Bluetooth Telemetry System for a Wearable Electrocardiogram

Ryan B (Ryan Benjamin) Green

Follow this and additional works at: https://scholarsjunction.msstate.edu/td

Recommended Citation
Green, Ryan B (Ryan Benjamin), "Bluetooth Telemetry System for a Wearable Electrocardiogram" (2013). Theses and Dissertations. 913.
https://scholarsjunction.msstate.edu/td/913

This Graduate Thesis is brought to you for free and open access by the Theses and Dissertations at Scholars Junction. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of Scholars Junction. For more information, please contact scholcomm@msstate.libanswers.com.
Bluetooth telemetry system for a wearable electrocardiogram

By

Ryan B Green

A Thesis
Submitted to the Faculty of
Mississippi State University
in Partial Fulfillment of the Requirements
for the Degree of Master of Science
in Electrical and Computer Engineering
in the Department of Electrical and Computer Engineering

Mississippi State, Mississippi

August 2013
Bluetooth telemetry system for a wearable electrocardiogram

By

Ryan B Green

Approved:

Erdem Topsakal
Associate Professor
Electrical and Computer Engineering
(Major Professor)

J. Patrick Donohoe
Professor
Electrical and Computer Engineering
(Committee Member)

Pan Li
Assistant Professor
Electrical and Computer Engineering
(Committee Member)

James E. Fowler
Professor
Electrical and Computer Engineering
(Graduate Coordinator)

Royce O. Bowden
Interim Dean of the Bagley College of Engineering
The rise of wireless networks has led to a new market in medicine: remote patient monitoring. Practitioners now desire to monitor the health conditions of their patients after hospital release. With the large number of cardiac related deaths and this new demand in medicine being the motivation, this study developed a Bluetooth® telemetry system for a wearable Electrocardiogram. This study also developed a compression t-shirt to hold the ECG and telemetry system. This device communicates the ECG signal of a patient to an Android device within the ISM frequency bands (2.4-2.48 GHz) where the data is displayed and stored in real time. This study is a stepping stone toward more portable heart monitoring that can communicate with the doctor in real time from remote locations.
DEDICATION

To my family and to my Lord.
ACKNOWLEDGEMENTS

The author would like to express his sincerest gratitude to the many people that, with selfless help and support, have made this master’s study possible. First I would like to thank Dr. Erdem Topsakal for spending time and effort to guide me in this endeavor and volunteering as a test subject for this project. I would also like to thank Mustafa Asili for also volunteering for be a test subject of this project as well. I would like to thank David Patterson for assisting me in developing the electrocardiogram circuit used in this study. I would also like to thank my roommate Richard Carley for volunteering his Samsung Tablet for this study. I would like to thank my mother, Leanne Green, for assisting me in the sewing and alterations made to make the ECG garment possible. Lastly, I would like to thank my sister, Kara Green, for asking “Why not button holes,” while discussing the ECG garment design.
TABLE OF CONTENTS

DEDICATION .................................................................................................................... ii
ACKNOWLEDGEMENTS ............................................................................................... iii
LIST OF FIGURES ........................................................................................................... vi

CHAPTER

I.  INTRODUCTION .................................................................................................1

II.  SYSTEM MODEL ...............................................................................................9

2.1  The ECG Garment ..................................................................................11
2.2  Electrocardiogram Amplifier Circuit .....................................................12
  2.2.1  Right Leg Drive Loop .................................................................13
  2.2.2  The Instrumentation Amplifier ..................................................17
  2.2.3  Filtering and Amplification Stage .............................................17
  2.2.4  Analog-To-Digital Converter Compliance Circuit ....................20

2.3  Telemetry Unit .........................................................................................21
  2.3.1  Telemetry System Hardware Overview ......................................22
  2.3.2  Telemetry Firmware Overview ................................................24

2.4  Android Application ...............................................................................26
  2.4.1  Graphical User Interface Overview ..........................................28
  2.4.2  Performance Overview (Java code) ..........................................30

III. PROTOTYPE AND FINAL HARDWARE DESIGN ..................................34

3.1  Prototype .................................................................................................34
3.2  Final Hardware Design ..........................................................................36
  3.2.1  Power Supply ...............................................................................36
  3.2.2  Bluetooth ....................................................................................37

3.3  Final System Overview ..........................................................................38

IV.  SYSTEM PERFORMANCE ........................................................................40

4.1  ECG Analog Circuit Performance .........................................................40
  4.1.1  Instrumentation Amplifier Output ............................................41
  4.1.2  Passive Filter Output .................................................................42
  4.1.3  Active Filter Output .................................................................43
4.1.4 AC Coupling Capacitor Output .....................................................45
4.1.5 Mixer Output..................................................................................46
4.1.6 Analog to Digital Converter Compliance Output ......................47
4.2 Android Application Performance ....................................................48

V. CONCLUSION AND FUTURE WORK ....................................................50

REFERENCES ..................................................................................................................51

APPENDIX

A. PROGRAMMING CODE ..............................................................................53
   A.1 BLUETOOTHCHAT.JAVA ...............................................................54
   A.2 MAIN.XML .........................................................................................68
   A.3 EKG_THESIS.ino................................................................................70
# LIST OF FIGURES

1.1 Graphical representation of an ECG signal .......................................................2
1.2 Standard 3-lead placement, MCL₁, and MCL₆ lead placement .......................4
1.3 The Holter Monitor. ...........................................................................................5
1.4 The percentage of ownership of smartphones in fiscal years 2010 and 2011...............7
2.1 Proposed System Model ....................................................................................9
2.2 ECG Garment With Color Coordinated Lead Placement ...............................11
2.3 Electrocardiogram Augmented Limb Lead topology. .................................12
2.4 ECG Amplifier Schematic ...............................................................................13
2.5 Signal and Power Amplitude Spectrum of a Typical ECG..............................14
2.6 Voltage drift due to motion..............................................................................15
2.7 Power Spectrum of ECG, P-Wave, T-Wave, QRS Complex, Motion Artifact, and Muscle Noise .................................................................16
2.8 Right Leg Drive Loop Schematic ....................................................................16
2.9 The Frequency Response of the Combined Passive Filters. .........................18
2.10 Non Inverting Filter Topology........................................................................19
2.11 Frequency Response of the Non Inverting Filter Topology .........................19
2.12 Gain vs. Frequency of the Filtering and Amplification Stage .......................20
2.13 Analog –To-Digital Converter Compliance Circuit .....................................21
2.14 RN-42 Class 2.0 Bluetooth Module .................................................................23
2.15 Telemetry System Schematic .......................................................................24
2.16 Algorithm for ATmega program ................................................................. 26
2.17 GUI for the Wearable ECG Application ................................................... 29
2.18 Flow Chart for Java Program .................................................................... 31
3.1 Bread Boarded Prototype ........................................................................... 35
3.2 Prototype Android application ................................................................. 35
3.3 Printed Circuit Board .................................................................................. 36
3.4 Power Supplies provided on the Final Hardware Design ......................... 37
3.5 Ground Plane Inset for RN-42 PCB placement .......................................... 38
3.6 Final System .............................................................................................. 39
4.1 ECG Output of the Instrumentation Amplifier .......................................... 41
4.2 ECG Output of the Passive Filters ............................................................. 42
4.3 Relative Amplitudes of the Filtering and Amplification Stage and the ECG signal .......................................................... 44
4.4 ECG Output of the Active Filter ................................................................. 44
4.5 ECG Output of the AC Coupling Capacitor .............................................. 45
4.6 ECG Output of the Mixer .......................................................................... 46
4.7 ECG Output of the ADC Compliance Circuit ........................................... 47
4.8 Android Application System Test Results ................................................. 48
CHAPTER I
INTRODUCTION

A new trend market in medicine is the emergence of remote patient monitoring. With the rise in wireless communication, wireless networks, and smart phones, this new trend has formed new innovations in health monitoring for doctors, patients, and athletes. Doctors and patients, with the assistance of wireless medical devices, will be able to monitor temperature, blood pressure, pulse, and blood glucose levels in more convenient ways. With the assistance of such technology, a patient released from a hospital setting can have health conditions monitored from a patient’s home, office, or recreational areas. The ability to monitor patients’ health conditions remotely can assist doctors in the diagnosis and treatment of their patients.

Not only will the medical community benefit from the emergence of remote health monitoring, but coaches and athletes will be beneficiaries as well. The monitoring of heart rate, temperature, and blood pressure will assist coaches and athletes to monitor activity effectiveness of workouts and athletic performance. One of the ways both doctors and athletes will want to take advantage of is the monitoring of a patient’s Electrocardiogram (ECG).

Electrocardiography is a preferred way to monitor the activity of the heart. The ECG is used for diagnosing anomalies and monitors the behavior of the heart. In a typical ECG system, there are alarms to alert medical practitioners of potential cardiac threats
through the monitoring the electrical activity of the heart. When the heart beats, electrical signals are sent from the top of the heart muscle to the bottom as the muscles compress and contract. Electrodes are used to detect this ECG signal shown in Figure 1.1 [1]. The P wave represents the atrial contractions. In the ECG signal, the P Wave is small due to the small muscle mass of the atrium in the heart. The ventricular contractions are shown in the QRS complex. The large spike that represents this complex is due to the muscle mass present in the ventricles. The ST section of the ECG signal is when the ventricles have the same electric potential. This segment is used for certain diagnosis including heart attacks and cardiac ischemia. When the electrical signal is recharging for the next ventricular contractions, the T wave appears [2]. Different aspects of the ECG measurements are used for abnormality diagnosis. The duration of the PR interval, for example, could indicate the presence of pre-excitation syndromes such as Wolff-Parkinson White Syndrome or even Atrium Ventricle (AV) blockages. The QT intervals could also show the presence of Coronary Heart Disease [3].

Figure 1.1  Graphical representation of an ECG signal
The P-R interval extends from the beginning of the P-Wave to the beginning of the QRS complex. The typical healthy time duration of the P-R interval is between 120 to 200 milliseconds [2]. The QRS duration extends from the end of the P-R interval to the beginning of the T-wave. The healthy and typical time duration of the QRS complex is between 60 to 100 milliseconds. The S-T interval extends from the end of the QRS complex to the beginning of the T-wave. The time duration of the S-T interval is typically 80 milliseconds [4]. The T-Wave extends from the end of the S-T interval and is the last aspect of one period of an ECG signal. The typical time for a T-Wave is 160 milliseconds.

The electrode placement is essential to display the ECG signal. While there are many configurations and topologies for electrode lead placement, there are certain topologies that are commonly used. The most common configurations are Modified Chest Lead 1 (MCL₁), MCL₆, V₁-V₆, standard 3-lead placement, and 12-lead placement. These topologies are used for their strengths in monitoring different aspects of the ECG signal. MCL₁ and MCL₆ are commonly used to determine widespread QRS complex rhythms [5]. The standard 3-lead placement, MCL₁, and MCL₆ are a 3-lead placement system and are shown in Figure 1.2 [6].
According to Joan Gomez, the peak to peak voltage of a typical ECG signal (acquired by a 3 electrode topology) is approximately 0.5 mV [7]. This peak to peak voltage is determined by the height of the QRS complex since the QRS complex contains the highest voltage. The P-wave amplitude is approximately 12% of the maximum value of the QRS complex. The T-wave is approximately equal to or less than 8% of the maximum value of the QRS complex [6].

This EKG technology has gone through an innovative process. The ECG monitoring systems of the past were wired systems. The patient undergoing an ECG test was confined to the area in which the wires could extend from an ECG circuit placed within a component rack. Often these patients were confined to a bed or chair. New developments in computers and wireless networks allowed for the innovation of wireless monitoring stations with a central station where nurses and practitioners can monitor multiple patients and can be alerted to any potential life threatening changes in condition.
New innovations in component size allow the ECG circuit size to be reduced and even made portable. The Holter monitor [8], shown in Figure 1.3, is a modern, portable way an ECG is used by the Mayo Clinic.

![The Holter Monitor](image)

**Figure 1.3** The Holter Monitor.

The Holter monitor is a body centric device that reads an ECG over a period of 1 to 3 days and is then analyzed by the doctor at a later time for the presence of anomalies. This system can be worn around the neck, placed in the pocket, or be clipped to a belt. While this is a system currently used, it does not employ a wireless telemetry system for health monitoring.

Another device that allows for remote ECG monitoring is the CardioNet MCOT device. The CardioNet MCOT is a neck worn system and uses the standard 3-Lead electrode placement topology for the ECG. The ECG signal is then sent to a handheld
device wirelessly. This signal is then transmitted to the CardioNet Monitoring Center. If abnormalities are detected, the physicians in charge of monitoring are immediately notified and local emergency responders can be notified. The doctors can receive data from the patient continuously either through the internet or through fax. While an effective system for a first alert system, the custom handheld device adds to the cost of the overall system. While the MCOT utilizes smartphone technology to view the data collected, the system does not utilize existing handheld devices for data logging and data distribution.

The popularity of smartphones has increased over the past few years. According to Nielsen [9], smartphone ownership in the United States of America has increased from 17% to 30% among 55 to 64 year olds and from 41% to 62% among 25 to 34 year olds. This is shown below in Figure 1.4.
The surging popularity of smartphones and tablets has opened the door to new innovations in many different fields. This growing market leads to a growing market in E-health. This trend in technology also yields a growing field in at home monitoring and a potential for lower costs for future monitoring systems. These smartphones come equipped with wireless communication capabilities including Bluetooth and Wi-Fi connectivity. These capabilities allow for smartphones to be used a communication relay between the ECG and the internet.

Another shortcoming with current at home monitoring systems is the usability. One fear among patients is that a data collecting session could need to be restarted and the electrodes need to be replaced. While these companies offer tutorials and training on
how to setup the equipment, the fear remains in the patient of invalidating results due to
user error.

The purpose of this research is to develop a Bluetooth telemetry system for an
ECG that can be mounted on a garment. The garment will be color coated to match the
colors of the ECG probe snaps in order to relieve some of the fear on the part of the
patient of reapplying the electrodes. The goals of this research include: ECG circuit
design, telemetry circuit design, microcontroller coding, and Android application
development. The prototype is constrained to fit within an area of a 3 by 5 inches index
card to minimize user discomfort. The system must communicate using the Bluetooth 2.1
class, so that the device can communicate with a smart phone with an Android Operating
System. The Android application will control the data flow, display the ECG signal
received, and log the data into a file to be sent via email. The efficacy of this design is
tested with three volunteers. The results of the measured ECG of each volunteer are
shown.
CHAPTER II
SYSTEM MODEL

The purpose of this study is to develop a wearable telemetry system for an ECG that can be monitored with an Android smart phone through a Bluetooth wireless protocol. Figure 2.1 shows the proposed system model for the wireless and wearable ECG system.

Figure 2.1  Proposed System Model
The system model is comprised of three major sections:

1. The ECG Garment
2. The Wireless ECG Circuit
3. The Android Display Application

The ECG Garment consists of a short sleeve compression t-shirt altered to allow electrodes to be monitored through the fabric. The electrodes are clipped through the fabric to a cable that connects to the ECG circuit. The ECG Garment is also altered to hold the ECG circuit on the chest area.

The ECG circuit consists of two parts: the ECG amplifier and the Telemetry Unit. The ECG amplifier takes the input from the 3 electrodes, conditions and amplifies the ECG signal to be used by the Telemetry Unit, which samples and broadcasts the ECG signal over the Bluetooth protocol to the Android Phone.

The Android Phone takes the data provided over Bluetooth, and displays a modified version of the data. The Android phone also communicates back with the Telemetry Unit to turn on and off the data from the circuit.

The remaining sections of this chapter are organized by the technical subsystems.

1. The ECG Garment
2. The ECG Amplifier Circuit
3. The Telemetry Unit
4. The Android Application

These individual sections will discuss the design and desired performance of each respective subsystem.
2.1 The ECG Garment

The ECG Garment is comprised of an altered short sleeve t-shirt. The shirt has a section of industrial strength Velcro on the chest which allows for the Wireless ECG circuit to be held in place. This allows for the ECG circuit to be worn on the body with reduced discomfort. The t-shirt has been altered to have a color coordinated button holes placed for a 3-lead ECG topology. The Button holes are colored white, black and, red respectfully for each ECG lead. The ECG Garment is shown below in Figure 2.2.

![ECG Garment With Color Coordinated Lead Placement](image)

The 3-Lead topology, shown in Figure 1.2, is a standard lead placement for an ECG. There is another topology similar to this standard topology, shown in Figure 2.3, called the “Electrocardiogram Augmented Limb Lead [10].” This topology allows for the reference electrode in the center front abdomen, as opposed to the typical placement of
the reference electrode on either the left side of the abdomen or on the left leg. This allows the red electrode to be placed on the abdomen.

Figure 2.3  Electrocardiogram Augmented Limb Lead topology.

The leads used in conjunction with the ECG Garment are fed into the ECG circuit by means of an ECG lead to $1/8^{th}$ inch stereo plug.

2.2  Electrocardiogram Amplifier Circuit

The amplifier for an ECG application conditions and amplifies the voltage difference between the electrodes placed on the patient’s body. The circuit of the ECG amplifier is shown below in Figure 2.4.
The design of this circuit is based on the ECG example circuit provided by the AD620 Instrumentation Amplifier data sheet [11]. The circuit shown is comprised of 4 parts.

1. The Right Leg Drive Loop (RLDL)
2. The Instrumentation Amplifier
3. Filtering and Amplification Stage
4. The Analog-To-Digital Converter (ADC) compliance circuit

2.2.1 Right Leg Drive Loop

The spectrum analysis of a typical ECG signal was attained in the book *Advanced Methods and Tools for ECG Data Analysis* [12]. The signal and power amplitudes of the ECG signal versus frequency is shown below in Figure 2.5.
As seen in Figure 2.5, the majority of the ECG signal is located between 0 and 25Hz. While a Typical ECG contains P-Waves, T-Waves, and QRS complexes, raw ECG data also contains noise and voltage drift.

The Right Leg Drive Loop (RLDL) is used to reduce the voltage drift and noise that occurs. The noise that occurs is often noise produced by the patient or 60Hz noise from surrounding sources. Voltage drift is often caused by the motion of the patient while using the ECG. Figure 2.6 shows the typical drifting effects due to motion [13]. This drift is reduced through the use of a feedback loop connected between the reference (red) electrode and the gain resistor of the instrumentation amplifier.
Figure 2.6 Voltage drift due to motion

The relative power spectrum of a Typical ECG with noise and drift (motion artifact) is shown below in Figure 2.7 [14]. In this figure, the relative power of the total ECG is shown as well as individual components such as the QRS complex, muscle noise and motion artifact. Notice, that the motion artifact spectrum, when present, can distort the ECG signal such that the P-Wave and the T-Wave cannot be read and renders the ECG signal unusable. This is helped remedied through the RLDL.
The RLDL Implemented utilizes a filtering topology for an operational amplifier. The RLDL used in this study uses a capacitance value of 10µF and a resistance of 1MΩ, yielding a cutoff frequency of .02Hz. The RLDL schematic is shown below in Figure 2.8.
The output of the operational amplifier is connected to the reference (Red) electrode.

### 2.2.2 The Instrumentation Amplifier

The instrumentation amplifier, shown earlier in Figure 2.4, is the AD620. The AD620 is a low noise and low cost instrumentation amplifier. The right (white) electrode output is connected to with the negative terminal of the AD620 and the left (black) electrode output is connected to the positive terminal. The instrumentation amplifier is used as a differential operational amplifier and the gain of the amplifier is set by one resistor, $R_G$. The gain of the amplifier can be calculated by the equation shown below [11].

\[
G = \frac{49.4k\Omega}{R_G} + 1
\]

In the development stage, different resistors were used. The most stable gain resistor, $R_G$, was found to be 150 $\Omega$. This yields an amplifier gain, $G$, of approximately 330.

### 2.2.3 Filtering and Amplification Stage

The Filtering and Amplification Stage reduces noise and drift as well as amplifying the ECG signal to a readable level. From Figures 2.5 and 2.7, we can see that the majority of an ECG signal lies between 0 and 25 Hz. The voltage drift due to motion however, extends from 0 Hz to roughly 5 Hz. This means that filtering out the voltage drift due to motion, also filters out some low frequency components of the ECG signal.
This stage is implemented with three stages: two stages of passive filters and one stage of active filtering. The two stages of filtering are accomplished with two cascading RC high-pass filters. The RLDL stage reduced some noise and some voltage drift; not all noise and drift are eliminated. These added stages further stabilize the differential voltage from the output of the instrumentation amplifier. The first stage is a passive high-pass filter to reduce any frequency below 0.2 Hz. An RC circuit where R = 33kΩ and C = 220μF yields a cutoff frequency of 0.22 Hz. The second stage is also a passive high pass filter to reduce frequencies lower than 1 Hz. Another RC circuit is assembled where R = 1kΩ and C = 100μF yields a cutoff frequency of 1.6 Hz.

The frequency response of the combined passive filters is shown below in Figure 2.9.

![Figure 2.9 The Frequency Response of the Combined Passive Filters.](image)

The third stage of the Filtering and Amplification stage is an active low-pass filter. This active low pass filter amplifies the ECG signal to a readable level for the
Analog-to-Digital Converter of the microcontroller and attenuates some noise produced by surrounding power sources.

In this topology, $R_1 = 3.3\, \text{k}\Omega$, $R_2 = 51\, \text{k}\Omega$, and $C = 0.1\, \mu\text{F}$ yields a -3dB cut-off frequency of 31.2 Hz and a maximum gain $G = 16.45$. The frequency response of the non inverting low-pass filter topology is shown in Figure 2.11.
The combined frequency response of all three stages, both passive and active, is shown below in Figure 2.12. This active filter yields a maximum gain of 15.32. The Filter has a -3dB bandwidth of 33.54 Hz centered at 18.82 Hz.

![Gain vs. Frequency of the Filtering and Amplification Stage](image)

Figure 2.12  Gain vs. Frequency of the Filtering and Amplification Stage

### 2.2.4 Analog-To-Digital Converter Compliance Circuit

The Analog-To-Digital Converter (ADC) Compliance Circuit manipulates the output of the amplifier and conditions it to fit within the specification of the microcontroller’s ADC. The microcontroller used in this study is the ATMega328P. The microcontroller has a built in 10-bit ADC and is limited to reading voltages between 0 and the voltage supply $V_s$. Typical $V_s$ for this microcontroller must be between 3V and 5V for the chip to work. In this study, the microcontroller is powered by a regulated 3.3V and therefore the ECG signal that needs to be sampled must have a voltage range from 0 to 3.3V.

The output of the amplifier is fed through a coupling capacitor of $2.2\mu$F to center the ECG signal at 0V. This signal then was offset using a simple mixer topology with the
ECG signal sent through one port and a DC voltage from a voltage divider circuit in another. The offset needed was determined to be 1.5V to effectively remove the negative voltage aspect of the ECG. The output of the mixer circuit is an ECG signal centered at 1.5V. In order to make sure no extraneous negative voltages and voltages higher that 3.3V were present, the signal was fed through a rectifier diode and over a 3.3V Zener diode. The schematic for the ADC compliance circuit is shown below in Figure 2.13.

![Schematic of ADC Compliance Circuit](image)

**Figure 2.13** Analog –To-Digital Converter Compliance Circuit

### 2.3 Telemetry Unit

The role of the telemetry unit is to sample the analog ECG signal from the ECG Amplifier circuit, and to broadcast it through a Bluetooth 2.0 protocol to a paired smartphone or tablet. This section is divided into two sub-sections:
1. Telemetry System Hardware Overview

2. Telemetry Firmware Overview

The Telemetry System Hardware Overview sub-section covers the hardware overview and the description of the major components used. The Telemetry Firmware Overview covers the method in which the system operates including the system algorithm and code.

2.3.1 Telemetry System Hardware Overview

The Telemetry System uses two major components: a microcontroller and a Bluetooth module. Both components are connected through a communication port and are powered by 3.3V.

The microcontroller used in this study is the ATMega328p from Atmel. This microcontroller contains 32kB of flash program memory, 6 analog channel input with a 10-bit ADC, and UART, SPI and I²C communication capabilities [15].

The microcontroller is used for sampling and data packaging. It then sends data packets to the Bluetooth module through the RS232 Serial protocol on the UART port. The microcontroller is also connected to an LED at Pin 9 to communicate visually with the user and developer.

The Bluetooth module used in this study is the RN-42 Class 2.0 Bluetooth Module from Roving Networks. The RN-42 takes data from the UART port connected to the Atmel microcontroller. This data received is then modulated and broadcast to a Bluetooth connected device through the antenna included in the RN-42 package [16]. The RN-42 Bluetooth module is shown below in Figure 2.14.
Along with wireless communication with a Bluetooth enabled device and UART communication to other devices, the RN-42 also allows for visual confirmation to the developer and user. The RN-42 has two pins that communicate the status of the connectivity and status of the module. Pin 21 of the Bluetooth module is a status pin. This pin toggles at different frequencies to alert the user of what state the module is in. When the device has been successfully connected to a device, this pin is held low. In this study, pin 21 is tied to a red LED to alert the user of the status of the Bluetooth module. Pin 19 is a connection pin. This pin is held high when, and only when, the device is connected to a device through a peer-to-peer network and held low for all other times [16]. In this study, pin 19 is connected to a green LED. The hardware schematic of the Telemetry System is shown below in Figure 2.15.
2.3.2 Telemetry Firmware Overview

This section discusses the program that is implemented on the microcontroller. The ATMega328P is commonly used in the Arduino Development Board. Arduino Development Boards use a programming language known as Arduino Processing. The Arduino Uno R3 bootloader was uploaded to the ATMega328P so that the Arduino Processing Code could be uploaded to the microcontroller.

Arduino Processing is a C-like programming language used often with the Arduino class of development boards. The programming language needs two major functions to run instead of the one `main` function required in C programs. The two major functions needed are the `setup` and `loop` functions.

The `setup` function is executed at the beginning of each power on or reset of the microcontroller. Typically, the `setup` function is used to initialize variables, set digital outputs, analog inputs, and initialize communication speeds. After the `setup` function, the `loop` function is executed. This function is repeated at the end of every `loop` function iteration. In a typical `loop` function, the bulk of the embedded system code is executed.
including basic math calculations, sensor readings, and communicating with other devices and components.

In this study, the setup function sets an A0 (Pin 23) as an analog input, initializes a Boolean variable isOn to false, and initializes the UART serial port at the data speed 115200 baud (bps). While Bluetooth 2.0 and Bluetooth 2.1 allows for a data rate up to 1Mbps, the data speed of 115200 baud is required to communicate with Android devices.

The loop function checks if there is any new information on the UART port. If the data from the data port is equal to “ON”, the isOn variable is set to true; Otherwise isOn is set to false. If isOn is true, the analog voltage is read on the A0 pin. This integer ADC value is then changed to a c-string, and sent to the Bluetooth module through the UART port. The program is then delayed by a sampling period of 10ms (or sampling frequency of 100 Hz). If isOn is false, the analog voltage is not read and nothing is sent to the Bluetooth Module. The loop function is then repeated. Below in Figure 2.16 is the flowchart of the program on the ATMega328p.
2.4 Android Application

An Android device is used for the data display. Android is an operating system developed and maintained by Google. Free application development is provided through the Eclipse Development environment. A typical Android application is comprised of two main files: an XML file for the Graphical User Interface (GUI) and a Java program to control functionality. The XML file is used for the aesthetics of the Android application.
In this file, the layout of the GUI is established as well the positioning of “widgets” on the GUI. The XML file also establishes what tools are available to both the user and the Java program (e.g. Buttons, Sliders, Labels, etc). The Java program controls the functionality of the program, and controls not only what functions to call and how to calculations, but also how the program listens for changes from the GUI. The program, for example, can change its behavior based on a single tap and a double tap on a button.

The application developed is based largely on the Bluetooth Chat application provided in the Android Development environment known as Eclipse [17]. In the original Bluetooth chat, the user can send “Text Messages” to another paired device. One example this original application was used for was to text an Arduino Uno through the Bluetooth Protocol. If the user typed in “Hello World” into the text field and sent it, the Arduino would respond with the same message at a randomizes delay back to the phone.

The application used in this study uses a similar structure to the provided example by Eclipse, but encompasses more capabilities. The application has added features of extra push buttons and a graphical display. The android application turns on and off the ECG signal from the ECG circuit. When the Android application allows data to flow to the device, the application receives data and converts the data to voltage readings. This converted data is displayed in a real time and stored in a text file.

The purpose of the smart phone application to provide the user with a display for the ECG signals using an existing protocol. Through the Bluetooth peer-to-peer network, the Android Application receives data from the wearable ECG system. This application has 5 primary functions.
1. Data flow management
2. Data reception
3. Data manipulation
4. Data display
5. Data Logging

The application developed will control and accept a periodic dataflow from the ECG circuit. This flow will be controlled by the push of a button widget on the phone. The application will process the data and display it in the form of a real-time graph. The application will also log this data in a text file accessible on the phone that can be transferred at a later time by the user.

This section is divided into two subsections:

1. Graphical User Interface Overview
2. Performance Overview

2.4.1 Graphical User Interface Overview

The GUI links the user to the Java program. This part of the application can contain buttons, text input, sliders, and pictures to guide the user to use the program more efficiently. The design of the GUI developed in this study can be shown below in Figure 2.17.
The GUI contains 3 major sections:

1. GraphView Display
2. ON/OFF Toggle Button
3. Debugging Text Input

The Debugging Text Input is used for the ECG and phone communication. If extra functionality is needed to be added to the microcontroller on the system that requires an external signal, this input is allotted to the developer to help control this. Currently, if the user inputs “ON” the data flow will start or resume. Anything else, the data flow will stop or remain stopped.

The GraphView Display graphs the incoming signal. This display comes from the GraphView library developed by Jonas Gehring and is licensed under the GNU Lesser General Public License (LGPL) [18]. The GraphView library allows for real time line plots to be implemented and also allows for previously acquired data to be viewed by scrolling the data.
The ON/OFF Toggle Button is the GUI’s main way of controlling the Bluetooth data flow. When the data is flowing, the Button reads “OFF” and is a red color. This indicates that if the user wants to stop the data flow, he or she could press the red OFF button. When no data is flowing, the button is green and reads “ON.” This indicates that if the user wants to start the data flow, he or she could press the green ON button.

2.4.2 Performance Overview (Java code)

The Java program associated with the application is where the majority of the functionality of the application exists. This program interacts with the XML script and allows the pressing of a button to perform a particular action or calculation. The Java program focuses on three major functionalities:

1. Real Time Line Graph
2. Bluetooth Communication
3. Data Logging

The flowchart of the Java program implemented is shown below in Figure 2.18.
The application starts by checking if the Toggle Button has been pressed. If this button has been pressed, the Boolean value `mMessageState` is toggled from either true to false or false to true. If the new value of `mMessageState` is true, a new text file is created using the date, hour and minute for the name. A String value `msg` is set to equal “ON” and the Toggle Button in the GUI has a color set to red and a text set to “OFF.” If the new value of `mMessageState` is false, `msg` is set to “OFF,” the button text set to “ON,” and the button color is set to green. Whether `mMessageState` is set to true or false, when the button is toggled, `msg` is sent through Bluetooth through the function `sendMessage()`.
The `sendMessage(String str)` is a function that sends the String variable `str` over a Bluetooth peer-to-peer network. First the function checks to see if the Bluetooth enabled device is connected to another Bluetooth device. If there is no connection, the application alerts the user that there is no Bluetooth connection and terminates the `sendMessage` function. If there is a Bluetooth connection and if the length of the message `str` is greater than zero, `str` is converted into bytes and written to the `BluetoothChatService` variable `mChatService` variable with its internal function `write(bytes)`.

After checking the Toggle Button status, the application checks to see if the Debugging Text Input send button has been pressed or the return key has been pressed. If either the send button has been entered or the return button key has been pressed while text is in the Debugging Text Input, the application will send the text to the Bluetooth connected device through the function `sendMessage` function.

If data is present in the Bluetooth buffer, the Bluetooth Handler function will process the data. If there exists new numerical data and the `mMessageState` Boolean is true, the application reads the bytes available, and converts the data using the function `Convert(String str)`.

The values received from the ECG Circuit are integer values ranging from 0 to 1023. The `Convert` function takes these values and converts them to voltage levels. First the convert function takes the string value of `str` and converts it to a double value called `ADCvalue`. This value is then converted to a voltage value `VoltageValue` through the Equation shown below.

\[ VoltageValue = \frac{ADCvalue \times 3.3}{1023} \]  

Equation 2.2
If $VoltageValue$ is greater than 3.3, indicating that there was a data package error, $VoltageValue$ is set to the previous value variable $previousValue$. If this value is less than or equal to 3.3, $previousValue$ is set to be the $VoltageValue$. The Convert function then returns $VoltageValue$.

The value returned from the Convert function is then added to the GraphView line graph using the function appendData found in the GraphView library. This value is then written into the text file opened by the Toggle button. The format the data is written in is $hh:mm:ss.SSSz$, $VoltageValue$, where $hh$ represents hour of the day, $mm$ represents the minute of the hour, $ss.SSS$ represents the second (to the thousandth place) of the minute, and $z$ represents the time zone. When the process of adding information to the file is done, the algorithm is repeated by checking the status of the Toggle Button.
CHAPTER III
PROTOTYPE AND FINAL HARDWARE DESIGN

3.1 Prototype

The Prototype of the hardware was implemented on a solder-less breadboard for a proof of concept study. In this prototype, the ECG circuit was implemented with through-hole versions of the components used in the final design, including an 8 pin Dual In-Line Package (DIP) version of an instrumentation amplifier and operational amplifiers. A 28 pin DIP version of the ATMega328P was used as a data sampler and sent data over the UART port to a Bluetooth Mate Silver Bluetooth Module. The Bluetooth Mate Silver module utilizes an RN-42 module as used in the final design, but requires a 5V power source with a logic level shifter to convert 3.3V RS232 serial communication to 5V RS232 and vice versa.
The Android Application was implemented on a Sharp FX Plus Smartphone. While sufficiently displaying data, the phone is a low end model and is prone to crashing in normal use. Below in Figure 3.2, some results are gathered by the Sharp FX Plus.

Figure 3.1   Bread Boarded Prototype

Figure 3.2   Prototype Android application
3.2 Final Hardware Design

The final hardware design was implemented on a printed circuit board. The size of the circuit board is 1.85 in. by 2.90 in. This meets the constraints of fitting within the area of an index card. The circuit board is able to fit in a metal container. This allows the circuit board to be portable and fit within pant pockets. The weight of the circuit is 23.56 grams. This allows the circuit to be light enough to be worn on a t-shirt.

![Printed Circuit Board](image)

Figure 3.3 Printed Circuit Board

3.2.1 Power Supply

The circuit is powered by two 9 Volt batteries. The power to the board is controlled by a Dual Pole Throw Switch. One end of the switch connects the positive end of the Power supply to the positive voltage line of the printed circuit board; the other end of the switch connects the negative terminal of the power supply to the negative voltage line of the printed circuit board. When the switch is thrown, both the positive and negative terminals are routed to unused pins and the circuit is broken.
Two red LEDs are used to indicate the power is being provided. One LED is used for the positive voltage supply and the second is used for the negative power supply.

Both the microcontroller and the Bluetooth module are powered by 3.3V. This power is supplied by a 3.3V voltage regulator. The voltage regulator output is also tied to another red LED which indicates that power is being provided through the regulator.

The power supplies are shown below in Figure 3.3.

![Figure 3.3 Power Supplies provided on the Final Hardware Design](image)

### 3.2.2 Bluetooth

The Bluetooth Module, RN-42, comes with a prefabricated antenna with the module. According the RN-42 datasheet [16], the placement of the antenna over a ground
plane causes reduced connectivity due to the subsequent altering of the antenna impedance for Bluetooth Frequencies (2.402 to 2.48 GHz [16]).

The ground plane covers the majority of the PCB area on the bottom side exempting the outer edges and the area immediately beneath the RN-42 antenna as shown in Figure 3.5.

![Ground Plane Inset for RN-42 PCB placement.](image)

**Figure 3.5** Ground Plane Inset for RN-42 PCB placement.

### 3.3 Final System Overview

The entire system is shown below in Figure 3.6. The power supply is placed in a shoulder strap to provide for some the convenience of the user. The power supply is connected to the circuit through a three line interwoven cable. The red line is the positive voltage, the green line is the negative voltage, and the green line is the common
reference. Electrodes are connected to the user on the inner side of the garment and connected electrically to the ECG circuit through an electrode clip to eighth inch stereo jack. The ECG communicates to a Bluetooth enabled Android tablet.

Figure 3.6 Final System
CHAPTER IV
SYSTEM PERFORMANCE

In this chapter, the system performance is discussed. This chapter will see how the ECG signal is conditioned through filtering and amplifying. The rest of the chapter is organized as follows:

1. ECG Analog Circuit Performance
2. Android Application Performance

4.1 ECG Analog Circuit Performance

This subsection discusses how the ECG signal change as it passes through the ECG circuit. The ECG signal is determined by the Instrumentation Amplifier. The amplifier is set to a gain of 330 and yields an amplified differential voltage in the output of the amplifier. The output of the Instrumentation Amplifier is fed through the passive filtering stage and then fed through the active filtering stage. The output of this active filter is then sent through an AC coupling capacitor to get rid of any unintended DC offset. The output of this capacitor is then fed through the simple mixer to add an intentional DC offset to the ECG signal for the ADC. The output of the mixer is fed through a rectifier diode and is clipped at 3.3V by a Zener diode.
This subsection is divided as follows:

1. Instrumentation Amplifier Output
2. Passive Filter Output
3. Active Filter Output
4. AC Coupling Capacitor Output
5. Mixer Output

4.1.1 Instrumentation Amplifier Output

The output of the Instrumentation Amplifier is shown below in Figure 4.1.

![ECG Output of the Instrumentation Amplifier](image)

Figure 4.1 ECG Output of the Instrumentation Amplifier
As shown a typical ECG signal from the output of the Instrumentation Amplifier has a peak-to-peak output of approximately 340 mV. One Aspect shown in Figure 4.1 is the drift due to motion. The voltage signal from the ECG swings upward as the user’s arms move in most. In this example the voltage swing moved up as the user clicked on the Stop button on the oscilloscope to capture the data. The data swings from a center voltage of -1.5V to a positive 1.5V depending on how the user is moving.

4.1.2 Passive Filter Output

The Passive filter is to help assist the RLDL with the added drift in the system. In order to accomplish this, the filters used are high-pass filters. The output of the filters is shown below in Figure 4.2.

Figure 4.2 ECG Output of the Passive Filters
As shown, much of the DC and lower frequencies are filtered out while higher frequencies remain. The next stage will help alleviate some of the distortion due to higher frequencies and the noise due to AC power sources apart from the ECG circuit.

4.1.3 Active Filter Output

The active filter provides two purposes. The first purpose is to filter out the noises due to the AC power sources and any noise caused by the body of the user. The second purpose is to amplify the ECG signal so the ECG signal can be read by the ADC. The Filtering and Amplification Stage attenuates out very low frequencies and AC power supply frequencies and amplifies the frequencies between 2 Hz and 36 Hz. Due to the nature of the voltage drift noise, some frequencies associated with the P-Wave and T-Wave are filtered out. The relative amplitudes of the ECG signal and the Filter can be seen below in Figure 4.3. The relate amplitudes are shown to show how similar the frequency responses are between the ECG signal and the filter output.
Figure 4.3  Relative Amplitudes of the Filtering and Amplification Stage and the ECG signal

Below is the ECG signal at the output of the Active Filter.

Figure 4.4  ECG Output of the Active Filter
The output of the Active Filter produces an ECG signal with a peak-to-peak voltage of 2.64 Volts. The majority of these voltages, however the majority of the signal is below the zero voltage level. There also exists a small amount of drift. The output of the AC coupling capacitor alleviates some of this burden.

### 4.1.4 AC Coupling Capacitor Output

The output of the AC coupling capacitor alleviates some of the burden of the voltage drift. The output of the AC coupling capacitor is shown in Figure 4.4 below.

![ECG Output of the AC Coupling Capacitor](image)
The output of the AC Coupling Capacitor varies between 2.4V and -200mV. While a good output of the amplifier, some negative voltages remain. This is taken care of by the Mixer stage which adds a small voltage so that no negative voltages remain.

4.1.5 Mixer Output

The output of the Mixer is shown in Figure 4.5. While the peak-to-peak voltage level has been reduced, the minimum voltage has been increased to approximately 800mV.

Figure 4.6 ECG Output of the Mixer
4.1.6 Analog to Digital Converter Compliance Output

While an overwhelming majority of the voltage is above the 0V level, any unforeseen circumstances that lead to a negative voltage could result in ruining the ACD of the ATMega328. In order to assure no negative voltages leak through, an ADC Compliance Circuit is applied to the voltage. This circuit consists of 2 parts. The first part is a rectifier diode to clip off any negative voltages. The second part is a Zener Diode to clip any voltages over 3.3V. The output of the ADC Compliance Circuit is shown below in Figure 4.6.

![ECG Output of the ADC Compliance Circuit](image)

Figure 4.7 ECG Output of the ADC Compliance Circuit
4.2 Android Application Performance

This section discusses the performance of the Android application. The system was connected to a volunteer and the system was initiated. The system was able to connect to the Bluetooth Module and was able to initiate data flow. When the data came into the Tablet’s buffer, the data was displayed and stored in a text file. This text file was saved onto the tablets hard drive and was able to be sent through the onboard email app available on a Samsung Galaxy Tab II tablet. Below in Figure 4.7 are some results of the Android Application test.

![Android Application System Test Results](image)

Figure 4.8 Android Application System Test Results

The system test was repeated with two extra volunteers and similar results were seen.

This system was tested on two Android Devices: the Sharp FX plus smartphone and the Samsung Galaxy Tab II. When implemented on the Sharp FX plus smartphone, the data amount of data received at a sampling frequency at 100 Hz would cause the application on the device to freeze or crash. This was remedied through two methods. The first method was to increase the sampling period to 20 ms (sampling frequency of
50Hz). This remedy reduced the signal quality of the ECG signal. The second remedy was to upload the application to a platform of higher quality and Random Access Memory (RAM). The application was downloaded to a Samsung Galaxy Tab II and the application allowed for the required 100Hz frequency.
CHAPTER V
CONCLUSION AND FUTURE WORK

In conclusion, a wearable Bluetooth telemetry system for an ECG was designed and tested. The study shows that an ECG system can be designed that can monitor the heart activity of a person and can be monitored by the means of a smartphone or tablet through Bluetooth.

Future work will consist of adding more functionality to the Android application, hardware improvements and additional sensors. The improvements to the Android application should include monitoring battery voltage levels. Hardware improvements should include miniaturizing the circuit further, one battery power supply, and packaging to hide and protect the circuit from the outside elements. Another aspect where hardware should be improved is exploring other wireless protocols available on smartphones and tablets. One such protocol is the Bluetooth 3.0 protocol. Bluetooth 3.0 is designed for audio transmission and can transmit data up to 24Mbit/s. Another such protocol that needs to be explored is the Bluetooth Low Energy (BLE) protocol. BLE allows for the communication to a wireless device using less energy and allows for longer functional times for the Bluetooth ECG device. Extra health sensors including temperature, humidity, and blood pressure should also be added into the system to add functionality of the whole system. The development of a Bluetooth ECG device should be explored to work for iOS devices including iPhone and iPad.
REFERENCES


APPENDIX A

PROGRAMMING CODE
A.1 BLUETOOTHCHAT.JAVA

The following program code is adapted from the BluetoothChat provided as an open source example program by The Android Open Source Project on the Java development environment Eclipse [17].

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

package com.example.android.BluetoothChat;

import java.io.File;
import java.io.FileOutputStream;
import java.io.OutputStreamWriter;
import java.text.SimpleDateFormat;
import java.util.Calendar;
import java.util.Date;

import com.example.rtgapr1.GraphView;
import com.example.rtgapr1.GraphViewSeries;
import com.example.rtgapr1.LineGraphView;
//import com.example.rtgapr1.R;

import android.app.Activity;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.content.Intent;
import android.graphics.Color;
import android.os.Bundle;
import android.os.Handler;
import android.os.Message;
import android.util.Log;
import android.view.KeyEvent;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.view.Window;
import android.view.View.OnClickListener;
import android.view.inputmethod.EditorInfo;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.EditText;
import android.widget.LinearLayout;
import android.widget.ListView;
import android.widget.TextView;
import android.widget.Toast;

/**
 * This is the main Activity that displays the current chat session.
 */
public class BluetoothChat extends Activity {
    public String time[];
    public Double Ekg[];
    public int index=0;
    public String FilePrefix = "ECG";
    public String FileName="";

    //File IO Stuff
    FileOutputStream fOut;
    OutputStreamWriter myOutWriter;

    // Debugging
    private static final String TAG = "BluetoothChat";
    private static final Boolean D = true;

    // Message types sent from the BluetoothChatService Handler
    public static final int MESSAGE_STATE_CHANGE = 1;
    public static final int MESSAGE_READ = 2;
    public static final int MESSAGE_WRITE = 3;
    public static final int MESSAGE_DEVICE_NAME = 4;
    public static final int MESSAGE_TOAST = 5;

    // Key names received from the BluetoothChatService Handler
public static final String DEVICE_NAME = "device_name";
public static final String TOAST = "toast";

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

// Layout Views
private TextView mTitle;
private ListView mConversationView;
private EditText mOutEditText;
private Button mSendButton;
private Button mToggle;

// Name of the connected device
private String mConnectedDeviceName = null;
// Array adapter for the conversation thread
private ArrayAdapter<String> mConversationArrayAdapter;
// String buffer for outgoing messages
private StringBuffer mOutStringBuffer;
// Local Bluetooth adapter
private BluetoothAdapter mBluetoothAdapter = null;
// Member object for the chat services
private BluetoothChatService mChatService = null;

// Global Variables
public Boolean mMessageState = false; // State for BT data communication (True->Flowing)[False->Stopped]
int data_length = 0;
String previous1 = "0";
String OrgBuffer = "";
LinearLayout layout;
GraphView graphView; /*= new LineGraphView(
    this // context
    , "GraphViewDemo" // heading
);*/
public GraphViewSeries exampleSeries;
// public Date time[];
// public Double EKG[];

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    if (D) Log.e(TAG, "+++ ON CREATE +++");
}
exampleSeries = new GraphViewSeries(new GraphView.GraphViewData[] {
    new GraphView.GraphViewData(1, 1.0d)
    , new GraphView.GraphViewData(2, 1.0d)
    , new GraphView.GraphViewData(3, 1.0d)
    , new GraphView.GraphViewData(4, 1.0d)
});

data_length = 4;

//LinearLayout layout = (LinearLayout)
findViewById(R.id.gview); //layout.addView(graphView);

// Set up the window layout
requestWindowFeature(Window.FEATURE_CUSTOM_TITLE);
setContentView(R.layout.main);
getWindow().setFeatureInt(Window.FEATURE_CUSTOM_TITLE, R.layout.custom_title);

// Set up the custom title
mTitle = (TextView) findViewById(R.id.title_left_text);
mTitle.setText(R.string.app_name);
mTitle = (TextView) findViewById(R.id.title_right_text);

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

// If the adapter is null, then Bluetooth is not supported
if (mBluetoothAdapter == null) {
    Toast.makeText(this, "Bluetooth is not available", Toast.LENGTH_LONG).show();
    finish();
    return;
}

@Override
public void onStart() {
    super.onStart();
    if(D) Log.e(TAG, "++ ON START ++");
}
// If BT is not on, request that it be enabled.
// setupChat() will then be called during onActivityResult
if (!mBluetoothAdapter.isEnabled()) {
    Intent enableIntent = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
    startActivityForResult(enableIntent, REQUEST_ENABLE_BT);
    // Otherwise, setup the chat session
} else {
    if (mChatService == null) setupChat();
}

@Override
public synchronized void onResume() {
    super.onResume();
    if (D) Log.e(TAG, "+ ON RESUME +");

    // Performing this check in onResume() covers the case in which BT was
    // not enabled during onStart(), so we were paused to enable it...
    // onResume() will be called when ACTION_REQUEST_ENABLE activity returns.
    if (mChatService != null) {
        // Only if the state is STATE_NONE, do we know that we haven't started already
        if (mChatService.getState() == BluetoothChatService.STATE_NONE) {
            // Start the Bluetooth chat services
            mChatService.start();
        }
    }
}

private void setupChat() {
    Log.d(TAG, "setupChat()");
    graphView = new LineGraphView(
        this // context
        , "GraphViewDemo" // heading
    );
    //Initialize Graph Area
    graphView.addSeries(exampleSeries); // data
    //graphView.setScalable(true);
    graphView.setViewPort(1, 200);
    //graphView.
graphView.setScrollable(true);
graphView.setManualYAxisBounds(3.0, 0.0);

//graphView.setScalable(true);
layout = (LinearLayout) findViewById(R.id.gview);
layout.addView(graphView);

// Initialize the array adapter for the conversation thread
mConversationArrayAdapter = new ArrayAdapter<String>(this, R.layout.message);
//mConversationView = (ListView) findViewById(R.id.in);
//mConversationView.setAdapter(mConversationArrayAdapter);

// Initialize the compose field with a listener for the return key
mOutEditText = (EditText) findViewById(R.id.edit_text_out);
mOutEditText.setOnEditorActionListener(mWriteListener);

// Initialize the send button with a listener that for click events
mSendButton = (Button) findViewById(R.id.button_send);
mSendButton.setOnClickListener(new OnClickListener() {
    public void onClick(View v) {
        // Send a message using content of the edit text widget
        TextView view = (TextView) findViewById(R.id.edit_text_out);
        String message = view.getText().toString();
        sendMessage(message);
    }
});

// Initialize a toggle Button to start and stop Data Flow
mToggle = (Button) findViewById(R.id.toggle);
mToggle.setOnClickListener(new OnClickListener() {
    public void onClick(View v) {
        mMessageState = !mMessageState;
        String msg;

        if (mMessageState)
        {
            String fPath;
            String FileDate;
            SimpleDateFormat fFormatter = new SimpleDateFormat("MM_dd_yyyy_hh-mm-ss");

            fPath = "F:\testFile.txt";
            FileDate = new Date().toString();
            fPath = fPath + FileDate;

            try {
                File file = new File(fPath);
                if (!file.exists())
                    file.createNewFile();

                FileWriter fw;
                fw = new FileWriter(fPath, true);
                fw.write(message + System.getProperty("line.separator");
                fw.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
});
try {
    //FileName = fPath;
    File myFile = new File(fPath);
    myFile.createNewFile();
    fOut = new FileOutputStream(myFile);
    OutputStreamWriter myOutWriter = new OutputStreamWriter(fOut);
    myOutWriter.append(FileName + 

    myOutWriter.append("\n");
    myOutWriter.append("\n");
    myOutWriter.close();
    fOut.close();
    Toast.makeText(getBaseContext(),
            "Writing SD"+FileName+"",
            Toast.LENGTH_SHORT).show();
} catch (Exception e) {
    Toast.makeText(getBaseContext(), e.getMessage(),
            Toast.LENGTH_SHORT).show();
}

msg = "ON";
msg = "OFF";
msg = "ON";
msg = "OFF";
// Initialize the BluetoothChatService to perform bluetooth connections
mChatService = new BluetoothChatService(this, mHandler);

// Initialize the buffer for outgoing messages
mOutStringBuffer = new StringBuffer("");

@Override
public synchronized void onPause() {
    super.onPause();
    if(D) Log.e(TAG, "- ON PAUSE -");
}

@Override
public void onStop() {
    super.onStop();
    if(D) Log.e(TAG, "-- ON STOP --");
}

@Override
public void onDestroy() {
    super.onDestroy();
    // Stop the Bluetooth chat services
    if (mChatService != null) mChatService.stop();
    if(D) Log.e(TAG, "--- ON DESTROY ---");
}

private void ensureDiscoverable() {
    if(D) Log.d(TAG, "ensure discoverable");
    if (mBluetoothAdapter.getScanMode() !=
        BluetoothAdapter.SCAN_MODE_CONNECTABLE_DISCOVERABLE) {
        Intent discoverableIntent = new Intent(BluetoothAdapter.ACTION_REQUEST_DISCOVERABLE);
        discoverableIntent.putExtra(BluetoothAdapter.EXTRA_DISCOVERABLE_DURATION, 300);
        startActivity(discoverableIntent);
    }

    /**
     * Sends a message.
     * @param message  A string of text to send.
     */
}
private void sendMessage(String message) {
    // Check that we're actually connected before trying anything
    if (mChatService.getState() != BluetoothChatService.STATE_CONNECTED) {
        Toast.makeText(this, R.string.not_connected, Toast.LENGTH_SHORT).show();
        return;
    }

    // Check that there's actually something to send
    if (message.length() > 0) {
        // Get the message bytes and tell the BluetoothChatService to write
        byte[] send = message.getBytes();
        mChatService.write(send);

        // Reset out string buffer to zero and clear the edit text field
        mOutStringBuffer.setLength(0);
        mOutEditText.setText(mOutStringBuffer);
    }
}

// The action listener for the EditText widget, to listen for the return key
private TextView.OnEditorActionListener mWriteListener =
    new TextView.OnEditorActionListener() {
        public Boolean onEditorAction(TextView view, int actionId, KeyEvent event) {
            // If the action is a key-up event on the return key, send the message
            if (actionId == EditorInfo.IME_NULL && event.getAction() ==
                KeyEvent.ACTION_UP) {
                String message = view.getText().toString();
                sendMessage(message);
            }
            if(D) Log.i(TAG, "END onEditorAction");
            return true;
        }
    };
mConversationArrayAdapter.clear();

break;

case BluetoothChatService.STATE_CONNECTING:
    mTitle.setText(R.string.title_connecting);
    break;

case BluetoothChatService.STATE_LISTEN:
    break;

case BluetoothChatService.STATE_NONE:
    mTitle.setText(R.string.title_not_connected);
    break;
}

break;

case MESSAGE_WRITE:
    byte[] writeBuf = (byte[]) msg.obj;
    // construct a string from the buffer
    String writeMessage = new String(writeBuf);
    mConversationArrayAdapter.add("Me: " + writeMessage);
    break;

case MESSAGE_READ:
    byte[] readBuf = (byte[]) msg.obj;
    // construct a string from the valid bytes in the buffer
    String readMessage = new String(readBuf, 0, msg.arg1);
    //

    //Add Unpacking function: EX-> (String)Unpack(String orig);

    //
    String num_str = Unpack(readMessage);

    if(isNumeric(num_str))
    {
        //CONVERT DATA
        num_str = String.valueOf(Convert(num_str));
        //PUT DATA INTO GRAPH
        //exampleSeries.
        GraphView.GraphViewData data = new
        GraphView.GraphViewData(data_length+1, Double.valueOf(num_str));
        exampleSeries.appendData(data, true);
        data_length++;

        try
        {
            if(FileName != ")")
            {
                File myFile = new File("/sdcard/"+FileName);
                fOut = new FileOutputStream(myFile,true);
            }
        }
    }
myOutWriter = new OutputStreamWriter(fOut);

SimpleDateFormat dFormat = new SimpleDateFormat("hh:mm:ss.SSSz");

myOutWriter.append(dFormat.format(Calendar.getInstance().getTime()) + ',

myOutWriter.append(num_str+"n";
myOutWriter.close();
fOut.close();
//Toast.makeText(getBaseContext(),"Appended
"+FileName+"",Toast.LENGTH_SHORT).show();

public void onActivityResult(int requestCode, int resultCode, Intent data) {
    if(D) Log.d(TAG, "onActivityResult " + resultCode);
    switch (requestCode) {
        case MESSAGE_DEVICE_NAME:
            // save the connected device's name
            mConnectedDeviceName = msg.getData().getString(DEVICE_NAME);
            Toast.makeText(getApplicationContext(), "Connected to "
                    + mConnectedDeviceName, Toast.LENGTH_SHORT).show();
            break;
        case MESSAGE_TOAST:
            Toast.makeText(getApplicationContext(), msg.getData().getString(TOAST),
                        Toast.LENGTH_SHORT).show();
            break;
        }
    }
}
case REQUEST_CONNECT_DEVICE:
    // When DeviceListActivity returns with a device to connect
    if (resultCode == Activity.RESULT_OK) {
        // Get the device MAC address
        String address = data.getExtras()
            .getString(DeviceListActivity.EXTRADEVICEADDRESS);
        // Get the BluetoothDevice object
        BluetoothDevice device = mBluetoothAdapter.getRemoteDevice(address);
        // Attempt to connect to the device
        mChatService.connect(device);
    }
    break;

case REQUEST_ENABLE_BT:
    // When the request to enable Bluetooth returns
    if (resultCode == Activity.RESULT_OK) {
        // Bluetooth is now enabled, so set up a chat session
        setupChat();
    } else {
        // User did not enable Bluetooth or an error occurred
        Log.d(TAG, "BT not enabled");
        Toast.makeText(this, R.string.bt_not_enabled_leaving,
            Toast.LENGTH_SHORT).show();
        finish();
    }
}

public String Unpack(String original)
{ /*
    int index1 = original.indexOf("(");
    int index2 = original.indexOf")(",0);

    int index,openIndex,closeIndex;
    openIndex = closeIndex = -1;
    index = 0;

    char[] charArray = original.toCharArray();

    while(index <charArray.length)
    {
        if(charArray[index] == '(')
            openIndex = index;
        if (charArray[index] == ')
            closeIndex = index;

        if(openIndex >=0 && closeIndex >=0)
break;

    index++;
    }

if(openIndex <0 || closeIndex <0)
{
    // If the data stream is incomplete add the buffer and see if it makes
    complete data
    OrgBuffer = OrgBuffer + original;
    charArray = OrgBuffer.toCharArray();

    openIndex = closeIndex = -1;
    index = 0;

    while(index <charArray.length)
    {
        if(charArray[index] == '(')
            openIndex = index;
        if (charArray[index] == ')')
            closeIndex = index;

        if(openIndex >=0 && closeIndex >=0 && closeIndex > openIndex)
            break;

        index++;
    }

if(openIndex >=0 && closeIndex >=0 && closeIndex > openIndex)
{
    previous1 = OrgBuffer.substring(openIndex+1, closeIndex);
    OrgBuffer = "";
    //return "1O: "+Integer.toString(openIndex)+" C: "+Integer.toString(closeIndex) +" "+ previous1;
    return previous1;
}
else
{
    //return "2O: "+Integer.toString(openIndex)+" C: "+Integer.toString(closeIndex) +" "+ previous1;
    return previous1;
}
previous1 = original.substring(openIndex+1, closeIndex); */
//return "O: " + Integer.toString(openIndex) + " C: " + Integer.toString(closeIndex) +
" " + original.substring(openIndex+1, closeIndex);
//return previous1;

return original;

}

double Convert(String str)
{
    double value = Double.valueOf(str);
    //if(ADC_bool && !Volt_bool && !Temp_bool){
    //} //else if (Volt_bool && !ADC_bool && !Temp_bool){

    value = value*5/1023;

    if(value>3.3)
        value=Double.valueOf(previous1);
    else if(value <.5)
        value=Double.valueOf(previous1);
    //}

    previous1 = String.valueOf(value);
    return value;
}

public static Boolean isNumeric(String str)
{
    try
    {
        double d = Double.parseDouble(str);
    }
    catch(NumberFormatException nfe)
    {
        return false;
    }
    return true;
}

@Override
public Boolean onCreateOptionsMenu(Menu menu) {
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.option_menu, menu);
    return true;
}

@Override
public Boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
    case R.id.scan:
        // Launch the DeviceListActivity to see devices and do scan
        Intent serverIntent = new Intent(this, DeviceListActivity.class);
        startActivityForResult(serverIntent, REQUEST_CONNECT_DEVICE);
        return true;
    case R.id.discoverable:
        // Ensure this device is discoverable by others
        ensureDiscoverable();
        return true;
    }
    return false;
}

A.2 MAIN.XML

The following program code is adapted from the BluetoothChat provided as an open source example program by The Android Open Source Project on the Java development environment Eclipse [17].

<?xml version="1.0" encoding="utf-8"?>
<!-- Copyright (C) 2009 The Android Open Source Project

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software

68
distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
>
    <LinearLayout
        android:id="@+id/gview"
        android:layout_width="match_parent"
        android:layout_height="164dp"
        android:orientation="vertical"
    />

    <ListView
        android:id="@+id/in"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_weight="1"
        android:stackFromBottom="true"
        android:transcriptMode="alwaysScroll"
    />

    <LinearLayout
        android:orientation="horizontal"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
    >
        <Button
            android:id="@+id/save"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Save" />
        <EditText android:id="@+id/edit_text_out"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
        />
    </LinearLayout>


A.3 EKG_THESIS.ino

boolean sendData;
int Apin = A0;
void setup()
{

digitalWrite(13,LOW);
Serial.begin(115200);
}

void loop()
{

char buf[100] = "";

int i;
String buffer;
if (Serial.available())
{
    i = 0;
    while(Serial.available() && i < 20)
    {
        buf[i] = Serial.read();
        i++;
        delay(10);
    }
Serial.println(buf);
buffer = buf;
if(buffer == "ON")
{
    sendData = true;
    digitalWrite(13,HIGH);
}
else
{
    sendData = false;
    digitalWrite(13,LOW);
}

int val;
String temp;
if(sendData)
{
    val = analogRead(Apin);
    sprintf(buf,"%d",val);
    Serial.println(buf);
}

delay(10);