application, an estimate of 15mA peak current would results to about $175mAh$ battery capacity.

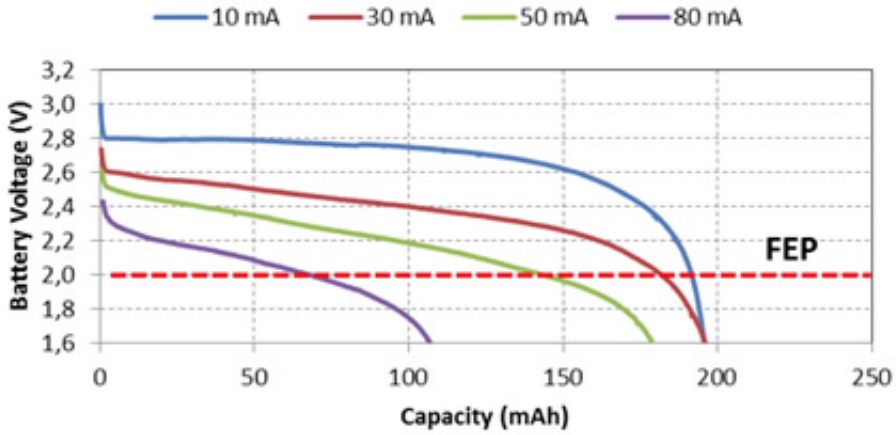


Figure 4.17: Peak Current Magnitude impact with 25ms pulse cycle on CR2032 Battery Capacity [16]

4.4.1 Maximizing Battery Capacity

Because of high peak current pulse loads for low power wireless applications, coin cell battery has voltage drop due to internal resistance and leads to degradation of battery capacity. Coin cell batteries are designed for low continuous standard load current of $<1mA$. They have high source impedance and do not sustain high loads. Internal resistance increases rapidly when battery is fully discharged. Because of this effect, few options were considered to maximize battery life. First option is to maximize the FEP margin such as limiting to a minimum supply voltage (in this case, $2.0V$ for AFE and nRF51). Another option is to minimize current drain by having the device to consume lowest peak current (of about $10mA$). The last option is by adding a capacitor in parallel with the coin cell[30, 16]. For this project, the third option was considered which will be discussed further in the next section.

4.4.2 Parallel Capacitor Calculation

One common technique to handle the high peak currents is the use of capacitor that acts as primary power source during high current periods. During low current periods, the battery will be the primary source and then recharges the capacitor.

The proposed circuit would have a large capacitor placed in parallel with the coin cell battery and a 10Ω series resistor which is used to monitor the behavior of supply current as shown in Figure 4.18.

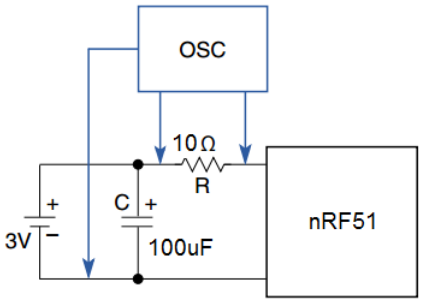


Figure 4.18: Proposed circuit for battery life improvement

There are also some considerations for choosing a parallel capacitor. It can be bypass with large value capacitor having low leakage using ceramic capacitor to reduce power loss due to battery's internal resistance. Leakage current rating of the capacitor must be checked to ensure that it is good enough for the desired battery life. Capacitor leakage current can continuously discharge battery and depends on factors such as capacitor value, nominal voltage rating, and production process used. Short peak current transients followed by long recovery time are better for coin cell (allow time for battery to recover).

4.4.3 Improving Battery Life

To improve the battery life, a high enough capacitor with low Equivalent Series Resistance (ESR) to deliver power for the duration needed with sufficient energy storage must be found. A parallel capacitor analysis was conducted for this project. For analysis, the nRF51 was set for $25ms$ connection interval, using a $3V$ CR2032 coin cell battery with a duty cycle of 20% (5ms @ $15mA$ awake, $20ms$ @ $< 1ma$ sleep)

In this project application, current is not constant and charging or discharging the capacitor over a resistor will go exponentially. Using the maximum supply voltage of 3V and a minimum supply voltage of 2V for device to be operational, a $1V$ difference is calculated. However, it is not desired to have the capacitor voltage to drop too much before some time. For analysis, the maximum allowed voltage was determined to be $200mV$. Equation 4.6 and 4.7 shows the exponential decay and RC time constant, respectively.

$$V_{\text{out}} = V_f - (V_f - V_i)^{e^{\frac{-t}{R \cdot C}}} \tag{4.6}$$

$$\tau = R \cdot C \tag{4.7}$$

Solving for the time constant based on Equation 4.7 with a maximum of $200mV$ drop, get:

$$\tau = R \cdot C = \frac{t}{ln(\frac{V_f - V_i}{V_{\text{out}} - V_f})} = \frac{5\text{ms}}{ln(\frac{3V-2V}{2.8V-3V})} = 0.022407s$$

Using time value calculated and the typical series resistance of coin cell battery (R=20Ω), the capacitor value was found to be:

$$C = \frac{\tau}{R} = \frac{22.407\text{ms}}{20\Omega} = 1120\mu F$$

To find which size would be most appropriate, different voltage drops were used from $50mV$ to $500mV$. Table 4.9 shows the result when varying the output voltage and its corresponding capacitor value.

Table 4.9: Final Voltage vs. Capacitor Value

Final Voltage (V)	Capacitor (μF)
2.95	4870
2.9	2370
2.8	1120
2.7	700
2.6	490
2.5	360

Based on Table 4.9, different capacitor values were simulated through PSpice varying from $100uF$ to $4700uF$ to show the current through the capacitor, load, and battery. The series resistance of coin cell battery was set to typical value of $Rbat = 20\Omega$. Figure 4.19 shows the circuit setup for testing currents with varying capacitor values.

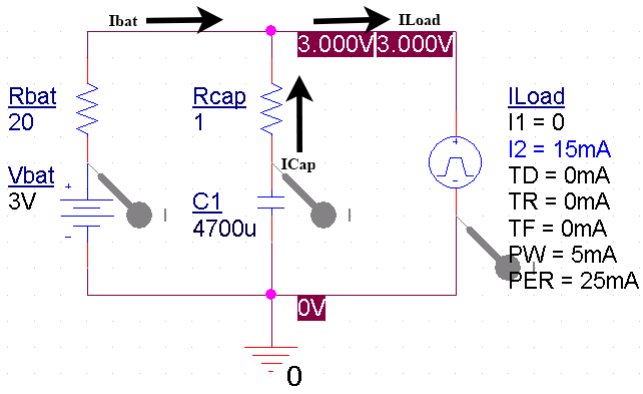


Figure 4.19: Battery Parallel Capacitor Circuit

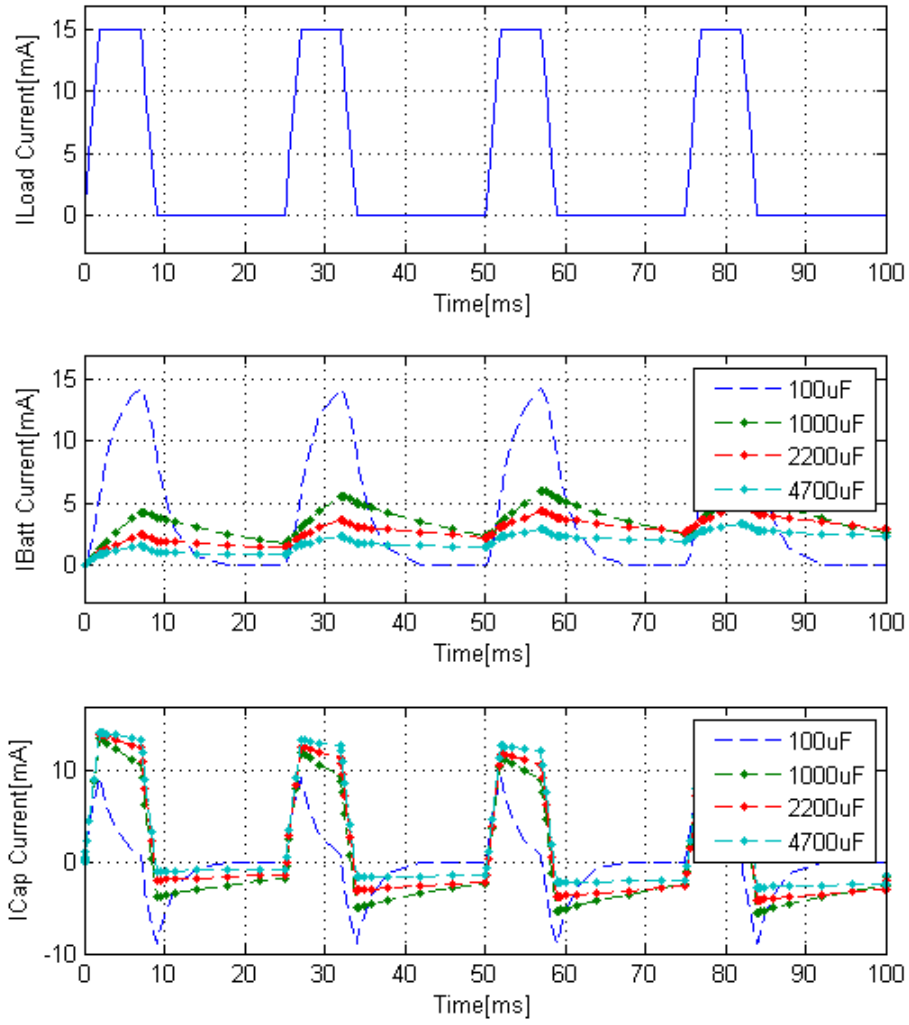


Figure 4.20: Transient Analysis (pulse load w/parallel capacitor circuit)

As shown in Figure 4.20, the capacitor value has direct impact to battery current consumption. The higher the capacitor value, more power is being supplied by the capacitor leading to a smoother battery current consumption. With only $100\mu F$ capacitor, it is not sufficient for the load current since the capacitor charges and discharges quickly that would still require for more battery current supply. The desired capacitor value would be as large as $2200\mu F$ which allows a smooth battery current of about $5mA$. The following Figure 4.4.3 and Figure 4.21 below shows current for load, battery, and capacitor respectively using a $2200\mu F$ capacitor.

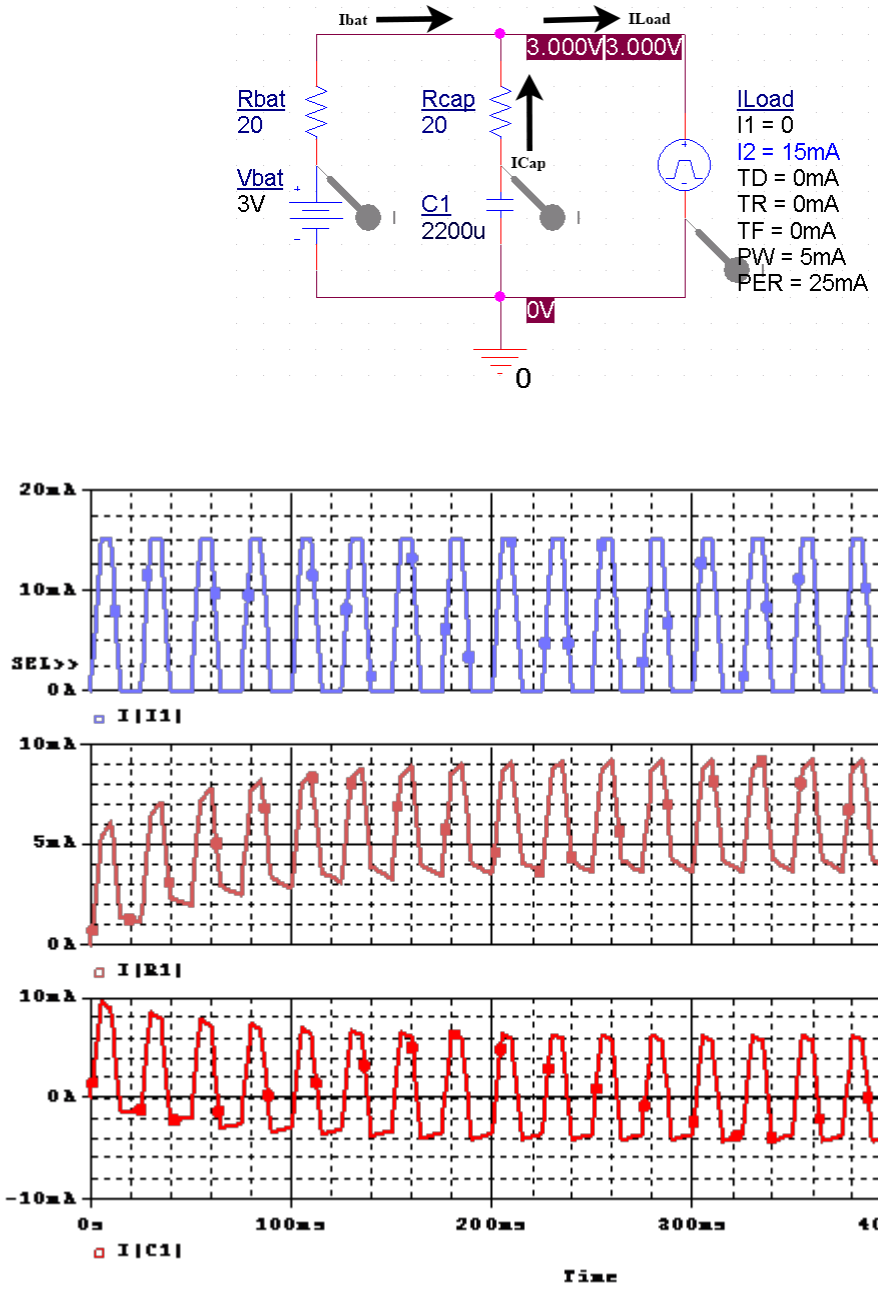


Figure 4.21: Current plot for load, battery, and capacitor respectively

Results show that a higher value capacitor would be needed. However, issues were encountered after conducting research on the available capacitor size that would be most appropriate for the device. This is due to the internal resistance of the capacitor with respect to size. The capacitor price with lowest ESR possible are generally high. In addition, bigger capacitors have larger ESR (i.e. $1000\mu F$ capacitor has about 10Ω ESR) which would greatly affect the charging and discharging of the capacitor. Thus, it was decided to be an option for future works since battery life was estimated to last for about two weeks.

5 User Interface

The final module in the general system block diagram for this project is the user interface. This block is where the visual results are displayed through electronic devices such as computers or smartphones after receiving data from the designed device. Due to increasing popularity and availability, a display through smartphone was chosen. However, another interface on laptop/ PC devices with the nRF51-dongle is also developed side by side.

5.1 Background Research

5.1.1 Mobile Platforms

Currently, the most prominent mobile platforms on the market are Android, Blackberry, iOS, Windows Phone, and Fire OS. Among those, the most widely used in the current generation are Android, iOS, and Windows Phone. Some of the criteria used to compare each platform are: available Software Development Kit (SDK), programming languages required to develop an app, development tool cost, and supports for Bluetooth communication. These criteria will be discussed further in the following subsections.

Android

The first popular mobile platform is Android by Google. The required software development kit (SDK) - Android Studio is available for installation through Android's developer site free of charges[31]. The SDK is compatible with Windows, Linux, and Mac operating systems. The programming language used for developing Android applications is Java with Android-specific APIs and frameworks. Even though there is no cost for Andrid developers, a registration fee of $25 is applied if the team decide to publish the app to Google Play Store. However, once registered, the team can immediately publish the app wihtout further inspection from Google[32].

iOS

Another popular mobile platform is iOS by Apple. The SDK for iOS is Xcode, which is free for download via Apple App Store but only for Mac OS X Lion users. Mac users with older versions of the operating system will need to buy the SDK and other operating system users such as Windows or Linux will have to use a virtual machine to buy and use this software[33]. The programming language for iOS app is mainly Objective-C. The resource library is available online at the iOS Dev Center[34]. Unlike Android, iOS source code is available only for developers that have an Apple Developer Account with an annual fee of $99. In order to publish the app to Apple App Store, a waiting time of approximately one

or two weeks is required. Moreover, Apple provides strict regulations when someone wants to publish an app to their store [35] [36] [37].

Windows Phone

The last mobile platform that was considered is the Windows Phone by Microsoft. Visual Studio is required to develop Windows Phone apps. There is a license fee for regular developers but Microsoft provides free licenses for students. The main programming languages for Windows Phone is Visual Basic or C#. Therefore, it is free to develop a Windows Phone app but a publication fee of $19/year is required for publishing the app to the Windows App store[38].

Comparison

In table 5.1, the three mobile platforms are compared in regards to which Software Development Kit is used, costs for development and publication, programming languages, and their support for Bluetooth low energy communication. Android app can be easily published on Google store while the iOS app and Windows Phone app might take a week waiting for approval. The Android platform itself is open-source. Testing the application through emulators will require some time. Android emulators are usually slow and take a while to load the application. It is preferable to use real Android devices for testing as they are faster. The iOS simulator is fast and more efficient. The three platforms use different programming languages (Java, Objective-C, and Visual Basic), have different software development kits (SDKs), and utilize different development tools.

Table 5.1: Mobile Platforms Comparison

Mobile Platform	Development Kit	Cost for Development	Cost for Publication	Programming Language
Android	Android Studio	Free	$25 fee for account	Java, C C++, HTML
iOS	Xcode	$99/year	Included with developer account	Objective-C
Windows Phone	Visual Studio	Free	$19/year for account	Visual Basic

5.1.2 App Builder

As an alternative, there are tools to design mobile application that do not require any extensive programming skills. Three popular web-based tools are the iBuildApp, AppyPie, and Conduit Mobile.

Appy Pie App

There is no coding required to build a mobile app from this tool. Appy Pie App supports iPhone, Android, and Windows Phone. The app builder provides an easy 3-step procedure to build a mobile app: choose a category, build the app, and publish the app. It also provides features such as analytical review of user usage, push notifications, social media integration and monetize (to earn money with ads). Some of the downsides of this app builder are the requirement for a subscription fee depending on the plan the customer uses and no support on Bluetooth devices [39].

iBuildApp

Another web-based tool that the group found is called iBuildApp. iBuildApp only supports iPhone and Android. This tool sets up a customizable layout of an app. It provides promotional tools such as social channels and built-in notification feature for users to view information easily [40].

Conduit Mobile

The third web-based app builder that was found is called the Conduit Mobile. Similar to the other previously mentioned platforms in this section, it requires no coding. It supports iPhone, Android and Windows Phone. Some features that this builder provides are: analytics of user usage to developers, push notifications, social media tools, and the ability to share the app. Conduit Mobile requires a monthly subscription fee for different plans: free for basic, $33 for Gold, and $83 for Platinum. One negative of this platform is it does not support Bluetooth devices [41].

Reviews

These building platforms allow an easier way of creating apps. They support any of the mobile operating systems mentioned in the previous section: iPhone, Android, and Windows Phone. There may be some limitations for Windows 8 phone since the Microsoft Company will release a newer version operating system. Features such as push notifications and social media sharing are also included on both platforms. One flaw in these object-oriented platforms is the limited support on wireless communication. The "do-it-yourself" platform is more beneficial for business firms[42].
Although these app builders require little to no programming experience, they offer a very limited set of features especially for data input and limited wireless communication - two significant factors for this MQP. There also cost more based on monthly subscription fee compared to developing our own app. These web-based app builders mostly target business, restaurants, blogs, and other firms that require only a small sets of feature but mainly social

media exposure. Moreover, customer reviews show that these platforms are buggy and slow and the customer support from these developer companies are not very satisfying, rarely offer further development beyond the very basic features of the apps[43].

5.1.3 Existing Applications

Existing mobile apps in the existing App Stores (Apple's App Store, Google Play Store and Windows Market) were also researched as possible references for what the MQP team chooses to pursue. Some of the more popular apps will be reviewed in the following paragraphs. *EMF Meter* (found in Google Play) uses the built-in magnetic sensor in phones to measure tri-axis magnetic fields. *Runtastic Receiver and Heart Rate monitor* is equipped with a chest strap that tracks hear rate. Lastly, the third project is *Wireless Body Sensor Networks (WBSN) system* for real-time ECG analysis application that started in 2011 which uses a smartphone app to display real-time biomedical monitoring.

EMF Detector

This app is available on both iPhone and Android but with different versions. The main purpose of the app is to track high magnetic field within the phone's proximity. The magnetic fields can be emitted through any surrounding electrical devices. The app will signal a warning beep if the phone is near radiation. The EMF detector detects strength of magnetic force fields using the built-in magnetic sensor of iPhone or the magnetometer of an Android phone. It is capable of measuring tri-axis magnetic fields and the measured values are displayed as an x-y-z plot with units of milligauss (mG). The sensitivity of sensor can also be adjusted to provide better results and to minimize interferences [44] [45].

Runtastic Heart Rate Monitor

This app aims solely at tracking the heart rate of the user. It is available on iOS, Android, and Windows Phone and it comes with a chest strap to detect the heart rate. It displays statistics through Bluetooth sensor of the phone. A headphone jack is built into the receiver so that users would be able to use the app during physical activity[46].

Wireless Body Sensor Networks ECG

This app is used for automatic pathology detection using a real-time wavelet-based electrocardiogram delineation system. This device also tracks heart rate real-time on a smartphone that uses FreeRTOS[47] operating system. It is equipped with ultra-low-power TI-MSP430 microcontroller with $8MHz$ clock frequency, $10KB$ RAM, and $48KB$ flash. It includes a wearable embedded platform called *Shimmer* which includes wireless body sensor nodes

(WBSN), attachable to a person's body. The *Shimmer* platform has two radio chips (Bluetooth and IEEE 802.15.4-compliant). The main purpose of this system is for biomedical monitoring and auto diagnosis[48].

5.1.4 Decision

Requirements

The user interface has certain requirements. In order to assess the feasibility of developing the user interface all the requirements must be considered. Familiarity and exposure of the developers to the mobile app development environment and language are one criteria to consider. In other words the target programming language must be easy to work with or easy to learn. Another criteria was availability of mobile device for testing. Since the designed app will need to be able to connect with the sensors and physical peripherals of the phone, using an emulator is out of the question. Therefore, access to device of the intended platform is essential to the development of such app. Finally, the cost for development and publication, affordability needs to be considered while choosing a developing environment. As an example Android has a one-time fee of $25 that provides unlimited publication for all apps that the original developer create. Other platforms require an annual fee for publication. Meanwhile Apple requires $99/year and Windows requires $19/year.

Final Decision

The team finally decided to develop the mobile application on Android platform for three reasons. The most important reason is the developer's better familiarity with the coding language of Android (HTML, Java and C) over Objective C or C#. The second reason was the availability of Android devices to the team. The team personally own four Android devices but only two iOS devices and no Windows Phone devices. And finally, the costs for developing and publishing Android app is significantly lower than the other two platforms. The SDK for Android - Android Studio is available free of charge to all developers. Moreover, Google Play Store only requires a one-time fee for publication while the other two app stores requires an annual fee.

5.2 Development

5.2.1 Structure

This Android app is developed using Google's standard Android SDK - Android Studio and the Apache Cordova library[49]. Cordova is a Web-to-mobile interface for building mobile applications through web development language. Specifically, it allows the user to use HTML, CSS and JavaScript to build a website interface and then transforms this web page into a mobile app. With Cordova, the team can develop a simple Android app much faster and

easier than to building a traditional Android app frm the ground up using Android Studio or Eclipse. Cordova also grants the app much greater compatibility to other mobile platforms. For example, an Android built using Cordova can be easily migrated to iOS or Windows Phone without changing a large portion of the code. The user simply need to replace the Android plug-in base platform provided by Cordova with one used with a different platform. At the time writing, Cordova supports a very wide variety of mobile platforms including iOS, Android, Windows Phone, Amazon Fire OS, Blackberry and Symbian[50]. A disadvantage of using Cordova is that it creates a small delay in processing of information input to the app due to the extra time to convert between the web interface and the Android API (see Figure 5.1). However, while testing the app, the team found no significant delay so this setback is not a big concern overall. The flow in figure 5.1 explains the code flow of the app.

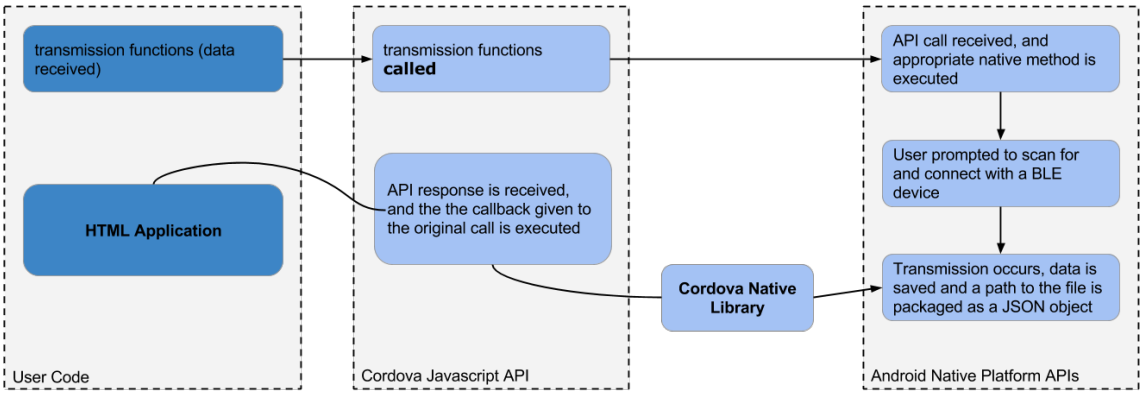


Figure 5.1: Flow Chart

5.2.2 Functionality

The Android application the group developed was named *HeartRate*. The app uses the BLE standard protocols to communicate with all the tested chips. In previous installments initially, the app was designed specifically to communicate with the TI CC2541 Keyfob. The communication was functionally successful when tested with simulated heart rate data from the chip. However, when sending data measured by the ADC of the TI CC2541 Keyfob, there seemed to be a few issues with stability and consistency of the signal. Coupled with some other reasons as discussed in section 3.1.2. After migrating to the nRF51422 the fundamental design of the both installments of the app didn't change. The only part that required modification was the BLE stack service used to communicate with the transmitting devices (in this case either the TI CC2541 or the nRF51-DK).

The interface of the app is displayed in the figure 5.2. The app has two buttons: one to refresh the list of detected devices and the other to disconnect with the currently connected transmitting device. An important note is that the app is designed solely as a receiver. So, although it has the ability to detect all advertising BLE devices, it will only connect with

transmitting devices. When attempting to connect with a receiver device, the app will still appear to connect but will not receive any data. In some cases, the app will crash after an extended time out. Another note is that the app is also designed to react with a specific advertising BLE service. Attempt to connect the app with a transmitting device with a advertising service different from the one specified in the design will either provide a false result or crash the app.

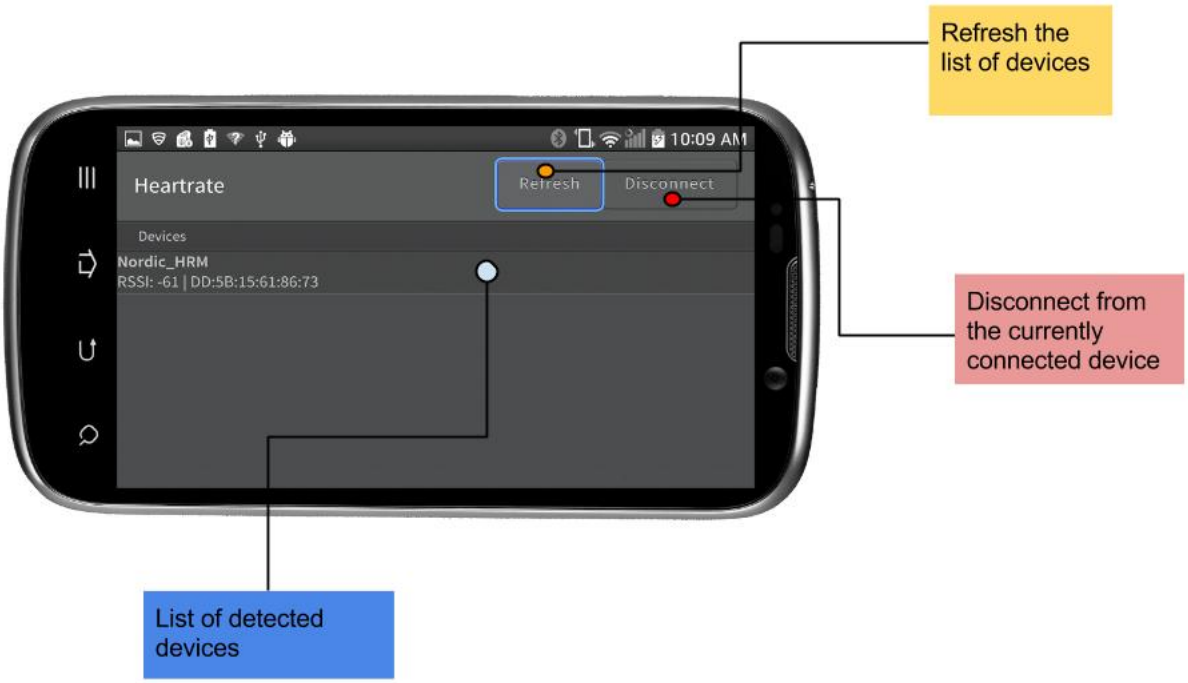


Figure 5.2: Functions of the main screen

For the complete code of the Android app, please check `https://github.com/lhnguyenwpi/MQP-icapps-14-15`.

5.2.3 Alternative solutions

The alternative solution for user interface is to connect the transmitting board (nRF51-DK) with its supporting dongle (nRF51-Dongle) and use a computer interface to graph the data. The dongle is flashed with the S120[51] proprietary software and the computer interface in this case is Python with a Dynamic Graphing Library call matplotlib [52]. The code for this interface can be seen in listing 1 in section 10. The code can be executed with Python but requires some extra packages to be installed. A suggested way to execute the code is to use a more graphic-oriented compiler of Python called Anaconda [53]. Another interface using C++ is also simultaneously developed. C++ offers a faster and more versatile graphic library and can process real-time signal processing but requires more modification through a UART interface to communicate with the nRF-Dongle.

6 System Design

The final general block diagram for this project is shown in Figure 6.1 which has four main blocks such as AFE (using AD8232), communications (using nRF51422), power (using coin cell battery), and user interface. Two revisions were fabricated for the printed circuit board (PCB) which will be discussed in the next sections.

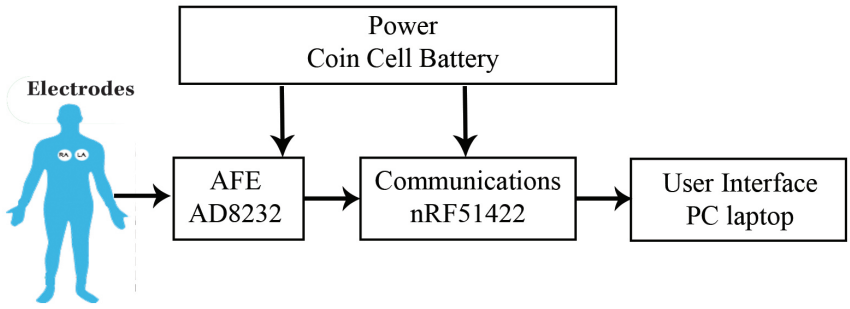


Figure 6.1: Final System Block Diagram

AFE

In order to condition the input, a total gain of 1100 was utilized for the ECG signal. In-amp gain has the fixed gain of 100 and the op-amp was configured for a gain of 11. The AFE block gain is set to 1100 with two-pole high-pass filter of $7Hz$ cutoff frequency and two-pole low-pass filter of $24Hz$ cutoff frequency. The gain and filtering configuration is shown in Figure 6.2

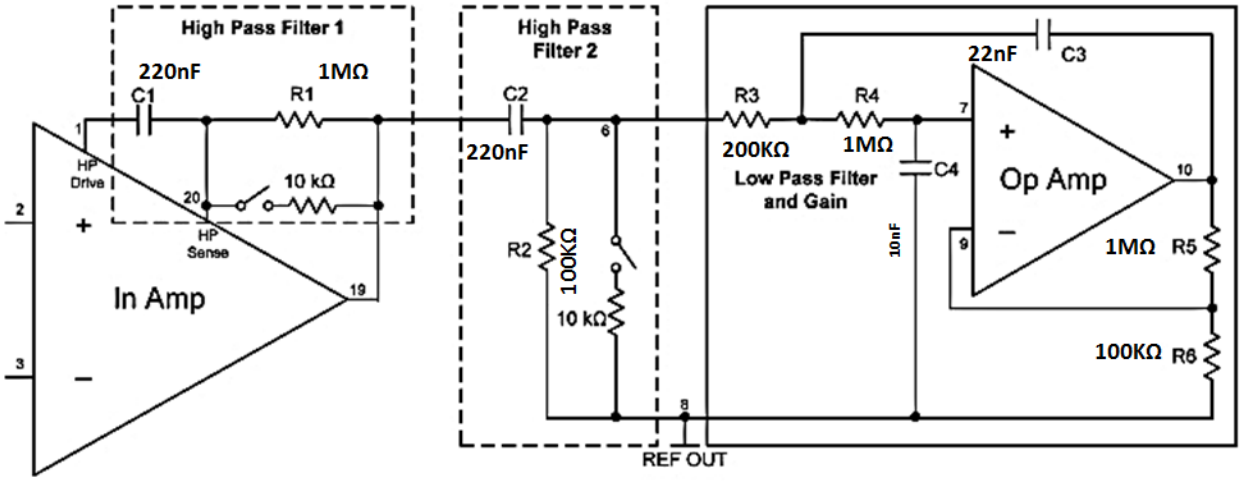


Figure 6.2: AFE Filter Configuration [17]

nRF51422 Hardware

This block includes the nRF51422 multi-protocol SoC, which is used in the BLE protocol mode. Everything in the communications block is contingent on the proper operation of the antenna. The reference design is 50Ω. The nRF51422 includes 30 pins that can be utilized for general purpose input/output. The interface to the AFE block $P0.30$ is used for digital I/O where the SDN_n pin of the AFE block is connected. The SDN_n pin of AFE is driven high (to $VS+$) in order to turn on the ADC. Pin $P0.05$ and $P0.06$ are connected to the $LO+$ and $L0-$ pins of AFE to utilize leads off detection capability.

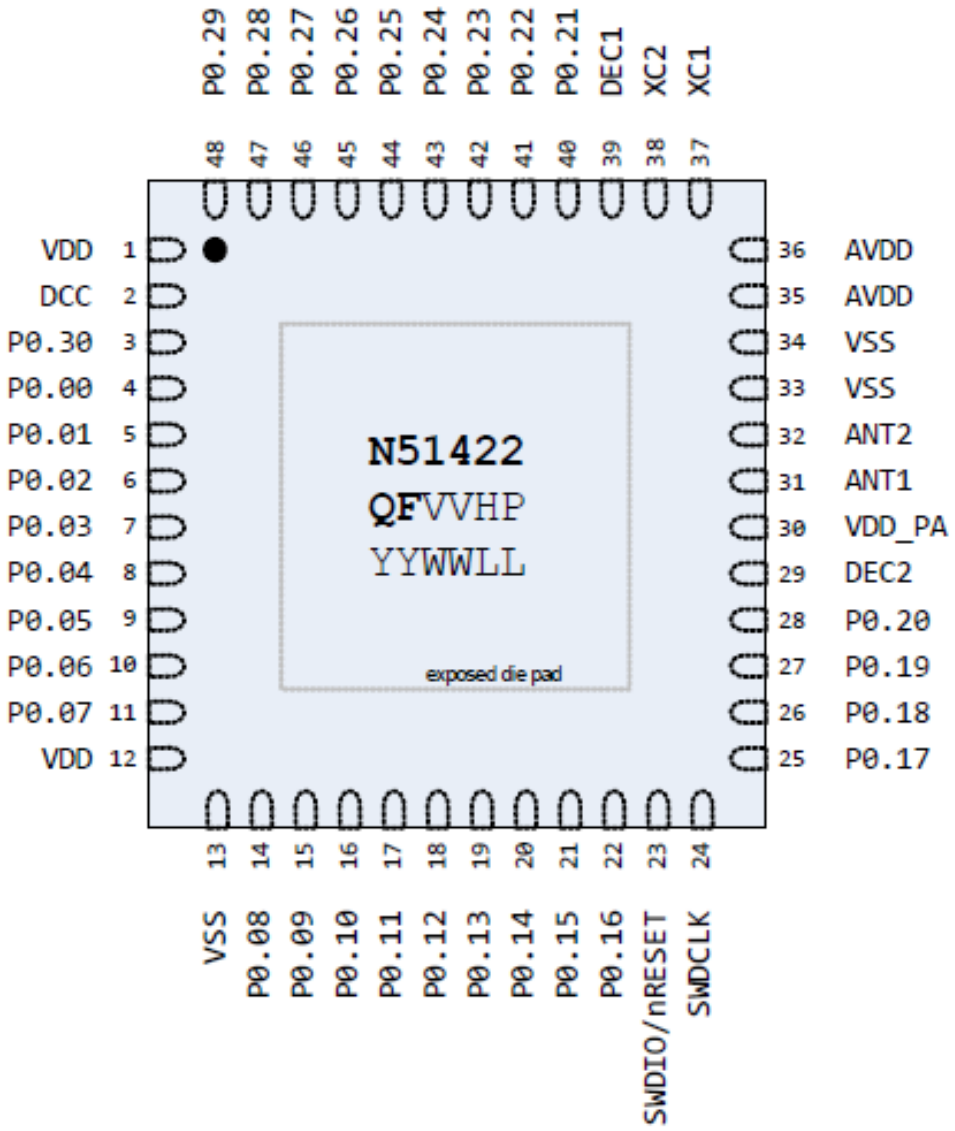


Figure 6.3: nRF51422 Pinout

nRF51422 Software

Figure 6.4 is the software flow chart used to program the nRF51422.

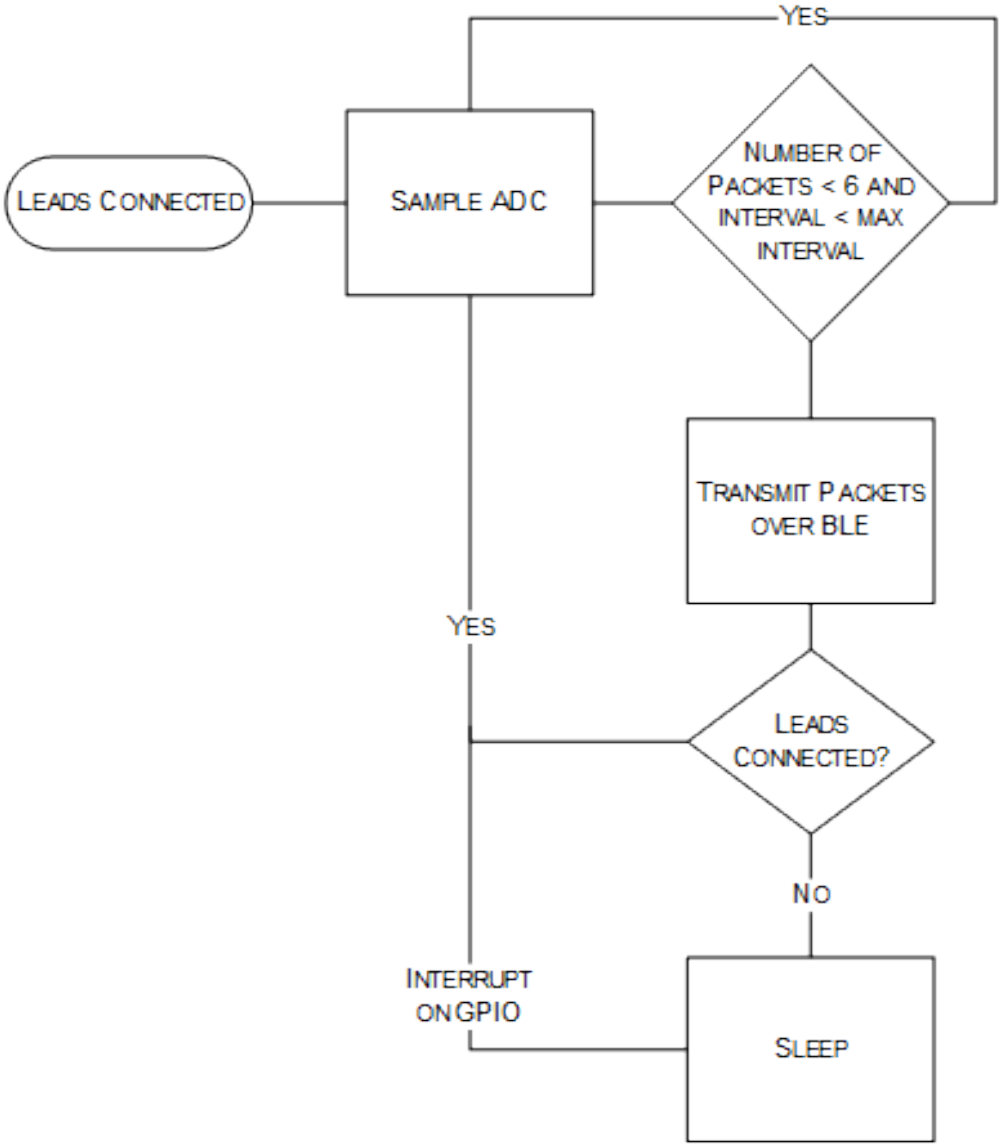


Figure 6.4: nRF51422 Embedded Program Flow Diagram

6.1 First Revision

The first revision of the design is shown in Figure 6.5. This revision of the system was designed for functionality testing purposes and power consumption measurements. The first design of PCB is a 2.580" x 3.678" board with 0805 size passive components for AFE block and 0402 size passive components for communication block. The communication block includes the nRF51422 (the BLE module) with a balun (for impedance matching) and a $\frac{\lambda}{4}$

monopole antenna. The additional test pins (10pin header) in the middle of the board is also intended for testing purposes.

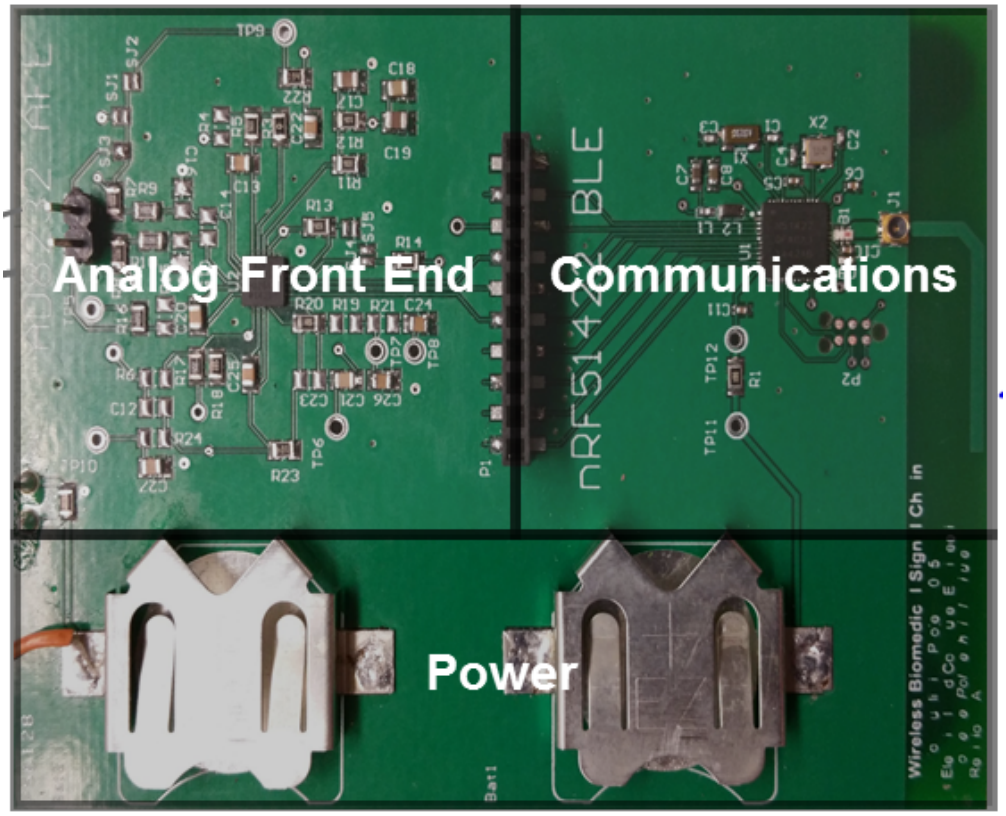


Figure 6.5: System Blocks First Revision

The PCB stack layers shown in Figure 6.6 include two signal plane, one ground plane and one split power plane. The idea behind having a split power plain was to be able to power each of the AFE and communication blocks independent of each other for independent testing and verification without any confounding issues from adjacent modules. Accordingly, as it is shown in Figure 6.5, the board includes two CR2032 battery clips.

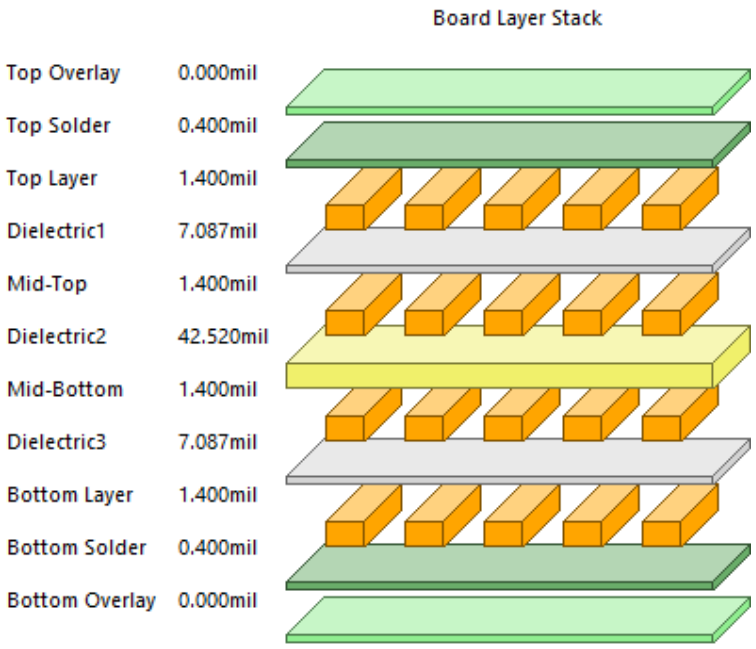


Figure 6.6: Stack Up Layers

Testing the first revision of the board functionality of each AFE and communications block was verified. While testing the end-to-end performance of the system, it was determined that the output of AFE block is not connected to the correct pin on nRF51422, meaning that the ECG signal does not make its path to the ADC of the communication block. As far as the interaction of the blocks is concerned, the system end-to-end performance was only partially verified in the first revision of the PCB board. This made another revision of the board necessary.

6.2 Final Revision

The final revision of the system shown in Figure 6.7 incorporates several modifications with respect to the first revision. See Appendix 10.2 and 10.4 for schematic diagrams of AFE and communications block, respectively. The main purpose of the final revision was to miniaturize the first PCB revision as well as connecting the AFE output pin to the appropriate pin of the communication block. The final revision was also designed to ensure the entire system runs off a single CR1632 coin battery. The dimension for the final PCB board is 2.739" x 1.394". The components are miniaturized (0402 size passives) and their spacing has been reduced. Also, the number of test points were decreased to maintain the size considerations. The test points that are placed at the edge of the final version board are ground and power pins as well as IAOUT, RLD, REFIN and REFOUT pins of the AFE for verifying the block operation. The VEXT test pin is added to allow for option of having an external voltage for

biasing the resistors at the input pins of AD8232. The final revision also includes an on/off slide switch to control powering of the device.

To reduce noise interference from motion artifacts and long wires, the electrode connector pads are provided at the bottom of the final PCB board. Although the design signal chain is tested for two-electrode application, it has the three-electrode measurement capability and can be adapted accordingly. An additional electrode connector pad at the bottom of the board provides the option for attaching the third electrode. The signal conditioning circuitry remained unchanged in the final revision of PCB.

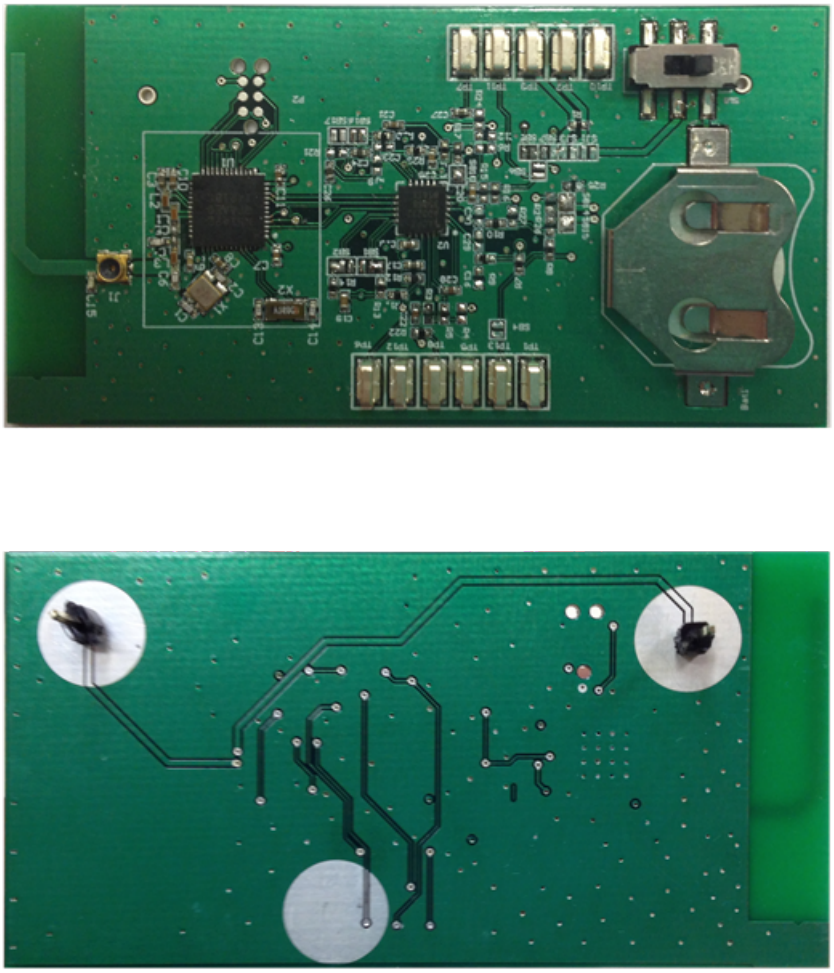


Figure 6.7: Second PCB Board Top View

The final revision PCB is a four-layer board as shown in Figure 6.8 with a single uniform power. The uniform power plane provides the capability to use a single $3V$ supply for the whole system, verify the system functionality, and re-estimate battery life.

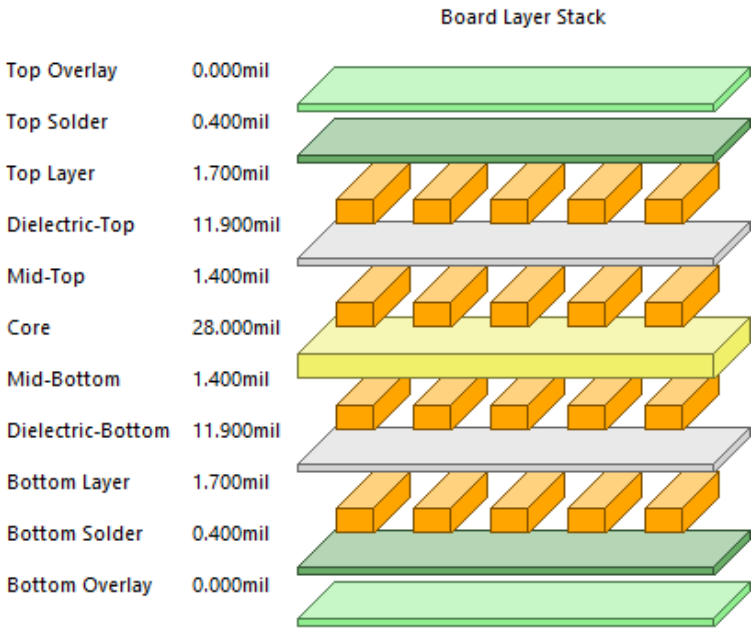


Figure 6.8: Second Revision Stack Up Layers

6.3 Specification Overview

The layout of the final revision is shown in Figure 6.9 showing the component placement and traces. Figure 6.10 shows the bottom layer of the board and the traces on the back of the board. As it is shown in Figures 6.10 and 6.9, a lot of ground vias are placed to avoid the long path for the current flowing on the surface of the board.

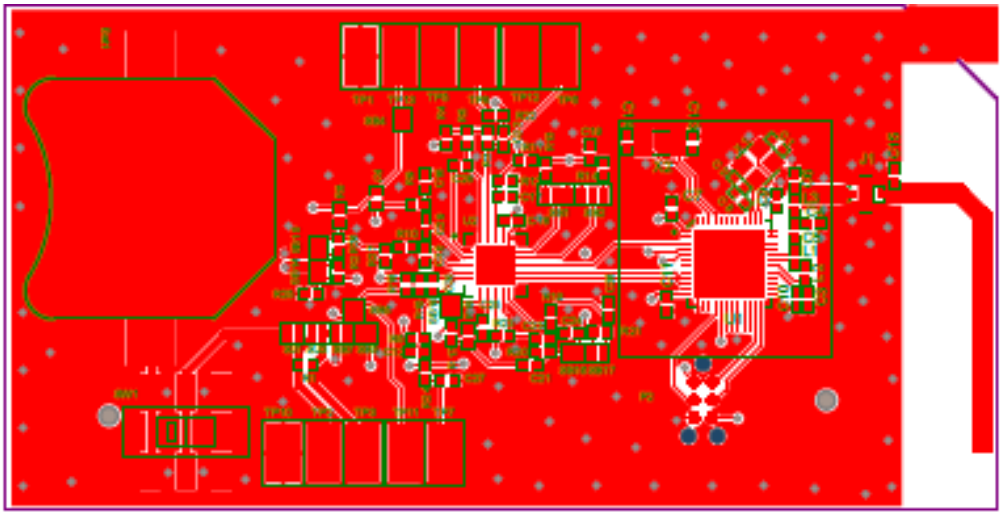


Figure 6.9: Second PCB Board Layout Front View

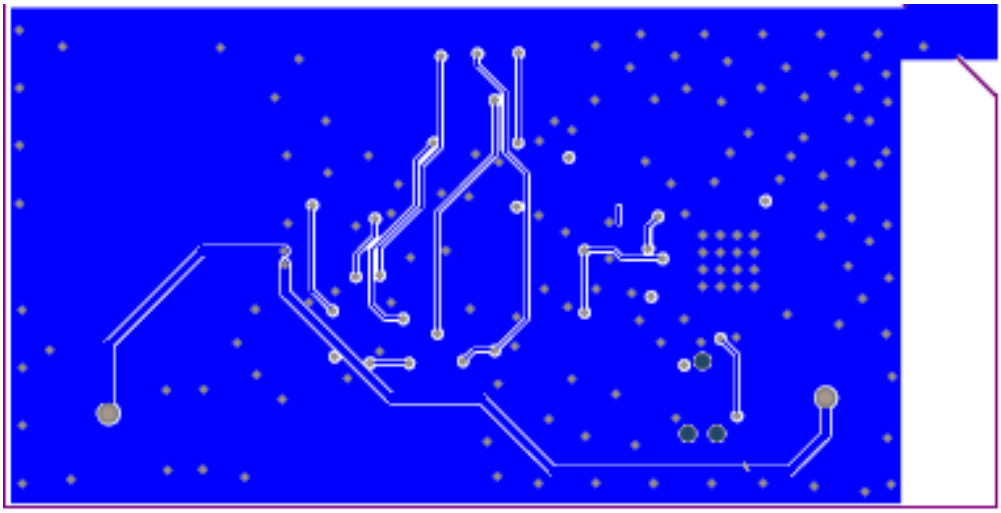


Figure 6.10: Second PCB Board Layout Bottom View

Figure 6.11 shows the block diagram of the final PCB design. It can condition the ECG signal using the two-electrode configuration. The device would be snapped onto a person's chest. The power source was modified to use the battery CR1632 coin cell. This was selected due to size and portability that would best fit for the final board design.

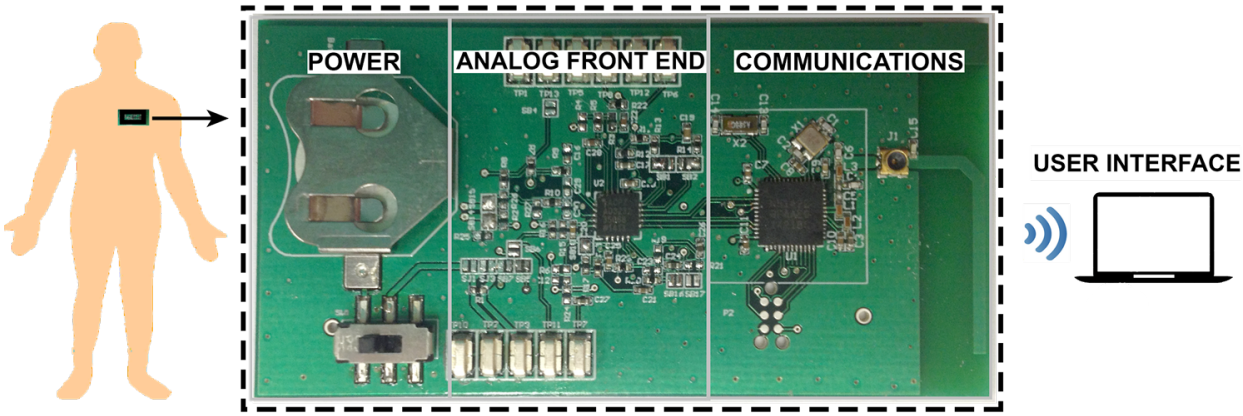


Figure 6.11: System Block Diagram

The CR1632 battery has similar characteristics to CR2032 coin cell except for its dimensions. Table 6.1 shows battery specifications of Energizer CR1632.

Table 6.1: Coin Cell CR1632 Specifications

Characteristic	Value
Nominal Voltage	3 V
Nominal Capacity	130 mAh
Dimension	16.0 mm x 3.2 mm

Based on the data sheet shown in Figure 6.12, the capacity for pulse drain is lower than the CR2032 as discussed in Chapter 4.4. For this project, an estimate of $100mAh$ is used for calculating battery life since the average pulsed-drain peak would be about $10mA$.

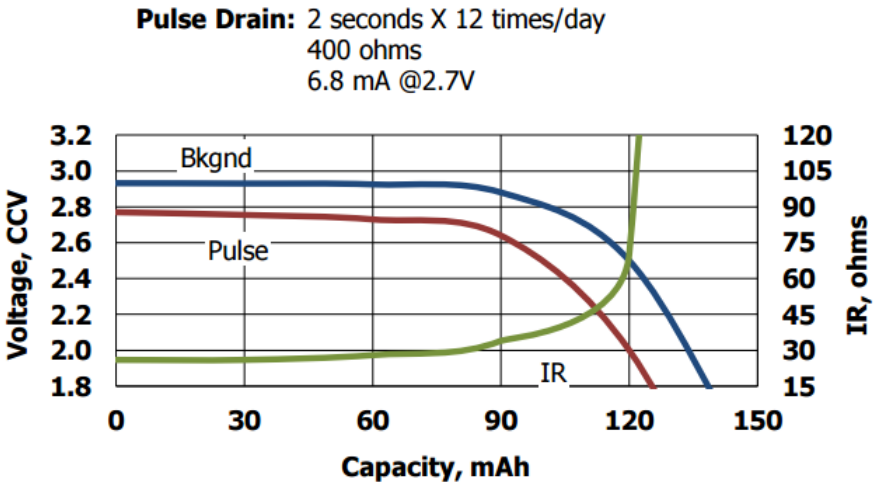


Figure 6.12: CR1632 Coin Cell Pulse Drain Characteristics [18]

7 Design Testing and Results

This chapter discusses the testing procedures performed and results acquired for the design. As discussed in the previous chapter, two revisions were prototyped for this project. The following sections explain more about functionality testing and results of each revision.

7.1 First Revision

Before finalizing the design, the first revision was produced so that the functionality of the system blocks AFE, communications, power, and user interface can be verified as separatly.

AFE

In order to verify the functionality of the AFE block, test setup shown in Figure 7.1 was used.

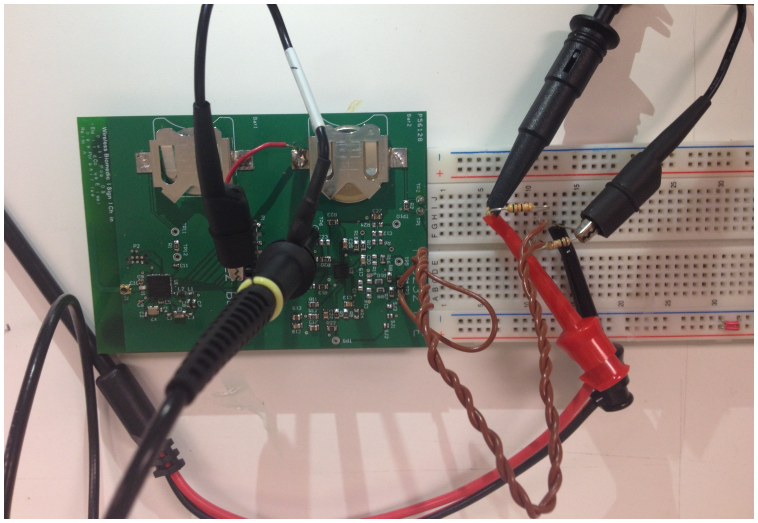


Figure 7.1: AFE Verification Setup

The two input electrodes were connected to test points TP2 and TP3 pins on the board. The cardiac mode of the function generator was used as an input of AFE with approximately $6Hz$ frequency and maximum $100mV$ amplitude. A voltage divider was added to have an input signal of about $1mVpp$. The oscilloscope channel is connected to the TP6 and ground to show the output of AD8232. Figure 7.2 shows AFE verification with two-electrode configuration using a raw ECG signal from function generator. The supply voltage for setup above was a CR2032 battery. The top signal Channel 2 of oscilloscope shows the input voltage to AFE and the bottom signal is the output result which was connected to Channel 1.

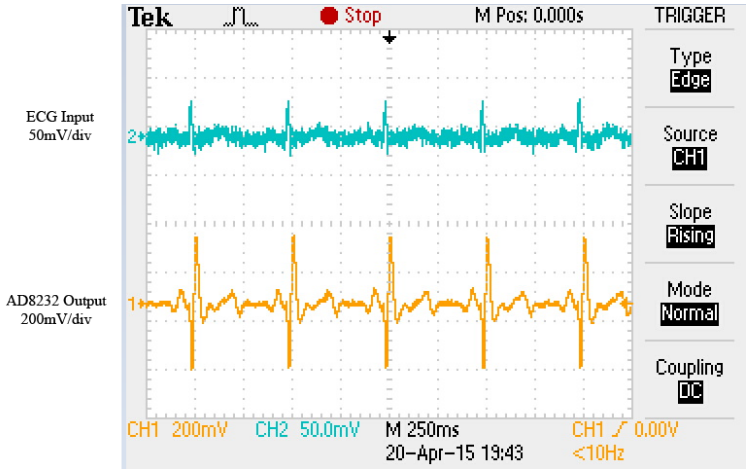


Figure 7.2: First Revision AFE Verification from Function Generator

The next step of testing was acquiring the real human signal. Therefore TP2 and TP3 pins on the board were connected to the ECG monitoring snap electrodes. The electrodes were placed near the heart. Figure 7.3 shows AFE verification using a real human heart signal using the two-electrode configuration.

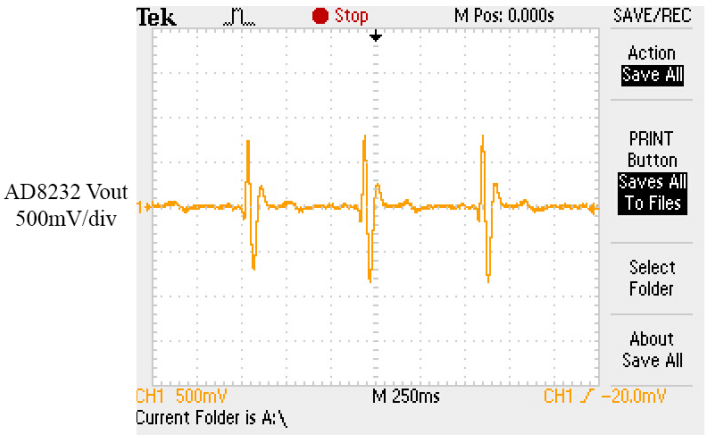


Figure 7.3: First Revision AFE Verification from Real Human Signal

Communications

Several functionality of the communication block was tested. The first step was to program nRF51422 using the debugging interface and make sure the device can be programmed as expected. The next step was to use a Bluetooth dongle to identify the device and make sure the BLE part of the design is advertising.

Using the second revision of the board, a custom firmware shown in the Appendix 10 was loaded to test that the BLE portion in the design works as expected.

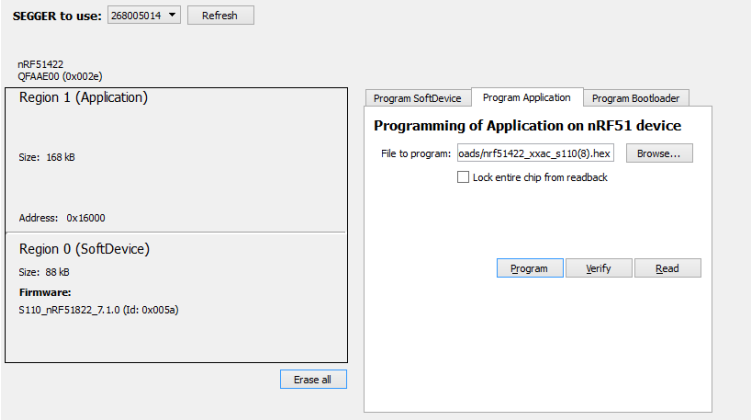


Figure 7.4: Device RAM mapping

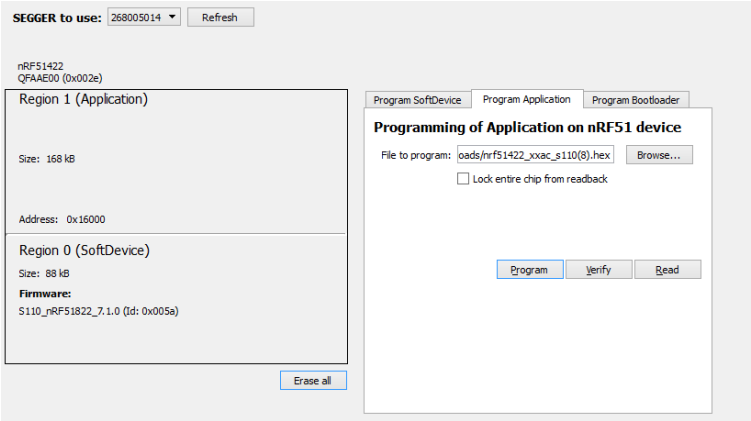


Figure 7.5: Loading Service

Power

For AFE, current consumption was measured across the 10Ω resistor and found to be about $160\mu A$. For the Communications, average energy consumption was measured across the 10Ω resistor for each connection interval. The test setup is shown in Figure 7.6. Finally, the expected battery life using CR1632 with corrected battery capacity was calculated. Parameters used for communications block in transmitting data can be found in Table 7.1.

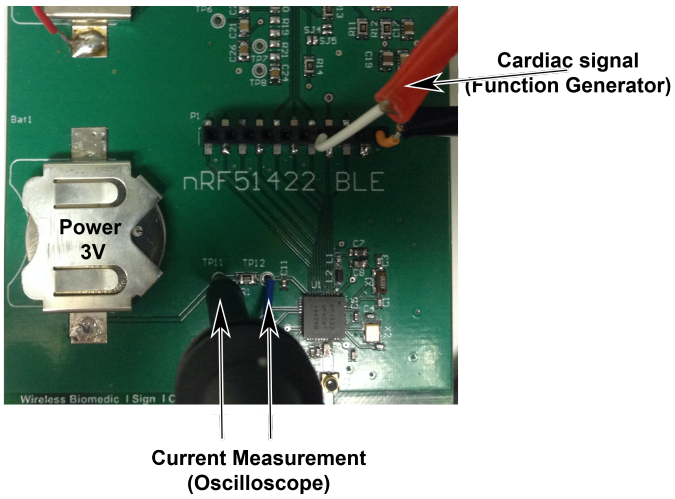


Figure 7.6: Communications Current Measurement Setup

Table 7.1: Current Consumption Measurement for Communications Section

Parameters	Value
Number of Packets	1
Number of Bytes	20
ADC Sampling	16 msec
Min Connection	15 msec
Max Connection	100 msec
Slave Latency	0
Supervisory Timeout	4000 msec

Connection interval was measured to be around $60ms$ as shown in Figure 7.7 where each peak with an average of $8mA$ constitutes a connection event. Otherwise, the device is in idle mode.

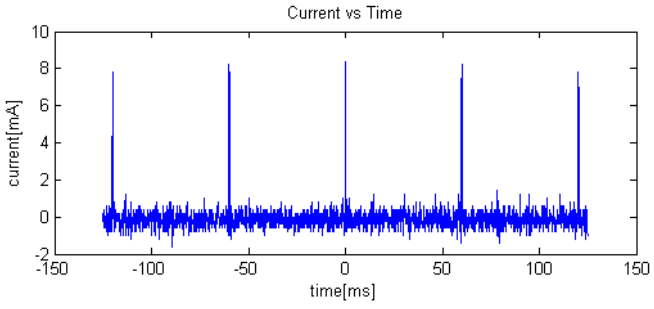


Figure 7.7: Current Consumption during BLE Connection

The total charge per interval was found by calculating the area under the current curve. The average current consumption during a single connection event for the nRF51 is shown in Figure 7.9. Current consumption for advertising event was also measured as measured to be $30.76\mu J$ and $14.94\mu J$ for advertising and connection event, respectively.

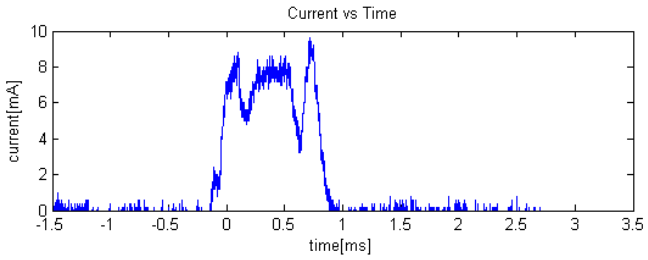


Figure 7.8: First Revision Current Consumption during Connection Event

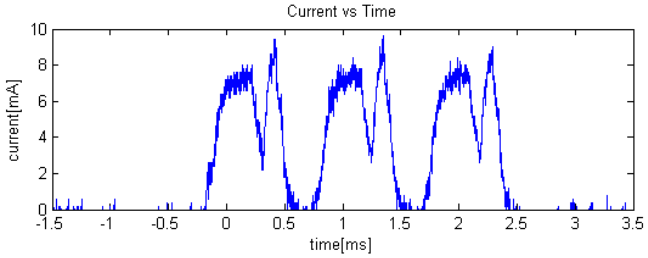


Figure 7.9: First Revision Current Consumption during Advertising Event

Average current per interval can be calculated by the charge divided by connection interval as shown in Equation 7.1.

$$I = \frac{Q}{t} = \frac{\text{charge}}{\text{connection interval}} = \frac{4.98\mu\text{C}}{60\text{ms}} = 0.08 \text{ mA} \tag{7.1}$$

In this case, the battery CR1632 coin cell was used due to size and portability. As shown in Figure 6.12, the capacity for pulse drain is lower than the CR2032 as discussed in Chapter 4. For this project, an estimate of $100mAh$ is used for calculating battery life since pulse drain average peak would be about $10mA$.

The expected battery life was found to be about 400 hours or 2.5 weeks as shown in Equation 7.2.

$$\text{Battery life} = \frac{\text{corrected battery capacity}}{I_{\text{ave (AFE + Communications)}}} = \frac{100 \text{ mAh}}{0.16 \text{ mA} + 0.08 \text{ mA}} = 400 \text{ hours} \tag{7.2}$$

User Interface

Adaptability

This application has not yet been tested on Android 5.x, or Android 2.x and 3.x, only on some versions of Android 4.x due to lack of access to devices running the former OS-es. The app functioned correctly on a Samsung Tab 4 running Android 4.3, a LG-G3 phone running Android 4.4 and a Samsung Galaxy S2 running Android 4.0. The app cannot run on Amazon Fire OS 3.8, which is heavily modified from Android 4.2.2 but this doesn't negate its ability to run on a standard version of 4.2.

Stability

The app crashes when connected with a non-transmitting device after time out or a sudden disconnection while transmitting. This is a common problem for BLE app when the communication lines are not handled delicately. It can be fixed by disabling and re-enabling the BLE services at the moment of disconnection. Cordova, however, does not provide this function in real time. A fix might be possible if the Android Native Platform APIs (see Figure 5.1). However, due to limited time scope, the group proceeded with this instability. On some occasion, the app successfully connects and receives data from transmitting devices with different advertising services than the one the app supposed to only recognize5.2.2. This is likely an error with the UUID service check of the app and will need to be further investigated.

Development issues

To verify the app's functionality, the team used an Adafruit BLE-Friend [54] with a nRF51822 core. The only difference between nRF51822 and nRF51422 (the core of nRF51-DK) is its peripheral supports for ANT. Adafruit BLE-Friend possesses all the BLE service channels available to the nRF51-DK. In addition, it has an UART interface that can be controlled through simple Python code. This UART interface allows quick alternations between different advertising services and simulation of data. The Adafruit was set to advertise using the same service as the manufacturer's HRM firmware for the nRF51-DK and then with an arbitrary unused service and continuously transmitted a random array of data. The app successfully received the simulated data through both of these services (see Figure 7.10 and Figure 7.12.

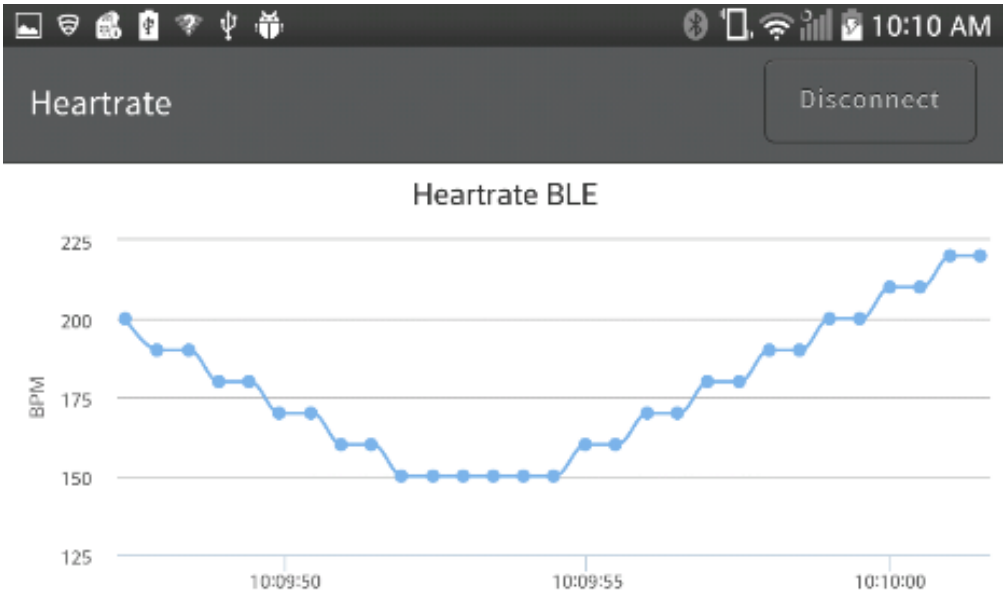


Figure 7.10: Test drive with manufacturer's HRM services

Figure 7.11: Using the manufacturer's Android app

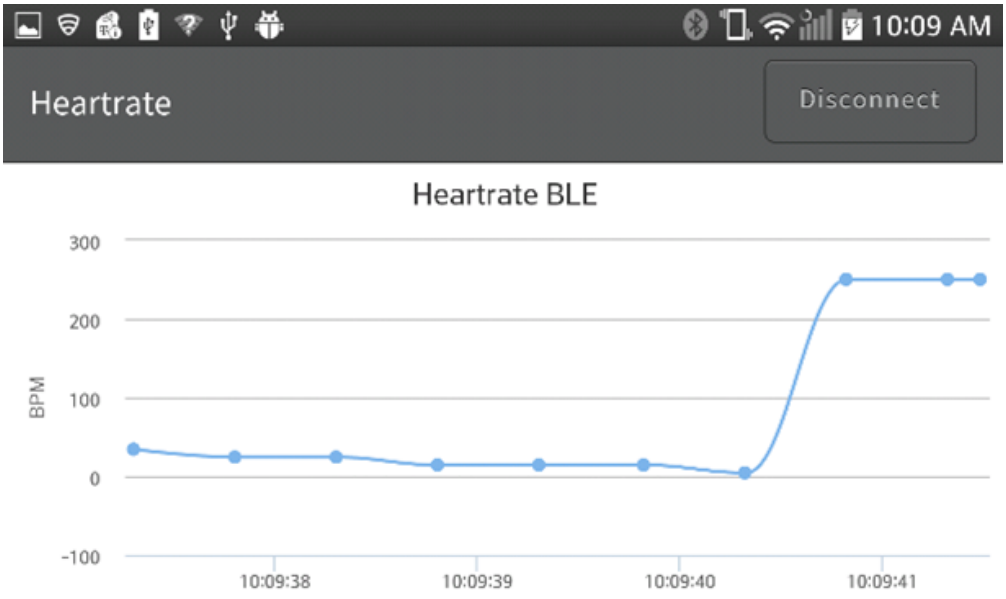


Figure 7.12: Test drive with Adafruit BLE-Friend custom services

The nRF51-DK board, using the same custom service as the Adafruit, can transmit the signal from its ADC to the nRF51-Dongle but cannot send these data to the Android app. To debug this issue, the team set up the nRF51-DK and the Adafruit BLE-Friend with a different custom advertising service and a new set of simulated data were used. This data was obtained using packets the nRF51-DK send to the nRF51-Dongle and thus, emulate the exact outputs of the nRF51-DK's ADC. Both devices transmitted the simulated data successfully to the Android app. This indicates that there might be some issues with how the data from the ADC of the nRF51-DK are sent to the Android app that the app cannot proceed. This issue is likely located in the firmware of the nRF51-DK.

Computer Interface

The Python script is able to receive the data and graph them successfully (see Figure 7.13). One problem remained with the script is its updating speed. Right now, the graphing speed of the script is a bit slow and is not matched with the updating speed of the nRF51 chip. The speed of the Python script can be changed by simply altering a few variables in the script (see Listing 1) but the transmitting speed of the nRF51 chip requires low-level code modification, which given the time constraint of this MQP is not feasible.

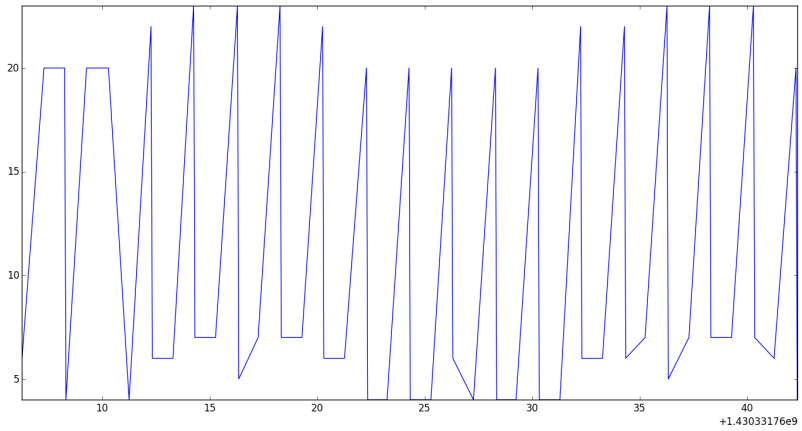


Figure 7.13: The Python Graphing with HRM firmware

7.2 Final Revision

For the final revision of the system, the functionality of the communication block of the system was confirmed. The nRF51422 was successfully programmed using a custom software made for the purpose of this project. The nRF51422 could advertise and establish a link with the Bluetooth dongle with Received Signal Strength Indication (RSSI) of 86dBm over the range of 10m. The communication block was also able to set the SDN pin of the AFE block to high which is required for operation mode of AD8232. However, the functionality of the AFE block was not verified due to time constraints. The user interface block was not modified during this revision.

8 Future Works

At this time, wireless ECG acquisition device that is capable of transmitting data to PC laptop wirelessesly using a Bluetooth USB adaptor. However, due to time constraints, issues were encountered on the final prototype design such as the AFE signal output being susceptible to power hum, better antenna and integration to a smartphone device. With this, future recommendations and enhancement of the product are presented which includes additional filtering, antenna impedance matching, code optimization, isolated grounds, power management, user interface, and design modifications. These are discussed in more detail in the following sub sections.

Additional Filtering

One issue encountered is that the ECG signal output is susceptible to power hum. Sometimes, the signal gets interference especially from power line. The current prototype is designed to have a band-pass filter from $0.2Hz$ to $200Hz$ which includes the AC noise of around 60Hz. One way to get rid of noise issue is to add a notch filter or band-stop filter with a narrow band specifically at the 60Hz frequency. More filters could also be added to remove harmonics that are usually caused by non-linear loads.

Antennta Impedance Matching

Although the final design is sufficiently able to perform the basic requirements, there are a few improvements that can be addressed for the reliability of the antenna.

$$f_0 = \frac{1}{2\pi\sqrt{LC}} \tag{8.1}$$

At frequencies above the self-resonance frequency, the capacitor behaves like an inductor and the inductor like a lumped capacitor with an inductor and resistor. We did not consider this parasitic effect or significantly look into modifying the antenna impedance in order to match the impedance of the antenna at 50Ω. Due to the difference in the manufactured board and the reference layout we neglected the dielectric change of the materials.

Code Optimization

For this project,any encryption or any extensive analysis on the timing of the BLE stack were not considered. Better analysis of the timing could be done for future works such that the interrupt generated due to an application timer, like the one utilized in this project for the period of the ADC sampling, would occur at a higher priority than other application and non-essential timers.

Isolated Grounds

In a future project, one could isolate the ground for the nRF51422 as digital ground and then have a separate analog ground for the AD8232, but instead in this project a single ground plane was utilized. This could potentially be the cause of some of the issues of noise due to the high frequency oscillations of the nRF51422 and current returning through the same ground as the AD8232.

Power Management

Another feature that could be added is a power management circuitry. This would regulate voltage from the battery so that it has a constant DC voltage supply and protect device from current surge. Since the goal of the project is to eventually have the device snap onto a person's chest, it is best to make sure that no surges or peaks happen that could stop a person's heart. Moreover, even though the device was expected to last for about 2 weeks, we recommend for future improvement to maximize battery life such as adding parallel capacitor with a size appropriate for the current consumption of device. Future work also include on using other types of power source such as some of the energy harvesting techniques mentioned earlier in Chapter 4.1, background research.

User Interface Integration

Due to limited time and experience developing a smartphone app, the group ended up on creating a program that could be used for PC laptops to display data results. However, this limits to users who have access to a Bluetooth adapter since it requires a BLE compatible device. In the future, we recommend developing an android phone based application that users could easily download from the App Store and easily integrate the device since current mobile devices have a built-in platform which supports Bluetooth Low Energy.

Design Modifications

If the project would be considered for future use in the medical field, future works on the overall design are recommended. This would include modifying the physical feature of the device such as including some sort of enclosure to be aesthetically pleasing and prevent damage to the components from contact to other materials. Moreover, we could add some led lights that would indicate the device status such being in use or not. This affect would then be considered to the power consumption of the whole system.

9 Conclusion

The purpose of this project was to design a low power and portable system with four primary modules such as analog front end (AFE), communications, power and user interface. The goals of this project, as demonstrated in the preceding sections, were largely met and the design provides an excellent booster for anyone looking to investigate wireless bio-potential transmission. The ECG data is then transmitted and displayed on a smartphone device for user-friendly interface. During the initial stage of the project, major issues were encountered particularly on the chosen communications integrated circuit for data processing since available resources were limited. In the end, the group decided on using the nRF51422. Despite the filters configured for the AFE, more issues on noise signal particularly from the 60Hz power line hum were encountered. For power, the use of a coin cell battery was chosen due to its portability, simplicity, and size consideration. The run-time was also estimated to be sufficiently long that it limits battery replacement to approximately two weeks.

References

[1] R. MacLeod and B. Birchler, "ECG Measurement and Analysis," Mar. 2011. [Online]. Available: http://www.sci.utah.edu/~macleod/bioen/be6000/labnotes/ecg/descrip.html

[2] Heart Risk Warner, "Meassure ECG, arrythmia detection," 2015. [Online]. Available: http://www.hrwproject.com/ecg.html

[3] C. Kitchin and L. Counts, *A designer's guide to instrumentation amplifiers*, 3rd ed. Analog Devices, 2006. [Online]. Available: http://www.analog.com/media/en/training-seminars/design-handbooks/58127566743127787 37Complete_In_Amp.pdf

[4] Analog Devices, "AD8236 40 uA Micropower Instrumentation Amplifier with Zero Crossover Distortion," 2009. [Online]. Available: http://www.analog.com/media/en/technical-documentation/data-sheets/AD8236.pdf

[5] A. Devices, "ADAS1000," 2013. [Online]. Available: http://www.analog.com/en/products/analog-to-digital-converters/ad-converters/adas1000.html

[6] Analog Devices, "AD8232: Single-Lead , Heart Rate Monitor Front End," 2012. [Online]. Available: http://www.analog.com/media/en/technical-documentation/data-sheets/AD8232.pdf

[7] ——, "Evaluating the AD8232 Single-Lead Heart Rate Monitor Front End," 2014. [Online]. Available: http://www.analog.com/media/en/technical-documentation/user-guides/AD8232-EVALZ_UG-514.pdf

[8] Texas Instruments, "CC2541 | Bluetooth / Bluetooth Low Energy | Wireless Connectivity | Description & parametrics." [Online]. Available: http://www.ti.com/product/cc2541

[9] ——, "CC2540F128, 2.4-GHz Bluetooth® low energy System-on-Chip," 2013. [Online]. Available: http://www.ti.com/lit/ds/symlink/cc2540.pdf

[10] Nordic Semiconductor, "nRF51422 Product Specification v3.1," 2014.

[11] ——, "S110 SoftDevice Specification v2.0," 2014.

[12] C. Simpson, "Texas Instruments: Characteristics of Rechargeable Batteries," 2011. [Online]. Available: http://www.ti.com/lit/an/snva533/snva533.pdf

[13] M. Dimitrijevic, "Lithium ion battery assembly challenges," 2012. [Online]. Available: http://www.ecnmag.com/articles/2011/01/lithium-ion-battery-assembly-challenges

[14] J. Kamath, Sandeep; Lindh, "SWRA347a: Application Note AN092 Measuring Bluetooth Low Energy Power Consumption," 2012. [Online]. Available: http://www.ti.com.cn/cn/lit/an/swra347a/swra347a.pdf

[15] Nordic Semiconductor, "nRF51 Development Kit: Developing with the MDK-ARM Microcontroller Development Kit User Guide," Oct. 2014. [Online]. Available: http://www.nordicsemi.com

[16] K. Furset and P. Hoffman, "High pulse drain impact on CR2032 coin cell battery capacity." [Online]. Available: http://cms.edn.com/contenteetimes/documents/schweber/c0924/c0924post.pdf

[17] J.-H. Broeders, "Predicting and Finding Your Limits!" 2012. [Online]. Available: http://www.analog.com/media/en/technical-documentation/technical-articles/Predicting-and-Finding-Your-Limits-MS-2385.pdf

[18] Energizer, "ENERGIZER CR1632 Product Sheet." [Online]. Available: http://data.energizer.com/PDFs/cr1632.pdf

[19] T. Kazmierski and S. Beeby, *Energy Harvesting Systems*. Springer, 2011.

[20] Energizer, "ENERGIZER CR2032 Product Sheet." [Online]. Available: http://data.energizer.com/PDFs/cr2032.pdf

[21] Analog Devices, "Capacitive Application," 2014.

[22] D. Mozaffarian, E. J. Benjamin, A. S. Go, D. K. Arnett, M. J. Blaha, M. Cushman, S. de Ferranti, J.-P. Després, H. J. Fullerton, V. J. Howard *et al.*, "Executive summary: Heart disease and stroke statistics—2015 update a report from the american heart association," *Circulation*, vol. 131, no. 4, pp. 434–441, 2015. [Online]. Available: http://circ.ahajournals.org/content/131/4/434

[23] J. Cox, F. Nolle, H. Fozzard, and G. Oliver, "Aztec, a preprocessing program for real-time ecg rhythm analysis," *Biomedical Engineering, IEEE Transactions on*, no. 2, pp. 128–129, 1968.

[24] J. Kruse and C. Redmond, "Detecting and Distinguishing Cardiac Pacing Artifacts," *Analog Dialogue*, vol. 46, no. 11, pp. 1–6, Nov. 2012. [Online]. Available: http://www.analog.com/library/analogDialogue/archives/46-11/pacing_artifacts.html

[25] R. Heydon, *Bluetooth low energy*. Prentice Hall, 2013.

[26] W. Bracke, P. Merken, R. Puers, S. Member, and C. V. Hoof, "Ultra-Low-Power Interface Chip for Autonomous Capacitive Sensor Systems," vol. 54, no. 1, pp. 130–140, 2007.

[27] C. Johnson, "Ultracapacitor Breakthrough May Recharge Energy Storage," Apr. 2014. [Online]. Available: http://www.eetimes.com/document.asp?doc_id=1322050

[28] Analog Devices, "ADP5061: Linear Battery Charger with Power Path and USB Mode Compatibility Data Sheet," 2013.

[29] Bluetooth SIG, "Bluetooth Core v4.0," Bluetooth SIG, Tech. Rep., 2013.

[30] J. Tyzzer, "Extending battery life in ultra low power wireless applications." [Online]. Available: http://www.low-powerdesign.com/121312-article-extending-battery-life.htm

[31] "Introduction to Android | Android Developers." [Online]. Available: http://developer.android.com/guide/index.html

[32] Google, "Android Developers," 2014. [Online]. Available: http://developer.android.com/index.html

[33] N. Gibson, "XCode on Windows: How to Develop for Mac or iOS on a PC," 2014. [Online]. Available: https://www.udemy.com/blog/xcode-on-windows/

[34] Apple Inc., "iOS Dev Center - Apple Developer," 2011. [Online]. Available: http://developer.apple.com/devcenter/ios/index.action

[35] Apple, "iOS Dev Center - Apple Developer," 2014. [Online]. Available: https://developer.apple.com/devcenter/ios/index.action

[36] ——, *Bluetooth Accessory Design Guidelines for Apple Products*. Cupertino, CA: Apple Inc., 2013. [Online]. Available: https://developer.apple.com/hardwaredrivers/BluetoothDesignGuidelines.pdf

[37] R. Shooter, "Simple iPhone app development tutorial | ricochetshooter on WordPress.com," 2012. [Online]. Available: http://ricochetshooter.wordpress.com/2012/02/12/simple-iphone-app-development-tutorial/

[38] Microsoft, "Develop universal Windows apps." [Online]. Available: http://dev.windows.com/en-us/develop

[39] AppyPie, "App Maker Appy Pie." [Online]. Available: http://www.appypie.com/

[40] IBuildApp, "iBuildApp - Create Android and iPhone App, Free, No Coding Required." [Online]. Available: http://ibuildapp.com/

[41] Conduit, "Como App Maker," 2014. [Online]. Available: http://www.como.com/

[42] S. Angeles, "Best App Maker 2014 - App Builders and Creators - BusinessNewsDaily," 2014. [Online]. Available: http://www.businessnewsdaily.com/4901-best-app-makers-creators.html

[43] J. Wallen, "Create a mobile app for your company with Conduit Mobile - TechRepublic," 2013. [Online]. Available: http://www.techrepublic.com/blog/smartphones/create-a-mobile-app-for-your-company-with-conduit-mobile/

[44] J. G. M. Bello, "Electromagnetic Detector:EMF Scanner on the App Store on iTunes," 2014. [Online]. Available: https://itunes.apple.com/us/app/electromagnetic-detector-emf/id836603095?mt=8

[45] SuperPhunLabs, "EMF Meter - Android Apps on Google Play," 2012. [Online]. Available: https://play.google.com/store/apps/details?id=com.superphunlabs.emf&hl=en

[46] Runtastic, "Runtastic Receiver and Heart Rate Monitor," 2014. [Online]. Available: https://www.runtastic.com/shop/usa/runtastic-receiver-heart-rate-monitor

[47] "FreeRTOS - Market leading RTOS (Real Time Operating System) for embedded systems with Internet of Things extensions." [Online]. Available: http://www.freertos.org/

[48] D. Rincon, Fransisco; Recas, Joaquin; Khaled, Nadia; Atienza, "Development and Evaluation of Multilead Wavelet-Based ECG Delineation Algorithms for Embedded Wireless Sensor Nodes," *IEEE Transactions of Information Technology in Biomedicine*, vol. 15, no. No. 6, pp. 854–863, 2011. [Online]. Available: http://infoscience.epfl.ch/record/167826/files/TITB2011-59977029.pdf

[49] Apache Cordova, "Apache Cordova," 2012. [Online]. Available: http://cordova.apache.org/

[50] "Apache Cordova API Supported Platforms." [Online]. Available: https://cordova.apache.org/docs/en/4.0.0/guide_support_index.md.html

[51] Nordic Semiconductor, "S120 SoftDevice Specification v2.1," pp. 1–63, 2014.

[52] "matplotlib: python plotting — Matplotlib 1.4.3 documentation." [Online]. Available: http://matplotlib.org/

[53] "Anaconda Scientific Python Distribution." [Online]. Available: https://store.continuum.io/cshop/anaconda/

[54] "Bluefruit LE Friend - Bluetooth Low Energy (BLE 4.0) - nRF51822 [v1.0] ID: 2267 - $24.95 : Adafruit Industries, Unique & fun DIY electronics and kits." [Online]. Available: https://www.adafruit.com/products/2267

10 Appendices

Listings


```
# Copyright (c) 2015 Nordic Semiconductor. All Rights Reserved.
#
# The information contained herein is property of Nordic Semiconductor ASA.
# Terms and conditions of usage are described in detail in NORDIC
# SEMICONDUCTOR STANDARD SOFTWARE LICENSE AGREEMENT.
# Licensees are granted free, non-transferable use of the information. NO
# WARRANTY of ANY KIND is provided. This heading must NOT be removed from
# the file.

"""
Example on use of s120_nrf51_ble_driver python binding library.

The example shows how to initialize the library and configure the nRF51 as
heart rate collector.
"""
import numpy as np

# Enthought imports
import matplotlib.pyplot as plt
import matplotlib.animation as animation
import time

# Add location of binding library to path
#import time
# Import the stuff for the graphing
import sys
sys.path.append("../..")
import platform
import traceback
if platform.system() == "Windows":
    import ctypes
    ref = ctypes.cdll.LoadLibrary('../../../../s120_nrf51_ble_driver')
else:
    raise Exception("Only Windows platform is supported")
import s120_nrf51_ble_driver as ble_driver
import ble_driver_util as util
# Import the binding library

import Tkinter
import threading
import matplotlib
import matplotlib.backends.backend_tkagg

class Plotter():
    def __init__(self,fig):
        self.root = Tkinter.Tk()
        self.root.state("zoomed")

```

```
        self.fig = fig
        t = threading.Thread(target=self.PlottingThread,args=(fig,))
        t.start()

    def PlottingThread(self,fig):
        canvas = matplotlib.backends.backend_tkagg.FigureCanvasTkAgg(fig, master=self.root)
        canvas.show()
        canvas.get_tk_widget().pack(side=Tkinter.TOP, fill=Tkinter.BOTH, expand=1)

        toolbar = matplotlib.backends.backend_tkagg.NavigationToolbar2TkAgg(canvas,
    self.root)
        toolbar.update()
        canvas._tkcanvas.pack(side=Tkinter.TOP, fill=Tkinter.BOTH, expand=1)

        self.root.mainloop()


class ECG():
    SERIAL_PORT = "COM5"
    TARGET_DEV_NAME = "ECG"
    MAX_PEER_COUNT = 1
    ECG_SERVICE_UUID = 0xFF02
    ECG_MEAS_CHAR_UUID = 0xFF05
    CCCD_UUID = 0x2902
    CCCD_NOTIFY = 0x01
    BLE_ADDRESS_LENGTH = 6
    connection_params = None
    scan_params = None
    connected_devices = 0
    connection_handle = 0
    service_start_handle = 0
    service_end_handle = 0
    ecg_char_handle = 0
    ecg_cccd_handle = 0
    connection_is_in_progress = False
    error_code = 0
    new_val = []
    time = []
    fig = 0

    def log_message_handler(self, severity, log_message):
        unused = severity
        try:
            print "[LOG MESSAGE]: {}".format(log_message)
        except Exception, ex:
            print "Exception: {}".format(str(ex))


    def ble_evt_handler(self, ble_event):
        try:
            if ble_event is None:
                print "Received empty ble_event"
                return

            evt_id = ble_event.header.evt_id

            if evt_id == ble_driver.BLE_GAP_EVT_CONNECTED:
```

```
105                 self.on_connected(ble_event.evt.gap_evt)
106
107             elif evt_id == ble_driver.BLE_GAP_EVT_DISCONNECTED:
108                 self.on_disconnected(ble_event.evt.gap_evt)
109
110             elif evt_id == ble_driver.BLE_GAP_EVT_ADV_REPORT:
111                 self.on_adv_report(ble_event.evt.gap_evt)
112
113             elif evt_id == ble_driver.BLE_GAP_EVT_TIMEOUT:
114                 self.on_timeout(ble_event.evt.gap_evt)
115
116             elif evt_id == ble_driver.BLE_GATTC_EVT_PRIM_SRVC_DISC_RSP:
117                 self.on_service_discovery_response(ble_event.evt.gattc_evt)
118
119             elif evt_id == ble_driver.BLE_GATTC_EVT_CHAR_DISC_RSP:
120                 self.on_characteristic_discovery_response(ble_event.evt.gattc_evt)
121
122             elif evt_id == ble_driver.BLE_GATTC_EVT_DESC_DISC_RSP:
123                 self.on_descriptor_discovery_response(ble_event.evt.gattc_evt)
124
125             elif evt_id == ble_driver.BLE_GATTC_EVT_WRITE_RSP:
126                 self.on_write_response(ble_event.evt.gattc_evt)
127
128             elif evt_id == ble_driver.BLE_GATTC_EVT_HVX:
129                 self.on_hvx(ble_event.evt.gattc_evt)
130
131             else:
132                 print "Received event with ID: {}".format(evt_id)
133
134         except Exception, ex:
135             print "Exception: {}".format(str(ex))
136             print traceback.extract_tb(sys.exc_info()[2])
137
138
139     def on_connected(self, gap_event):
140         print "Connection established"
141         self.connected_devices += 1
142         self.connection_handle = gap_event.conn_handle
143         self.connection_is_in_progress = False
144         self.start_service_discovery()
145
146
147     def on_disconnected(self, gap_event):
148         print "Disconnected, reason: 0x{0:02X}".format(gap_event.params.disconnected.reason)
149         self.connected_devices -= 1
150         self.connection_handle = 0
151
152
153     def on_adv_report(self, gap_event):
154         address_pointer = gap_event.params.adv_report.peer_addr.addr
155         address_list = util.uint8_array_to_list(address_pointer, self.BLE_ADDRESS_LENGTH)
156         address_string = "".join("{0:02X}".format(byte) for byte in address_list)
157         adv_data_pointer = gap_event.params.adv_report.data
158         adv_data_length = gap_event.params.adv_report.dlen
159         adv_data_list = util.uint8_array_to_list(adv_data_pointer, adv_data_length)
160
161         parsed_data = self.parse_adv_report(ble_driver.BLE_GAP_AD_TYPE_COMPLETE_LOCAL_NAME,
```

```
                                        adv_data_list)

        if not parsed_data:
            parsed_data = self.parse_adv_report(ble_driver.BLE_GAP_AD_TYPE_SHORT_LOCAL_NAME,
                                                adv_data_list)

        if not parsed_data:
            return

        peer_device_name = "".join(chr(element) for element in parsed_data)

        print "Received advertisment report, address: 0x{}, device_name: {}".format(
            address_string, peer_device_name)

        if peer_device_name != self.TARGET_DEV_NAME:
            return

        if self.connected_devices >= self.MAX_PEER_COUNT:
            return

        if self.connection_is_in_progress:
            return

        self.err_code = ble_driver.sd_ble_gap_connect(gap_event.params.adv_report.peer_addr,
                                                      self.scan_params,
                                                      self.connection_params)

        if self.err_code != ble_driver.NRF_SUCCESS:
            print "Connection request failed, reason {}".format(self.err_code)

        self.connection_is_in_progress = True


    def on_timeout(self, gap_event):
        source = gap_event.params.timeout.src
        if source == ble_driver.BLE_GAP_TIMEOUT_SRC_CONN:
            self.connection_is_in_progress = False
        elif source == ble_driver.BLE_GAP_TIMEOUT_SRC_SCAN:
            self.start_scan()


    def parse_adv_report(self, adv_type, adv_data):
        length = len(adv_data)
        index = 0
        while index < length:
            field_length = adv_data[index]
            field_type = adv_data[index + 1]
            if field_type == adv_type:
                offset = index + 2
                parsed_data = adv_data[offset: offset + field_length - 1]
                return parsed_data
            index += (field_length + 1)
        return None


    def on_service_discovery_response(self, gattc_event):
        if gattc_event.gatt_status != ble_driver.NRF_SUCCESS:
```

```
219             print "Error. Service discovery failed. Error code
    0x{0:X}".format(gattc_event.gatt_status)
220             return
221         count = gattc_event.params.prim_srvc_disc_rsp.count
222         if count == 0:
223             print "Error. Service not found"
224             return
225         print "Received service discovery response"
226 
227         service_list =
    util.service_array_to_list(gattc_event.params.prim_srvc_disc_rsp.services, count)
228         service_index = 0  # We requested to discover Heart Rate service only, so selecting
    first result
229         service = service_list[service_index]
230 
231         self.service_start_handle = service.handle_range.start_handle
232         self.service_end_handle = service.handle_range.end_handle
233 
234         print "UUID: 0x{0:04X}, start handle: 0x{1:04X}, end handle: 0x{2:04X}".format(
235             service.uuid.uuid, self.service_start_handle, self.service_end_handle)
236 
237         self.start_characteristic_discovery()
238 
239 
240     def on_characteristic_discovery_response(self, gattc_event):
241         if gattc_event.gatt_status != ble_driver.NRF_SUCCESS:
242             print "Error. Characteristic discovery failed. Error code 0x{0:X}".format(
243                 gattc_event.gatt_status)
244             return
245 
246         count = gattc_event.params.char_disc_rsp.count
247 
248         print "Received characteristic discovery response, characteristics count:
    {}".format(count)
249 
250         char_list = util.char_array_to_list(gattc_event.params.char_disc_rsp.chars, count)
251 
252         for i in range(0, count):
253             characteristic = char_list[i]
254 
255             print "Handle: 0x{0:04X}, UUID: 0x{1:04X}".format(characteristic.handle_decl,
256                                                               characteristic.uuid.uuid)
257 
258             if characteristic.uuid.uuid == self.ECG_MEAS_CHAR_UUID:
259                 self.ecg_char_handle = characteristic.handle_decl
260 
261         self.start_descriptor_discovery()
262 
263 
264     def on_descriptor_discovery_response(self, gattc_event):
265         if gattc_event.gatt_status != ble_driver.NRF_SUCCESS:
266             print "Error. Descriptor discovery failed. Error code 0x{0:X}".format(
267                 gattc_event.gatt_status)
268             return
269 
270         count = gattc_event.params.desc_disc_rsp.count
271 
```

```
272         print "Received descriptor discovery response, descriptor count: {}".format(count)
273 
274         desc_list = util.desc_array_to_list(gattc_event.params.desc_disc_rsp.descs, count)
275         for i in range(0, count):
276             descriptor = desc_list[i]
277             print "Handle: 0x{0:04X}, UUID: 0x{1:04X}".format(descriptor.handle,
    descriptor.uuid.uuid)
278 
279             if descriptor.uuid.uuid == self.CCCD_UUID:
280                 self.ecg_cccd_handle = descriptor.handle
281 
282         print "Press enter to toggle notifications"
283 
284 
285     def on_write_response(self, gattc_event):
286         print "Received write response"
287         if gattc_event.gatt_status != ble_driver.NRF_SUCCESS:
288             print "Error. Write operation failed. Error code
    0x{0:X}".format(gattc_event.gatt_status)
289             return
290 
291 
292     def on_hvx(self, gattc_event):
293         if gattc_event.gatt_status != ble_driver.NRF_SUCCESS:
294             print "Error. Handle value notification failed. Error code 0x{0:X}".format(
295                 gattc_event.gatt_status)
296             return
297 
298         length = gattc_event.params.hvx.len
299         data_array = gattc_event.params.hvx.data
300         data_list = util.uint8_array_to_list(data_array, length)
301 
302         data_list_string = ", ".join("{0:02X}".format(el) for el in data_list)
303         self.time.append(time.time())
304         self.new_val.append(int(data_list[0]))
305 
306         #stuff here
307         print "Received handle value notification, handle: 0x{0:04X}, value: 0x{1}".format(
308             gattc_event.params.hvx.handle, data_list_string)
309 
310 
311     def start_scan(self):
312         self.error_code = ble_driver.sd_ble_gap_scan_start(self.scan_params)
313 
314         if self.error_code != ble_driver.NRF_SUCCESS:
315             print "Scan start failed"
316             return
317 
318         print "Scan started"
319 
320 
321     def start_service_discovery(self):
322         print "Discovering primary services"
323         start_handle = 0x0001
324 
325         srvc_uuid = ble_driver.ble_uuid_t()
326         srvc_uuid.type = ble_driver.BLE_UUID_TYPE_BLE
```

```
        srvc_uuid.uuid = self.ECG_SERVICE_UUID

        self.error_code =
    ble_driver.sd_ble_gattc_primary_services_discover(self.connection_handle, start_handle,
                                                                                srvc_uuid)

        if self.error_code != ble_driver.NRF_SUCCESS:
            print "Failed to discover primary services"
            return self.error_code

        return ble_driver.NRF_SUCCESS


    def start_characteristic_discovery(self):
        print "Discovering characteristics"
        handle_range = ble_driver.ble_gattc_handle_range_t()
        handle_range.start_handle = self.service_start_handle
        handle_range.end_handle = self.service_end_handle

        self.error_code =
    ble_driver.sd_ble_gattc_characteristics_discover(self.connection_handle, handle_range)

        return self.error_code


    def start_descriptor_discovery(self):
        print "Discovering descriptors"

        handle_range = ble_driver.ble_gattc_handle_range_t()

        if self.ecg_char_handle == 0:
            print "Error. No ECG characteristic handle has been found"
            return

        handle_range.start_handle = self.ecg_char_handle
        handle_range.end_handle = self.service_end_handle

        ble_driver.sd_ble_gattc_descriptors_discover(self.connection_handle, handle_range)


    def set_ecg_cccd(self, value):
        cccd_list = [value, 0]
        print "cccd_list = {}".format(cccd_list)
        cccd_array = util.list_to_uint8_array(cccd_list)

        write_params = ble_driver.ble_gattc_write_params_t()
        write_params.handle = self.ecg_cccd_handle
        write_params.len = len(cccd_list)
        write_params.p_value = cccd_array.cast()
        write_params.write_op = ble_driver.BLE_GATT_OP_WRITE_REQ
        write_params.offset = 0

        ble_driver.sd_ble_gattc_write(self.connection_handle, write_params)


    def init_connection_params(self):
        self.connection_params = ble_driver.ble_gap_conn_params_t()
```

```
382         self.connection_params.min_conn_interval = util.msec_to_units(30, util.UNIT_1_25_MS)
383         self.connection_params.max_conn_interval = util.msec_to_units(60, util.UNIT_1_25_MS)
384         self.connection_params.conn_sup_timeout = util.msec_to_units(4000, util.UNIT_10_MS)
385         self.connection_params.slave_latency = 0
386 
387 
388     def init_scan_params(self):
389         self.scan_params = ble_driver.ble_gap_scan_params_t()
390         self.scan_params.active = 1
391         self.scan_params.interval = util.msec_to_units(200, util.UNIT_0_625_MS)
392         self.scan_params.window = util.msec_to_units(150, util.UNIT_0_625_MS)
393         self.scan_params.timeout = 0x1000
394 
395     def main(self):
396         ble_driver.sd_rpc_serial_port_name_set(self.SERIAL_PORT)
397         ble_driver.sd_rpc_serial_baud_rate_set(115200)
398         ble_driver.sd_rpc_log_handler_set(self.log_message_handler)
399         ble_driver.sd_rpc_evt_handler_set(self.ble_evt_handler)
400         self.error_code = ble_driver.sd_rpc_open()
401         self.fig = matplotlib.pyplot.figure()
402         Plotter(self.fig)
403         if self.error_code != ble_driver.NRF_SUCCESS:
404             print "Failed to open the nRF51 BLE Driver, error code:
    0x{0:X}.".format(self.error_code)
405             return
406 
407         self.init_connection_params()
408         self.init_scan_params()
409         self.start_scan()
410         #self.set_ecg_cccd(self.CCCD_NOTIFY)
411         ccd_notify = 0
412         val = 0
413         previous = time.clock()
414 
415         while True:
416             cur = time.clock()
417             if((cur-previous) >= 1):
418                 previous = time.clock()
419                 self.fig.clear()
420                 self.fig.gca().grid()
421                 self.fig.gca().relim()
422                 self.fig.gca().autoscale_view(True,True,True)
423                 self.fig.gca().plot(self.time[-100:],self.new_val[-100:])
424                 self.fig.canvas.draw()
425             if(ccd_notify != 1):
426                 sys.stdin.readline()
427                 ccd_notify ^= self.CCCD_NOTIFY
428                 print "set_hrm_cccd({})".format(ccd_notify)
429                 self.set_ecg_cccd(ccd_notify)
430 
431         self.error_code = ble_driver.sd_rpc_close()
432 
433         if self.error_code != ble_driver.NRF_SUCCESS:
434             print "Failed to close the nRF51 BLE Driver, error code:
    0x{0:X}".format(self.error_code)
435             return
436 
```

```
437 if __name__ == "__main__":
438         ecg = ECG()
439         ecg.main()
440         quit()
```

Listing 1: Python Interface

```c
/* Copyright (c) 2014 Nordic Semiconductor. All Rights Reserved.
 *
 * The information contained herein is property of Nordic Semiconductor ASA.
 * Terms and conditions of usage are described in detail in NORDIC
 * SEMICONDUCTOR STANDARD SOFTWARE LICENSE AGREEMENT.
 *
 * Licensees are granted free, non-transferable use of the information. NO
 * WARRANTY of ANY KIND is provided. This heading must NOT be removed from
 * the file.
 *
 */
/** @example examples/ble_peripheral/ble_app_hrs/main.c
 *
 * @brief Heart Rate Service Sample Application main file.
 *
 * This file contains the source code for a sample application using the Heart Rate service
 * (and also ecg and Device Information services). This application uses the
 * @ref srvlib_conn_params module.
 */

#include <stdint.h>
#include <string.h>
#include "nordic_common.h"
#include "nrf.h"
#include "app_error.h"
#include "nrf51_bitfields.h"
#include "ble.h"
#include "ble_hci.h"
#include "ble_srv_common.h"
#include "ble_advdata.h"
#include "ble_dis.h"
#ifdef BLE_DFU_APP_SUPPORT
#include "ble_dfu.h"
#include "dfu_app_handler.h"
#endif // BLE_DFU_APP_SUPPORT
#include "ble_conn_params.h"
#include "boards.h"
#include "softdevice_handler.h"
#include "app_timer.h"
#include "device_manager.h"
#include "pstorage.h"
#include "app_trace.h"
#include "app_gpiote.h"
#include "bsp.h"
#include "ble_ecg.h"

#define IS_SRVC_CHANGED_CHARACT_PRESENT      0
    /**< Include or not the service_changed characteristic. if not enabled, the server's
    database cannot be changed for the lifetime of the device*/

#define WAKEUP_BUTTON_ID                     0
    /**< Button used to wake up the application. */
#define BOND_DELETE_ALL_BUTTON_ID            0
    /**< Button used for deleting all bonded centrals during startup. */

#define DEVICE_NAME                          "ECG"                                /**< Name
    of device. Will be included in the advertising data. */
```

```
#define MANUFACTURER_NAME                       "nRF"                           /**< Manufacturer. Will be passed to Device Information Service. */
#define APP_ADV_INTERVAL                        40                              /**< The advertising interval (in units of 0.625 ms. This value corresponds to 25 ms). */
#define APP_ADV_TIMEOUT_IN_SECONDS              180                             /**< The advertising timeout in units of seconds. */

#define APP_TIMER_PRESCALER                     0                               /**< Value of the RTC1 PRESCALER register. */
#define APP_TIMER_MAX_TIMERS                    (7+BSP_APP_TIMERS_NUMBER)       /**< Maximum number of simultaneously created timers. */
#define APP_TIMER_OP_QUEUE_SIZE                 2                               /**< Size of timer operation queues. */
#define SLAVE_LATENCY                           0                               /**< Slave latency. */
#define CONN_SUP_TIMEOUT                        MSEC_TO_UNITS(4000, UNIT_10_MS) /**< Connection supervisory timeout (4 seconds). */

#define FIRST_CONN_PARAMS_UPDATE_DELAY          APP_TIMER_TICKS(5000, APP_TIMER_PRESCALER) /**< Time from initiating event (connect or start of notification) to first time sd_ble_gap_conn_param_update is called (5 seconds). */
#define NEXT_CONN_PARAMS_UPDATE_DELAY           APP_TIMER_TICKS(30000, APP_TIMER_PRESCALER)/**< Time between each call to sd_ble_gap_conn_param_update after the first call (30 seconds). */
#define MAX_CONN_PARAMS_UPDATE_COUNT            3                               /**< Number of attempts before giving up the connection parameter negotiation. */

#define ECG_LEVEL_MEAS_INTERVAL                 APP_TIMER_TICKS(8, APP_TIMER_PRESCALER) /**< ecg level measurement interval (ticks). */
//Lou: The above is the interval for the ADC measurement
#define MIN_CONN_INTERVAL                   MSEC_TO_UNITS(15, UNIT_1_25_MS)             /**< Minimum connection interval (7.5 ms). */
#define MAX_CONN_INTERVAL                   MSEC_TO_UNITS(100, UNIT_1_25_MS)              /**< Maximum connection interval (15 ms). */



#define MAX_CONN_PARAMS_UPDATE_COUNT            3                               /**< Number of attempts before giving up the connection parameter negotiation. */

#define SEC_PARAM_TIMEOUT                       30                              /**< Timeout for Pairing Request or Security Request (in seconds). */
#define SEC_PARAM_BOND                          1                               /**< Perform bonding. */
#define SEC_PARAM_MITM                          0                               /**< Man In The Middle protection not required. */
#define SEC_PARAM_IO_CAPABILITIES               BLE_GAP_IO_CAPS_NONE            /**< No I/O capabilities. */
#define SEC_PARAM_OOB                           0                               /**< Out Of Band data not available. */
#define SEC_PARAM_MIN_KEY_SIZE                  7                               /**< Minimum encryption key size. */
#define SEC_PARAM_MAX_KEY_SIZE                  16                              /**< Maximum encryption key size. */

#define DEAD_BEEF                               0xDEADBEEF                      /**< Value used as error code on stack dump, can be used to identify stack location on
```

```
   stack unwind. */
#ifdef BLE_DFU_APP_SUPPORT
#define DFU_REV_MAJOR                           0x00                                        /**
   DFU Major revision number to be exposed. */
#define DFU_REV_MINOR                           0x01                                        /**
   DFU Minor revision number to be exposed. */
#define DFU_REVISION                            ((DFU_REV_MAJOR << 8) | DFU_REV_MINOR)      /**
   DFU Revision number to be exposed. Combined of major and minor versions. */
#endif // BLE_DFU_APP_SUPPORT


static uint16_t                                 m_conn_handle = BLE_CONN_HANDLE_INVALID;
   /**< Handle of the current connection. */
static ble_gap_adv_params_t                     m_adv_params;
   /**< Parameters to be passed to the stack when starting advertising. */
static ble_ecg_t                                m_ecg;
   /**< Structure used to identify the ecg service. */

static app_timer_id_t                           m_ecg_timer_id;
   /**< ecg timer. */
static dm_application_instance_t                m_app_handle;
   /**< Application identifier allocated by device manager */

static bool                                     m_memory_access_in_progress = false;
   /**< Flag to keep track of ongoing operations on persistent memory. */
#ifdef BLE_DFU_APP_SUPPORT
static ble_dfu_t                                m_dfus;
   /**< Structure used to identify the DFU service. */
#endif // BLE_DFU_APP_SUPPORT


/**@brief Callback function for asserts in the SoftDevice.
 *
 * @details This function will be called in case of an assert in the SoftDevice.
 *
 * @warning This handler is an example only and does not fit a final product. You need to
   analyze
 *          how your product is supposed to react in case of Assert.
 * @warning On assert from the SoftDevice, the system can only recover on reset.
 *
 * @param[in]   line_num   Line number of the failing ASSERT call.
 * @param[in]   file_name  File name of the failing ASSERT call.
 */
void assert_nrf_callback(uint16_t line_num, const uint8_t * p_file_name)
{
    app_error_handler(DEAD_BEEF, line_num, p_file_name);
}


/**@brief Function for performing ecg measurement and updating the ecg Level characteristic
 *        in ecg Service.
 */
static void ecg_level_update(uint8_t level)
{
    uint32_t err_code;
    err_code = ble_ecg_level_update(&m_ecg, level);
    if ((err_code != NRF_SUCCESS) &&
```

```
        (err_code != NRF_ERROR_INVALID_STATE) &&
        (err_code != BLE_ERROR_NO_TX_BUFFERS) &&
        (err_code != BLE_ERROR_GATTS_SYS_ATTR_MISSING)
        )
    {
        APP_ERROR_HANDLER(err_code);
    }
}


/**@brief Function for handling the ecg measurement timer timeout.
 *
 * @details This function will be called each time the ecg level measurement timer expires.
 *
 * @param[in]   p_context   Pointer used for passing some arbitrary information (context) from the
 *                          app_start_timer() call to the timeout handler.
 */
static void ecg_level_meas_timeout_handler(void * p_context)
{

        uint32_t p_is_running = 0;

        sd_clock_hfclk_request();
        while(! p_is_running) {                             //wait for the hfclk to be available
                sd_clock_hfclk_is_running((&p_is_running));
        }

    NRF_ADC->TASKS_START = 1;                               //Start ADC sampling
}

/**@brief Function for the Timer initialization.
 *
 * @details Initializes the timer module. This creates and starts application timers.
 */
static void timers_init(void)
{
    uint32_t err_code;

    // Initialize timer module.
    APP_TIMER_INIT(APP_TIMER_PRESCALER, APP_TIMER_MAX_TIMERS, APP_TIMER_OP_QUEUE_SIZE, false);

    // Create timers.
    err_code = app_timer_create(&m_ecg_timer_id,
                                APP_TIMER_MODE_REPEATED,
                                ecg_level_meas_timeout_handler);
    APP_ERROR_CHECK(err_code);
}


/**@brief Function for the GAP initialization.
 *
 * @details This function sets up all the necessary GAP (Generic Access Profile) parameters of the
```

```
182  *          device including the device name, appearance, and the preferred connection
       parameters.
183  */
184 static void gap_params_init(void)
185 {
186     uint32_t                err_code;
187     ble_gap_conn_params_t   gap_conn_params;
188     ble_gap_conn_sec_mode_t sec_mode;
189 
190     BLE_GAP_CONN_SEC_MODE_SET_OPEN(&sec_mode);
191 
192     err_code = sd_ble_gap_device_name_set(&sec_mode,
193                                           (const uint8_t *)DEVICE_NAME,
194                                           strlen(DEVICE_NAME));
195     APP_ERROR_CHECK(err_code);
196 
197     err_code = sd_ble_gap_appearance_set(BLE_APPEARANCE_GENERIC_HEART_RATE_SENSOR);
198     APP_ERROR_CHECK(err_code);
199 
200     memset(&gap_conn_params, 0, sizeof(gap_conn_params));
201 
202     gap_conn_params.min_conn_interval = MIN_CONN_INTERVAL;
203     gap_conn_params.max_conn_interval = MAX_CONN_INTERVAL;
204     gap_conn_params.slave_latency     = SLAVE_LATENCY;
205     gap_conn_params.conn_sup_timeout  = CONN_SUP_TIMEOUT;
206 
207     err_code = sd_ble_gap_ppcp_set(&gap_conn_params);
208     APP_ERROR_CHECK(err_code);
209 }
210 
211 
212 /**@brief Function for initializing the Advertising functionality.
213  *
214  * @details Encodes the required advertising data and passes it to the stack.
215  *          Also builds a structure to be passed to the stack when starting advertising.
216  */
217 static void advertising_init(void)
218 {
219     uint32_t      err_code;
220     ble_advdata_t advdata;
221     uint8_t       flags = BLE_GAP_ADV_FLAGS_LE_ONLY_GENERAL_DISC_MODE;
222 
223     ble_uuid_t adv_uuids[] =
224     {
225         {BLE_UUID_ECG_SERVICE,                  BLE_UUID_TYPE_BLE},
226         {BLE_UUID_DEVICE_INFORMATION_SERVICE,   BLE_UUID_TYPE_BLE}
227     };
228 
229     // Build and set advertising data.
230     memset(&advdata, 0, sizeof(advdata));
231 
232     advdata.name_type               = BLE_ADVDATA_FULL_NAME;
233     advdata.include_appearance      = true;
234     advdata.flags.size              = sizeof(flags);
235     advdata.flags.p_data            = &flags;
236     advdata.uuids_complete.uuid_cnt = sizeof(adv_uuids) / sizeof(adv_uuids[0]);
237     advdata.uuids_complete.p_uuids  = adv_uuids;
```

```
238
239     err_code = ble_advdata_set(&advdata, NULL);
240     APP_ERROR_CHECK(err_code);
241
242     // Initialize advertising parameters (used when starting advertising).
243     memset(&m_adv_params, 0, sizeof(m_adv_params));
244
245     m_adv_params.type        = BLE_GAP_ADV_TYPE_ADV_IND;
246     m_adv_params.p_peer_addr = NULL;                             // Undirected advertisement.
247     m_adv_params.fp          = BLE_GAP_ADV_FP_ANY;
248     m_adv_params.interval    = APP_ADV_INTERVAL;
249     m_adv_params.timeout     = APP_ADV_TIMEOUT_IN_SECONDS;
250 }
251
252
253 #ifdef BLE_DFU_APP_SUPPORT
254 static void advertising_stop(void)
255 {
256     uint32_t err_code;
257
258     err_code = sd_ble_gap_adv_stop();
259     APP_ERROR_CHECK(err_code);
260
261     err_code = bsp_indication_set(BSP_INDICATE_IDLE);
262     APP_ERROR_CHECK(err_code);
263 }
264
265
266 /** @snippet [DFU BLE Reset prepare] */
267 static void reset_prepare(void)
268 {
269     uint32_t err_code;
270
271     if (m_conn_handle != BLE_CONN_HANDLE_INVALID)
272     {
273         // Disconnect from peer.
274         err_code = sd_ble_gap_disconnect(m_conn_handle,
    BLE_HCI_REMOTE_USER_TERMINATED_CONNECTION);
275         APP_ERROR_CHECK(err_code);
276         err_code = bsp_indication_set(BSP_INDICATE_IDLE);
277         APP_ERROR_CHECK(err_code);
278     }
279     else
280     {
281         // If not connected, then the device will be advertising. Hence stop the
    advertising.
282         advertising_stop();
283     }
284
285     err_code = ble_conn_params_stop();
286     APP_ERROR_CHECK(err_code);
287 }
288 /** @snippet [DFU BLE Reset prepare] */
289 #endif // BLE_DFU_APP_SUPPORT
290
291
292 /**@brief Function for initializing services that will be used by the application.
```

```
 *
 * @details Initialize the Heart Rate, ecg and Device Information services.
 */
static void services_init(void)
{
    uint32_t       err_code;
    ble_ecg_init_t ecg_init;
    ble_dis_init_t dis_init;
    // Initialize ecg Service.
    memset(&ecg_init, 0, sizeof(ecg_init));

    // Here the sec level for the ecg Service can be changed/increased.
    BLE_GAP_CONN_SEC_MODE_SET_OPEN(&ecg_init.ecg_level_char_attr_md.cccd_write_perm);
    BLE_GAP_CONN_SEC_MODE_SET_OPEN(&ecg_init.ecg_level_char_attr_md.read_perm);
    BLE_GAP_CONN_SEC_MODE_SET_NO_ACCESS(&ecg_init.ecg_level_char_attr_md.write_perm);

    BLE_GAP_CONN_SEC_MODE_SET_OPEN(&ecg_init.ecg_level_report_read_perm);

    ecg_init.evt_handler          = NULL;
    ecg_init.support_notification = true;
    ecg_init.p_report_ref         = NULL;
    ecg_init.initial_ecg_level   = 0;

    err_code = ble_ecg_init(&m_ecg, &ecg_init);
    APP_ERROR_CHECK(err_code);

    // Initialize Device Information Service.
    memset(&dis_init, 0, sizeof(dis_init));

    ble_srv_ascii_to_utf8(&dis_init.manufact_name_str, (char *)MANUFACTURER_NAME);

    BLE_GAP_CONN_SEC_MODE_SET_OPEN(&dis_init.dis_attr_md.read_perm);
    BLE_GAP_CONN_SEC_MODE_SET_NO_ACCESS(&dis_init.dis_attr_md.write_perm);

    err_code = ble_dis_init(&dis_init);
    APP_ERROR_CHECK(err_code);

#ifdef BLE_DFU_APP_SUPPORT
    /** @snippet [DFU BLE Service initialization] */
    ble_dfu_init_t   dfus_init;

    // Initialize the Device Firmware Update Service.
    memset(&dfus_init, 0, sizeof(dfus_init));

    dfus_init.evt_handler    = dfu_app_on_dfu_evt;
    dfus_init.error_handler  = NULL; //service_error_handler - Not used as only the switch
   from app to DFU mode is required and not full dfu service.
    dfus_init.evt_handler    = dfu_app_on_dfu_evt;
    dfus_init.revision       = DFU_REVISION;

    err_code = ble_dfu_init(&m_dfus, &dfus_init);
    APP_ERROR_CHECK(err_code);

    dfu_app_reset_prepare_set(reset_prepare);
    /** @snippet [DFU BLE Service initialization] */
#endif // BLE_DFU_APP_SUPPORT
}
```

```
349
350
351
352 /**@brief Function for starting application timers.
353  */
354 static void application_timers_start(void)
355 {
356     uint32_t err_code;
357
358     // Start application timers.
359     err_code = app_timer_start(m_ecg_timer_id, ECG_LEVEL_MEAS_INTERVAL, NULL);
360     APP_ERROR_CHECK(err_code);
361 }
362
363
364 /**@brief Function for starting advertising.
365  */
366 static void advertising_start(void)
367 {
368     uint32_t err_code;
369     uint32_t count;
370
371     // Verify if there is any flash access pending, if yes delay starting advertising until
372     // it's complete.
373     err_code = pstorage_access_status_get(&count);
374     APP_ERROR_CHECK(err_code);
375
376     if (count != 0)
377     {
378         m_memory_access_in_progress = true;
379         return;
380     }
381
382     err_code = sd_ble_gap_adv_start(&m_adv_params);
383     APP_ERROR_CHECK(err_code);
384
385     err_code = bsp_indication_set(BSP_INDICATE_ADVERTISING);
386     APP_ERROR_CHECK(err_code);
387 }
388
389
390 /**@brief Function for handling the Connection Parameters Module.
391  *
392  * @details This function will be called for all events in the Connection Parameters Module
    which
393  *          are passed to the application.
394  *          @note All this function does is to disconnect. This could have been done by
    simply
395  *                setting the disconnect_on_fail config parameter, but instead we use the
    event
396  *                handler mechanism to demonstrate its use.
397  *
398  * @param[in]   p_evt   Event received from the Connection Parameters Module.
399  */
400 static void on_conn_params_evt(ble_conn_params_evt_t * p_evt)
401 {
402     uint32_t err_code;
```

```
403
404     if (p_evt->evt_type == BLE_CONN_PARAMS_EVT_FAILED)
405     {
406         err_code = sd_ble_gap_disconnect(m_conn_handle, BLE_HCI_CONN_INTERVAL_UNACCEPTABLE);
407         APP_ERROR_CHECK(err_code);
408     }
409 }
410
411
412 /**@brief Function for handling a Connection Parameters error.
413  *
414  * @param[in]   nrf_error   Error code containing information about what went wrong.
415  */
416 static void conn_params_error_handler(uint32_t nrf_error)
417 {
418     APP_ERROR_HANDLER(nrf_error);
419 }
420
421
422 /**@brief Function for initializing the Connection Parameters module.
423  */
424 static void conn_params_init(void)
425 {
426     uint32_t               err_code;
427     ble_conn_params_init_t cp_init;
428
429     memset(&cp_init, 0, sizeof(cp_init));
430
431     cp_init.p_conn_params                  = NULL;
432     cp_init.first_conn_params_update_delay = FIRST_CONN_PARAMS_UPDATE_DELAY;
433     cp_init.next_conn_params_update_delay  = NEXT_CONN_PARAMS_UPDATE_DELAY;
434     cp_init.max_conn_params_update_count   = MAX_CONN_PARAMS_UPDATE_COUNT;
435     cp_init.start_on_notify_cccd_handle    = m_ecg.ecg_level_handles.cccd_handle;
436     cp_init.disconnect_on_fail             = false;
437     cp_init.evt_handler                    = on_conn_params_evt;
438     cp_init.error_handler                  = conn_params_error_handler;
439
440     err_code = ble_conn_params_init(&cp_init);
441     APP_ERROR_CHECK(err_code);
442 }
443
444
445 /**@brief Function for handling the Application's BLE Stack events.
446  *
447  * @param[in]   p_ble_evt   Bluetooth stack event.
448  */
449 static void on_ble_evt(ble_evt_t * p_ble_evt)
450 {
451     uint32_t err_code;
452
453     switch (p_ble_evt->header.evt_id)
454     {
455         case BLE_GAP_EVT_CONNECTED:
456
457             err_code = bsp_indication_set(BSP_INDICATE_CONNECTED);
458             APP_ERROR_CHECK(err_code);
459             m_conn_handle = p_ble_evt->evt.gap_evt.conn_handle;
```

```
460             break;
461
462         case BLE_GAP_EVT_DISCONNECTED:
463             err_code = bsp_indication_set(BSP_INDICATE_IDLE);
464             APP_ERROR_CHECK(err_code);
465             advertising_start();
466             break;
467
468         case BLE_GAP_EVT_TIMEOUT:
469
470             if (p_ble_evt->evt.gap_evt.params.timeout.src ==
    BLE_GAP_TIMEOUT_SRC_ADVERTISEMENT)
471             {
472                 err_code = bsp_indication_set(BSP_INDICATE_IDLE);
473                 APP_ERROR_CHECK(err_code);
474
475                 // enable buttons to wake-up from power off
476                 err_code = bsp_buttons_enable( (1 << WAKEUP_BUTTON_ID) | (1 <<
    BOND_DELETE_ALL_BUTTON_ID) );
477                 APP_ERROR_CHECK(err_code);
478
479                 // Go to system-off mode (this function will not return; wakeup will cause
    a reset).
480                 err_code = sd_power_system_off();
481                 APP_ERROR_CHECK(err_code);
482             }
483             break;
484
485         default:
486             // No implementation needed.
487             break;
488     }
489 }
490
491
492 /**@brief Function for handling the Application's system events.
493  *
494  * @param[in]   sys_evt   system event.
495  */
496 static void on_sys_evt(uint32_t sys_evt)
497 {
498     switch(sys_evt)
499     {
500         case NRF_EVT_FLASH_OPERATION_SUCCESS:
501         case NRF_EVT_FLASH_OPERATION_ERROR:
502
503             if (m_memory_access_in_progress)
504             {
505                 m_memory_access_in_progress = false;
506                 advertising_start();
507             }
508             break;
509
510         default:
511             // No implementation needed.
512             break;
513     }
```

```
514 }
515
516
517 /**@brief Function for dispatching a BLE stack event to all modules with a BLE stack event
    handler.
518  *
519  * @details This function is called from the BLE Stack event interrupt handler after a BLE
    stack
520  *          event has been received.
521  *
522  * @param[in]   p_ble_evt   Bluetooth stack event.
523  */
524 static void ble_evt_dispatch(ble_evt_t * p_ble_evt)
525 {
526     dm_ble_evt_handler(p_ble_evt);
527     //ble_hrs_on_ble_evt(&m_hrs, p_ble_evt);
528     ble_ecg_on_ble_evt(&m_ecg, p_ble_evt);
529     ble_conn_params_on_ble_evt(p_ble_evt);
530 #ifdef BLE_DFU_APP_SUPPORT
531     /** @snippet [Propagating BLE Stack events to DFU Service] */
532     ble_dfu_on_ble_evt(&m_dfus, p_ble_evt);
533     /** @snippet [Propagating BLE Stack events to DFU Service] */
534 #endif // BLE_DFU_APP_SUPPORT
535     on_ble_evt(p_ble_evt);
536 }
537
538
539 /**@brief Function for dispatching a system event to interested modules.
540  *
541  * @details This function is called from the System event interrupt handler after a system
542  *          event has been received.
543  *
544  * @param[in]   sys_evt   System stack event.
545  */
546 static void sys_evt_dispatch(uint32_t sys_evt)
547 {
548     pstorage_sys_event_handler(sys_evt);
549     on_sys_evt(sys_evt);
550 }
551
552
553 /**@brief Function for initializing the BLE stack.
554  *
555  * @details Initializes the SoftDevice and the BLE event interrupt.
556  */
557 static void ble_stack_init(void)
558 {
559     uint32_t err_code;
560
561     // Initialize the SoftDevice handler module.
562     SOFTDEVICE_HANDLER_INIT(NRF_CLOCK_LFCLKSRC_XTAL_20_PPM, false);
563
564 #if defined(S110) || defined(S310)
565     // Enable BLE stack
566     ble_enable_params_t ble_enable_params;
567     memset(&ble_enable_params, 0, sizeof(ble_enable_params));
568     ble_enable_params.gatts_enable_params.service_changed = IS_SRVC_CHANGED_CHARACT_PRESENT;
```

```
569     err_code = sd_ble_enable(&ble_enable_params);
570     APP_ERROR_CHECK(err_code);
571 #endif
572 
573     // Register with the SoftDevice handler module for BLE events.
574     err_code = softdevice_ble_evt_handler_set(ble_evt_dispatch);
575     APP_ERROR_CHECK(err_code);
576 
577     // Register with the SoftDevice handler module for BLE events.
578     err_code = softdevice_sys_evt_handler_set(sys_evt_dispatch);
579     APP_ERROR_CHECK(err_code);
580 }
581 
582 
583 /**@brief Function for handling the Device Manager events.
584  *
585  * @param[in]   p_evt   Data associated to the device manager event.
586  */
587 static uint32_t device_manager_evt_handler(dm_handle_t const * p_handle,
588                                            dm_event_t const  * p_event,
589                                            api_result_t        event_result)
590 {
591     APP_ERROR_CHECK(event_result);
592 
593     switch(p_event->event_id)
594     {
595 #ifdef BLE_DFU_APP_SUPPORT
596         case DM_EVT_DEVICE_CONTEXT_LOADED: // Fall through.
597         case DM_EVT_SECURITY_SETUP_COMPLETE:
598             dfu_app_set_dm_handle(p_handle);
599             break;
600         case DM_EVT_DISCONNECTION:
601             dfu_app_set_dm_handle(NULL);
602             break;
603 #endif // BLE_DFU_APP_SUPPORT
604     }
605 
606     return NRF_SUCCESS;
607 }
608 
609 
610 /**@brief Function for the Device Manager initialization.
611  */
612 static void device_manager_init(void)
613 {
614     uint32_t                err_code;
615     dm_init_param_t         init_data;
616     dm_application_param_t register_param;
617 
618     // Initialize persistent storage module.
619     err_code = pstorage_init();
620     APP_ERROR_CHECK(err_code);
621 
622     // Clear all bonded centrals if the Bonds Delete button is pushed.
623     err_code =
    bsp_button_is_pressed(BOND_DELETE_ALL_BUTTON_ID,&(init_data.clear_persistent_data));
624     APP_ERROR_CHECK(err_code);
```

```
625
626     err_code = dm_init(&init_data);
627     APP_ERROR_CHECK(err_code);
628
629     memset(&register_param.sec_param, 0, sizeof(ble_gap_sec_params_t));
630
631     register_param.sec_param.timeout      = SEC_PARAM_TIMEOUT;
632     register_param.sec_param.bond         = SEC_PARAM_BOND;
633     register_param.sec_param.mitm         = SEC_PARAM_MITM;
634     register_param.sec_param.io_caps      = SEC_PARAM_IO_CAPABILITIES;
635     register_param.sec_param.oob          = SEC_PARAM_OOB;
636     register_param.sec_param.min_key_size = SEC_PARAM_MIN_KEY_SIZE;
637     register_param.sec_param.max_key_size = SEC_PARAM_MAX_KEY_SIZE;
638     register_param.evt_handler            = device_manager_evt_handler;
639     register_param.service_type           = DM_PROTOCOL_CNTXT_GATT_SRVR_ID;
640
641     err_code = dm_register(&m_app_handle, &register_param);
642     APP_ERROR_CHECK(err_code);
643 }
644
645 /**@brief Function for the Power manager.
646  */
647 static void power_manage(void)
648 {
649     //uint32_t err_code = sd_app_evt_wait();
650     //APP_ERROR_CHECK(err_code);
651 }
652 static void adc_init(void)
653 {
654     /* Enable interrupt on ADC sample ready event*/
655     NRF_ADC->INTENSET = ADC_INTENSET_END_Msk;
656     sd_nvic_SetPriority(ADC_IRQn, NRF_APP_PRIORITY_LOW);
657     sd_nvic_EnableIRQ(ADC_IRQn);
658
659     NRF_ADC->CONFIG = (ADC_CONFIG_EXTREFSEL_None << ADC_CONFIG_EXTREFSEL_Pos) /* Bits
    17..16 : ADC external reference pin selection. */
660                                     | (ADC_CONFIG_PSEL_AnalogInput2 << ADC_CONFIG_PSEL_Pos)
                    /*!< Use analog input 2 as analog input. */
661                                     | (ADC_CONFIG_REFSEL_VBG << ADC_CONFIG_REFSEL_Pos)
                        /*!< Use internal 1.2V bandgap voltage as reference for conversion.
    */
662                                     | (ADC_CONFIG_INPSEL_AnalogInputNoPrescaling <<
    ADC_CONFIG_INPSEL_Pos) /*!< Analog input specified by PSEL with no prescaling used as
    input for the conversion. */
663                                     | (ADC_CONFIG_RES_8bit  << ADC_CONFIG_RES_Pos);
                                /*!< 8bit ADC resolution. */
664
665     /* Enable ADC*/
666     NRF_ADC->ENABLE = ADC_ENABLE_ENABLE_Enabled;
667 }
668 void ADC_IRQHandler(void)
669 {
670   uint8_t adc_result;
671
672   NRF_ADC->EVENTS_END = 0;
673     adc_result = NRF_ADC->RESULT;
674     ecg_level_update(adc_result);
```

```
675     NRF_ADC->TASKS_STOP = 1;
676 
677     //Release the external crystal
678     sd_clock_hfclk_release();
679 }
680 
681 
682 /**@brief Function for application main entry.
683  */
684 int main(void)
685 {
686     // Initialize.
687     ble_stack_init();
688     timers_init();
689     APP_GPIOTE_INIT(1);
690 
691     uint32_t err_code = bsp_init(BSP_INIT_LED | BSP_INIT_BUTTONS, APP_TIMER_TICKS(100,
    APP_TIMER_PRESCALER), NULL);
692     APP_ERROR_CHECK(err_code);
693 
694     device_manager_init();
695     gap_params_init();
696     advertising_init();
697     services_init();
698     //sensor_sim_init();
699     conn_params_init();
700     // Start execution.
701     adc_init();
702 
703     application_timers_start();
704     advertising_start();
705 
706 
707 
708     // Enter main loop.
709     for (;; )
710     {
711             power_manage();
712     }
713 }
```

Listing 2: nRF51-DK custom firmware

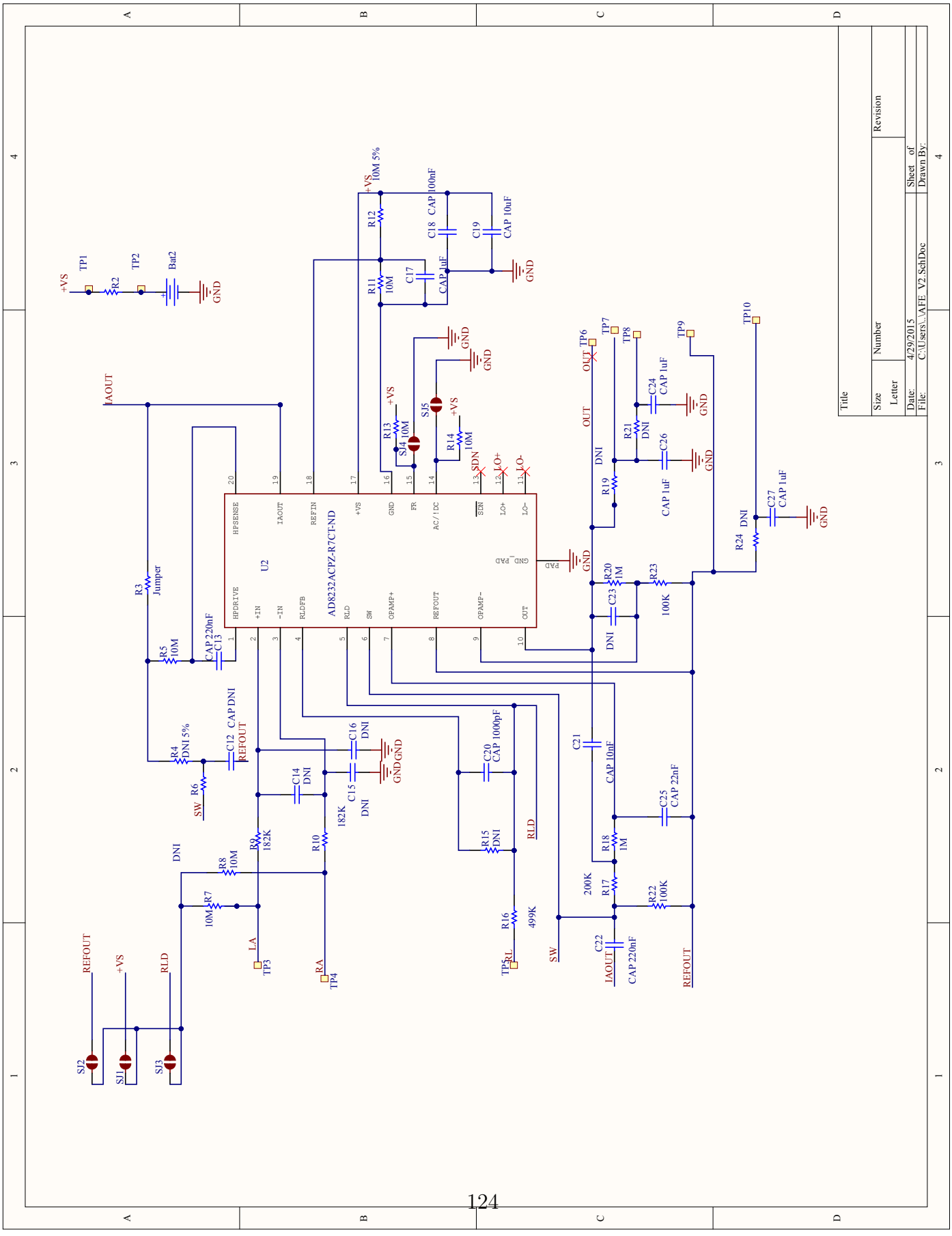
```
var a;

$(function () {
    $(document).ready(function () {
        Highcharts.setOptions({
            global: {
                useUTC: false
            }
        });

        $('#container').highcharts({
            chart: {
                type: 'spline',
                animation: Highcharts.svg, // don't animate in old IE
                marginRight: 10,
                events: {
                    load: function () {

                        // set up the updating of the chart each second
                        var series = this.series[0],
                            maxSamples = 30,
                            count = 0;

                        setInterval(function () {
                            if(a[1]){
                            var x = (new Date()).getTime(), // current time
                                y = a[1];
                            series.addPoint([x, y], true, (++count >= maxSamples));
                            }
                        }, 16);
                    }
                }
            },
            title: {
                text: 'Heartrate BLE Custom'
            },
            xAxis: {
                type: 'datetime',
                tickPixelInterval: 150
            },
            yAxis: {
                title: {
                    text: 'BPM'
                },
                plotLines: [{
                    value: 0,
                    width: 1,
                    color: '#808080'
                }]
            },
            tooltip: {
                formatter: function () {
                    return '<b>' + this.series.name + '</b><br/>' +
                        Highcharts.dateFormat('%Y-%m-%d %H:%M:%S', this.x) + '<br/>' +
                        Highcharts.numberFormat(this.y, 2);
                }
            },
```

```
            legend: {
                enabled: false
            },
            exporting: {
                enabled: false
            },
            series: [{
                name: 'Heartrate',
                data: []
            }]
        });
    });
});

/*change to HRM SDK = 180D, 2A37*/
var heartrate = {
    service:       "0000FF02-0000-1000-8000-00805f9b34fb",
    data:          "0000FF05-0000-1000-8000-00805f9b34fb",
}
var num = 1;
var app = {
    initialize: function() {
        this.bindEvents();
        detailPage.hidden = true;
        disconnect.hidden = true;
    },
    bindEvents: function() {
        document.addEventListener('deviceready', this.onDeviceReady, false);
        refreshButton.addEventListener('touchstart', this.refreshDeviceList, false);
        disconnectButton.addEventListener('touchstart', this.disconnect, false);
        deviceList.addEventListener('touchstart', this.connect, false); // assume not
   scrolling
    },
    onDeviceReady: function() {
        app.refreshDeviceList();
    },
    refreshDeviceList: function() {
        deviceList.innerHTML = ''; // empties the list
        // scan for all devices
        ble.scan([], 5, app.onDiscoverDevice, app.onError);
    },
    onDiscoverDevice: function(device) {
            var listItem = document.createElement('li'),
                html = '<b>' + device.name + '</b><br/>' +
                    'RSSI: ' + device.rssi + ' | ' +
                    device.id;

            listItem.dataset.deviceId = device.id;  // TODO
            listItem.innerHTML = html;
            deviceList.appendChild(listItem);     },
    connect: function(e) {
        var deviceId = e.target.dataset.deviceId,
            onConnect = function() {
                ble.notify(deviceId, heartrate.service, heartrate.data,
   app.onheartrateData, app.onError);
                disconnectButton.dataset.deviceId = deviceId;
                app.showDetailPage();
```

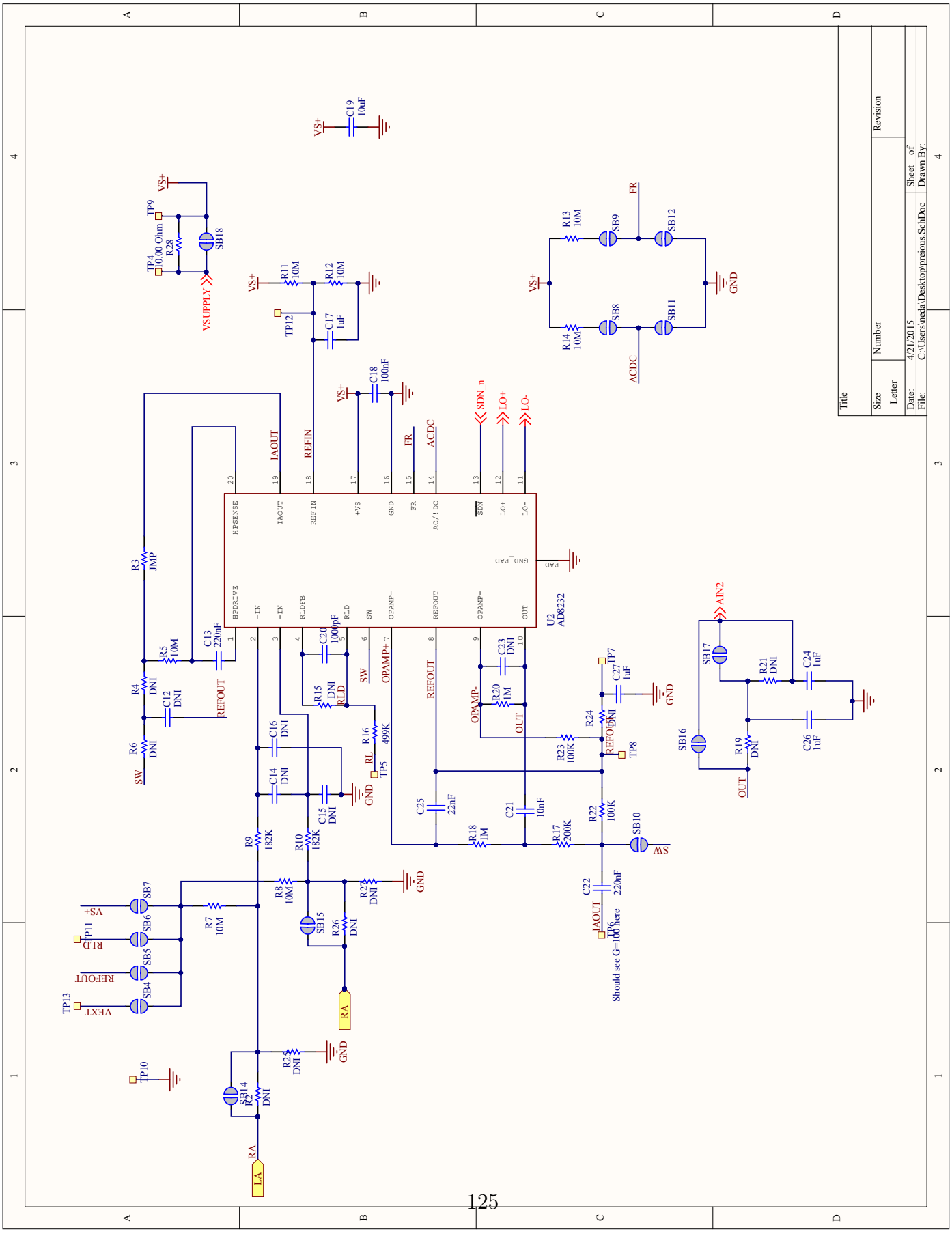
```
            };

        ble.connect(deviceId, onConnect, app.onError);
    },
    onheartrateData: function(data) {
        console.log(data);
        a = new Uint8Array(data);

    },
    disconnect: function(event) {
        var deviceId = event.target.dataset.deviceId;
        ble.disconnect(deviceId, app.showMainPage, app.onError);
    },
    showMainPage: function() {
        disconnect.hidden = true;
        mainPage.hidden = false;
        refresh.hidden = false;
        detailPage.hidden = true;
    },
    showDetailPage: function() {
        disconnect.hidden = false;
        refresh.hidden = true;
        mainPage.hidden = true;
        detailPage.hidden = false;
    },
    onError: function(reason) {
        alert("ERROR: " + reason); // real apps should use notification.alert
    }
};
```

Listing 3: Android app web interface

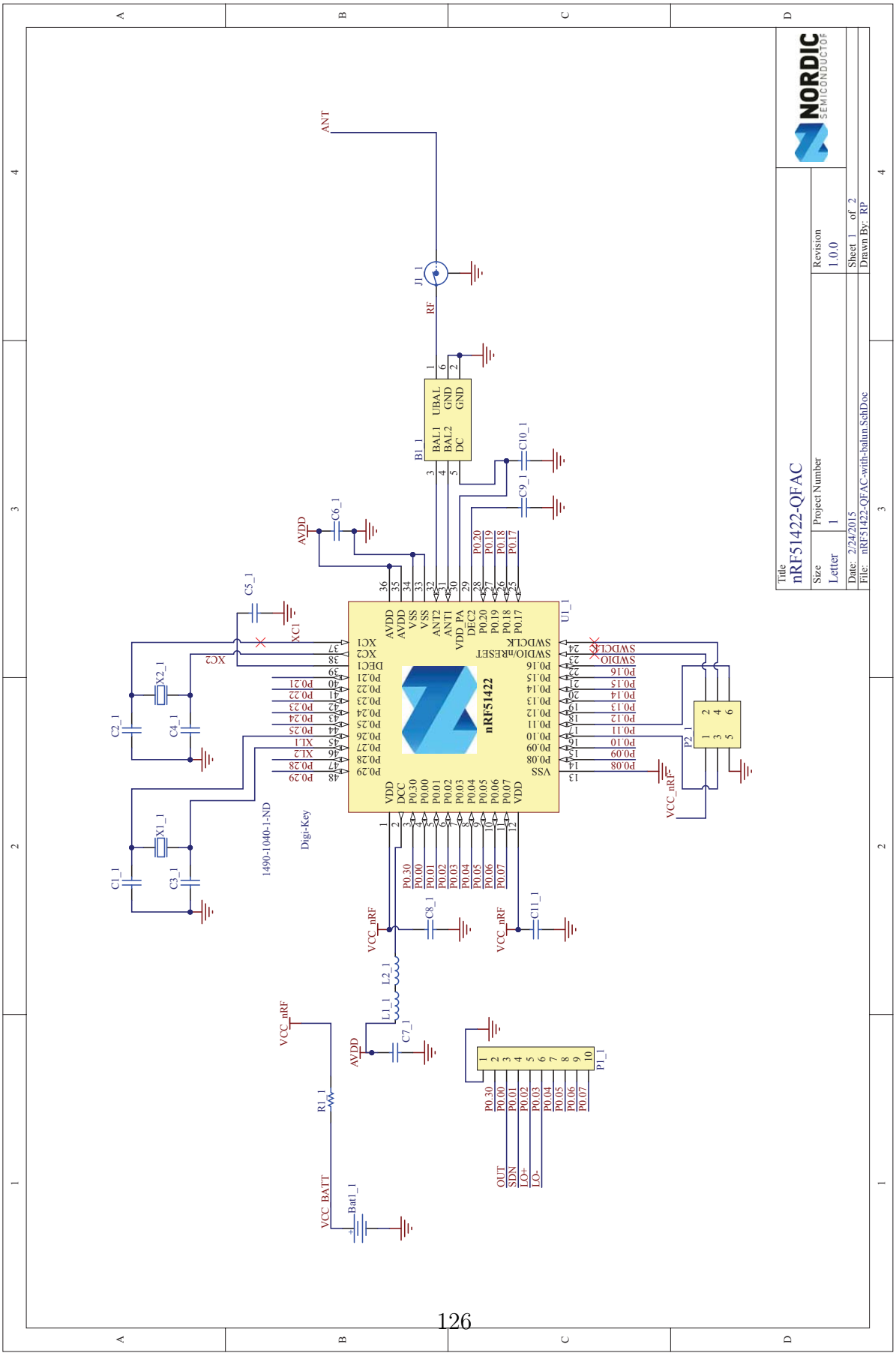
10.1 AFE First Revision Schematic



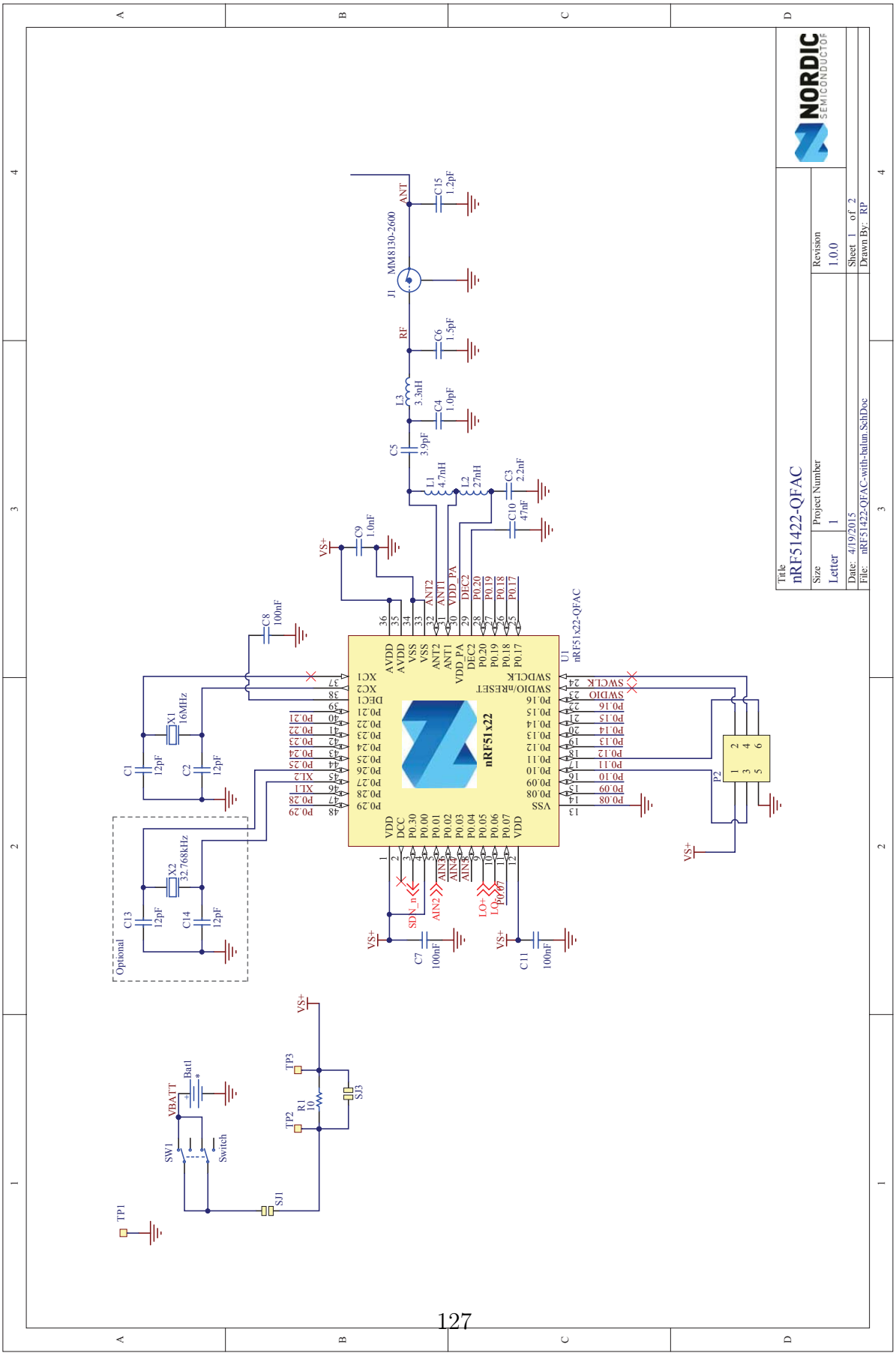
10.2 AFE Second Revision Schematic



10.3 Communication Block First Revision Schematic

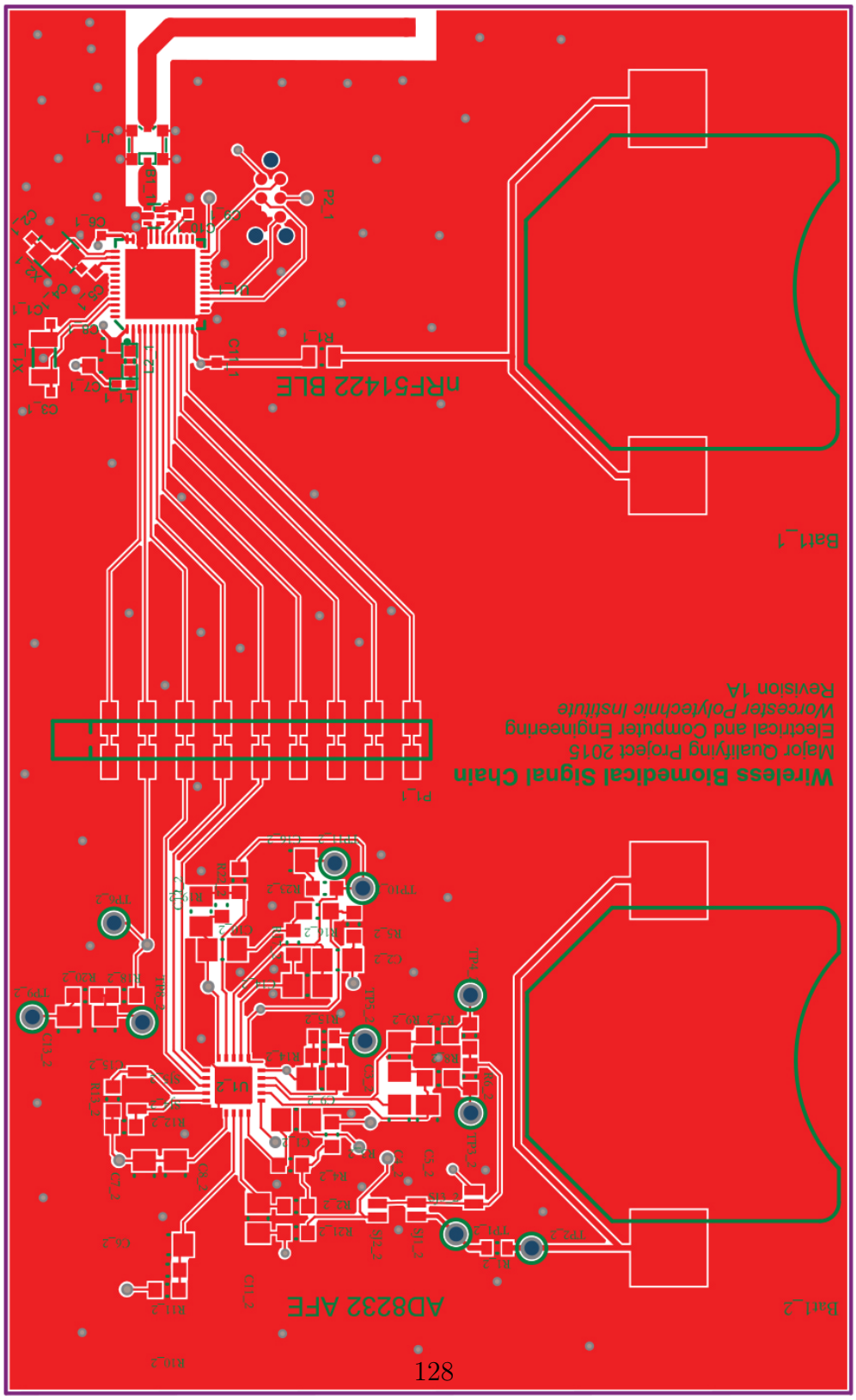


10.4 Communication Block Second Revision Schematic

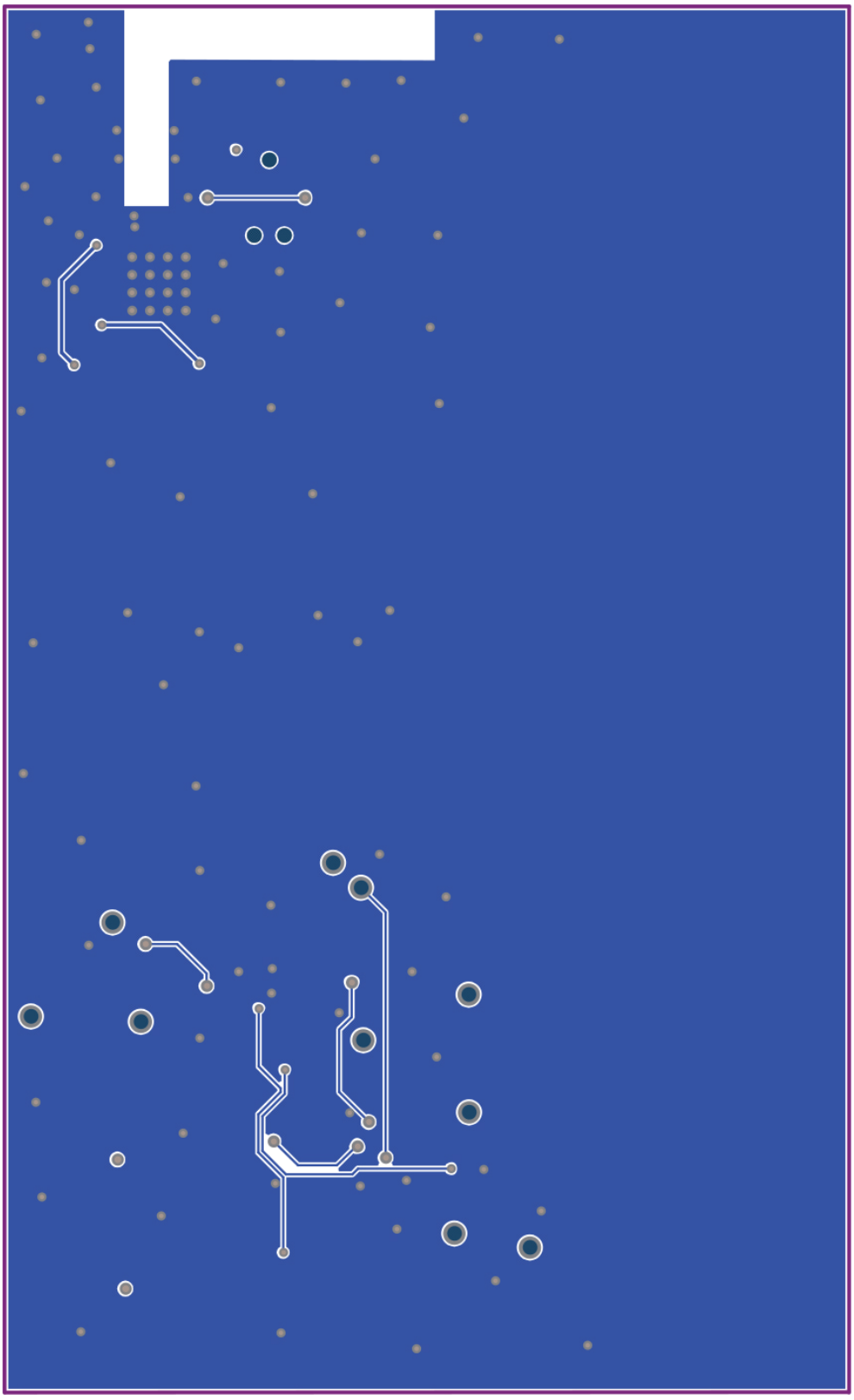


10.5 PCB Prints

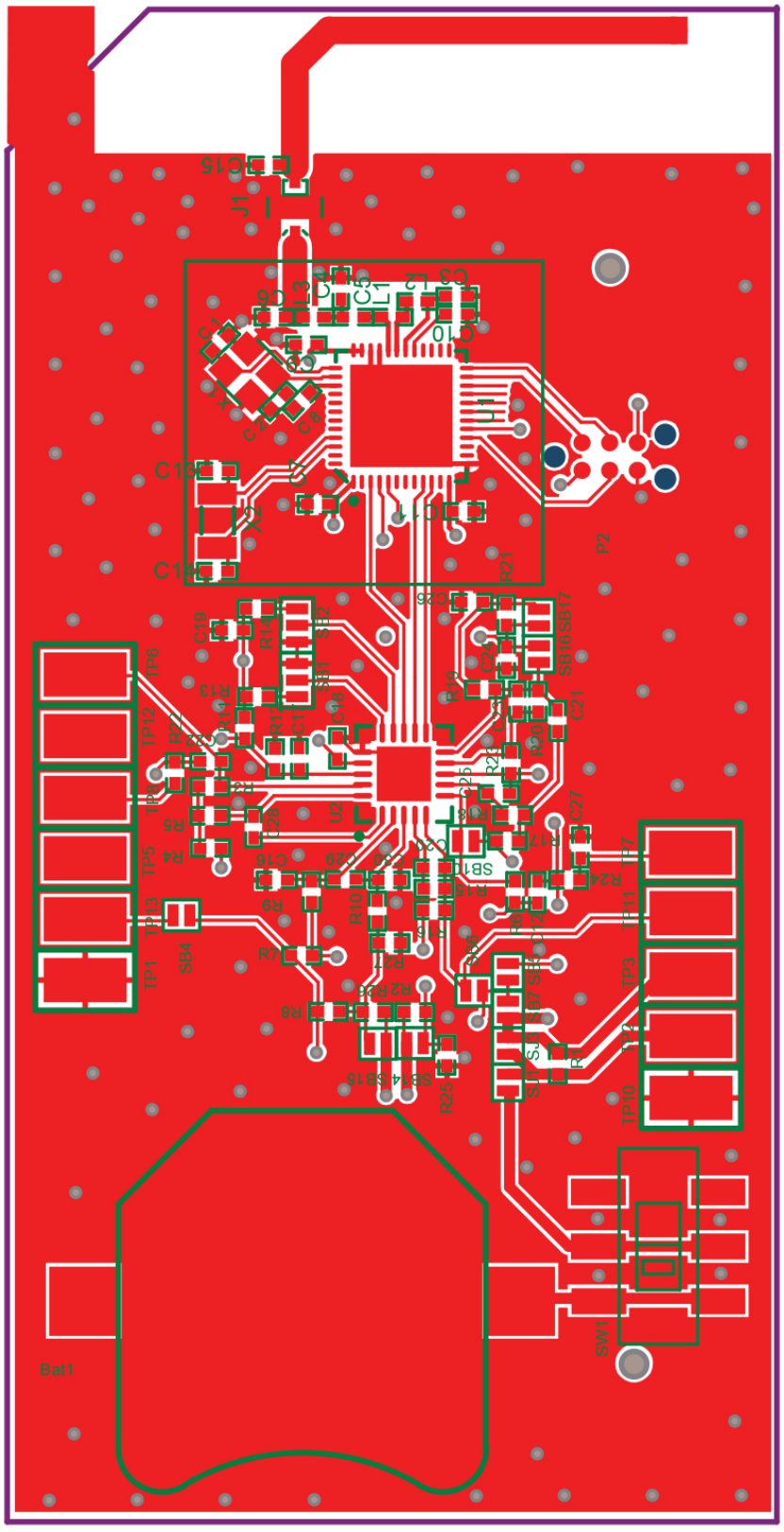
10.5.1 First Revision Top View



10.5.2 First Revision Bottom View



10.5.3 Second Revision Top View



10.5.4 Second Revision Bottom View