FJL - IMUO

WORCESTER POLYTECHNIC INSTITUTE

Wilson

Dan Sullivan (dansullivan@wpi.edu)
Jibin Mathew (jmathew1991@wpi.edu)
Ryan Kimmel (ryankimmel@wpi.edu)

Advisor: Prof. Fred Looft (fjlooft@wpi.edu)

25 April 2013

Abstract

The goal of this project was to create a companion device that could work in tandem with an Android tablet or phone to display information that would be of interest to an individual who was looking for an electronic companion, much like Wilson was a companion in the movie Castaway. The Wilson device uses various sensors to collect information about its acceleration, rotation, temperature, and altitude. Wilson then transmits this data via Bluetooth to an Android application that displays selected information for remote viewing.

Acknowledgements

The Wilson MQP team would like to thank our advisor, Prof. Looft, for guiding us through the project.

We would also like to thank WPI's ECE shop, which made hardware easily accessible to us.

Additionally, we would like to thank the people who will read this report. We point out that, for further information, you can contact our advisor, Prof. Looft, at fjlooft@wpi.edu. Appendix A – Supplementary Materials provides a list of items that Prof. Looft has and can make available to you.

Table of Contents

1 Introduction

In the movie Cast Away (2000), Tom Hanks plays a man stranded on a deserted island. His only companion was a volleyball that he befriended to pass the time. The volleyball's name was Wilson and it is the main inspiration behind this project. We wanted to make a device that would provide the user with as much information as possible about its position and current surroundings. The Wilson device is a collection of sensors including an altimeter and an accelerometer to collect data. By using Bluetooth, the Wilson device sends sensor data to a "Wilson App", which can be run on an Android device. The application displays the sensor data graphically. The following sections will go into detail about the system design, implementation, and testing.

2 Background

This project assumes a basic understanding of selected topics with which the average user may not be familiar. The following sections provide information in order to establish a foundation for the remainder of this report.

2.1 Android

Android Inc. originally developed the Android operating system (OS) for touch screen applications. In 2005, Google acquired Android Inc. (Elgin, 2005). Android OS is Linux-based and open source. This allows anyone to develop for and with Android software.

Phone

The Android OS was originally developed for smart phone and tablet computers and thus can be found running on many devices all around the world. Although each device on which Android runs may be different, the core of Android is robust enough to provide a useful operating system for mobile devices. According to the International Data Corporation "Android had a worldwide smart phone market share of 75% during the third quarter of 2012" (International Data Corporation, 2013).

Applications

The applications written for Android are written in a Java-like code style that emphasizes object-oriented programming. Each application may be put on the Android application market, called the Google Play store. To put an app on the market, however, you need to gain Android developer status, which can be achieved through a simple application process.

Development

Fortunately, Google makes it very easy for developers to develop apps for the Android operating system. The Android Software Development Kit (SDK) is freely available and contains numerous examples that are extremely useful for getting started.

2.2 Inter Integrated Circuit (I2C)

I^2C is a communication protocol designed to use very little board space and to interact with many devices. In a standard I^2C implementation, a single master device is able to communicate with various slave devices over just two wires. The wires used are serial data and clock. By addressing each transmission, the I^2C master is able to connect to multiple slave devices simultaneously, as seen in Figure 1.

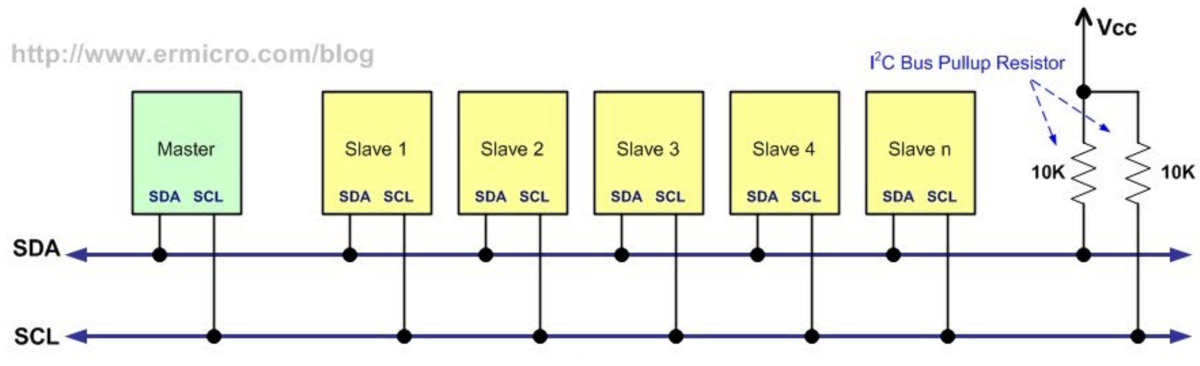


Figure 1: High Level I²C Example with Multiple Slave Devices[1]

Because there is only one data line and there is no multiplexing, I²C is half-duplex. This means that at any given time, the master is either sending or receiving data but these actions cannot happen concurrently. This can be a drawback since it means that I²C will be inherently slower than other communication protocols as each transmission requires the overhead of transmitting the slave's address. Figure 2 shows typical I²C transmit and receive bit patterns.

Master Transmit Example

START	SLAVE ADDRESS	WRITE	ACK	
	DATA		ACK	STOP

Master Receive Example

START	SLAVE ADDRESS	READ	ACK	
	DATA		NACK	STOP

Figure 2: I²C Transmit and Receive Example Bit Patterns (Master: Blue; Slave: Orange)

2.3 Universal Asynchronous Receiver/Transmitter (UART)

UART is a bidirectional serial communications protocol. A typical implementation of UART will involve a single master communicating with a single slave. The protocol calls for each device to have a transmit port and a receive port. The master's transmit is connected to the slave's receive and vice versa. This allows the master and slave to communicate. Figure 3 shows this configuration.

[1] http://www.ermicro.com/blog/?p=744

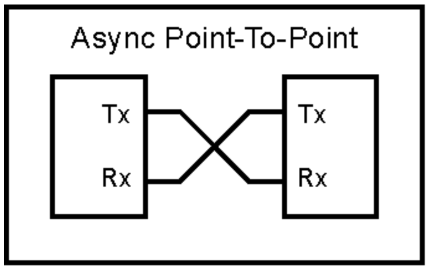


Figure 3: Simple UART Connection Diagram[2]

Since transmit and receive can operate independently of each other, UART is full duplex. This independency, paired with the fact that UART does not require the overhead of addressing, means that UART can achieve much higher throughput than I2C. Unfortunately, the typical UART configuration does not allow multiple slave devices, which is a significant drawback.

2.4 MSP430 Microcontroller and Launchpad

Texas Instruments (TI) makes a microcontroller line titled the MSP430. These mixed signal processors have a range of modules built in depending on the model. Some of the modules that can be incorporated include various types of communication busses such as I^2C and UART. Additionally, the microcontroller may have an analog to digital converter. Almost every MSP430 includes multiple timers, which can act independent of the main system clock. TI designed MSP430 microcontrollers to use very little power, and as such, they each have various low power modes, which allow the developer to conserve as much power as possible as often as possible. Figure 4 shows a TI Launchpad platform and an MSP430 device.

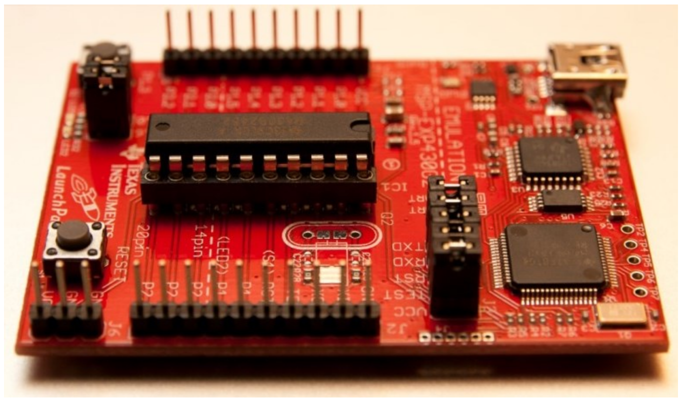


Figure 4: MSP430G2452 on a TI Launchpad Development Board[3]

The MSP430 microcontrollers can be either surface mount or dual in-line packaged, which allows for PCB or breadboard development. To assist developers, Texas Instruments developed a breakout board for the MSP430 called the Launchpad. The Launchpad provides easily accessible headers as well as built-in switches and LEDs. Figure 4 shows the TI Launchpad development board.

[2] http://www.sampson-jeff.com/article/interface/int.htm
[3] http://orpix.org/2012/05/08/ti-launchpad/

2.5 Inertial Measurement Unit (IMU)

An IMU is an electronics module that collects and reports linear acceleration and orientation. A mixture of devices including accelerometers, gyroscopes, and magnetometers take measurements. IMUs are essential to inertial navigation systems such as those found in aircraft.

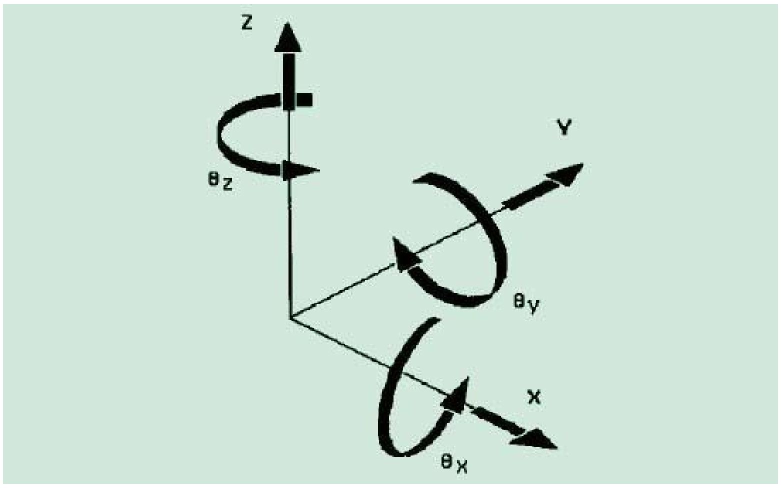


Figure 5: 6 Degrees of Freedom[4]

The IMU contains three orthogonally accelerometers that are used to measure linear acceleration in the x, y, and z direction. Three gyroscopes measure acceleration about each of these axes (yaw, pitch, and roll). Finally, the absolute direction of magnetic north is determined using a 3-axis compass.

2.6 Global Positioning System (GPS)

A GPS is a navigation system consisting of 24 satellites owned by the U.S. Department of Defense. This system has been open to civilian use since the 1980s. It is a freely available resource usable in any part of the world.

The satellites orbit the earth, constantly transmitting signal information. The GPS receiver then uses this information to calculate the user's location. The GPS must use a minimum of three satellites to calculate a 2D position, and four satellites for a 3D position. Using this data, the receivers can calculate other useful information such as speed, direction, and distance.

[4] http://2.bp.blogspot.com/-LuRps3CRaHc/T5rxG1CECyI/AAAAAAAAAEQ/e3b_GNya6Yw/s1600/fig+2.jpg

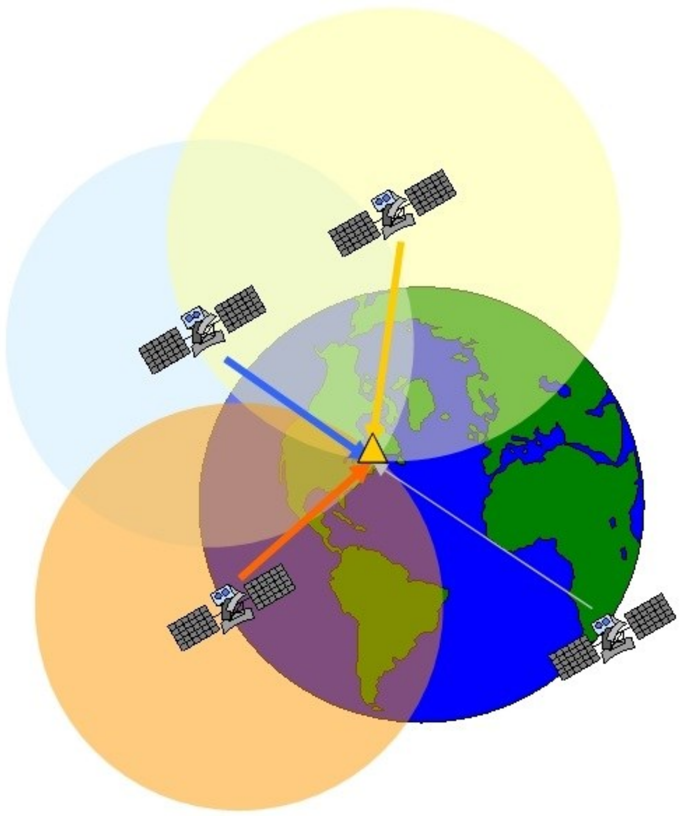


Figure 6: GPS Triangulation[5]

The Wide Area Augmentation System (WAAS) provides information to GPS receivers to boost their accuracy and reliability of position estimates. GPS signals are received by Wide Area Reference Stations (WRS), which is then forwarded to the WAAS Master Station (WMS) (Federal Aviation Administration, n.d.). The WMS then creates WAAS augmentation messages that hold information on GPS errors so that the GPS receivers may remove those errors for better accuracy on position. Many times, there are GPS signal errors caused by timing, orbiting errors, or even disturbances in the ionosphere. For instance, if the satellite clock timing is off by a couple of seconds. GPS receivers can then use this WAAS information accurately readjust the timing. WAAS allows location calculations to be accurate to be within 3m.

Despite the precision of GPS modules, they have one major flaw. Since their location is dependent on information from satellites, GPS receivers will not work if the receiver cannot obtain a strong-enough signal from four or more satellites. As a result, GPS receivers generally will not work indoors and underground where satellite signals cannot easily penetrate.

2.7 Bluetooth

Bluetooth is a wireless standard for short distance data exchange created by Ericsson in 1994. The key features of Bluetooth are its low power and low cost. Bluetooth networking transmissions can use as

[5] http://www.3grt.fr/Comprendre_le_GPS.html

little as 1mW output power with a frequency between 2.402GHz and 2.480GHz. A Bluetooth connection has a range of 10 meters and thus avoids interfering with other wireless signals.

2.8 Altimeter

An altimeter is an instrument that measures the altitude of an object versus sea level. It usually gets the measurement by measuring the air pressure using an internal barometer. Another method of measuring air pressure is by using a high precision radar altimeter that measures the transmission time between the device and a satellite. Hikers and climbers can use altimeters to help determine location of the user. Aircraft also use altimeters so the pilots can determine where they are in the sky (or on the ground).

2.9 Summary

This section ultimately provides the reader with background information required to understand the remainder of the report. The sections are written to be independent of each other, and can thus serve as a reference throughout the report.

3 Project Statement

The goal of the Wilson MQP was to develop an electronic device that could display its location and environmental data in an interesting way. Attaining this goal consisted of three main stages. First, we chose external sensors that could monitor various environmental factors. Second, we developed an Android application that could display the data in an interesting way. Finally, we interfaced these two components with each other to create the final product.

3.1 Project Objectives

Based on the project goals, the objectives of this project were to:

- Interface the system to a reasonable number of sensors to provide environmental data to higher-level applications.
- Develop at least one original Android application to interface with external sensors.
- Successfully interface with at least one Bluetooth device.
- Include solar charging with the system.
- Consolidate components into a single prototype device.

3.2 Methods

The following sections contain information on the design methods used to fulfill each of the project objectives.

Interfacing External Sensors

A single I^2C bus interconnected the external sensors. This I^2C bus allowed a master controller to poll each sensor in order to retrieve various pieces of information such as altitude, temperature, and acceleration.

Original Android Applications

The objective for the main application was to display the sensor information in a creative and unique way. In order to test the functionality of various components, we designed simple Android applications that we used as stepping stones for the final application.

Interface with Bluetooth Device

The Android development board has Bluetooth capability that allows it to connect with external Bluetooth devices. Such devices include Bluetooth keyboards, headsets, cameras, or even another Android device. This would allow users to have multiple ways to interact with the device.

Ultimately, the Android development board used Bluetooth to connect with a separate module, which contained each of the external sensors. This allowed the sensors to be device independent and was easier to implement than direct communication would have been.

Solar Charging and Consolidating Components

The last objective was to combine all sensors and the development board into a single unit with solar charging capability. As a result, the final system design was incorporated into a single self-contained unit.

The development board runs on a rechargeable battery pack. Small photovoltaic cells on the back of the unit can recharge this battery pack. The final unit also has a small fan to reduce heat on the board. Two separate acrylic boxes house the board and the measurement devices. This allows Android devices other than the development board to utilize the external components.

3.3 Schedule

The list below shows the general schedule of the project. The schedule breaks up the project's highlights by term. The schedule covers the accomplishments at each stage of the project.

- A Term
  - Determine Project Direction
  - Component Selection
  - Overall Project Planning and Timeline
- B Term
  - I^2C Communication and Testing
  - Android Development and Communication with Internal Components
- C Term
  - Android Application with User-Triggered Color Changes
  - Connect Android to Existing Bluetooth GPS
  - Fabricate Box to Hold All Components
  - Develop External Component Controller on MSP430
  - Develop Altimeter Drivers for MSP430
  - Develop IMU Drivers for MSP430
- D Term
  - Acrylic Boxes Completed
  - Basic Graphing Software Implementation on Android
  - Tested Android to External Controller Communication
  - Final MSP430 and Android Tweaks for Interoperability
  - Android Graphing of External Component Data

3.4 Summary

The overall focus of this project was to meet the four project goals. These goals were to interface with external sensors, to create original Android applications, to interface with a Bluetooth device and to consolidate all of the components into one module. By meeting these objectives, we were able to achieve our goal of displaying location and environmental data in an interesting way.

4 System Design

The purpose of this section is to provide a high-level understanding of how the system functions. After explaining the system design, we will discuss the features and considerations of each component. Footnotes will provide a link to the product website when a new component is introduced.

4.1 Overall System Design

The Wilson MQP ultimately consists of six main components, shown in the block diagram of Figure 7. We will briefly describe each component and how it interacts with its neighbors.

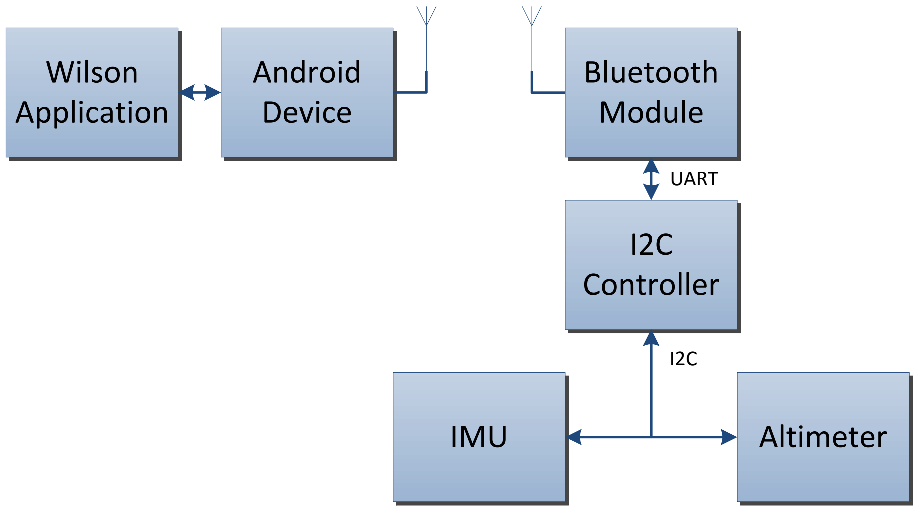


Figure 7: High-level Block System Block Diagram

The highest-level component is the Wilson Application. This application can be run on any Android device whose operating system is at least 4.0.3 (Ice Cream Sandwich).

The next component is the Android Device. While we initially used a dedicated Android development board, we discovered that the development board could be substituted with any Bluetooth-enabled Android-based processor board that has a minimum software version of Android 4.0.3.

Next is the Bluetooth Module. This component serves as a bridge between Bluetooth and UART.

The I²C Controller gathers data from the external components on the I²C bus and immediately passes it to the Android device via the Bluetooth Module.

Finally, there are two external components. Each component connects to the I²C bus and is a slave to the I²C Controller. The first component is an IMU. More information about IMUs is given in the Background section. This device measures the acceleration in the x, y, and z directions. Additionally, it measures the rotation about each axis. The altimeter provides the barometric pressure and the environmental temperature.

4.2 Component Selection

In order to choose components for Wilson objectively, we identified and selected different criteria for each type of component. Next, we assigned a weighting factor to these criteria based on which ones we felt were more significant for the product. Finally, we researched individual components to determine which best fit the criteria we had created. The following sections detail this process.

Development Board

Originally, our project required that we implement Linux or Android on a device. However, as we researched devices that would support that requirement, we found the development board in Figure 8.

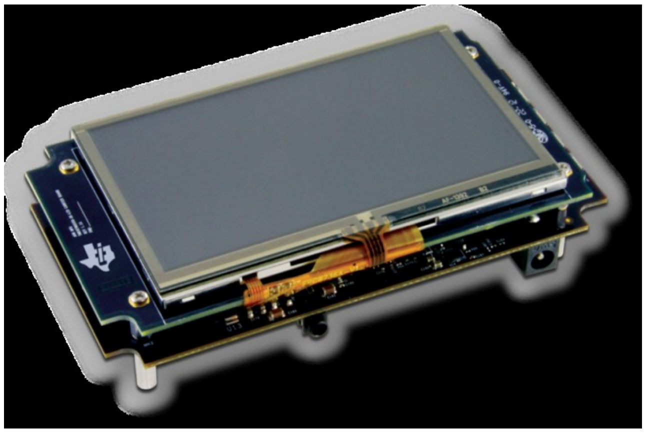


Figure 8: Texas Instruments' AM335x Starter Kit[6]

TI designed this development board to run Linux or the Android operating system, which eliminated the need to implement the Android operating system on a microcontroller. For this reason, we decided to use the development board, which would allow our team to focus more on integrating the external components rather than dealing with operating systems. In addition, TI designed the board with developers in mind, and so TI includes various example code projects and programming support for the module. Unfortunately, as the project evolved, we discovered that a typical smart phone could replace this component. Using smart phones had other benefits such as faster processors, capacitive touch screens, and a more intuitive button layout.

GPS

Table 1 shows each of the GPS components we considered. The main criteria we considered were the unit price, average power consumption, interface, and size. Ease to Implement is directly related to the Connection field. Since I^2C is addressable and allow multiple devices to share a common communication bus, the ease of implementation was a very important factor. Choosing an I^2C compatible device would ultimately allow Wilson to communicate with all of its sensors over a single port.

[6] http://processors.wiki.ti.com/index.php/AM335x_Starter_Kit

Table 1: GPS Component Analysis

Name	Price ($)	Power (mW)	Connection	Price	Size	Power	Ease to Implement	Totals
Weights				*3*	*3*	*4*	*5*	*N/A*
EM-410	44.90	198	UART	5	3	4	2	50
D2523T	79.95	495	UART	1	4	2	2	33
GP-2106	49.95	117	UART	4	5	4	2	53
Navigatron v2	59.90	158	I2C	3	3	3	5	55

The Totals column in Table 1 shows that the Navigatron V2, seen in Figure 9, seemed to be the best choice GPS. In addition to using I2C, this GPS has low power requirements, which is ideal for mobile applications. As an added bonus, this module also supports the WAAS, which can greatly improve the accuracy of a GPS. The Background section of this report contains more information about WAAS. Unfortunately, the GPS is not included in the final Wilson product due to a lack of documentation for the module. For this reason, we suggest that the reader avoid using this component.

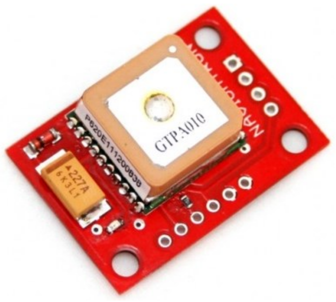


Figure 9: Navigatron V2 GPS[7]

IMU

The chief concerns for our IMU were the cost, size, power consumption, and the ease to implement. All of the devices we researched had low power consumption, and they were all similarly small. This meant the deciding factors were cost and the ease to implement. Most of the devices supported I2C, which we had already determined to be the ideal communication bus. This left price as the primary determining factor. Table 2 shows an abridged analysis of the components.

Table 2: IMU Component Analysis

Name	Price ($)	Power (mW)	Connection	Price	Size	Power	Ease to Implement	Accel	Gyro	Magne	Totals
Weights				*3*	*5*	*4*	*3*	*5*	*5*	*3*	
Sen-10724	100	24	I2C	3	4	4	4	5	5	5	131
Razor	125	30	Serial	3	4	4	4	5	5	5	131
Sen-10121	65	23	I2C	4	5	4	4	5	5	1	127
MPU-6050	40	13	I2C	5	5	5	5	5	5	1	137

[7] http://www.flytron.com/sensors/180-i2c-gps-for-multiwii-and-others.html

Table 2 shows that the MPU-6050 was the clear winner. While the other IMUs measure acceleration on nine axes, the MPU-6050 only measures acceleration on six axes. This is acceptable since we are not expressly concerned about the magnetometer. By eliminating those axes, we can reduce the cost by nearly a third.

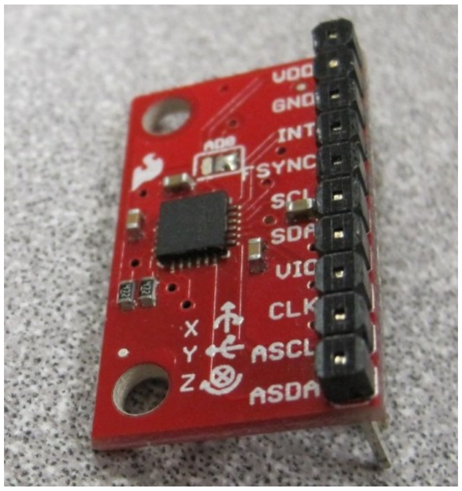


Figure 10: Sparkfun's MPU-6050 IMU[8]

The IMU in Figure 10 has sensors measuring six axes. It monitors acceleration in the typical x-, y-, and z-axis and it monitors the acceleration about each of these axes. This module can communicate using I2C, which made it ideal for the final Wilson device. The MPU-6050 uses a register-based approach to I2C. The Background section discusses the advantages and disadvantages of this approach. However, it is sufficient to say that using a register-based approach allows for easier communication with an I²C master.

Altimeter

The MS5607 Altimeter Module is a highly accurate temperature and altimeter sensor, which measures atmospheric pressure to determine the altitude. The sensor provides a resolution of 20 cm. It uses an I²C interface, which is very desirable due to the limited communication lines on the development board and the MSP430. The low power requirements are beneficial since the final product is battery powered. Additionally, the altimeter has different operation modes, which allow the user to strike a balance between the device's reading accuracy and its speed. The module is also extremely small (5 x 3 x 1 mm) which works well with the mobile aspects of Wilson.

[8] https://www.sparkfun.com/products/11028

Figure 11: Parallax's Altimeter Module (MS5607)[9]

Bluetooth Module

This Bluetooth module is an attachment designed for TI's Launchpad development board. It is the only module we found which could be easily interfaced directly with the Launchpad. The module allows an MSP430 chip to communicate via Bluetooth. Within the scope of this project, this board allows the MSP430 to relay information from the external sensors back to the Android device. A particular benefit of this device is the amount of support and examples that are available. These examples allowed our team to rapidly develop an interface between the MSP430 microcontroller and this device.

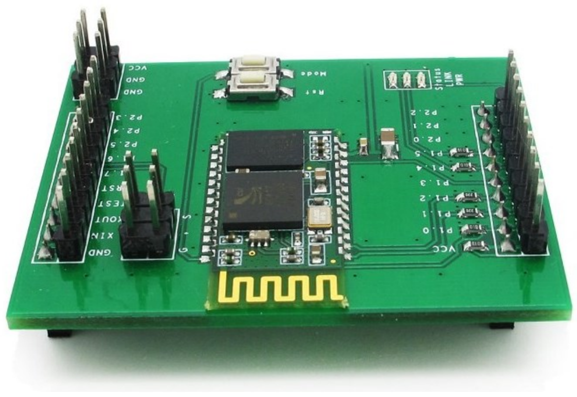


Figure 12: ITead Studio's BT BoosterPack Module[10]

Batteries

Four rechargeable AA Nickel-Metal Hydride batteries will power the development board. By connecting the 1.2 V, 2100 mAh batteries in series, we are able to provide the board with 4.8 V for about 2100 mAh. One of our power tests showed that the fully charged batteries could power the development board for at least four hours.

Solar Panel

To supplement the energy stored in the batteries, we added energy harvesting to Wilson. We considered the three solar panels in Table 3.

[9] http://www.parallax.com/Store/Sensors/AccelerationTilt/tabid/172/ProductID/780/List/0/Default.aspx
[10] http://imall.iteadstudio.com/bt-boosterpack.html

Table 3: Solar Panel Component Analysis

Name	Price ($)	Power (W)	Size (mm)
9V Parallax	10	1.000	135x135x2.8
6V Parallax	10	1.000	125x63
Small Solar Panel	20	0.500	95x63

We chose to use Parallax's 6V solar panel (Figure 13) since it has a large enough output voltage to charge the batteries, and provides a reasonable peak output power of 1W. Additionally, this model solar panel is similar in size to the development board. This enables both components to be included in the same housing on opposing sides. In effect, this means that the solar panel will be out of sight and idle until the user decides to orient the box so that it can charge.

Figure 13: Parallax's 6V Solar Panel[11]

Enclosure

We chose to use Acrylic to house the different components. Acrylic is very durable and easy to laser cut. The main acrylic box housed the development board, the batteries, solar panel, and a 5V fan. The second box contains the IMU, altimeter, TI Launchpad with MSP430 microcontroller, and the BT BoosterPack. The casing is made of clear acrylic, which allows those using or viewing the device to see the internal components.

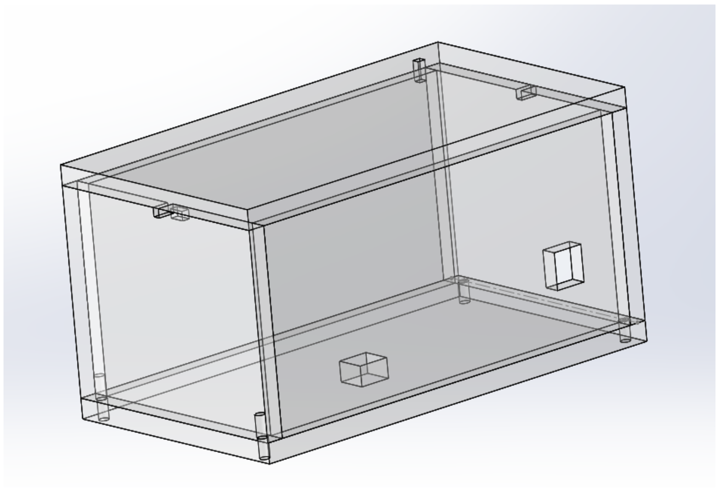


Figure 14: CAD Drawing of Wilson Acrylic Enclosure

[11] http://www.parallax.com/tabid/768/ProductID/619/Default.aspx

4.3 Summary

This section provides insight into the components we selected for the final design of Wilson. To reiterate, Appendix A – Supplementary Materials has information about obtaining the trade studies for each of these components. Now that the components have been introduced, we can begin to discuss how each component was implemented to realize the high-level system design at the beginning of this section.

5 System Implementation

This section discusses the steps taken and the challenges faced in implementing the system given in System Design. The implementation starts with the high-level software development and progresses to the I²C bus controller, and finally to the implementation of the individual sensor modules.

5.1 Final System Architecture

Below is the system architecture of the Wilson Device, demonstrating the pin layout and connections between the MSP430 Launchpad, the Bluetooth Booster Pack, and the altimeter and the accelerometer.

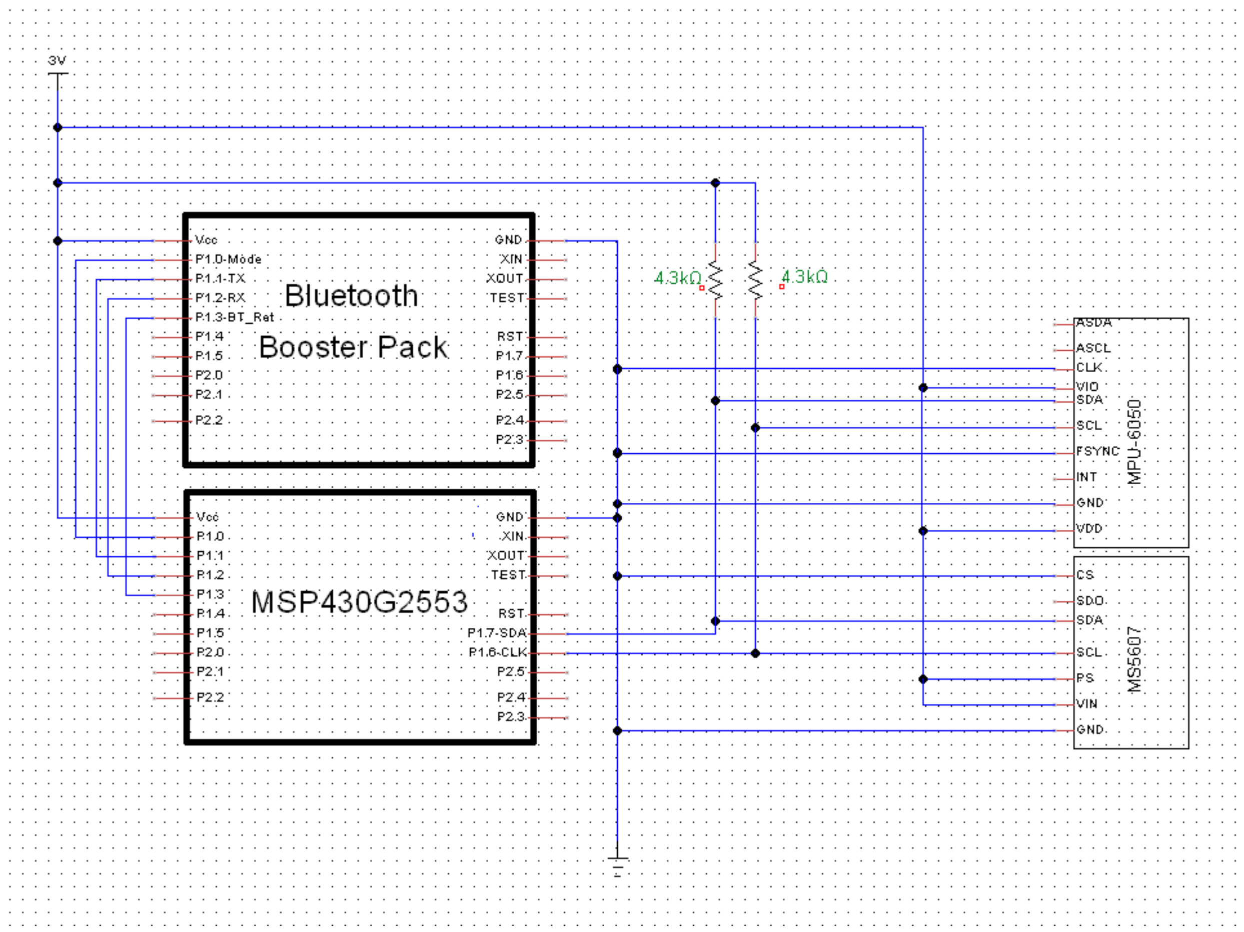


Figure 15: Wilson External Component Circuit Diagram

5.2 Android Applications

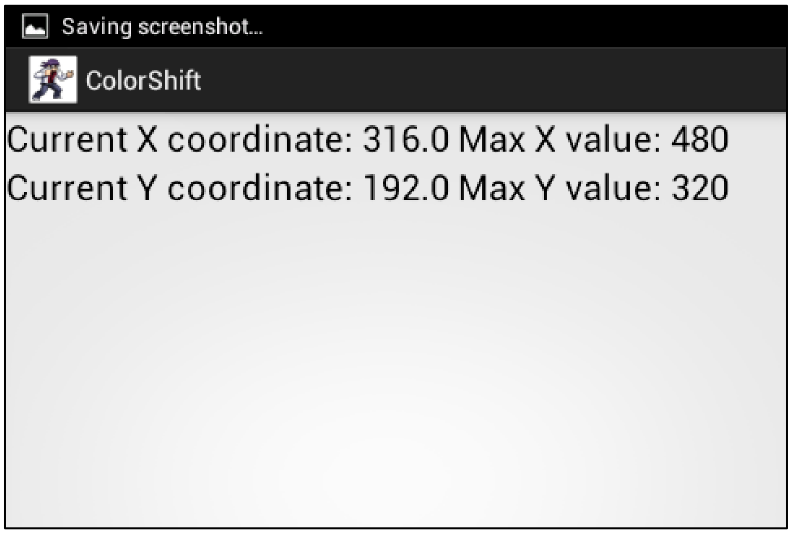
The following section will go into detail about how each of the Android applications were developed and implemented throughout the project. The code referenced in the section below can be found in a digital data archive accessible through Professor Looft. The applications we discuss below can be run on any device that is running Android 4.0.3 (Ice Cream Sandwich). These apps were developed and tested using the Samsung Galaxy Stratosphere II (Samsung, 2013) and Sony Xperia Mini (Sony Mobile Communications AB, 2013) smart phones.

Hello World Application

When first beginning to program in an unfamiliar language or development environment, it is often good to set up a program that outputs the string "Hello World" to the user. We decided to use this approach for initial development of applications for the Android development board. Starting with the Hello World application allowed us to understand the basic flow of an Activity in Android. To get started we first had to install the Android Software Development Kit (SDK). The Android SDK is a collection of examples and libraries made available to developers to aid them when developing applications for the Android OS (Google Inc., 2013). We also needed to install a development environment in which to code. The development environment could have been any one of the many environments to choose from, for example, one could also use Eclipse, or more specifically, the Android Development Tools (ADT) plugin (Google Inc., 2013). Using ADT, we were able to open and view several examples that are included with the SDK. These examples allowed us to learn how to develop code for an Android OS device and ultimately create more complex applications in the future.

Color Changing Application

During the development of our project, we realized that we would like to be able to have some applications that would interact with the user based on what they did. In this case, we decided to make a color changing application that would change the color of the screen based on where the user's finger was on the touch screen.



Current X coordinate: 316.0 Max X value: 480
Current Y coordinate: 192.0 Max Y value: 320

Figure 16-Screenshot of Color Changing Application

The color changing application works by first dynamically fetching the "Display Metrics" of the Android device the application is running on. The display metrics give us the maximum X and maximum Y screen coordinates. Once we have the maximum coordinate values we then scale the current finger press location coordinates to be a percentage of the maximum values. Once we have the scaled values we use a method in the Color library called HSVToColor to translate the scaled coordinate values into some linear expression of color (This method returns an integer in standard αRGB format). Once we have the color integer we call the setBackgroundColor method to update the background color to that of the color integer.

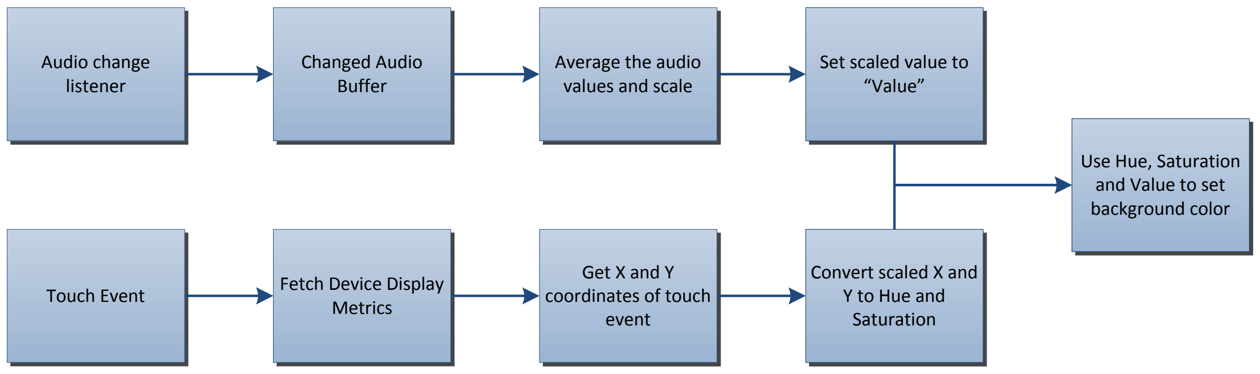

Audio change listener
Changed Audio Buffer
Average the audio values and scale
Set scaled value to "Value"
Use Hue, Saturation and Value to set background color
Touch Event
Fetch Device Display Metrics
Get X and Y coordinates of touch event
Convert scaled X and Y to Hue and Saturation

Figure 17: Color Changing Application Code Flow

In the later stages of our project, we decided to make this application slightly more complicated by sensing the ambient noise from the device microphone and using this data to affect the color of the background. This part of the application actually introduced significant complexity because the program now tries to pass audio information in "semi-real" time. Adding the audio in to this application required the development of Variable Change Listeners to look at the audio buffer regularly to check to see if it has changed. Ultimately, we used the audio data by averaging the absolute value of the audio buffer values, scaling that value, and then setting it as the "Value" in the Hue Saturation Value model of color expression.

Accelerometer Application

While developing for the Android device we thought that it would be good to show that we could poll data from the android device's onboard sensors. The accelerometer read application is a very basic implementation of the onboard sensor querying methods, which are natively available to android devices. This application generates a button on screen and waits for the user to press the button. Once the user has pressed the on screen button, the device queries the onboard accelerometer and retrieves the x, y, and z accelerations in m/s^2. Fortunately, the Android OS provides data as a float value directly instead of necessitating conversion from some analog value.

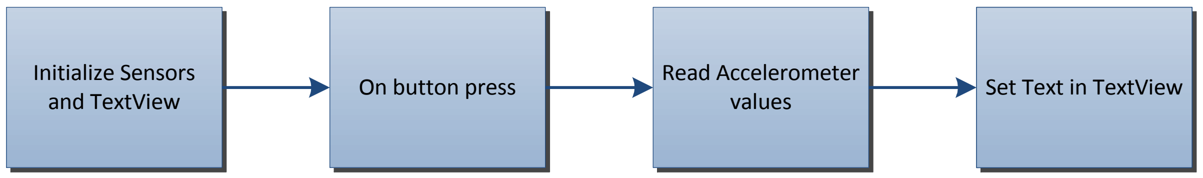
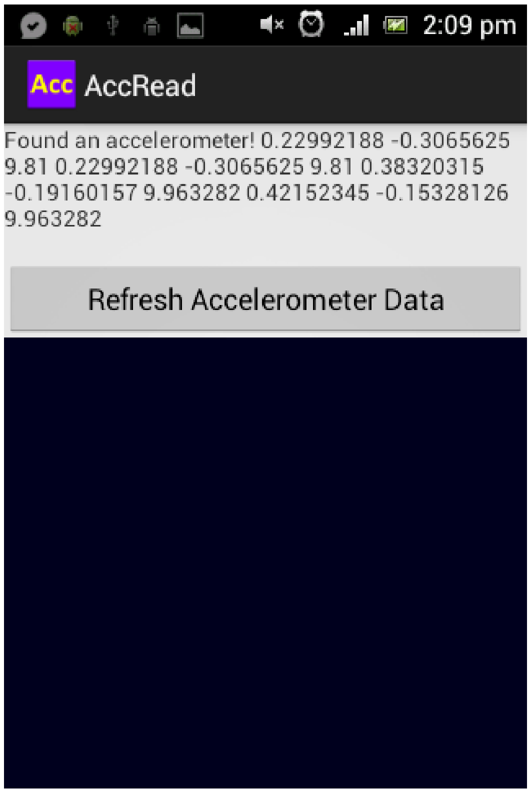


Figure 18: Onboard Accelerometer Application Code Flow



```
Found an accelerometer! 0.22992188 -0.3065625
9.81 0.22992188 -0.3065625 9.81 0.38320315
-0.19160157 9.963282 0.42152345 -0.15328126
9.963282
```

Figure 19-Screenshot of Accelerometer Read Application

Bluetooth GPS Application

After our project moved into the Bluetooth development phase, we needed our Android device to communicate via Bluetooth to our MSP430 module. In order to do this we needed to set up an application that could transmit commands and receive a data stream. After trying to teach ourselves about how Android communicates over Bluetooth based on many StackOverflow.com posts, we tried to put together a simple data stream reading application. After attempting to develop a working data stream read app we found a sample application in the Android SDK called “Bluetooth Chat”. After installing this application on the device and changing a few parameters specific to our setup, we were able to communicate with a BT-Q1000 Bluetooth GPS device. When the user opens the application, a prompt will appear to ask the user if they want to turn on the Bluetooth adapter (if it is not on already). Next, the user can look through a list of paired Bluetooth devices and select one with which to try to initialize a connection. After a connection is established, the application printed the received data stream in a series of messages on the screen of the Android device.

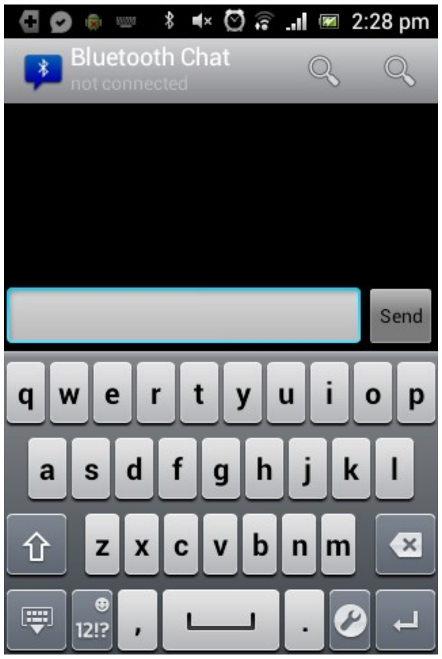


Figure 20-Screenshot of Bluetooth Chat App

Data Graphing Application

The purpose of the data graphing application is to get a feel for how to use the graphing libraries that are available to the Android operating system. This application gave us a background for how to use various data plotting libraries to graph the incoming data stream from our MSP430.

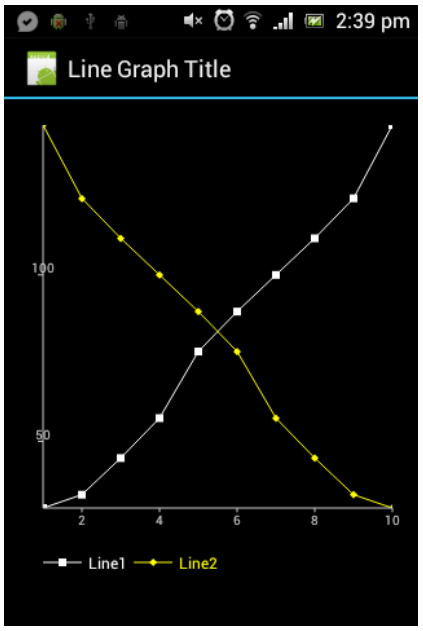


Figure 21-Screenshot of Chart Application

5.3 I²C Controller

In order for the software to communicate with the sensor modules, we needed an I²C bus controller. The controller would serve the purpose of tying all the devices together in a way that the Android software could communicate easily with every module.

Android Development Board

The first approach we attempted was to connect the I²C bus directly to the development board. The advantage to this solution was that the development board already had an I²C port labeled on it. Unfortunately, after researching methods to interface with this port, we were unable to determine any simple way to do so. Since the development-board was running the Android operating system, access to low-level communications ports were buried inside the Android kernel. This abstraction of software from hardware is generally an advantage on the Android, since it provides a clean and uniform platform on which applications can be run. However, for the purposes of this project, the abstraction served as a staunch barrier, which ultimately shifted the direction of the project.

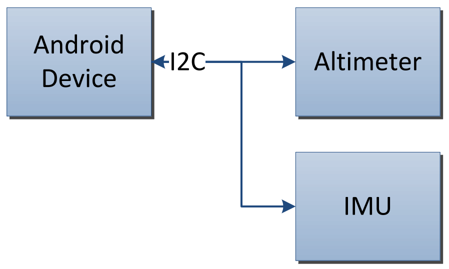


Figure 22: Android to Sensors – Via Direct Connection

After determining that the I²C port of the Android development board was prohibitively difficult to interface with, we considered other communication options. The option that seemed most appealing was the Android operating system's native support of Bluetooth. Once we had decided to use Bluetooth as the new communication method, the question then became how to interface Bluetooth with an I²C bus.

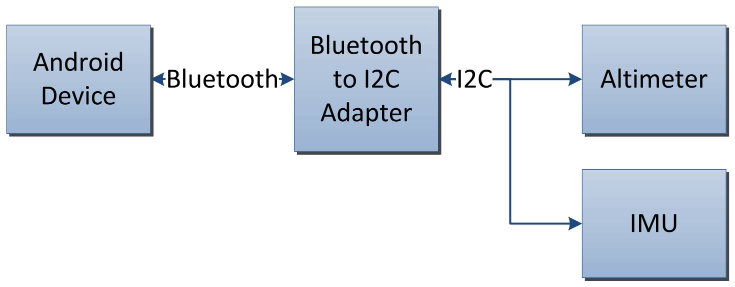


Figure 23: Android to Sensors - Via Bluetooth/I²C Adapter

Bluetooth I²C Master

Our first attempt to solve the Bluetooth to I²C conversion was to buy a Bluetooth module that would act as an I²C master. That is to say, any device could connect to this Bluetooth module, and the module would accept commands that would be converted to I²C commands and pushed out to the bus.

Unfortunately, our research showed that only one company has ever made a device such as this, and the device has long since been discontinued. This meant that it would be necessary to implement our own device to convert between a Bluetooth module and the I²C bus.

MSP430 Microcontroller

The MSP430 microcontroller has an onboard I²C module that was used to test each of the external sensor modules. Naturally, we considered using the MSP430 as the Bluetooth to I²C converter. We were fortunate enough to find a Bluetooth module designed for the MSP430 Launchpad Development board. The Bluetooth module communicated with the MSP430 via UART. Figure 24 shows this configuration.

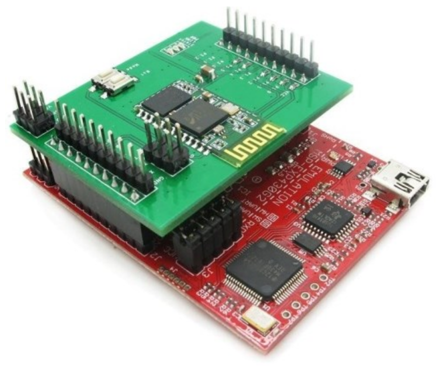


Figure 24: BT BoosterPack with Launchpad[12]

5.4 I^{2}C Sensor Modules

By using the MSP430 microcontroller to control the I²C bus, we were able to test each of the following modules.

Altimeter

The first module we tested with was the altimeter. The altimeter provided an analog-to-digital readout from a temperature and a pressure sensor. Additionally, the altimeter gave the user the option to choose the oversampling ratio. This allowed the user to choose the tradeoff between a low oversampling ratio (lower resolution, but faster to computer) and a high oversampling ratio (high resolution, slow to compute).

One issue with the altimeter was that it did not follow the de-facto I²C standard. While the I²C specification allows a master to send or receive an arbitrary amount of data to or from a slave, a more specific way of sending and retrieving data has been widely adopted. Generally, the master will first send a single byte with a register address. Next, the slave either reads or writes to or from that register on the slave. This provides a uniform interface between the master and many slaves, which use this same method for communicating data. The altimeter did not conform to this unwritten standard, which created an obstacle to writing clean and effective I²C drivers on the MSP430 microcontroller. Specifically, in addition to the capability to read and write to slave registers, the I²C drivers needed to be able to accept variable length responses when communicating with the altimeter.

[12] http://imall.iteadstudio.com/bt-boosterpack.html

Another problem we encountered with the altimeter was that it became stuck in the I²C acknowledge state. This means that any time it was powered on, the device would force the I²C data line to a low voltage. This low voltage caused the entire I²C bus to stall. The manufacturer has documented this infrequently occurring issue. As a result, we were able to follow the manufacturer's instructions to reset the altimeter from this state and thus solve the issue.

GPS

The next module we considered interfacing with was the GPS. While the GPS at first seemed promising, we found that the module came with very sparse documentation. Furthermore, it was the only GPS module we could find which communicated over I2C. This eliminated the possibility of finding a replacement GPS. Given more time, we could perhaps find a way to implement the GPS module. Alternatively, given more time, we could consider a solution that involved a GPS communicating via UART, which is standard for Bluetooth modules. This would greatly increase the complexity of the data acquisition since it would require a separate set of drivers and a separate UART port on the microcontroller. This is particularly a problem on the MSP430 currently in use. The final MSP430 configuration is only equipped with two communication ports. One of them uses I²C to communicate with the external components while the other uses non-addressable UART to communicate with the Bluetooth module. Thus, adding any non-I²C device (i.e. a typical GPS unit) would require upgrading the microcontroller.

IMU

Finally, after having troubles with the GPS module, we interfaced the IMU to the MSP430. Unlike the altimeter, the IMU used the standard I²C format where the master always sends a register address to the slave, and then either reads or writes at that address. Figure 25 shows this de-facto standard.

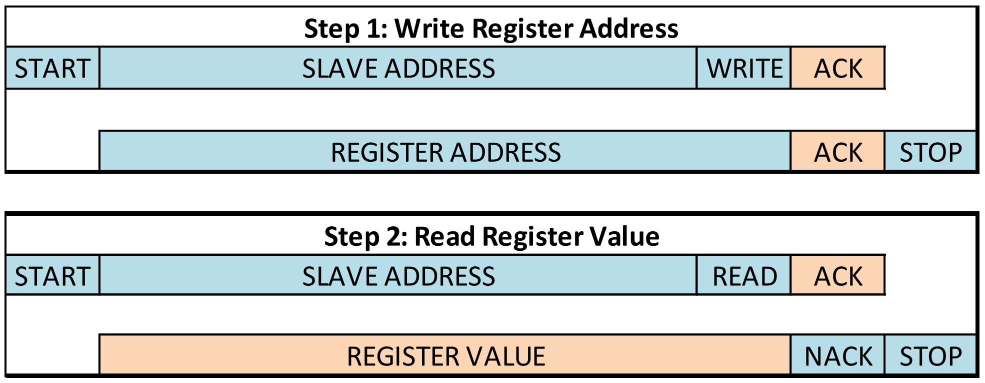


Figure 25: Example of Register-Based Read (Blue: Master; Orange: Slave)

Additionally, the IMU supports sequential reads. That is to say that if the master reads a register address but continues reading after the value has been returned, the IMU will send the master the next sequential register address. Typically, the master would read one byte per register from the IMU. For example, it might request 0x00 and receive "4". However, if the master tries to read three bytes at 0x00, the IMU will send {"4", "5", "6"}.

Table 4: Example of Sequentially Available Data

Register (1 byte)	Value (1 byte)
0x00	4
0x01	5
0x02	6

Sequential reads are encouraged because the IMU updates the register values periodically. Reading register values in a sequential manner ensures that each received value is from the same time sample. This is particularly important because the IMU uses two registers to store acceleration data for each axis. Ultimately, this flexibility in allowing sequential reads is what enabled us to write drivers that could read the altimeter's variable length data fields in addition to interfacing with this IMU.

5.5 Power

Four 1.2V 2100mAh AA Nickel Metal Hydride rechargeable batteries power Wilson. The batteries are trickle charged by a 6V solar panel at up to 167mA. This combination of rechargeable batteries and solar charging allow Wilson to be energy independent. The main components that need power are the Launchpad and MSP430, the altimeter, the Bluetooth transmitter, and the accelerometer.

Table 5: Component Power Analysis

Component	Voltage	Current	Power
Launchpad & MSP430	3.5 V	2 mA	7 mW
Altimeter MS5607	3.5 V	1 µA	3.5 µW
IMU MPU-6050	3.5 V	5.5 mA	19 mW
BT BoosterPack	3.5 V	30 mA	105 mW
Parallax 6V Solar Panel	6 V	166.6 mA (peak)	1 W (peak)

When Wilson is running, it uses 131 mA. Thus, when solely the 2100mAh batteries power the device, it can run for about 16 hours.

Additionally, the solar panel is able to charge the batteries at up to 167mA. This means it would take about 20 hours to recharge the batteries fully. While this seems like a long time, it is important to note how little the batteries are actually being drained by the application, and even still, they are only significantly drained when the device is use.

5.6 Summary

Overall many different approaches were taken to find a clear path to designing the final product. The difficulties in Android development and I^{2}C driver design were two difficult hurdles that were overcome during the Wilson development. The MSP430, which initially was used to test the different sensors, has become part of the final design. Using I2C, the MSP430 reads the data from the altimeter and the IMU and transmits the information through a Bluetooth adapter to the Android development board. Our Android application on the board communicates to the MSP430 via Bluetooth sending commands and receiving data, which is then displayed in a user-friendly chart application.

6 System Software and Testing

This section provides the details of our application and final system. We begin by discussing the application that takes in Bluetooth data from the MSP430 and displays that data in graphical format. We then go into detail about the final physical system as well, describing how each component is connected, and how it interacts with the system. Finally, we discuss our testing process throughout the project.

6.1 Wilson Android Application

The final Wilson Android application graphically displays the data that the Wilson device continuously sends. The final application can be broken down into three main functions. These functions, also called "activities", are the Tab Layout, Bluetooth Chat, and Charting functions.

Tab Layout

The Wilson application uses a tab layout to provide the most user-friendly interface. This is the main activity that runs the Wilson application. The application has four tabs, which are the Bluetooth Chat tab, Accelerometer tab, Temperature tab, and the Altitude tab. Each tab contains its own activity that runs when a user selects that tab.

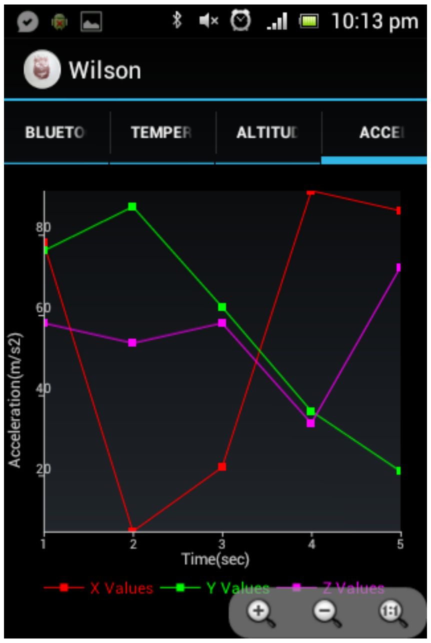


Figure 26: Tab Layout Screenshot

There was a particular issue when porting the Bluetooth Chat application into its own tab. The Bluetooth Chat application used an "Actionbar", which is a window that displays the name of the application and provides user options at the top of the screen. The conflict was that both the tab layout and the Actionbar were competing for the same space, which caused the application to crash upon opening. The Actionbar contained the button used to connect with the Wilson device. This button was necessary in order to create a Bluetooth connection between the Wilson device and the Wilson application.

The solution was to remove the Actionbar entirely and move the connect option to a different options menu. The user can access this option menu by using a hardware button found on the Android device. This was the only feasible solution to connect to the Wilson device without losing the tab layout.

Bluetooth Chat

As mentioned in the System Implementation section, we built our Bluetooth communication activity off an existing sample called Bluetooth Chat. This tab first establishes a secure RFCOMM socket, which allows two-way communication with the Wilson device. This application accepts sensor data from the Bluetooth module and I^2C controller. This application then checks the prefix and suffix for our protocol. The prefixes we are looking for are "AL" for altitude, "AC" for acceleration, and "TE" for temperature. After the prefix is recognized, we also check to see if the suffix, $, is present. If both of these are recognized the application then parse the string in order to separate the data into an array of integers. This array format is required for the graphing activity. The parser takes in the buffered message string and eliminates the prefix and suffix. The resulting string is a series of data separated by commas. The application splits the string using the comma as a delimiter. The result is stored into an array which is then is parsed into an integer using basic methods.

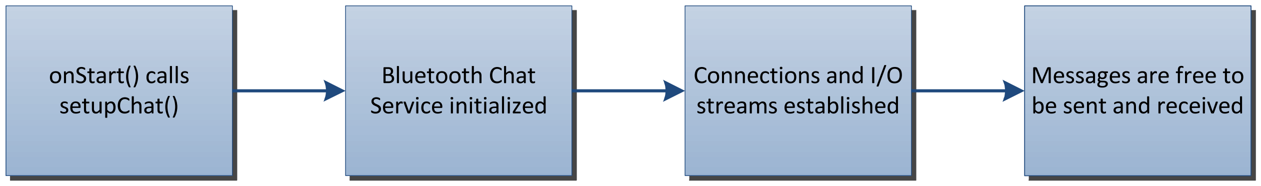

onStart() calls setupChat()
Bluetooth Chat Service initialized
Connections and I/O streams established
Messages are free to be sent and received

Figure 27: Bluetooth Chat Code Flow

Graphing

To display data received from the Wilson device, we used a software library called "aChartEngine". This library is able to create graphs from an array of integers. The charting process consists of four main activities. We will use the altimeter tab as an example to explain the graphing process. When the user selects the Altimeter tab, a new thread begins to run. The thread runs forever, requesting a new data point every second.

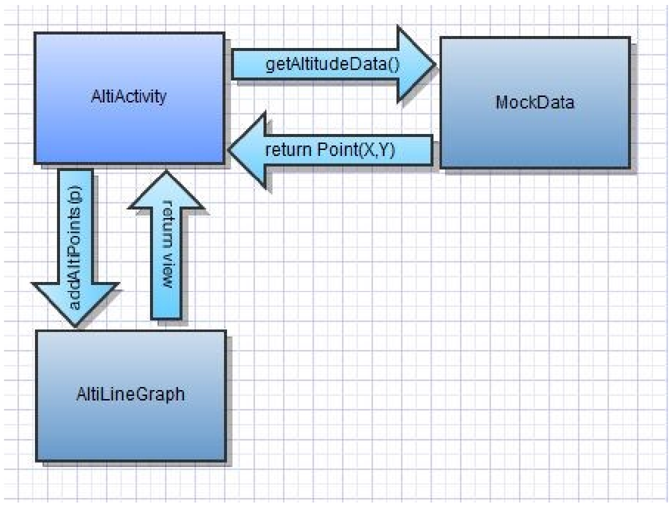


Figure 28: Graphing Code Flow Diagram

The data point is requested from the MockData class that returns the data from the Wilson device. The sensor data is received from the Bluetooth chat activity and placed in an array of integers. It is returned as a point containing the x and y value to be plotted.

From here, the data is sent to the AltiLineGraph class, which contains the formatting and setup for the chart. Options to change the type of graph, color of the line, axis labels, shape of data points, and other options can be found in this class. The x and y values are set to be plotted by the chart engine and from here the graph on the android device is refreshed and plotted with the new point.

6.2 External Component Module

The external component module consists of three parts: the I²C controller, the Bluetooth adapter, sensors.

The I²C controller runs in a continuous loop. For each iteration of the loop, the controller retrieves data from each sensor. The controller does this by using the I²C and sensor specific drivers that were developed during the system implementation. Once the I²C controller collects the data, it puts the data into a specific format that the Android application can read. FIGURE shows the format that the Android application expects.

Table 6: Android Data Packing Format (values in square brackets)

Data	Format
Temperature	TE$[temperature]$
Altitude	AL$[pressure]$
Acceleration	AC$[x-axis],[y-axis],[z-axis],[yaw],[pitch],[roll]$

Once the data is packed, the I²C controller sends it to the Bluetooth module. The Bluetooth module sends the data to the mobile Android device, which can then display the data.

6.3 Systems Testing

After completing the Android Application and the sensor software, we then had to test the interaction between the two and tweak the software in order for the system to work as expected. The Bluetooth chat must be able to receive the data in the correct format and the graphing portion of the application should be able to take the formatted data and display it correctly.

Bluetooth Chat

There were several phases of testing for the Bluetooth chat once during communication with a Bluetooth GPS, and again during the Tabbed layout and Graphing incorporation. The first round of testing consisted of simply attempting to connect to the GPS device and receive the data output from the GPS. The second round of testing was more involved because we needed to receive data from our own device. The first step to communicating with our own device was to make sure that the Bluetooth Chat App was able to communicate with it. Once a connection was established and we could communicate to our device and receive data back we could then move towards a communication protocol. Since we were in control of the data flow and protocol used to communicate, we were able to set some simple codes to buffer and receive data reliably.

Graphing

The main method for testing the graphing portion of the Wilson app was done using randomly generated values. Within the MockData class is a method that randomly generates a number between 0 and 100. This number was used as the y value in an x, y coordinate pair. We determined that the aChartEngine dynamically adjusts the y scale to the highest y value.

6.4 Summary

This section describes the various final applications that we developed. The first application is the Bluetooth chat application, which shows the user what commands are passing between the I^2C module and the graphing application. The graphing application receives data from the sensors and plots it into different tabbed windows.

7 Summary and Conclusions

The summary and conclusions section aims to provide an overview of what has been discussed throughout the paper as well as draw conclusions about the project as a whole.

7.1 Objectives

The purpose of this section is to look at our initial goals and compare them with what we accomplished. The objectives can be seen below along with a description of their result.

Interface the system to a reasonable number of sensors to provide environmental data to higher-level applications.

The Wilson device's I^2C controller collects data from two external sensors, an altimeter, and an IMU.

Develop at least one original Android application to interface with external sensors.

The Wilson application is designed to graph the data that it receives from external sensors.

Successfully interface with at least one Bluetooth device.

The Wilson application communicates with the Wilson device using a Bluetooth channel.

Include solar charging with the system.

The Wilson device is battery powered and can be recharged using a solar panel.

Consolidate components into a single prototype device.

We designed a box to house the components for the Wilson device.

7.2 Recommendations for Future Work

The final step in evaluating this project is to discuss what could be done in the future to potentially improve the project. In this case, we believe that, although this project was a good systems design experience, it did not ultimately produce a marketable product. One potential avenue of future work for this project might be adding features that would make it stand out in the market. Right now, a typical smart phone has sensors that are comparable to those on the Wilson device.

Works Cited

Elgin, B. (2005, August 16). *Google Buys Android for Its Mobile Arsenal*. Retrieved from Bloomberg Business Week: http://www.businessweek.com/stories/2005-08-16/google-buys-android-for-its-mobile-arsenal

Federal Aviation Administration. (n.d.). *Navigation Programs - WAAS - How It Works*. Retrieved April 2013, from Federal Aviation Administration: http://www.faa.gov/about/office_org/headquarters_offices/ato/service_units/techops/navservices/gnss/waas/howitworks/

Google Inc. (2013). *ADT Plugin | Android Developers*. Retrieved 2013, from Android Developers: http://developer.android.com/tools/sdk/eclipse-adt.html

Google Inc. (2013). *Android SDK | Android Developers*. Retrieved 4 15, 2013, from Android Developers: http://developer.android.com/sdk/index.html

International Data Corporation. (2013). *Android Marks Fourth Anniversary Since Launch with 75.0% Market Share in Third Quarter, According to IDC*. Retrieved March 19, 2013, from IDC: The premier global market intelligence firm: http://www.idc.com/getdoc.jsp?containerId=prUS23771812#.UUipwxy7JAo

Samsung. (2013). *Samsung Galaxy Stratosphere II*. Retrieved 2013, from Samsung US: http://www.samsung.com/us/mobile/cell-phones/SCH-I415SAAVZW

Sony Mobile Communications AB. (2013). *Xperia mini*. Retrieved 2013, from Sony Smartphones US: http://www.sonymobile.com/us/products/phones/xperia-mini/

Appendix A – Supplementary Materials

This appendix provides a list of supplemental materials held by the Wilson team's advisor, Professor Looft. To acquire any of these materials, contact Professor Looft at fjlooft@wpi.edu.

Hardware

- AM335x Development Kit
- Parallax MS5607 Altimeter
- Sparkfun MPU-6050 Inertial Measurement Unit
- Level Shifter
- USB to I^{2}C Communication Module (x2)
- Navigatron v2 GPS Receiver
- TI Launchpad with MSP430G2553 Microcontroller

Software

- Android Code
  - Hello World App
  - Onboard Accelerometer Read App
  - Bluetooth GPS Chat App
  - External Bluetooth Reader App
  - Color Changing App
  - Wilson Charting & Tab App
- MSP430 I^{2}C Controller Code
  - UART Drivers
  - I^{2}C Drivers
  - Altimeter Drivers
  - IMU Drivers

Research and Misc.

- Block Diagrams
- Component Trade Study
- All Android Application Code
- All MSP430 Application and Drivers Code