Acknowledgements:
We would like to thank the following people for their advice, tools, friendship, and Inspiration to complete this project:
Xuefu Zhou – Everly James - Elvin Stepp - Dave Tashjian
This document is a partial fulfillment of the Requirements for the degree of Bachelor of Science In Electrical or Computer Engineering Technology At the College of Engineering & Applied Science
By TIDIANE DIABY and CHRIS HARKINS Bachelor of Science University of Cincinnati April 2013
Faculty Advisor: XUEFU ZHOU
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABSTRACT</td>
<td>4</td>
</tr>
<tr>
<td>INTRODUCTION</td>
<td>5</td>
</tr>
<tr>
<td>Problem</td>
<td>5</td>
</tr>
<tr>
<td>Solution</td>
<td>5</td>
</tr>
<tr>
<td>CREDIBILITY</td>
<td>6</td>
</tr>
<tr>
<td>GOALS AND METHODOLOGY</td>
<td>6</td>
</tr>
<tr>
<td>DISCUSSION</td>
<td>8</td>
</tr>
<tr>
<td>DESIGN OBJECTIVES</td>
<td>8</td>
</tr>
<tr>
<td>Discrete Hardware</td>
<td>8</td>
</tr>
<tr>
<td>Robust Hardware</td>
<td>9</td>
</tr>
<tr>
<td>Long battery life</td>
<td>9</td>
</tr>
<tr>
<td>Ease of use</td>
<td>10</td>
</tr>
<tr>
<td>Use in many different applications</td>
<td>10</td>
</tr>
<tr>
<td>TECHNICAL APPROACH</td>
<td>11</td>
</tr>
<tr>
<td>Hardware</td>
<td>11</td>
</tr>
<tr>
<td>Analog front end</td>
<td>12</td>
</tr>
<tr>
<td>Microcontroller</td>
<td>12</td>
</tr>
<tr>
<td>Bio-Potential Signal Acquisition</td>
<td>13</td>
</tr>
<tr>
<td>Memory Card</td>
<td>14</td>
</tr>
<tr>
<td>Wireless communication</td>
<td>15</td>
</tr>
<tr>
<td>Mcp73833 (battery charger)</td>
<td>16</td>
</tr>
<tr>
<td>Smaller size</td>
<td>17</td>
</tr>
<tr>
<td>Software</td>
<td>18</td>
</tr>
<tr>
<td>Bluetooth ECG Monitor</td>
<td>18</td>
</tr>
<tr>
<td>Android APP User Interface</td>
<td>18</td>
</tr>
<tr>
<td>Enabling Bluetooth Pictures</td>
<td>19</td>
</tr>
<tr>
<td>Data collection</td>
<td>20</td>
</tr>
<tr>
<td>Data interpretation</td>
<td>20</td>
</tr>
<tr>
<td>Calibration</td>
<td>21</td>
</tr>
<tr>
<td>Noise</td>
<td>21</td>
</tr>
</tbody>
</table>
ABSTRACT

The result of this project is an off-the-shelf Android Bluetooth ECG monitoring device that communicates wirelessly to a computer or smartphone. There were two parts of the project that was designed, the user interface software and the wireless ECG monitoring device. The user interface is a software program for a computer and/or smartphone that controls the wireless ECG device function, set user options, and displays the data results. The wireless ECG monitoring device has components that allow for sensing heart electrical activity, signal conditioning, data conversion, data storage and wireless communication. Software code was written and tested on computer and smartphone as well as the wireless ECG monitoring device. Writing the software code took about three months to be complete and function perfectly. While writing the software code, the design for the wireless ECG monitoring device component placement and enclosure were designed, which took about two months to complete.

Figure 1 Typical Screen-shots of ECG monitoring
INTRODUCTION

This report outlines the process we followed while researching, designing, and constructing the Android Bluetooth ECG Monitor.

Problem

Heart disease causes almost 25% of all deaths in the United States each year but it is something that’s detectable and preventable if caught early on. Mobile ECG monitoring devices used as preventive prognoses already exist (e.g. Holter Monitors) but they are unwieldy and require seeing a doctor before being able to use them, which limits how often a person can use them as well as a person’s ability to get one at all. After a day or two of monitoring, the patient takes the unit back and the data is then analyzed. A person spends days waiting for results when potential problems can happen at any time during monitoring. Using an app on a smartphone and/or a computer that communicates with a wireless ECG monitoring device will allow a person to see the results immediately. The smartphone will not only display the heart activity but also save the data to be used for further analysis at any time as well as informing the user of any current problems that may or may not need to be addressed immediately.

Solution

The Android Bluetooth ECG monitor device detects heart electrical activity on the surface of the human body and display it on a screen. Not only this device automatically identifies anomalies in heart-rate, but also alerts contacts in seconds which allows help and treatment quickly. An integrated wireless chip-set, such as Bluetooth and/or ZigBee, was used to wirelessly transfer the sensor data to a computer and/or smartphone post-processing device. It also serves as a
testing platform to evaluate multiple types of wireless systems, and the differences between types of sensors. With a more comfortable monitoring system, it would be possible to monitor the heart of elderly or sick people continuously, while they continue with normal life. It also allows for personal use in the home when an individual wishes to check their heart, much like when people measure their blood sugar, pulse, or temperature. With this wireless device, people could log their day, and analyze the data later, making it possible for individuals to go about their normal routine and therefore produce more accurate stress readings.

CREDIBILITY

Tidiane Diaby & Chris Harkins

We have experience with developing and writing software programs that detect signals from measurement devices and convert it into useful data to be stored. Classes required and taken in the Electrical Engineering Technology program has built a strong foundation for circuit design, writing software, and signal processing. Prior work experience at a company that designed and manufactured medical equipment has been experienced, which allowed for familiarity with medical technology design.

GOALS AND METHODOLOGY

- Sense body electrical activity: Must be able to read heart electrical activity from the body. The purpose of this is to read heart electrical activity and determine abnormalities for preventive medical purposes.
• **Evaluate Various Body Electrodes:** In order to establish the best solution for a clean signal and have modules that are the least intrusive as possible, we evaluated all possible sensor types. We’ve analyzed "wet" and "dry" sensors and determined the best solution for our design.

• **Use off the shelf components:** To insure manufacturer friendly construction and cost efficiency we used off the shelf (non-custom) components as well as some readily available and manufactured analog components for amplification and signal conditioning. We’ve used an off the shelf microcontroller (i.e. PIC24FJGB004). We’ve will use off the shelf wireless transceivers to transfer information to local base station (i.e. Bluetooth, ZigBee, 802.11).

• **Processing of Information:** All information was processed eventually in order to make the data readable and user-friendly.

Information that was given to the base station (microcontroller) needed algorithms to detect heart problems from data.

• **Evaluation of Wireless Protocols:** Given the vast extent of protocols available currently, we’ve evaluated different wireless transceivers to determine energy consumption, reliability, throughput, bandwidth, etc. We’ve found the best fit for our product to be a Class2 Bluetooth.

• **Must be low Power:** Given that this will be a wireless system, the device has an independent power source. This power source was able to last long enough to validate the need for it to be unobtrusive. In other words, it should last long enough that a 15min doctor visit is less appealing than using our product.

• **Must be light weight:** To insure patient comfort, the modules should be the least intrusive as possible in weight. Our device should be light weight as to be comfortable for the patient.

• **Must be small in size:** To insure patient comfort, each module should be the least intrusive as possible in size. Our device should be small as to limit interference with the patient.
DISCUSSION

Project Concept

The concept of Wireless ECG Monitoring Device is; the monitor is connected to a patient by adhesive patches. The monitor records patient data and transmits. The signal is captured by an

After looking at similar systems it became apparent that there are many products that partially fulfill our design requirements, however none of them meet all of the required components of this project. Several systems, such as PiiX, BMS, and MEMSWear, have wireless sensor nodes but none of those were capable of displaying data on a smartphone. There were other devices like the LifeSync Wireless ECG that could do a large number of sensor nodes, but all of those systems display data on a PC through wires.

The wireless protocols used were varied across all the standard methods, though most utilized the 2.4GHz band. All of the wireless devices were also battery powered, most did not reveal what kind of battery or how long the device could run.

DESIGN OBJECTIVES

Discrete Hardware

Our device should be small as to limit interference with the patient. To insure patient comfort, each module should be the least intrusive as possible. In an application like this, a flexible PCB is required to ensure the hardware remains physically robust and not become dislodged during use.
With a purpose-built compact printed circuit board design, and optimizing the analog circuitry to enable smaller components, the weight of production hardware could be drastically reduced.

**Robust Hardware**

An important consideration was to ensure that the strapped-case assembly would be robust, as the last thing we wanted during the construction and testing of this device was to have the hardware damaged in case of falling on the ground. This required attention to detail and an understanding of the user imposed stresses to the device’s application during all stages of hardware design.

As a protective measure, the hardware was placed in a plastic enclosure to help dampen vibrations and shock. Hardware components such as the battery and battery charger were secured in place within the enclosure with glue. This ensured that the electrical connections of these components would not be stressed mechanically.

**Long battery life**

Having a battery failure in middle of collecting ECG data, especially when it results in the loss of data, can be frustrating. To this end, we specified all components of our hardware to be low power devices. In particular, the ADAS1000 uses just a few mW of power. The most power-hungry device is the SD Card which requires approximately 400mA of current. All other components use an insignificant amount of current. We specified a 2000mAH lithium polymer battery because of its small size and mass, which produces a theoretical 12 hours of data collecting time. Practically speaking, the device should operate for at least 24 hours before it requires recharging. This should last long enough that the time required for a doctor’s visit is less appealing than using our product.
Ease of use

To use our device, the user must place the ECG electrode on the chest, power on the device in each by using a slide on/off switch, and start the app on the phone. There is a Start/Stop button on the App display.

Use in many different applications

The current market for a wireless, multi-channel, multi-sensor bio-monitoring system is huge, as indicated by the number of devices trying to fill this spot. While devices exist we have an opportunity to bring a device to the market that addresses all the deficiencies of current devices. With a more comfortable monitoring system, it would be possible to monitor the vital signs of elderly or sick people continuously, while they continue with normal life. It also allows for personal use in the home when an individual wishes to check their vital signs, much like when people measure their blood sugar, pulse, or temperature. This device also allows for athletes to monitor and track their hearts condition during exercising.
TECHNICAL APPROACH

Hardware

The Mobile ECG monitor's hardware design consists of a small microcontroller circuit (PIC), a data acquisition system for bio potential applications signal (ADAS1000), Bluetooth, and a memory card for data storage. This device measures heart electrical activity on the surface of the human body known as ECG. The data is then transmitted, via Bluetooth wireless protocol, to a transceiver installed in an Android device (Samsung Galaxy Note 2). For our initial design, we determined that the use of sensor (wet vs. dry) and amplifier would be ideal for building our device. However, feedback from analog device web site suggested using a single chip (ADAS1000) which includes sensors and amplifiers. Based on this advice, we modified our design to eliminate all the sensor and amplifier circuitries. The hardware is also composed of a PIC microcontroller, a Bluetooth RN-42-SM, a SanDisk 4GB micro SD card, a Lithium-Polymer battery, and a recharging board for Li-Po.

Figure 2 Prototyping Hardware of ECG monitoring
**Analog front end**

**ADAS1000: Bio-Potential signal acquisition**

This component is used to sense bio-potential signals, thoracic impedance, pacing artifacts, and lead on/off status. The component incorporates a programmable gain low noise differential preamplifier, a fixed gain anti-aliasing filter, buffers, and a 2MHz successive Approximation Register (SAR) converter using 1024 x oversampling to achieve optimal performance. The resulting converted data as well as lead on/off data, respiration data, and pacing artifacts data is sent to a microcontroller via SPI interface.

**Microcontroller**

As the project developed, we have decided to utilize the PIC24FJ64GB002 family of microcontrollers because of their ease of use, general availability, and flexibility to create powerful applications with pre-existing libraries. During the early design stages, we recognized that we had to minimize the size of the hardware and
planned to use a generic ADAS1000 chip mounted on a 64 pin LQPF Adapter board.

This microcontroller (PIC24FJ64GB002) is used as an interface medium between the ADAS1000, Memory Chip, and the Bluetooth chip as well as averaging calculations and other functions. Using the Microchip’s compiler and IDE interface is how the software code will be developed.

**Bio-Potential Signal Acquisition**

Our bio-potential signal acquisition sensor was manufactured by Analog Front End Device. The ADAS1000 series are available in two package options: either a 56-lead LFCSP or a 64-lead LQFP package; they are specified over −40°C to +85°C temperature range. The ADAS1000 is a full-featured, 5-channel ECG including respiration and pace detection, while the ADAS1000-1 offers only ECG channels with no respiration or pace features. Similarly, the ADAS1000-2 is a subset of the main device and is configured for gang purposes with only the ECG channels enabled (no respiration, pace, or right leg drive). We’ve limited the use of ADAS1000 to only 5-channel ECG. It was designed to simplify the task of acquiring and ensuring quality ECG signals. It has provided a low power, small data acquisition system for our bio-potential signal acquisition. The front end of an ECG must be able to deal with extremely weak signals ranging from 0.5 mV to 5.0
mV, combined with a dc component of up to ±300 mV resulting from the electrode-skin contact, plus a common-mode component of up to 1.5 V, resulting from the potential between the electrodes and ground. The useful bandwidth of an ECG signal, depending on the application, can range from 0.5 Hz to 50 Hz for a monitoring application in intensive care units up to 1 kHz for late-potential measurements (pacemaker detection). A standard clinical ECG application has a bandwidth of 0.05 Hz to 100 Hz. ECG signals may be corrupted by various kinds of noise.

**Memory Card**

![Figure 5 MICRO-SD CARD SanDisk (4GB)](image)

This component was used to store the data that is sent from the ADAS1000 AFE component. The component receives data required for storage from the microcontroller via SPI interface. It is capable of saving 4GB of data. We have noticed that the original calculation on data space needed was incorrect because we were assuming a 16 bit data but did not account for space data and time
stamping. Therefore, we did an update and end up using the SanDisk adaptable SD Card.

**Wireless communication**

![Block Diagram](image)

We chose the RN-42-SM protocol because of its low power and highly flexible Bluetooth socket module. This module supports SPP/DUN and HCI Bluetooth interface protocols. It is simple to design in and fully certified. With its high performance on chip antenna and support for Bluetooth Enhanced Data Rate (EDR), the RN-42 delivers up to 3 Mbps data rate for distances to 100M / 20M. The RN-42 socket module was the perfect method for adding Bluetooth wireless capability to our products without redesign and saved us significant time and money.
**Mcp73833 (battery charger)**

The MCP73833 is a highly advanced linear charge management controller for use in space-limited, cost sensitive applications. It draws power through a mini-USB port and has three status LEDs (device ON, charging, and charge complete). This charger works with batteries up to 5000mAh. A lithium-polymer or lithium-ion battery is attached via either a JST 2.0mm connector or via pin pads on the board. By default, the charge current is 470mA. This can be changed to 100mA by cutting a trace. The MCP73833 is available in a 10-Lead, 3 mm x 3 mm DFN package or a 10-Lead, MSOP package. Along with its small physical size, the low number of external components required makes the MCP73833 ideally suited for portable applications. For applications charging from a USB port, the MCP73833 can adhere to all the specifications governing the USB power bus. The MCP73833 employs a constant current/constant voltage charge algorithm with selectable preconditioning and charge termination. The constant voltage regulation is fixed with four available options: 4.20V, 4.35V, 4.40V, or 4.50V, to accommodate new, emerging battery charging requirements. The constant current value is set with one external resistor. The MCP73833 limits the charge current based on die temperature during high power or high ambient conditions. This thermal regulation optimizes the charge cycle time while maintaining device reliability. Several options are available for the preconditioning threshold, preconditioning current value, charge termination value, and automatic recharge threshold. The preconditioning value and charge termination value are set as a ratio, or percentage, of the programmed constant current value. Preconditioning can be set to 100%. The MCP73833 is fully specified over the ambient temperature ranging from -40°C to +85°C.
Smaller size

Components used for prototype design were dual in-line package (DIP). These types of components are bulkier than surface mount technology chips. Changing from DIP to SMT would allow a single board roughly two inch by 1 inch to be used, reducing size greatly.
Software

Our product includes software written for the Bluetooth ECG Monitor, the embedded environment, as well as a custom Android APP, the Mobile environment.

Bluetooth ECG Monitor

The ECG monitor software was written in C programming language using Microchip’s MPLAB X Integrated Development Environment (IDE). MPLAB-X-IDE was used because the microcontroller used for the embedded environment is Microchip’s PIC24FJ64GB002.

The software initializes the ADAS1000 bio-potential data acquisition system; SanDisk microSD flashed based memory device, and the RN-42 Bluetooth socket module.

Android APP User Interface

The Android APP software was written in Java programming language using Eclipse Integrated Development Environment (IDE) with Android Developer Tools (ADT) plugin. The Android APP is a User’s Interface (UI) to the Bluetooth ECG Monitor. This UI gives the user the options to: turn on Bluetooth, connect to a Bluetooth device (the Bluetooth ECG Monitor), start recording ECG data, stop recording ECG data, display current Beats per Minute (B.P.M.), send data from microSD to Android device, and erase micro SD data.
Enabling Bluetooth Pictures
Data collection

Address, data, and the read/write bits are all in the same word. Data is updated on the rising edge of Chip Select (CS) or the first cycle of the following word. For all write commands to the ADAS1000 the data-word is 32 bits. Similarly, read-data is 32 bits (address bits and data bits). For register reads, data is shifted out during the next word.

The general data packet structure is: header, left arm, left leg, right arm. For the 2 kHz Frame Rate a 32-bit data format is used, where the register address is encapsulated in the upper byte, identifying the word within the frame.

The Data Ready (DRDY) pin is used to indicate that a Frame of data is available to be read. If DRDY is logic high the ADAS1000 is still converting the data, when DRDY is logic low a data Frame is available to be read. The entire Frame must be read before DRDY state can change back to converting status.

The serial clock rate was determined by using a formula to determine a minimum clock rate.

Formula is: \( SCLK (\text{min}) = \text{frame\_rate} \times \text{words\_per\_frame} \times \text{bits\_per\_word} \).

Frame Rate = 2kHz, Words per Frame = 4, Bits per Word = 32

\( Sclk(\text{min}) = 256kHz \)

Data interpretation

The channel data value (header, left arm, left leg, and right arm) is read most significant bit first. The data value is a signed twos compliment format ranging from \( -VREF \) to \( +VREF \).
Calibration

Calibrations are done using software settings available on the ADAS1000 chip. These gain calibrations are used in conjunction with the Right Leg Drive to allow accurate readings of bio-potential signals.

Noise

The main sources of noise were:

• Power-line interference: 50–60 Hz pickup and harmonics from the power mains

• Electrode contact noise: variable contact between the electrode and the skin, causing baseline drift

• Motion artifacts: shifts in the baseline caused by changes in the electrode-skin impedance

• Muscle contraction: electromyogram-type signals (EMG) are generated and mixed with the ECG signals

• Respiration, causing drift in the baseline

• Electromagnetic interference from other electronic devices, with the electrode wires serving as antennas, and

• Noise coupled from other electronic devices, usually at high frequencies.

For meaningful and accurate detection, steps have been taken to filter out or discard all these noise sources.
This budget includes everything necessary to build this project. It should be noted that all software tools used (Multisim, Microchip’s MPLAB X, Android APP software) are publicly available at no charge.
PROBLEMS ENCOUNTERED

Shipping delay

The major problem we encountered was first the shipment of our main component (ADAS1000) which took almost a month to get it. Although it did not get here on time, we were able to build the project and present it. The second problem we encountered was writing the android code for our App.
CONCLUSION

There is a gap in the ECG technology that currently doesn’t allow for the data leading up to some form of heart failure to be captured. A portable ECG device that is constantly monitoring a patient’s heart could not only save their life directly by calling for help in the case of heart failure, it can also save many lives indirectly by providing researchers with data outlining a heart’s activity leading up to heart failure.

An algorithm outline has been developed that can read the signal from a portable ECG and perform two main tasks with it. The first is to store the data for in depth analysis in a laboratory later. The second task is to perform some basic dynamic live analysis on the incoming data to determine if the user is experiencing a medical emergency. If this is the case then the smartphone will take appropriate action depending on the severity of the emergency.

The two smartphone giants, iOS and Android, were taken into consideration for development of our app. Android provided many more freedoms regarding their marketplace, resource allocation on the device, and expandable storage options. Taking these factors into account, developing the ECG app for Android is the recommended choice.
APPENDIX A

Microcontroller Code

/**************************INCLUDE FILES LIST**************************/

#include "p24Fxxx.h"
#include "xc.h"
#include "PPS.h"
#include "spi.h"
#include "ADAS1000.h"

//ADAS1000 CHIP SELECT PIN INITIALIZATION
#define ADAS_CS_TRIS TRISBbits.TRISB2 = 0  //Chip Select an output on RB2 pin
#define ADAS_CS PORTBbits.RB2       //Chip Select on RB2 pin

/******************************ADAS1000 SPI FUNCTION DEFINITIONS******************************/

void ADAS_SPI_ENABLE()
{
    SPI1STATbits.SPIEN = 1; //ENABLE SPI1
}

void ADAS_SPI_DISABLE()
{
    SPI1STATbits.SPIEN = 0; //DISABLE SPI1
}

void ADAS_PPS_INIT()
{
    __builtin_write_OSCCONL(OSCCON & 0xBF); //Unlock Registers
    RPINR20bits.SDI1R = 7; //SDI1 on RP07 PIN16
    RPOR5bits.RP10R = 7;   //SDO1 on RP10 PIN21
}
RPOR5bits.RP11R = 8;  //SCK1 on RP11 PIN22
__builtin_write_OSCCONL(OSCCON | 0x40);  //Lock Registers

void ADAS_SPI_INIT()
{
    ADAS_PPS_INIT();
    //ADAS1000 SPI1 COMM CONFIGURATION
    ADAS_SPI_DISABLE();  //Disable SPI Port
    SPI1BUF = 0;         //Clear SPI Buffer
    IFS0bits.SPI1IF = 0; //Clear Interrupt Flag
    IEC0bits.SPI1IE = 0; //DISABLE INTERRUPT
    SPI1STATbits.SPIROV = 0; //Clear OverFlow Bit
    SPI1CON1 = 0x033F;
    /*SPI1CON1 SETTINGS*/
    Internal SPI CLK enabled, SDOx pin controlled by module, 8-bit comm,
    Data_in sampled at end of output time, SDO changes on Active to Idle clock,
    SSx pin not used, Idle clock state is low, Master Mode, Sec=1:1, Pri=1:1*/
    SPI1CON2 = 0x0000;
    /*SPI1CON2 SETTINGS*/
    Framed SPI disabled, Frame sync pulse output, Frame sync pulse active low,
    Frame sync pulse before first bit clock, enhance buffer diabled*/
    ADAS_SPI_ENABLE();    //ENABLE SPI PORT
}

/*******ADAS1000 FUNCTION DEFINITIONS***********/
void ENABLE_ADAS()
{
    ADAS_CS = 0;  //SELECT ADAS1000 CHIP
}

void DISABLE_ADAS()
{ ADAS_CS = 1; //DESELECT ADAS1000 CHIP }

void ADAS_ECGCTL_INIT()
{
    ADAS_SPI_INIT();    //READY ADAS-MCU SPI COMMUNICATION
    ENABLE_ADAS();      //ENABLE ADAS1000 SPI COMMUNICATION
    WriteSPI1(ECGCTL1); //WRITE 4-BYTE CMD TO ECGCTL CONTROL REGISTER
    WriteSPI1(ECGCTL2);
    WriteSPI1(ECGCTL3);
    WriteSPI1(ECGCTL4);
    ReadSPI1();         //READ SPI1BUF TO CLEAR FLAGS
    DISABLE_ADAS();     //DISBALE ADAS1000 SPI COMMUNICATION
}

void ADAS_FRMCTL_INIT()
{
    ADAS_SPI_INIT();    //READY ADAS-MCU SPI COMMUNICATION
    ENABLE_ADAS();      //ENABLE ADAS1000 SPI COMMUNICATION
    WriteSPI1(FRMCTL1); //WRITE 4-BYTE CMD TO FRMCTL CONTROL REGISTER
    WriteSPI1(FRMCTL2);
    WriteSPI1(FRMCTL3);
    WriteSPI1(FRMCTL4);
    ReadSPI1();         //READ SPI1BUF TO CLEAR FLAGS
    DISABLE_ADAS();     //DISBALE ADAS1000 SPI COMMUNICATION
}

void ADAS_INIT()
{
    ADAS_ECGCTL_INIT();
    ADAS_FRMCTL_INIT();
void ADAS_READ_DATA()
{
    ADAS_SPI_INIT();

    //SEND ADAS1000 READ COMMAND
    ENABLE_ADAS();
    WriteSPI1(FRAMES1);
    WriteSPI1(FRAMES2);
    WriteSPI1(FRAMES3);
    WriteSPI1(FRAMES4);
    DISABLE_ADAS();

    //READ ADAS1000 HEADER DATA
    ENABLE_ADAS();
    HEADER_RX[1] = ReadSPI1(SPI1BUF);
    HEADER_RX[2] = ReadSPI1(SPI1BUF);
    HEADER_RX[3] = ReadSPI1(SPI1BUF);
    HEADER_RX[4] = ReadSPI1(SPI1BUF);
    DISABLE_ADAS();

    //READ ADAS1000 Left Arm (LA) DATA
    ENABLE_ADAS();
    LA_RX[1] = ReadSPI1(SPI1BUF);
    LA_RX[2] = ReadSPI1(SPI1BUF);
    LA_RX[3] = ReadSPI1(SPI1BUF);
    LA_RX[4] = ReadSPI1(SPI1BUF);
    DISABLE_ADAS();

    //READ ADAS1000 Left Leg (LL) DATA
    ENABLE_ADAS();
LL_RX[1] = ReadSPI1(SPI1BUF);
LL_RX[2] = ReadSPI1(SPI1BUF);
LL_RX[3] = ReadSPI1(SPI1BUF);
LL_RX[4] = ReadSPI1(SPI1BUF);
DISABLE_ADAS();

//READ ADAS1000 Right Arm (RA) DATA
ENABLE_ADAS();
RA_RX[1] = ReadSPI1(SPI1BUF);
RA_RX[2] = ReadSPI1(SPI1BUF);
RA_RX[3] = ReadSPI1(SPI1BUF);
RA_RX[4] = ReadSPI1(SPI1BUF);
DISABLE_ADAS();

ADAS_SPI_DISABLE();
public class MainActivity extends Activity implements Button.OnClickListener
{

    //MESSAGE TYPES SENT FROM BLUETOOTHRFCOMMCLIENT 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 BLUETOOTHRFCOMMCLIENT 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;

//BT-UART CONSTANTS
private static final int MAX_SAMPLES = 640;
private static final int MAX_LEVEL = 240;
private static final int DATA_START = (MAX_LEVEL + 1);
private static final int DATA_END = (MAX_LEVEL + 2);
private static final byte REQ_DATA = 0x00;
private static final byte CKH = 0x00;

//RUN/PAUSE STATUS
private boolean bReady = false;

//RECEIVE DATA
private int[] ch1_data = new int[MAX_SAMPLES/2];
private int[] ch2_data = new int[MAX_SAMPLES/2];
private int dataIndex = 0;
private int dataIndex1 = 0;
private int dataIndex2 = 0;
private boolean bDataAvailable = false;

//LAYOUT VIEWS
private TextView mBTStatus;
private Button mConnectButton;
private ToggleButton run_button;

//NAME OF CONNECTED DEVICE
private String mConnectedDeviceName = null;
private BluetoothAdapter mBluetoothAdapter = null;

private BluetoothRfcommClient mRfcommClient = null;

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

    // Set up the window layout
    requestWindowFeature(Window.FEATURE_NO_TITLE);
    setContentView(R.layout.main);

    // 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 BT is not on, request that it be enabled.
    if (!mBluetoothAdapter.isEnabled())
    {
        Intent enableIntent = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
        startActivityForResult(enableIntent, REQUEST_ENABLE_BT);
    }
    // else {if (mRfcommClient == null) setupMonitor();}
}

@Override
public synchronized void onResume()
{
    super.onResume();
    if (mRfcommClient != null) // Only if the state is STATE_NONE, do we know that we haven't started already
    {
        if (mRfcommClient.getState() == BluetoothRfcommClient.STATE_NONE)
        {
            mRfcommClient.start(); // Start the BlueTooth RFCOMM services
        }
    }
}

@Override
public void onClick(View v)
{
    int buttonID;
buttonID = v.getId();
switch (buttonID)
{
    case R.id.btn_runtoggle :
        if(run_button.isChecked()) {sendMessage(new String(new byte[] {REQ_DATA})); bReady = true;}
        else {bReady = false;}
    
        if(run_button.isChecked()) {sendMessage(new String(new byte[] {CKH})); bReady = true;}
        else {bReady = false;}
        break;
}

@Override
public void onDestroy()
{
    super.onDestroy();
    if (mRfcommClient != null) mRfcommClient.stop();// Stop the BlueTooth RFCOMM services
}

/**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 (mRfcommClient.getState() != BluetoothRfcommClient.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 BluetoothRfcommClient to write
   byte[] send = message.getBytes();
   mRfcommClient.write(send);
}
}

// private void setupMonitor()
{
   mBTStatus = (TextView) findViewById(R.id.txt_btstatus);
   mConnectButton = (Button) findViewById(R.id.button_connect);
   mConnectButton.setOnClickListener(new OnClickListener()
   {
      public void onClick(View arg0) {BTConnect();}
   });

   run_button = (ToggleButton) findViewById(R.id.tbtn_runtoggle);
   run_button.setOnClickListener(this);

   // Initialize the BluetoothRfcommClient to perform BlueTooth connections
   mRfcommClient = new BluetoothRfcommClient(this, mHandler);
}

// private void BTConnect()
{

// Initialize the BluetoothRfcommClient to perform BlueTooth connections
mRfcommClient = new BluetoothRfcommClient(this, mHandler);
}
Intent serverIntent = new Intent(this, DeviceListActivity.class);
startActivityForResult(serverIntent, REQUEST_CONNECT_DEVICE);

// HANDLER TO GET INFORMATION BACK FROM BLUETOOTH_RFCLIENT
private final Handler mHandler = new Handler()
{
    @Override
    public void handleMessage(Message msg)
    {
        //
        switch (msg.what)
        {
        //
        case MESSAGE_STATE_CHANGE:
            switch (msg.arg1)
            {
            //
            case BluetoothRfcommClient.STATE_CONNECTED:
                mBTStatus.setText(R.string.title_connected_to);
                mBTStatus.append("\n" +
                mConnectedDeviceName);
                break;

            //
            case BluetoothRfcommClient.STATE_CONNECTING:
                mBTStatus.setText(R.string.title_connecting);
                break;

            //
            case BluetoothRfcommClient.STATE_NONE:
                mBTStatus.setText(R.string.title_not_connected);
                break;
        }
```java
break;

// TO-DO: IMPLEMENT RECEIVE DATA BUFFERING

case MESSAGE_READ:
    byte[] readBuf = (byte[]) msg.obj;   //
    int data_length = msg.arg1;         //
    //
    for (int x = 0; x < data_length; x++)
    {
        //
        int raw = UByte(readBuf[x]);
        //
        if (raw > MAX_LEVEL)
        {
            //
            if (raw == DATA_START) {bDataAvailable = true;dataIndex = 0;dataIndex1 = 0;dataIndex2 = 0;}
            //
            else if ((raw == DATA_END) || (dataIndex >= MAX_SAMPLES))
            {
                //
                bDataAvailable = false;dataIndex = 0;dataIndex1 = 0;dataIndex2 = 0;
                if (bReady)
                {MainActivity.this.sendMessage(new String(new byte[] { REQ_DATA }));} // SEND "REQ_DATA" AGAIN
            }
        }
    }
    // VALID DATA
    else if ((bDataAvailable) && (dataIndex < (MAX_SAMPLES)))
    {
        //
    }
```
if ((dataIndex++) % 2 == 0)
    ch1_data[dataIndex1++] = raw;  // EVEN DATA
    //
    else ch2_data[dataIndex2++] = raw;
    // ODD DATA
    
break;

// SAVE CONNECTED DEVICE'S NAME
case MESSAGE_DEVICE_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;
    
}
}

// SIGNED TO UNSIGNED
private int UByte(byte b)
{
    // IF NEGATIVE
    if (b < 0) return (int) ((b & 0x7F) + 128);
    // IF POSITIVE
    else return (int) b;
public void onActivityResult(int requestCode, int resultCode, Intent data) {
    switch (requestCode) {
    //
    case REQUEST_CONNECT_DEVICE:
        // When DeviceListActivity returns with a device to connect
        if (resultCode == Activity.RESULT_OK) {
            // Get the device MAC address
            String address =
                data.getExtras().getString(DeviceListActivity.EXTRA_DEVICE_ADDRESS);
            // Get the BLuetoothDevice object
            BluetoothDevice device =
                mBluetoothAdapter.getRemoteDevice(address);
            // Attempt to connect to the device
            mRfcommClient.connect(device);
        }
        break;
    // When the request to enable BlueTooth returns
    case REQUEST_ENABLE_BT:
        // BlueTooth is now enabled, so set up monitor
        if (resultCode == Activity.RESULT_OK) {
            setupMonitor();
        }
        break;
    // User did not enable BlueTooth or an error occurred
    else
Toast.makeText(this, R.string.bt_not_enabled_leaving,
Toast.LENGTH_SHORT).show();
finish();
break;
APPENDIX C

Schematics

Figure 9 ECG monitor circuit diagram
Figure 10 ADAS1000 shield PCB

*DVDD can be operated from 1.65V to 3.6V. In this circuit, 3.3V is used for interfacing purposes.

NOTES
1. In this circuit, the AGND, DGN and REFIN are all tied into one ground plane. The digital ground and digital circuitry can be split into a separate ground, with connection made to AGN at the ADAS1000 device. The configuration shown here uses RA, LA, LL, V1 and RL, leaving 1 spare ADA path that can be used for other measurements. Alternatively, this can be ganged with a slave ADAS1000 device to achieve a nine ECG + one RL or 12-lead measurement.
APPENDIX D

Datasheets

Evaluation Board User Guide
ADAS1000SDZ

One Technology Way • P.O. Box 9106 • Norwood, MA 02062-9106, U.S.A. • Tel: 781.329.4700 • Fax: 781.461.3113 • www.analog.com

ADAS1000 ECG Front-End Evaluation Board for Demonstration and Development

FEATURES
Biopotential signals in; digitized signals out
6 electrodes (5 acquisition channels and 1 driven lead)
2 × ADAS1000 (master/slave) for 1 to 12 lead electrode measurements
AC and DC lead-off detection
Pace detection
Optional thoracic impedance measurement (internal/external)
Selectable reference lead
Lead or electrode data available
Calibration features (DAC and 1 mV square/sinewave)
Low or high speed data output rates
Serial Interface SPI°/-QSPI°/-DSP-compatible

CUSTOMER INTERFACING OPTIONS
Direct access to ADAS1000 serial interface (J4)
USB interface via Analog Devices, Inc., Blackfin°-based SDP-B controller board and GUI

APPLICATIONS
ECG
Patient monitor
Holter monitor
Cardiac defibrillators

GENERAL DESCRIPTION
The primary function of the EVAL-ADAS1000 evaluation board is to demonstrate the ADAS1000 integrated ECG device for medical instrumentation. A full description of the ADAS1000 is available in the data sheet and should be consulted when utilizing this user guide. This evaluation board is used to evaluate the ADAS1000 and all variants.

The evaluation board is ideal for exploring concepts and adopting the ADAS1000 into advanced medical systems. The board can be operated as part of an end user’s system via the ADAS1000 SPI interface or as a standalone evaluation of the ADAS1000 using the support of the Analog Devices system demonstration platform controller board (SDP-B), and a standard PC (running Windows® XP, Windows® Vista or Windows® 7 32-bit and 64-bit) to run the Analog Devices evaluation software.

The SDP-B board, which is a DSP-based controller board is separate from the evaluation board, is required to run the Analog Devices evaluation software and is used for data transfer from evaluation board to PC via USB interface. It does not come as part of the evaluation kit and can be ordered separately (EVAL-SDP-CB1Z). It can be reused with many other Analog Devices evaluation modules.

For more information, visit the Analog Devices healthcare home page.
ADAS100 Functional Block Diagram

ADAS100x ECG AFE Roadmap

<table>
<thead>
<tr>
<th>Part Number</th>
<th>Number of Electrodes</th>
<th>Extra Features</th>
<th>Input Noise, 0.05 Hz to 150 Hz (µV p-p)*</th>
<th>CM I/P Range (V)</th>
<th>Supply Rail (V)</th>
<th>Temperature</th>
<th>Package</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADAS1000</td>
<td>5 ECG electrodes + RLD</td>
<td>Respiration pace + I/F</td>
<td>10</td>
<td>±1</td>
<td>3.3 to 5.5</td>
<td>−40°C to +85°C</td>
<td>56-lead LF CSP, 64-lead LQFP</td>
</tr>
<tr>
<td>ADAS1000-2</td>
<td>5 ECG electrodes (companion chip for gang purposes)</td>
<td>—</td>
<td>10</td>
<td>±1</td>
<td>3.3 to 5.5</td>
<td>−40°C to +85°C</td>
<td>56-lead LF CSP</td>
</tr>
<tr>
<td>ADAS1000-1</td>
<td>5 ECG electrodes + RLD (no pace algorithm, includes pace I/F)</td>
<td>Pace I/F</td>
<td>10</td>
<td>±1</td>
<td>3.3 to 5.5</td>
<td>−40°C to +85°C</td>
<td>56-lead LF CSP</td>
</tr>
</tbody>
</table>

*Noise performance measured in 2 kHz data rate with a gain of 28, providing a ±0.5 V common-mode input range.
Universal Serial Bus Features:
- USB v2.0 On-The-Go (OTG) Compliant
- Dual Role Capable – can act as either Host or Peripheral
- Low-Speed (1.5 Mbps) and Full-Speed (12 Mbps) USB Operation in Host mode
- Full-Speed USB Operation In Device mode
- High-Precision PLL for USB
- 0.25% Accuracy using Internal Oscillator – No External Crystal Required
- Internal Voltage Boost Assist for USB Bus Voltage Generation
- Interface for Off-Chip Charge Pump for USB Bus Voltage Generation
- Supports up to 32 endpoints (16 bidirectional):
  - USB module can use any RAM location on the device as USB endpoint buffers
- On-Chip USB Transceiver
- Interface for Off-Chip USB Transceiver
- Supports Control, Interrupt, Isochronous and Bulk Transfers
- On-Chip Pull-up and Pull-Down Resistors

Power Management Modes:
- Selectable Power Management modes with nanoWatt XLP Technology for Extremely Low Power:
  - Deep Sleep mode allows near total power-down (20 nA typical and 500 nA with RTCC or WDT),
  - along with the ability to wake-up on external triggers or self-wake on programmable WDT or RTCC alarm
  - Extreme low-power DSBOR for Deep Sleep, LPROR for all other modes
  - Sleep mode shuts down peripherals and core for substantial power reduction, fast wake-up
  - Idle mode shuts down the CPU and peripherals for significant power reduction, down to 4.5 µA typical
  - Doze mode enables CPU clock to run slower than peripherals
  - Alternate Clock modes allow on-the-fly switching to a lower clock speed for selective power reduction during Run mode down to 15 µA typical

Special Microcontroller Features:
- Operating Voltage Range of 2.0V to 3.6V
- Self-Programmable under Software Control
- 5.5V Tolerant Input (digital pins only)
- High-CURRENT Sink/Source (18 mA/18 mA) on All I/O Pins
- Flash Program Memory:
  - 10,000 erase/write cycle endurance (minimum)
  - 20-year data retention minimum
  - Selectable write protection boundary
- Fail-Safe Clock Monitor Operation:
  - Detects clock fail and switches to on-chip FRG oscillator
- On-Chip 2.5V Regulator
- Power-on Reset (POR), Power-up Timer (PWRT) and Oscillator Start-up Timer (OST)
- Two Flexible Watchdog Timers (WDT) for Reliable Operation:
  - Standard programmable WDT for normal operation
  - Extreme low-power WDT with programmable period of 2 ms to 26 days for Deep Sleep mode
- In-Circuit Serial Programming™ (ICSP™) and In-Circuit Debug (ICD) via 2 Pins
- JTAG Boundary Scan Support

<table>
<thead>
<tr>
<th>PIC24FJ Device</th>
<th>Pins</th>
<th>Program Memory (Bytes)</th>
<th>SPAM (Bytes)</th>
<th>Remapable Peripherals</th>
<th>Remappable Peripherals</th>
</tr>
</thead>
<tbody>
<tr>
<td>32GB002</td>
<td>28</td>
<td>32K</td>
<td>8K</td>
<td>15 5 5 5 2 2 2 9 3 Y Y Y</td>
<td></td>
</tr>
<tr>
<td>64GB002</td>
<td>28</td>
<td>64K</td>
<td>8K</td>
<td>15 5 5 5 2 2 2 9 3 Y Y Y</td>
<td></td>
</tr>
</tbody>
</table>
Class 1 / Class 2 Bluetooth® Socket Module

Features

- Socket module with UART interface
- 3.3/5V logic CMOS I/O (RS-232 as well)
- Fully qualified Bluetooth 2.1/2.0/1.2/1.1 module
- Bluetooth v2.1+EDR support
- Low power (8-30mA connected, 2mA idle)
- UART supports baud rates from 1200 to 3Mbit
- Sustained SPP data rates - 240Kbps (slave), 300Kbps (master)
- HCI data rates - 1.5Mbps sustained, 3.0Mbps burst in HCI mode
- HCI mode, or SPP/DUN software stacks available.
- Embedded Bluetooth stack profiles included (requires no host stack): GAP, SDP, RFCOMM and L2CAP, with SPP, DUN and HID profiles.
- RS232 on board with power enable on IO pin.
- Bluetooth SIG Qualified, End Product Listing
- Class 1 high power amplifier (RN-41-SM only) with on board ceramic RF chip antenna.
  - Certifications: FCC, ICS, CE
  - Environmentally friendly, RoHS compliant

Applications

- Cable replacement
- Barcode scanners
- Measurement and monitoring systems
- Industrial sensors and controls
- Medical devices
- Asset tracking

Description

The RN-41-SM / RN-42-SM is a though hole, low power and highly flexible Bluetooth socket module. This module supports SPP/DUN and HCI Bluetooth interface protocols, is simple to design in and fully certified. With its high performance on chip antenna and support for Bluetooth® Enhanced Data Rate (EDR), the RN-41 / RN-42 delivers up to 3 Mbps data rate for distances to 100M / 20M. The RN-41/RN-42 socket module is the perfect method for adding Bluetooth wireless capability to existing products without redesign, saving you significant time and money.