Remote iOS Monitoring System

Project Design Report

Design Team 08

Pierre Hall
Benjamin Kasmin
Philip Steele
Yikun Wang

Dr. Subramaniya Hariharan

November 15, 2012
# Table of Contents

Abstract ............................................................................................................... 1

1.0 Problem Statement .................................................................................. 1
   1.1 Need ........................................................................................................... 1
   1.2 Objective .................................................................................................. 1
   1.3 Background .............................................................................................. 2
   1.4 Marketing Requirements ......................................................................... 4
   1.5 Objective Tree ......................................................................................... 4

2.0 Design Requirement Specification ......................................................... 5
   2.1 Calculations .............................................................................................. 7
      2.1.1 Measured Variables ........................................................................... 7
      2.1.2 Velocity Calculations ....................................................................... 7
      2.1.3 Calculated Variables and Equations ............................................... 8
      2.1.4 Additional Calculation Information ............................................... 8

3.0 Accepted Technical Design ...................................................................... 9
   3.1 Level 0 Software Block Diagram ......................................................... 9
   3.2 Level 0 Hardware Block Diagram ......................................................... 10
   3.3 Level 1 Software Block Diagram ......................................................... 11
   3.4 Level 1 Hardware Block Diagram ......................................................... 12
   3.5 Level 2 Software Block Diagrams ....................................................... 14
      3.5.1 Level 2 Software Server Module .................................................. 14
      3.5.2 Level 2 Software Wireless Module ............................................... 15
      3.5.3 Level 2 Software App Module ...................................................... 15
      3.5.4 Level 2 Software Digest ............................................................... 16

3.6 Level 2 Hardware Block Diagrams .......................................................... 18
   3.6.1 Level 2 Hardware Normalizing PCB ................................................ 18
   3.6.2 Level 2 Hardware Microprocessor Module ..................................... 19
   3.6.3 Level 2 Hardware Digest .................................................................. 20

3.7 Hardware Schematics and Theory of Operation ..................................... 21
   3.7.1 Arduino Connection Map ................................................................... 21
   3.7.2 RiOS Interconnect Schematic ............................................................ 22
   3.7.3 Normalization PCB Interface ............................................................ 23
   3.7.4 Speed Sensing Circuit ....................................................................... 24
   3.7.5 Thermal Sensor Circuit ..................................................................... 25
   3.7.6 Voltage/Current Sensing Circuit ...................................................... 26
   3.7.7 Sensor Normalization Amplification Circuit ...................................... 27

3.8 Software Flow Diagrams and Theory of Operation ................................ 29
   3.8.1 State of Charge Calculation ............................................................... 29
   3.8.2 Discharging (Serial Connection) Flow Diagram ................................ 30
   3.8.3 Server module ................................................................................... 32
   3.8.4 Data Integrity Flow Chart ................................................................. 34
   3.8.5 System Context Diagram .................................................................. 35
   3.8.6 Data Flow Diagram ........................................................................... 36
   3.8.7 Application Development ................................................................ 37

4.0 Parts List .................................................................................................... 42
5.0 Project Schedules .................................................................................................................. 43
6.0 Design Team Information ................................................................................................... 44
7.0 Conclusions and Recommendations ................................................................................. 45
8.0 References .......................................................................................................................... 45
9.0 Appendices .......................................................................................................................... 46

List of Figures

Figure 1: Objective Tree ............................................................................................................. 5
Figure 2: Level 0 Software Block Diagram ............................................................................... 9
Figure 3: Level 0 Hardware Block Diagram ............................................................................. 10
Figure 4: Level 1 Software Block Diagram ............................................................................. 11
Figure 5: Level 1 Hardware Block Diagram ............................................................................. 12
Figure 6: Level 2 Software Block Diagram; Server Module ................................................... 14
Figure 7: Level 2 Software Block Diagram; Wireless Module ................................................ 15
Figure 8: Level 2 Software Block Diagram; App Module ......................................................... 15
Figure 9: Level 2 Hardware Block Diagram; Normalizing PCB .............................................. 18
Figure 10: Level 2 Hardware Block Diagram; Microprocessor ............................................... 19
Figure 11: Arduino Connection Map ......................................................................................... 21
Figure 12: Schematic of Box Interconnect .............................................................................. 22
Figure 13: Normalization PCB Interface .................................................................................. 23
Figure 14: Speed Sensing Circuit ............................................................................................. 24
Figure 15: Output Voltage vs. Hall Voltage .......................................................................... 24
Figure 16: Thermal Sensor Schematic ..................................................................................... 25
Figure 17: Simulation of Thermal Sensor Output Voltage ....................................................... 25
Figure 18: Current Sensing Circuit .......................................................................................... 26
Figure 19: Voltage Measurement Divider ............................................................................. 26
Figure 20: Simple Non-Inverting Amplifier Circuit ................................................................. 27
Figure 21: State-of-Charge Calculation Pseudocode .............................................................. 29
Figure 22: Redpark SDK Overview ......................................................................................... 30
Figure 23: Sample Python Script ............................................................................................. 31
Figure 24: Data Integrity Flow Chart ...................................................................................... 34
Figure 25: System Context Diagram ....................................................................................... 35
Figure 26: Data Flow Diagram ............................................................................................... 36
Figure 27: Application Storyboard ......................................................................................... 37
Figure 28: Table View Controller ........................................................................................... 38
Figure 29: Tabbed View Controller; Discharging State ........................................................... 39
Figure 30: Tabbed View Controller; Charging State ............................................................... 40
Figure 31: Tabbed View Controller; Advanced ...................................................................... 41
Figure 32: Fall 2012 Gantt Chart ............................................................................................ 43
Figure 33: Spring 2013 Gantt Chart ......................................................................................... 44
**List of Tables**

Table 1: Design Requirements ................................................................. 6
Table 2: Level 0 Software Module ............................................................ 9
Table 3: Level 0 Hardware Module ............................................................ 10
Table 4: Level 1 Software Module; Microprocessor .................................. 11
Table 5: Level 1 Software Module; Server ................................................ 12
Table 6: Level 1 Hardware Module; Interface PCB ..................................... 13
Table 7: Level 1 Hardware Module; Wireless ........................................... 13
Table 8: Level 1 Hardware Module; Microprocessor .................................. 13
Table 9: Level 2 Software Module; Server ................................................ 14
Table 10: Level 2 Software Module; Wireless .......................................... 15
Table 11: Level 2 Software Module; Application .................................... 16
Table 12: Level 2 Hardware Module; Normalizing PCB .......................... 18
Table 13: Level 2 Hardware Module; Microprocessor ............................. 19
Table 14: Database Server Performance Diagnostic ............................... 32
Abstract

There is a need for an easily implementable system dedicated to monitoring and manipulation of data regarding battery characteristics of an electric vehicle from remote locations. In response to this dilemma, the Remote iOS (RiOS) will be able to remotely monitor these characteristics to provide an iOS device with necessary information so that a user may make appropriate action in response to this data. This will be accomplished using a network of microprocessor controlled sensors, coupled with cellular communication to broadcast data wirelessly to any iOS device. While it is connected locally, the RiOS will automatically double as a user defined instrument cluster for any host device. This provides the user with real time, custom information regarding the electric vehicle status.

1.0 Problem Statement

1.1 Need

There is a need for a convenient and effective method to remotely monitor the charging characteristics and battery health of an electric vehicle that is convenient, cost-effective, and simple to integrate into an existing vehicle.

1.2 Objective

The objective of this project is to design and build a system to remotely monitor the battery condition of an electric vehicle from a smartphone. The design process will consist of writing the application code and integrating it with the hardware to be installed on the vehicle. The build will consist of taking the design and performing tests to ensure its proper functionality.
1.3 Background

Patent: Battery Monitoring

An architecture for organizing information in a battery monitoring system. It consists of a hierarchy of the levels, in which the first level contains information relating to the battery variables. The next level contains information on battery analysis, and the third level contains information on battery-specific maintenance parameters. The fourth and final level in the hierarchy records measurements on the capacity of the battery.

Patent: Battery Monitoring Device and Batteries

A communication means between the battery monitoring device that receives information from the monitoring device host system. Upon receipt of the measuring parameters, the monitoring host system signals the actual battery monitoring device to perform the measurements, then converts the information to digital serial data. Essentially, if the type of battery and/or construction layout and battery configuration changes, the state of charge can be accurately measured and monitored by this battery monitoring device.

Electrical Engineer’s Reference: Chapter 11, Section 12

To collect raw data to be later processed, there is a need for a network of sensing devices that will be able to detect a physical response, and convert it into an electrical signal. The devices used in this process (Transducers) are clearly explained in chapter 11.12 of the Electrical Engineer’s Reference Book, 16th Edition. The sensors that will be used in this project will be both passive, and active in nature. They will be used to transfer the physical signals of heat (Passive) and speed (Active) into electrical responses which can then be sent as raw serial data to the primary monitoring system.
Transducers will be ideal for this project because they will supply the Arduino transmitter with the proper style of raw data. This data will then be available for monitoring, and processing on the other end of the communication channel.

IEEE article: Application for battery monitoring

This article discusses a PC application developed by the authors, which is used to monitor battery status remotely over Wi-Fi using two levels of controls. The lower level of control is a board with an 8-bit microprocessor, and the high-level control is a board with a 32-bit MCU with NAND Flash and main memory (Access Point). The application receives data from the access point router and is able to send requests over Wi-Fi to check different monitoring sensors. The article demonstrates that it is possible to use a Wi-Fi connection as the bridge to transfer data to an application accurately in real-time.
1.4 Marketing Requirements

1 – Must integrate easily into the charging system of the host device
2 – Pushes data remotely over a wireless connection
3 – Data should be processed in real-time
4 – Integrates with iOS device (software application will be written for an iPhone)
5 – Relatively inexpensive
6 – Intuitive—should require no particular skills or background for the user to learn
7 – Can be utilized when the host device itself is not operating
8 – Monitoring device can be powered off of host electronics, or separately powered
9 – Measures data statistics accurately
10 – Properly calculates battery conditions and parameters with multiple battery types

1.5 Objective Tree

The figure below is a graphical representation of the marketing requirements; it establishes a hierarchy for the needs and priorities of the design.
2.0 Design Requirement Specification
<table>
<thead>
<tr>
<th>Marketing Requirements</th>
<th>Engineering Requirements</th>
<th>Justification</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,4,6</td>
<td>The monitoring device shall be constructed in such a way that it will not jeopardize the</td>
<td>No accidents should occur with the installation of the monitoring device</td>
</tr>
<tr>
<td></td>
<td>safety of the host system</td>
<td></td>
</tr>
<tr>
<td>9,10</td>
<td>The monitoring device shall accurately approximate battery voltage and current with an</td>
<td>The pack voltage and current must be known in order to perform proper system</td>
</tr>
<tr>
<td></td>
<td>error of ±1%</td>
<td>analysis</td>
</tr>
<tr>
<td>9,10</td>
<td>The monitoring device shall closely monitor the peak temperature of the battery pack</td>
<td>The temperature of the pack will need to be known in order to inform the user</td>
</tr>
<tr>
<td></td>
<td>with an error of ±1 deg Celsius</td>
<td>of unsafe operating conditions</td>
</tr>
<tr>
<td>1,4,8</td>
<td>The monitoring device shall supply 5V max and 1A max for iOS devices</td>
<td>If these rated conditions are exceeded, the iOS device(s) will potentially be</td>
</tr>
<tr>
<td></td>
<td></td>
<td>damaged</td>
</tr>
<tr>
<td>1, 7, 8</td>
<td>Monitoring device must be able to function from the operating power of the host system</td>
<td>In order to function, the system must be able to operate at host system output</td>
</tr>
<tr>
<td></td>
<td></td>
<td>power</td>
</tr>
<tr>
<td>1, 3, 4, 9, 10</td>
<td>When connected locally (in the host system's discharging state), the monitoring device</td>
<td>Real-time data is necessary to display the intended functions</td>
</tr>
<tr>
<td></td>
<td>application should display real-time data</td>
<td></td>
</tr>
<tr>
<td>2, 4, 7, 9, 10</td>
<td>When connected wirelessly (in the host system's charging state), the monitoring device</td>
<td>Gives the user appropriate information to take action regarding the status of</td>
</tr>
<tr>
<td></td>
<td>application shall refresh sensor data every 5 minutes or until requested by the user</td>
<td>the battery</td>
</tr>
<tr>
<td>4, 5, 6</td>
<td>The monitoring device application should be intuitive</td>
<td>In order to be operated by a diverse set of users, the application must be</td>
</tr>
<tr>
<td></td>
<td></td>
<td>intuitive, with a short learning curve</td>
</tr>
<tr>
<td>1, 6</td>
<td>The monitoring device shall be no greater than 110 cubic centimeters in volume</td>
<td>Monitoring device should be as small as possible and as light as possible</td>
</tr>
<tr>
<td></td>
<td></td>
<td>without compromising its intended functionality</td>
</tr>
<tr>
<td>1,3,4,6</td>
<td>The monitoring device application shall display the battery pack voltage, temperature,</td>
<td>The user must know these quantities to maximize operability</td>
</tr>
<tr>
<td></td>
<td>and the current drawn by the host system</td>
<td></td>
</tr>
<tr>
<td>9,10</td>
<td>The monitoring device application shall calculate and display the host system's state of</td>
<td>The state of charge needs to be known to indicate to the user the amount of</td>
</tr>
<tr>
<td></td>
<td>charge</td>
<td>available energy stored</td>
</tr>
<tr>
<td>1,4,6,10</td>
<td>The monitoring device application must allow the user to define different battery</td>
<td>The monitoring device must be compatible with multiple battery chemistries</td>
</tr>
<tr>
<td></td>
<td>specifications</td>
<td></td>
</tr>
<tr>
<td>3,4,10</td>
<td>The monitoring device application shall define the estimated remaining distance left of</td>
<td>The user must have an estimate of remaining useable range in order to take</td>
</tr>
<tr>
<td></td>
<td>the host system</td>
<td>appropriate action</td>
</tr>
</tbody>
</table>

Table 1: Design Requirements
2.1 Calculations

2.1.1 Measured Variables

\( \text{Temp}(t): \) Temperature as a function of time

\( \dot{X}(t): \) Velocity as a function of time

\( V(t): \) Voltage as a function of time

\( I(t): \) Current as a function of time

2.1.2 Velocity Calculations

One of the most important values required for operation of a motor vehicle is the value of the vehicle’s velocity. This value will enable the operator to make proper considerations to control the vehicle. The classical method to determine linear velocity is

\[ V \left[ \frac{m}{s} \right] = v_i + a \times t, \]

where \( V \) is the instantaneous velocity, \( a \) is the acceleration, and \( t \) is the duration of time for which the body has been accelerating.

For this design, however, the calculation of linear velocity has to be measured in terms of rotational velocity. This calculation is accomplished using the below equation:

\[ V \left[ \frac{m}{s} \right] = C \times RPM \tag{1} \]

Thus, the velocity is equal to the circumference \( C \) of the rotating body multiplied by the rotations per minute \( RPM \). For the structure of the circuit that is measuring rotations, one pulse of the circuit represents one rotation. Given that the circumference of a circle is \( 2\pi \) multiplied by the radius of the circle, and with observance to SI units, the calculation of linear velocity can be expressed by (2) below:

\[ V \left[ \frac{m}{s} \right] = 4\pi \frac{p}{60} \tag{2} \]

where \( p \) is the number of pulses produced at the output.
2.1.3 Calculated Variables and Equations

\[ Power \ [P(t)] = V(t) \times I(t) \]  \hspace{1cm} (3)

\[ Torque \ [T(t)] = \frac{P(t)}{\dot{X}(t)} \]  \hspace{1cm} (4)

\[ Energy \ [E(t)] = \int P(t) \, dt \]  \hspace{1cm} (5)

\[ Energy \ stored \ [E_{\text{stored}}] \approx V \times Q \]  \hspace{1cm} (6)

\[ Charge \ [Q] = C \times V \]  \hspace{1cm} (7)

\[ Stored \ Capacitance \ [C_s] = \frac{\Delta Q}{\Delta V} \]  \hspace{1cm} (8)

\[ Efficiency \ [{\eta}] = \frac{P}{\dot{X}} \]  \hspace{1cm} (9)

\[ State \ of \ Charge \ [SOC] = \int I(t) \, dt \]  \hspace{1cm} (10)

2.1.4 Additional Calculation Information

In order to produce accurate outputs, certain battery information will need to be known in advance for the app to replicate a traditional battery calibration process. The app will be pre-programmed with “common” models on standard battery chemistries in the event that this information is easily obtained on the host battery, and to eliminate unnecessary calibration time before using the system. The alternative to pre-programming is an actual calibration period where the app collects and stores data to generate a standard model of the unknown battery characteristics. This will allow the app to make proper suggestions on the health and/or state of different battery types and configurations.
3.0 Accepted Technical Design

3.1 Level 0 Software Block Diagram

Figure 2 below is the Level 0 Software Block Diagram. It shows that the input will be the serial data from the hardware (micro controller) along with data from the server database. The output will be the vehicle status in both the charging and discharging states shown on iOS device's screen. Our iOS app will process the serial input data and output the calculated results in an intuitive, user-friendly fashion.

![Remote Monitoring System Software Block Diagram](image)

Figure 2: Level 0 Software Block Diagram

<table>
<thead>
<tr>
<th>Module</th>
<th>Remote Monitoring System Software</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inputs</td>
<td>Serial Data from Hardware</td>
</tr>
<tr>
<td>Outputs</td>
<td>Status displayed on iOS Device</td>
</tr>
<tr>
<td>Functionality</td>
<td>Receives Data, Performs Calculations, and Displays Status on iOS Device</td>
</tr>
<tr>
<td>Designers</td>
<td>Yikun Wang</td>
</tr>
</tbody>
</table>

Table 2: Level 0 Software Module
3.2 Level 0 Hardware Block Diagram

Figure 3 shows the Level 0 Hardware Block Diagram of the RiOS. The RMS will take analog sensor data and convert this data into a useable format. This format will be serial data, which is capable of being transmitted through a wireless or local communication channel. In this manner, the iOS application will be able to receive information to perform calculations in real-time and display the desired data to the user.

![Remote Monitoring System Hardware Block Diagram](image)

**Figure 3: Level 0 Hardware Block Diagram**

<table>
<thead>
<tr>
<th>Module</th>
<th>Remote Monitoring System Hardware</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Inputs</strong></td>
<td></td>
</tr>
<tr>
<td>Analog Sensor Data</td>
<td></td>
</tr>
<tr>
<td>? VDC</td>
<td></td>
</tr>
<tr>
<td><strong>Outputs</strong></td>
<td>Serial/Wi-Fi Data Streams</td>
</tr>
<tr>
<td><strong>Functionality</strong></td>
<td>Receives Analog Sensor Data and provides a Serial Data Stream to a Communication</td>
</tr>
<tr>
<td><strong>Designers</strong></td>
<td>Benjamin Kasmin</td>
</tr>
</tbody>
</table>

**Table 3: Level 0 Hardware Module**
3.3 Level 1 Software Block Diagram

Figure 4 shows the Level 1 Software Block Diagram of the RiOS. The RiOS will take analog sensor data and convert this data into a useable format. This format will be serial data, which is capable of being transmitted through a wireless or local communication channel. In this manner, the iOS application will be able to receive information to perform calculations in real-time and display the desired data to the user.

Two defined code sets will be utilized to discriminate data in the different usage modes of the system, (charging while vehicle is stored and discharging, which is during use of the vehicle) each of which is user selectable from the iOS application interface. In this manner, code can be clearly written and optimized for each operation mode. When the vehicle is charging, data will be transmitted wirelessly from the RiOS hardware and queried to the application via a server. When the vehicle is discharging, the iOS device is wired to the RiOS hardware, and thus data will be transmitted by way of a wired serial protocol for improved data integrity and real-time processing of data.

![Figure 4: Level 1 Software Block Diagram](image)

<table>
<thead>
<tr>
<th>Module</th>
<th>Microprocessor Parsing Module</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inputs</td>
<td>Serial Data from Processor</td>
</tr>
<tr>
<td>Outputs</td>
<td>Internal Data Stream (Real-Time)</td>
</tr>
<tr>
<td>Functionality</td>
<td>Retrieves and Parses Data</td>
</tr>
<tr>
<td>Designers</td>
<td>Yikun Wang</td>
</tr>
</tbody>
</table>

Table 4: Level 1 Software Module; Microprocessor
### 3.4 Level 1 Hardware Block Diagram

The RiOS will take analog sensor inputs and convert these inputs into separate serial data outputs to be utilized by the iOS device. This process occurs by utilizing a “normalizing” interface PCB that proportionally converts measured values into scaled 0-5V pulses that can be processed further by a microprocessor. The microprocessor will parse and process the input data in such a manner that each signal can be output as a serial representation digitally.

The microprocessor then feeds data to the iOS device in two states, with two separate interfaces; one local communication channel and the other remotely via a wireless connection. The local communication channel (serial cable) will be utilized while the host device is running, and the wireless communication channel while the host is idle and charging. This streamlines the calculation methodology by allowing simultaneous processing of signals to be separated by the state of the RiOS and the connection type of the iOS device.

![Figure 5: Level 1 Hardware Block Diagram](image-url)
### Table 6: Level 1 Hardware Module; Interface PCB

<table>
<thead>
<tr>
<th>Module</th>
<th>Interface PCB Module</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inputs</td>
<td>Analog Sensor Data</td>
</tr>
<tr>
<td>Outputs</td>
<td>Normalized Sensor Data, 0-5V P-P</td>
</tr>
<tr>
<td>Functionality</td>
<td>Proportionally Scales Sensor Measurements to be Read and Processed</td>
</tr>
<tr>
<td>Designers</td>
<td>Benjamin Kasmin</td>
</tr>
</tbody>
</table>

### Table 7: Level 1 Hardware Module; Wireless

<table>
<thead>
<tr>
<th>Module</th>
<th>Wireless Module</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inputs</td>
<td>Processed Data from Microprocessor Internal Bus</td>
</tr>
<tr>
<td>Outputs</td>
<td>Wireless Serial Data for iOS Device</td>
</tr>
<tr>
<td>Functionality</td>
<td>Wireless Gateway to Transmit Data Remotely to iOS Device</td>
</tr>
<tr>
<td>Designers</td>
<td>Benjamin Kasmin, Philip Steele</td>
</tr>
</tbody>
</table>

### Table 8: Level 1 Hardware Module; Microprocessor

<table>
<thead>
<tr>
<th>Module</th>
<th>Microprocessor Module</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inputs</td>
<td>Normalized Sensor Data, Power</td>
</tr>
<tr>
<td>Outputs</td>
<td>Processed and Parsed Serial Data Stream</td>
</tr>
<tr>
<td>Functionality</td>
<td>Processes and Calculates Analog Inputs; Locally Streams Serial Data to External Transmission Devices</td>
</tr>
<tr>
<td>Designers</td>
<td>Benjamin Kasmin, Philip Steele</td>
</tr>
</tbody>
</table>


3.5 Level 2 Software Block Diagrams

3.5.1 Level 2 Software Server Module

Figure 6: Level 2 Software Block Diagram; Server Module

<table>
<thead>
<tr>
<th>Module</th>
<th>Server Module</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inputs</td>
<td>Serial Data from Processor; PHP Queries</td>
</tr>
<tr>
<td>Outputs</td>
<td>Refined Data Stream for iOS Application</td>
</tr>
<tr>
<td>Functionality</td>
<td>Configures Data for iOS Application from Wireless Communication with Processor</td>
</tr>
<tr>
<td>Designers</td>
<td>Yikun Wang</td>
</tr>
</tbody>
</table>

Table 9: Level 2 Software Module; Server
3.5.2 Level 2 Software Wireless Module

![Block Diagram of Wireless Module](image)

**Figure 7: Level 2 Software Block Diagram; Wireless Module**

<table>
<thead>
<tr>
<th>Module</th>
<th>Wireless Module</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inputs</td>
<td>Data from Processor Bus</td>
</tr>
<tr>
<td>Outputs</td>
<td>Streamed Wireless Serial Data</td>
</tr>
<tr>
<td>Functionality</td>
<td>Transmits Processed Data over a Wireless Network</td>
</tr>
<tr>
<td>Designers</td>
<td>Yikun Wang</td>
</tr>
</tbody>
</table>

**Table 10: Level 2 Software Module; Wireless**

3.5.3 Level 2 Software App Module

![Block Diagram of App Module](image)

**Figure 8: Level 2 Software Block Diagram; App Module**
3.5.4 Level 2 Software Digest

The software design yields raw data from sensors as inputs, and produces processed vehicle status information to user as outputs. The raw data of the host vehicle’s battery current, voltage, temperature and speed will be collected by multiple sensors and then passed to the Arduino board. By using a Wi-Fi module or cellular module, the Arduino board will have access to Internet. After getting raw data from the sensors, the Arduino board will process and send data it collects into our Linux, Apache, MySQL, or PHP based server’s database via either Wi-Fi or cell using PHP method 1. The iOS app queries the database using PHP method 2 to get most current vehicle status information (current, voltage, temperature, speed) and then do the calculation to get the remaining life of the battery pack and other information. The iOS/web app then will show status information to the user via a user-friendly GUI.

PHP Method 1: jia.php

One way for Arduino board to transfer data collected by sensors to our database is using a cellular module. The idea behind it is to let our Arduino board have internet access and then send data by calling PHP method 1. Jia.php takes four inputs: current, voltage, speed, and temperature, which are the four raw data parameters that we will collect from sensors. This method should be called as:
After the Arduino board gets values of current, voltage, speed, temperature of the vehicle, it should then, using wget (or another suitable Arduino function), visit the URL with the values it retrieved via the https protocol. When this php code it called, it will add the four inputs it takes into our database along with a timestamp and an automatically generated ID number.

PHP Method 2: get.php

This PHP code is used for our iOS app to query the database in order to resolve the most recent status values:

https://yikun.net/get.php?type=%5Bvalue_type%5D

Therein, which value_type is one of the raw data types (current, voltage, speed, temperature). It returns the latest value of value_type as a string. This returned string then will be processed in our iOS app.

The iOS application is segmented into two “tabs”, one for the charging state of the Host and one for the discharging state of the Host. The application does not discriminate between modes; rather the user chooses the mode of usage through the application’s graphical user interface. The usage modes only differ in what data is displayed to the user, and the method of data acceptance (wireless, or wired). This is a design decision based solely on practical use and keeping the functions of the application user-friendly. All calculations will be performed in the application code except for SOC (State of Charge), which to maintain the desired accuracy must be performed on the microprocessor initially, and then streamed to the iOS device.
3.6 Level 2 Hardware Block Diagrams

3.6.1 Level 2 Hardware Normalizing PCB

![Diagram](image)

Figure 9: Level 2 Hardware Block Diagram; Normalizing PCB

<table>
<thead>
<tr>
<th>Module</th>
<th>Interface PCB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inputs</td>
<td>Analog Velocity Voltage</td>
</tr>
<tr>
<td></td>
<td>Analog Temperature Voltage</td>
</tr>
<tr>
<td></td>
<td>Analog Battery Current</td>
</tr>
<tr>
<td></td>
<td>Analog Battery Voltage</td>
</tr>
<tr>
<td>Outputs</td>
<td>Normalized Velocity Signal</td>
</tr>
<tr>
<td></td>
<td>Normalized Temperature Signal</td>
</tr>
<tr>
<td></td>
<td>Normalized Battery Current and Voltage</td>
</tr>
<tr>
<td>Functionality</td>
<td>Proportionally Steps-Down Analog Signal</td>
</tr>
<tr>
<td></td>
<td>Inputs into Useable Serial Data</td>
</tr>
<tr>
<td>Designers</td>
<td>Benjamin Kasmin</td>
</tr>
</tbody>
</table>

Table 12: Level 2 Hardware Module; Normalizing PCB
3.6.2 Level 2 Hardware Microprocessor Module

A: Voltage
B: Current
C: Velocity
D: Heat
E: State of Charge

Figure 10: Level 2 Hardware Block Diagram; Microprocessor

<table>
<thead>
<tr>
<th>Module</th>
<th>Microprocessor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inputs</td>
<td>Normalized Serial Data</td>
</tr>
<tr>
<td>Outputs</td>
<td>Wireless Serial Data</td>
</tr>
<tr>
<td></td>
<td>Processed Data Stream</td>
</tr>
<tr>
<td></td>
<td>Calculated SOC (State-of-Charge)</td>
</tr>
<tr>
<td>Functionality</td>
<td>Feeds Processed Information Data to Application</td>
</tr>
<tr>
<td>Designers</td>
<td>Pierre Hall</td>
</tr>
</tbody>
</table>

Table 13: Level 2 Hardware Module; Microprocessor
3.6.3 Level 2 Hardware Digest

The RMS will take four analog sensor inputs (speed, temperature, voltage, and current) and convert these inputs into separate serial data outputs to be utilized by the iOS device. This process occurs by utilizing a “normalizing” interface PCB that proportionally converts measured values into scaled 0-5V pulses that can be processed further by a microprocessor. The microprocessor will parse and process the input data in such a manner that each signal can be output as a serial representation digitally. All calculations other than SOC (state-of-charge) will be performed on the iOS device and application. State-of-charge will be calculated by the microprocessor instead of the iOS application in order to minimize timing delays that may demean the user experience.

The microprocessor feeds data to the iOS device in two states, with two separate interfaces; one local communication channel and the other remotely via a wireless connection. The local communication channel (serial cable) will be utilized while the host device is running, and the wireless communication channel while the host is idle and charging. This streamlines the calculation methodology by allowing simultaneous processing of signals to be separated by the state of the RMS and the connection type of the iOS device. In the charging state of the Host (wireless connection of RMS), only voltage, current, and state-of-charge are needed, as the Host is static. In the discharging state of the Host, the RMS is thus coupled wirelessly, and all sensor data is necessary for proper operation. Both the hardware and software portion of the RMS can distinguish the state of the Host based on the method of connection to the iOS device. The iOS application will then be able to receive the parsed and processed information to perform calculations in real-time and display intended data to the user in an intuitive GUI.
3.7 Hardware Schematics and Theory of Operation

3.7.1 Arduino Connection Map

![Diagram showing Arduino connection map.](image)

Figure 11: Arduino Connection Map

The above figure shows the mapping of the connections that must be made to interface the sensors to the Red Park serial interface. This figure assumes that the input signals from the sensors have already been normalized, which must be done using additional hardware.
3.7.2 RiOS Interconnect Schematic

The above figure shows the schematic of the interconnections that will take place between the sensors, the Arduino, and the Red Park cable. This represents the physical connection, which will be utilized while the host device is connected locally. It can be seen that the voltages coming from the sensors are fed into an interface board where normalization will take place, making the signals suitable for the analog inputs of the microprocessor.
3.7.3 Normalization PCB Interface

The above figure is the schematic for the interconnections of the normalization PCB. The input signals are the raw analog signals from the sensors, and the outputs are normalized signals, which are fed into the microprocessor. The values of R5-R8 are left variable to accommodate for the changes in battery chemistry, which will result in corresponding changes in battery current and voltage.
3.7.4 Speed Sensing Circuit

Figure 14: Speed Sensing Circuit

Figure 15: Output Voltage vs. Hall Voltage
Figure 14 is a rendering of the amplification circuit that is to be used to amplify the analog signal sensed from the hall element. This simulation assumes that the hall element will be driven with the 3.3VDC source from the microprocessor circuit. Figure 15 shows the plot of input hall voltage from the sensor versus the output voltage sent to the microprocessor. The amplitude of this signal is sufficient to drive the input pin of the microprocessor, causing a high pulse, which will be used to calculate velocity.

3.7.5 Thermal Sensor Circuit

![Figure 16: Thermal Sensor Schematic](image)

![Figure 17: Simulation of Thermal Sensor Output Voltage](image)
Figure 16 is a rendering of the amplification circuit that will be used to amplify the analog signal from the thermal sensor. This simulation assumes that the thermal sensor will be driven using the 3.3VDC from the microprocessor circuit. The temperature range of the thermal sensor is -40°C to 125°C, and yields a voltage full-scale output of 100mV to 1.7V, which, after amplification, is sufficient to drive the input pin of the microprocessor circuit. Figure 17 shows the simulation of input temperature voltage to output voltage.

3.7.6 Voltage/Current Sensing Circuit

![Figure 18: Current Sensing Circuit](image1)

![Figure 19: Voltage Measurement Divider](image2)
Figure 18 above displays a simple in-line shunt resistance to measure the vehicle current. A voltage is measured across a conducting element with finite resistance; the voltage across this element is then amplified for input requirements of the microprocessor and is proportional to the current intended to be measured (by Ohm’s Law, \( V = I \times R \)). The conductive element is used due to the prevalence of very high current ranges in the electric vehicles that need to be measured. Current can be accurately determined as long as the true resistance of the conductive element is known. A simple shunt resistance would not suffice because extremely high amounts of power, and thus heat, must be dissipated across it.

To measure voltage, a simple voltage division circuit (Fig. 19) is used to normalize the maximum acceptable values of specified vehicle voltage to less than 5V DC, to be processed by the microprocessor.

3.7.7 Sensor Normalization Amplification Circuit

![Figure 20: Simple Non-Inverting Amplifier Circuit](image)
The above figure is a simple non-inverting amplifier circuit. This circuit is used to produce a voltage gain between the input voltage and the output voltage, which is controlled by the resistors R1 and R2. To determine the gain of the circuit consider the following calculation:

Using nodal analysis at the $V_-$ node

$$\frac{V_- - 0}{R_1} + \frac{V_- - V_O}{R_2} = 0$$

For an ideal amplifier:

$$V_+ = V_- = V_i$$

Therefore,

$$\frac{V_i}{R_1} + \frac{V_i}{R_2} = \frac{V_O}{R_2}$$

Resulting in a voltage gain of:

$$G = \frac{V_O}{V_i} = 1 + \frac{R_2}{R_1}$$

This circuit is used in the design to amplify sensor signals for suitable use in a microprocessor circuit. For the velocity sensing strategy, the range of output is 0 – 100mV, and the required voltage at the microprocessor is 0-5V, therefore, a gain of 50 is required. If the value of R2 is arbitrarily chosen as 100kΩ, then the calculated value of R1 is found to be 2.04kΩ.

For the thermal sensor, the range of available voltage is 100mV – 1.7V, thus a gain of 2.94 is required. If a value of R1 is arbitrarily chosen to be 1kΩ, then the calculated value of R2 is found to be 1.94kΩ.
3.8 Software Flow Diagrams and Theory of Operation

3.8.1 State of Charge Calculation

Figure 21: State-of-Charge Calculation Pseudocode

```
//Benjamin Kasmin 11-13-12
//State of Charge Pseudocode for use with arduino module
//Goal is to integrate battery current with time
Variable Sample_time
READ Amplitude_of_battery_current
Perform Calculations
    For i=1:Entire_Discharging_Period
        State_of_Charge_Instant = Sample_time*Amplitude_of_battery_current(i)
    End
State_of_Charge_Tot = sum(State_of_Charge_Instant)
State_of_Charge_Display = (State_of_Charge_Tot/Total_Battery_SOC)*100
Write State_of_Charge_Display
```

Figure 21 above displays a block of pseudocode written for the state of charge calculation, which will be done on the Arduino microprocessor itself. This block of code shows the technique and flow for how the calculation will be performed. It integrates (sums) the magnitudes of the measured current values over a continuous time interval. The input stream is output to the serial or wireless interfaces, while the instantaneous value at a given point of time is sent in real-time to the iOS device in the Host device’s discharging state, and sampled once per second in the Host device’s charging state.
3.8.2 Discharging (Serial Connection) Flow Diagram

Redpark SDK (Objective-C class RscMgr: RscMgr.h and libRscMgr.a)

iOS App Project

- Initialize RscMgr

Set RscMgr as Delegate

Event Occurrence: Data from Serial Cable Sent to App

Event Information sent back to RscMgr instance; accessed by SDK Functions

Figure 22: Redpark SDK Overview
For the wired connection (Host discharging state), the Redpark TTL Serial Cable will be used as the connection tool. The Redpark cable has a built-in RS-232 serial to TTL adapter, so that it can be connected directly between the Arduino microprocessor and IOS device. This cable comes with its own SDK that provides an Objective-C class RscMgr that will be used as the functional class for data transfer. In the iOS application code, a new instance of the RscMgr class is initialized, and then is set as the delegate. This instance of RscMgr works as an event listener. Whenever there is a new event at the serial port, information about such events will be caught by this instance of the RscMgr class. Afterwards, the properties of the newly occurring events can be determined by using functions provided from the RscMgr class.

```python
from urllib import urllib
import time
print 'time spent to update DB: '
updatetime=0
for i in range(10):
    t1=time.time()
    r = urllib.urlopen(url).read()
    t2=time.time()
    # print r
    utime=(t2-t1)*1000
    updatetime = updatetime+utime
    print 'test'+str(i)+"\t"+str(utime)+" ms"
    # 1st one always return a large value
    # time.sleep(2)

url = 'https://yikun.net/get.php?type=tree'
# query timestamp from DB will return largest string -> this query takes longest time
print 'query timestamp: '
querytime=0
for i in range(10):
    t1=time.time()
    r = urllib.urlopen(url).read()
    t2=time.time()
    qtime=(t2-t1)*1000
    querytime = querytime+qtime
    print 'test'+str(i)+"\t"+str(qtime)+" ms"
print 'Average update time: '+str(updatetime/10)+" ms"
print 'Average query time: '+str(querytime/10)+" ms"
```

Figure 23: Sample Python Script
3.8.3 Server module

A python script was written to test general connectivity and database performance via a desktop computer in the Senior Design Lab (Room 507) on the University’s network connection, resulting in the following:

<table>
<thead>
<tr>
<th>C:\Users\student\Desktop&gt;server.py</th>
<th>time spent to update DB (transfer data to server):</th>
</tr>
</thead>
<tbody>
<tr>
<td>test1 1029.99997139 ms</td>
<td>test2 720.00002861 ms</td>
</tr>
<tr>
<td>test3 710.000038147 ms</td>
<td>test4 700.000047684 ms</td>
</tr>
<tr>
<td>test5 829.999923706 ms</td>
<td>test6 720.00002861 ms</td>
</tr>
<tr>
<td>test7 720.00002861 ms</td>
<td>test8 730.000019073 ms</td>
</tr>
<tr>
<td>test9 710.000038147 ms</td>
<td>test10 710.000038147 ms</td>
</tr>
</tbody>
</table>

query timestamp (get data from server to app):

| test1 710.000038147 ms             | test2 719.9999790192 ms                          |
| test3 730.000019073 ms             | test4 769.999980927 ms                            |
| test5 790.0000200272 ms            | test6 730.000019073 ms                            |
| test7 709.999979728 ms             | test8 690.00005722 ms                             |
| test9 690.00005722 ms              | test10 799.999952316 ms                           |

Average update time: 758.000016212 ms (time to send data to database)

Average query time: 733.999991417 ms (time to query data from database)

based on ping to the server[67.212.231.51] using senior design lab desktop:

Ping statistics for 67.212.231.51:
- Packets: Sent = 4, Received = 4, Lost = 0 (0% loss).
- Approximate round trip times in milliseconds:
  - Minimum = 77ms, Maximum = 78ms, Average = 77ms

Table 14: Database Server Performance Diagnostic
From the above results, based on a network connection that has average ping/delay to the server of 77ms, it takes 758ms on average to send data to the database and 734ms to query data from database. To conclude, there is an average server delay/database processing of 0.76s. According to ‘Cellular Standards for the Third Generation’ section of the International Telecommunication Union’s article ‘About mobile technology and IMT-2000’, 3G speeds in a moving vehicle are, on average, 384 kbit/s (48kbyte/s). ("Cellular Standards for the Third Generation". ITU. 1 December 2005. Archived from the original on 24 May 2008.). When sending data to the database, 116 bytes of data will be sent. When querying and receiving data from database, 148 bytes of data will be sent and received. Using 3G baud rates of 48 kbyte/s and server processing time of 0.76s, it is seen that:

Uploading data to the server takes approximately:

\[
\frac{116\text{bytes}}{48\text{kbytes/s}} + 0.76\text{s} = 0.0024\text{s} + 0.76\text{s} = 0.7624\text{s}
\]

Sending queries to request/receive data from database are approximately:

\[
\frac{148\text{bytes}}{48\text{kbytes/s}} + 0.76\text{s} = 0.003\text{s} + 0.76\text{s} = 0.763\text{s}
\]
3.8.4 Data Integrity Flow Chart

Figure 24: Data Integrity Flow Chart
3.8.5 System Context Diagram

Figure 25: System Context Diagram

Figure 24 above establishes a system of checks and balances of sorts for proper acquisition and processing of data between the RiOS hardware and application. In the event that there is an issue with transmission between either the wireless transmission mode and/or the serial communication channel, the application will alert the user in the event of an error such that the user can take further action to rectify the issue. The system context diagram (Fig. 25) gives a large-scale overview of how data is intended to travel between all hardware and software components of the RiOS and the user.
3.8.6 Data Flow Diagram

![Data Flow Diagram]

Figure 26: Data Flow Diagram
3.8.7 Application Development

![Application Storyboard](image)

Figure 27: Application Storyboard

XCode, a native Macintosh OSX program, is used for App development. XCode allows developers to code in various manners. The app development scheme in this project is what is termed a “Storyboard”. Storyboards allow the user to view their code in a graphical manner, and the ability to code more in-depth in the corresponding views. The user can use different gestures to create simple lines of code in header files (.h) and implementation files (.m). After the simple “frame” code is added, the user writes more detailed code to implement exactly what is intended.
When creating the demo app for this project, a table view controller (Fig. 28) is used as the navigation controller in conjunction with a tabbed view controller (Fig. 29). The table view controller is used as the initial perspective of the application. Table view is the best option for this app because it displays a “table” of objects that the user can click to proceed to a directly related new view. After the table row is clicked, the app goes to a tabbed view. In the tabbed view, there are three different views that the user can choose, each displaying pertinent information. For the sake of the demo app, the table view uses a nested method rather than a dynamic method to better demonstrate the app flow. When developing the final app, the nested table is changed to a dynamic view.
In the tabbed view mode, the user will be able to manually select three different views. The first shows the app in the vehicle’s discharge state, the second displays the charging state, and the third an “advanced” view for additional relevant information. Figure 29 above demonstrates the app in discharge mode, in which the app will display the information appropriate during the vehicles usage.
Figure 30 above demonstrates the app in the charging mode, while the vehicle is idle. In this state, the app will only display the information appropriate during vehicle charging, such as general battery statistics and the estimated amount of time left until the charge cycle completes.
In Figure 31, the app displays an “advanced” view. In this view the user will be able to swipe across the screen to view different plots. The plots will show detailed information and calculations beyond the scope of the average user. SOC vs. Time, Voltage vs. Time, Temperature vs. Time, Current vs. Time, and a plot of the battery charging characteristics over time are all intended uses for the advanced tab.
### 4.0 Parts List

#### SENIOR DESIGN PARTS REQUEST ORDERING FORM

<table>
<thead>
<tr>
<th>#</th>
<th>Refdes</th>
<th>Part Num.</th>
<th>Description</th>
<th>Suggested Vendor</th>
<th>Vendor Part Num.</th>
<th>Website Link</th>
<th>Cost</th>
<th>Unit</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SS440R</td>
<td>SENSOR HALL EFFECT UNIPOLAR TO92</td>
<td>Digikey</td>
<td>480-3735-ND</td>
<td><a href="http://www.digikey.com/scripts/dksearch/dksus.dll?vendor=0&amp;keywords=SS440R">http://www.digikey.com/scripts/dksearch/dksus.dll?vendor=0&amp;keywords=SS440R</a></td>
<td>1.80</td>
<td>1.80</td>
<td>1.80</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>MCP9701</td>
<td>IC SENSOR THERMAL 3.1V TO-92-3</td>
<td>Digikey</td>
<td>MCP9701A-ETO-ND</td>
<td><a href="http://www.digikey.com/product-detail/en/MCP9701A-E%2FTO/MCP9701A-E%2FTO-ND/1212511">http://www.digikey.com/product-detail/en/MCP9701A-E%2FTO/MCP9701A-E%2FTO-ND/1212511</a></td>
<td>0.34</td>
<td>0.34</td>
<td>0.34</td>
<td></td>
</tr>
</tbody>
</table>

Total: $7.83
### 5.0 Project Schedules

<table>
<thead>
<tr>
<th>ID</th>
<th>Task Name</th>
<th>Duration</th>
<th>Start</th>
<th>Finish</th>
<th>Resource Names</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Final Design Report</td>
<td>29 days</td>
<td>Wed 10/17/12</td>
<td>Mon 11/26/12</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Abstract</td>
<td>12 days</td>
<td>Sun 10/14/12</td>
<td>Mon 10/28/12</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Software Design</td>
<td>30 days</td>
<td>Wed 10/17/12</td>
<td>Tue 11/27/12</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Modules</td>
<td>30 days</td>
<td>Wed 10/17/12</td>
<td>Tue 11/27/12</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>GUI</td>
<td>11.38 days</td>
<td>Mon 10/15/12</td>
<td>Tue 10/30/12</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Discharge</td>
<td>11.38 days</td>
<td>Mon 10/15/12</td>
<td>Tue 10/30/12</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Instrument Clutter</td>
<td>11 days</td>
<td>Mon 10/15/12</td>
<td>Mon 10/29/12</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Plotting</td>
<td>11 days</td>
<td>Mon 10/15/12</td>
<td>Mon 10/29/12</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Pseudo Code/State Diagram</td>
<td>11 days</td>
<td>Mon 10/15/12</td>
<td>Mon 10/29/12</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Charge</td>
<td>11.38 days</td>
<td>Mon 10/15/12</td>
<td>Tue 11/13/12</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>Charge State</td>
<td>11 days</td>
<td>Mon 10/29/12</td>
<td>Mon 11/12/12</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>Plotting</td>
<td>11 days</td>
<td>Mon 10/29/12</td>
<td>Mon 11/12/12</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>Pseudo Code/State Diagram</td>
<td>11 days</td>
<td>Mon 10/29/12</td>
<td>Mon 11/12/12</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>App Design</td>
<td>11 days</td>
<td>Mon 10/29/12</td>
<td>Mon 11/12/12</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>App Interface/Board</td>
<td>11 days</td>
<td>Mon 10/29/12</td>
<td>Mon 11/12/12</td>
<td>Pierre Hall</td>
</tr>
<tr>
<td>16</td>
<td>App Testing</td>
<td>11 days</td>
<td>Mon 10/29/12</td>
<td>Mon 11/12/12</td>
<td>Pierre Hall</td>
</tr>
<tr>
<td>17</td>
<td>Calculations</td>
<td>6 days</td>
<td>Mon 10/15/12</td>
<td>Fri 10/19/12</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>Charging</td>
<td>4 days</td>
<td>Mon 10/15/12</td>
<td>Thu 10/18/12</td>
<td>Pierre Hall</td>
</tr>
<tr>
<td>19</td>
<td>Discharge</td>
<td>4 days</td>
<td>Mon 10/15/12</td>
<td>Thu 10/18/12</td>
<td>Pierre Hall</td>
</tr>
<tr>
<td>20</td>
<td>Calibration</td>
<td>4 days</td>
<td>Mon 10/15/12</td>
<td>Thu 10/18/12</td>
<td>Pierre Hall</td>
</tr>
<tr>
<td>21</td>
<td>Communication</td>
<td>11 days</td>
<td>Mon 10/15/12</td>
<td>Mon 10/29/12</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>Charge</td>
<td>4 days</td>
<td>Mon 10/15/12</td>
<td>Thu 10/18/12</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>Pseudo Code/State Flow</td>
<td>4 days</td>
<td>Mon 10/15/12</td>
<td>Thu 10/18/12</td>
<td>Yikun Yang</td>
</tr>
<tr>
<td>24</td>
<td>Network SDK Functions</td>
<td>11 days</td>
<td>Mon 10/15/12</td>
<td>Mon 10/29/12</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>Pseudo Code/State Flow</td>
<td>11 days</td>
<td>Mon 10/15/12</td>
<td>Mon 10/29/12</td>
<td>Yikun Yang</td>
</tr>
<tr>
<td>26</td>
<td>Hardware Design</td>
<td>28 days</td>
<td>Wed 10/17/12</td>
<td>Fri 11/23/12</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>Modules</td>
<td>28 days</td>
<td>Wed 10/17/12</td>
<td>Fri 11/23/12</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>Sensors</td>
<td>30 days</td>
<td>Mon 10/15/12</td>
<td>Fri 11/23/12</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>Velocity</td>
<td>5 days</td>
<td>Mon 10/15/12</td>
<td>Fri 10/19/12</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>Block Diagram</td>
<td>5 days</td>
<td>Sun 10/14/12</td>
<td>Thu 10/18/12</td>
<td>Ben Kazmin</td>
</tr>
<tr>
<td>31</td>
<td>Schematic</td>
<td>5 days</td>
<td>Sun 10/14/12</td>
<td>Thu 10/18/12</td>
<td>Ben Kazmin</td>
</tr>
<tr>
<td>32</td>
<td>Simulation</td>
<td>5 days</td>
<td>Sun 10/14/12</td>
<td>Thu 10/18/12</td>
<td>Ben Kazmin</td>
</tr>
<tr>
<td>33</td>
<td>Thermal</td>
<td>5 days</td>
<td>Mon 10/22/12</td>
<td>Fri 10/26/12</td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>Block Diagram</td>
<td>5 days</td>
<td>Sun 10/21/12</td>
<td>Thu 10/25/12</td>
<td>Ben Kazmin</td>
</tr>
<tr>
<td>35</td>
<td>Schematic</td>
<td>5 days</td>
<td>Sun 10/21/12</td>
<td>Thu 10/25/12</td>
<td>Ben Kazmin</td>
</tr>
<tr>
<td>36</td>
<td>Simulation</td>
<td>5 days</td>
<td>Sun 10/21/12</td>
<td>Thu 10/25/12</td>
<td>Ben Kazmin</td>
</tr>
<tr>
<td>37</td>
<td>Voltage</td>
<td>5 days</td>
<td>Mon 10/15/12</td>
<td>Fri 10/19/12</td>
<td></td>
</tr>
<tr>
<td>38</td>
<td>Block Diagram</td>
<td>5 days</td>
<td>Sun 10/14/12</td>
<td>Thu 10/18/12</td>
<td>Philip Steele</td>
</tr>
<tr>
<td>39</td>
<td>Schematic</td>
<td>5 days</td>
<td>Sun 10/14/12</td>
<td>Thu 10/18/12</td>
<td>Philip Steele</td>
</tr>
<tr>
<td>40</td>
<td>Simulation</td>
<td>5 days</td>
<td>Sun 10/14/12</td>
<td>Thu 10/18/12</td>
<td>Philip Steele</td>
</tr>
<tr>
<td>41</td>
<td>Current</td>
<td>6 days</td>
<td>Sun 10/21/12</td>
<td>Fri 10/26/12</td>
<td></td>
</tr>
<tr>
<td>42</td>
<td>Block Diagram</td>
<td>6 days</td>
<td>Sun 10/21/12</td>
<td>Fri 10/26/12</td>
<td>Philip Steele</td>
</tr>
<tr>
<td>43</td>
<td>Schematic</td>
<td>6 days</td>
<td>Sun 10/21/12</td>
<td>Fri 10/26/12</td>
<td>Philip Steele</td>
</tr>
<tr>
<td>44</td>
<td>Simulation</td>
<td>6 days</td>
<td>Sun 10/21/12</td>
<td>Fri 10/26/12</td>
<td>Philip Steele</td>
</tr>
<tr>
<td>45</td>
<td>Interface</td>
<td>16 days</td>
<td>Mon 10/28/12</td>
<td>Mon 11/18/12</td>
<td></td>
</tr>
<tr>
<td>46</td>
<td>Normalize</td>
<td>11 days</td>
<td>Mon 10/28/12</td>
<td>Mon 11/18/12</td>
<td></td>
</tr>
<tr>
<td>47</td>
<td>PCB Design</td>
<td>11 days</td>
<td>Mon 10/28/12</td>
<td>Mon 11/18/12</td>
<td></td>
</tr>
<tr>
<td>48</td>
<td>Layout</td>
<td>11 days</td>
<td>Mon 10/29/12</td>
<td>Mon 11/12/12</td>
<td>Ben Kazmin</td>
</tr>
<tr>
<td>49</td>
<td>Velocity</td>
<td>11 days</td>
<td>Mon 10/29/12</td>
<td>Mon 11/12/12</td>
<td>Ben Kazmin</td>
</tr>
<tr>
<td>50</td>
<td>Thermal</td>
<td>11 days</td>
<td>Mon 10/29/12</td>
<td>Mon 11/12/12</td>
<td>Ben Kazmin</td>
</tr>
<tr>
<td>51</td>
<td>Voltage</td>
<td>11 days</td>
<td>Mon 10/29/12</td>
<td>Mon 11/12/12</td>
<td>Ben Kazmin</td>
</tr>
<tr>
<td>52</td>
<td>Current</td>
<td>11 days</td>
<td>Mon 10/29/12</td>
<td>Mon 11/12/12</td>
<td>Ben Kazmin</td>
</tr>
<tr>
<td>53</td>
<td>Communication</td>
<td>10 days</td>
<td>Sun 11/12/12</td>
<td>Thu 11/18/12</td>
<td></td>
</tr>
<tr>
<td>54</td>
<td>Serial</td>
<td>8 days</td>
<td>Sun 11/4/12</td>
<td>Thu 11/8/12</td>
<td></td>
</tr>
<tr>
<td>55</td>
<td>Research</td>
<td>5 days</td>
<td>Sun 11/4/12</td>
<td>Thu 11/8/12</td>
<td>Ben Kazmin</td>
</tr>
<tr>
<td>56</td>
<td>Wireless</td>
<td>6 days</td>
<td>Sun 11/4/12</td>
<td>Thu 11/8/12</td>
<td></td>
</tr>
<tr>
<td>57</td>
<td>Research</td>
<td>5 days</td>
<td>Sun 11/4/12</td>
<td>Thu 11/8/12</td>
<td>Yikun Yang</td>
</tr>
<tr>
<td>58</td>
<td>The Box</td>
<td>12 days</td>
<td>Sun 11/4/12</td>
<td>Mon 11/19/12</td>
<td></td>
</tr>
<tr>
<td>59</td>
<td>Wire Routing</td>
<td>12 days</td>
<td>Sun 11/4/12</td>
<td>Mon 11/19/12</td>
<td>Ben Kazmin</td>
</tr>
<tr>
<td>60</td>
<td>Construction</td>
<td>12 days</td>
<td>Sun 11/4/12</td>
<td>Mon 11/19/12</td>
<td>Ben Kazmin</td>
</tr>
<tr>
<td>61</td>
<td>Design</td>
<td>12 days</td>
<td>Sun 11/4/12</td>
<td>Mon 11/19/12</td>
<td></td>
</tr>
<tr>
<td>62</td>
<td>Dimensions</td>
<td>12 days</td>
<td>Sun 11/4/12</td>
<td>Mon 11/19/12</td>
<td>Ben Kazmin</td>
</tr>
<tr>
<td>63</td>
<td>Microcontroller/Program</td>
<td>12 days</td>
<td>Sun 11/4/12</td>
<td>Mon 11/19/12</td>
<td>Ben Kazmin</td>
</tr>
<tr>
<td>64</td>
<td>Parts Request Form</td>
<td>30 days</td>
<td>Wed 10/17/12</td>
<td>Tue 11/27/12</td>
<td></td>
</tr>
<tr>
<td>65</td>
<td>Budget (Estimated)</td>
<td>30 days</td>
<td>Wed 10/17/12</td>
<td>Tue 11/27/12</td>
<td>Pierre Hall</td>
</tr>
<tr>
<td>66</td>
<td>Implementation Gantt Chart</td>
<td>30 days</td>
<td>Wed 10/17/12</td>
<td>Tue 11/27/12</td>
<td>Pierre Hall</td>
</tr>
<tr>
<td>67</td>
<td>Conclusions and Recommendations</td>
<td>30 days</td>
<td>Wed 10/17/12</td>
<td>Tue 11/27/12</td>
<td>Pierre Hall</td>
</tr>
</tbody>
</table>

**Figure 32: Fall 2012 Gantt Chart**
<table>
<thead>
<tr>
<th>ID</th>
<th>Task Name</th>
<th>Duration</th>
<th>Start</th>
<th>Finish</th>
</tr>
</thead>
<tbody>
<tr>
<td>55</td>
<td>Wireless Communication</td>
<td>1 day?</td>
<td>Mon 1/14/13</td>
<td>Tue 1/15/13</td>
</tr>
<tr>
<td>56</td>
<td>Communication Development</td>
<td>1 day?</td>
<td>Mon 1/14/13</td>
<td>Tue 1/15/13</td>
</tr>
<tr>
<td>57</td>
<td>Serial Communication</td>
<td>1 day?</td>
<td>Mon 1/14/13</td>
<td>Tue 1/15/13</td>
</tr>
<tr>
<td>58</td>
<td>Red Park Cable</td>
<td>1 day?</td>
<td>Mon 1/14/13</td>
<td>Tue 1/15/13</td>
</tr>
<tr>
<td>59</td>
<td>Red Park SDK</td>
<td>1 day?</td>
<td>Mon 1/14/13</td>
<td>Tue 1/15/13</td>
</tr>
<tr>
<td>60</td>
<td>Wireless Communication</td>
<td>1 day?</td>
<td>Mon 1/14/13</td>
<td>Tue 1/15/13</td>
</tr>
<tr>
<td>61</td>
<td>Server</td>
<td>1 day?</td>
<td>Mon 1/14/13</td>
<td>Tue 1/15/13</td>
</tr>
<tr>
<td>62</td>
<td>Communication Speeds</td>
<td>1 day?</td>
<td>Mon 1/14/13</td>
<td>Tue 1/15/13</td>
</tr>
<tr>
<td>63</td>
<td>Microprocessor</td>
<td>1 day?</td>
<td>Mon 1/14/13</td>
<td>Tue 1/15/13</td>
</tr>
<tr>
<td>64</td>
<td>Calculated Data</td>
<td>1 day?</td>
<td>Mon 1/14/13</td>
<td>Tue 1/15/13</td>
</tr>
<tr>
<td>65</td>
<td>SOC</td>
<td>1 day?</td>
<td>Mon 1/14/13</td>
<td>Tue 1/15/13</td>
</tr>
<tr>
<td>66</td>
<td>Sampling Time</td>
<td>1 day?</td>
<td>Mon 1/14/13</td>
<td>Tue 1/15/13</td>
</tr>
<tr>
<td>67</td>
<td>Sending Data</td>
<td>1 day?</td>
<td>Mon 1/14/13</td>
<td>Tue 1/15/13</td>
</tr>
<tr>
<td>68</td>
<td>Wireless</td>
<td>1 day?</td>
<td>Mon 1/14/13</td>
<td>Tue 1/15/13</td>
</tr>
<tr>
<td>69</td>
<td>Serial</td>
<td>1 day?</td>
<td>Mon 1/14/13</td>
<td>Tue 1/15/13</td>
</tr>
<tr>
<td>70</td>
<td>Test Software</td>
<td>14 days</td>
<td>Tue 1/15/13</td>
<td>Tue 1/29/13</td>
</tr>
<tr>
<td>71</td>
<td>Revise Software</td>
<td>7 days</td>
<td>Tue 1/29/13</td>
<td>Tue 2/5/13</td>
</tr>
<tr>
<td>72</td>
<td>MIDTERM: Demonstrate Software</td>
<td>7 days</td>
<td>Tue 2/5/13</td>
<td>Tue 2/12/13</td>
</tr>
<tr>
<td>73</td>
<td>SDC &amp; FA Software Approval</td>
<td>0 days</td>
<td>Tue 2/12/13</td>
<td>Tue 2/12/13</td>
</tr>
<tr>
<td>74</td>
<td>System Integration</td>
<td>39 days</td>
<td>Tue 2/12/13</td>
<td>Sat 3/23/13</td>
</tr>
<tr>
<td>75</td>
<td>Assemble Complete System</td>
<td>14 days</td>
<td>Tue 2/12/13</td>
<td>Tue 2/26/13</td>
</tr>
<tr>
<td>76</td>
<td>Test Complete System</td>
<td>21 days</td>
<td>Tue 2/26/13</td>
<td>Tue 3/19/13</td>
</tr>
<tr>
<td>77</td>
<td>Revise Complete System</td>
<td>25 days</td>
<td>Tue 2/26/13</td>
<td>Sat 3/23/13</td>
</tr>
<tr>
<td>78</td>
<td>Demonstration of Complete System</td>
<td>0 days</td>
<td>Sat 3/23/13</td>
<td>Sat 3/23/13</td>
</tr>
<tr>
<td>79</td>
<td>Develop Final Report</td>
<td>99 days</td>
<td>Mon 1/14/13</td>
<td>Tue 4/23/13</td>
</tr>
<tr>
<td>80</td>
<td>Write Final Report</td>
<td>99 days</td>
<td>Mon 1/14/13</td>
<td>Tue 4/23/13</td>
</tr>
<tr>
<td>81</td>
<td>Submit Final Report</td>
<td>0 days</td>
<td>Tue 4/23/13</td>
<td>Tue 4/23/13</td>
</tr>
<tr>
<td>82</td>
<td>Martin Luther King Day - University closed</td>
<td>0 days</td>
<td>Mon 1/14/13</td>
<td>Mon 1/14/13</td>
</tr>
<tr>
<td>83</td>
<td>Spring Recess</td>
<td>6 days</td>
<td>Mon 3/25/13</td>
<td>Sun 3/31/13</td>
</tr>
<tr>
<td>84</td>
<td>Project Demonstration and Presentation</td>
<td>0 days</td>
<td>Fri 4/19/13</td>
<td>Fri 4/19/13</td>
</tr>
<tr>
<td>85</td>
<td>Faraday Banquet</td>
<td>0 days</td>
<td>Fri 4/26/13</td>
<td>Fri 4/26/13</td>
</tr>
<tr>
<td>86</td>
<td>Senior Design Expo</td>
<td>0 days</td>
<td>Wed 4/24/13</td>
<td>Wed 4/24/13</td>
</tr>
</tbody>
</table>

Figure 33: Spring 2013 Gantt Chart

### 6.0 Design Team Information

Pierre Hall - Electrical Engineer, Project Leader  
Benjamin Kasmin - Electrical Engineer, Hardware Manager  
Philip Steele - Electrical Engineer, Archivist  
Yikun Wang - Computer Engineer, Software Manager
7.0 Conclusions and Recommendations

RiOS is going to be a challenging and rewarding project. At this time it is still uncertain how stepping the voltage down to make it a safe level for the micro controller will be performed or if manual normalization will be necessary depending on the chips that can be purchased. On the software side of the project, communication is going to play major key. If there is no communication no data can be collected or calculated and be displayed to the user. The app still has more development that needs to take place but with the research done it is possible to complete the project. Retrieving the communicated information is the key on the app side as well as displaying the data to the user.

8.0 References

9.0 Appendices

Data Sheets

<table>
<thead>
<tr>
<th>Component</th>
<th>URL</th>
</tr>
</thead>
</table>

iOS Code

Table View Implementation File

```objective-c
// TableViewController.m
// SeniorDesign
// Created by Pierre Hall on 11/11/12.
// Copyright (c) 2012 Pierre Hall. All rights reserved.

#import "TableViewController.h"
#import "TabbedViewController.h"

@interface TableViewController ()
@end

@implementation TableViewController

@synthesize vehicleList;

-(id)initWithStyle:(UITableViewStyle)style {
    self = [super initWithStyle:style];
    if (self) {
        // Custom initialization
    }
    return self;
}
```
-(void)viewDidLoad
{
    [super viewDidLoad];

    vehicleList = [[[NSArray alloc] initWithObjects:@"Vehicle 1",
        @"Vehicle 2", @"Vehicle 3", nil];

    self.title = @"My Vehicles";
}

-(void)didReceiveMemoryWarning
{
    [super didReceiveMemoryWarning];
}

#pragma mark - Table view data source

- (NSInteger)tableView:(UITableView *)tableView
numberOfSectionsInTableView:(NSInteger)section
{
    // Return the number of sections.
    return 1;
}

- (NSInteger)tableView:(UITableView *)tableView
numberOfRowsInSection:(NSInteger)section
{
    // Return the number of rows in the section.
    return vehicleList.count;
}

- (UITableViewCell *)tableView:(UITableView *)tableView
cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
    //create an NSString object that we can use as the reuse
    identifier
    static NSString *CellIdentifier = @"Cell";

    //Check to see if we cab reuse a cell from a row that has
    just rolled off the screen

    UITableViewCell *cell = [tableView
        dequeueReusableCellWithIdentifier: CellIdentifier];

    //If there are no cells to be reused, create a new cell

    if (cell == nil) {
        cell = [[UITableViewCell alloc]
initWithStyle:UITableViewControllerCellStyleDefault reuseIdentifier:
CellIdentifier];
}

//Set the text attribute to whatever we are currently looking
at in our array

cell.textLabel.text = [vehicleList
objectAtIndex:indexPath.row];

//Set the detail disclosure indicator

//Return the cell
return cell;

} */

/*
// Override to support conditional editing of the table view.
-(BOOL)tableView:(UITableView *)tableView
canEditRowAtIndexPath:(NSIndexPath *)indexPath{
    // Return NO if you do not want the specified item to be
    // editable.
    return YES;
}
*/

/*
// Override to support editing the table view.
-(void)tableView:(UITableView *)tableView
commitEditingStyle:(UITableViewCellEditingStyle)editingStyle
forRowAtIndexPath:(NSIndexPath *)indexPath{
    if (editingStyle == UITableViewCellEditingStyleDelete) {
        // Delete the row from the data source
        [tableView deleteRowsAtIndexPaths:@[indexPath]
withRowAnimation:UITableViewRowAnimationFade];
    }
    else if (editingStyle == UITableViewCellEditingStyleInsert) {
        // Create a new instance of the appropriate class, insert
        // it into the array, and add a new row to the table view
    }
} */
/*
// Override to support rearranging the table view.
-(void)tableView:(UITableView *)tableView
moveRowAtIndexPath:(NSIndexPath *)fromIndexPath
toIndexPath:(NSIndexPath *)toIndexPath
{
}
*/

/*
// Override to support conditional rearranging of the table view.
-(BOOL)tableView:(UITableView *)tableView
canMoveRowAtIndexPath:(NSIndexPath *)indexPath
{
 // Return NO if you do not want the item to be re-orderable.
 return YES;
}
*/

//*/
-(void)prepareForSegue:(UIStoryboardSegue *)segue
sender:(id)sender
{
 //Create an instance of our TabbedViewController
 TabbedViewController * TVC = [[TabbedViewController alloc]
 init];

 //Set the TVC to the destinationViewController property of
 the segue
 TVC = [segue destinationViewController];

 //Get indexPath
 NSIndexPath * path = [self.tableView
 indexPathForSelectedRow];

 NSString * theVehicle = [vehicleList objectAtIndex:path.row];

 TVC.vehicleNumber = path.row;
 TVC.vehicleName = theVehicle;
}
//*/

//-(void)tableView:(UITableView*)tableView
didSelectRowAtIndexPath:(NSIndexPath *)indexPath
//{{
 // TabbedViewController * TVC = [[TabbedViewController alloc]
 init];
 //
 // [self.navigationController pushViewController:TVC
animated:YES]; //
//}}

@end

Table View Header File
//
//  TableViewController.h
//  SeniorDesign
//
//  Created by Pierre Hall on 11/11/12.
//  Copyright (c) 2012 Pierre Hall. All rights reserved.
//
#import <UIKit/UIKit.h>

@interface TableViewController : UITableViewController
@property (strong, nonatomic) NSArray * vehicleList;
@end

Tabbed View Implementation File
//
//  TabbedViewController.m
//  SeniorDesign
//
//  Created by Pierre Hall on 11/12/12.
//  Copyright (c) 2012 Pierre Hall. All rights reserved.
//
#import "TabbedViewController.h"

@interface TabbedViewController ()
@end

@implementation TabbedViewController
@synthesize vehicleNumber, vehicleName;

-(id)initWithNibName:(NSString *)nibNameOrNil bundle:(NSBundle *)nibBundleOrNil {
    self = [super initWithNibName:nibNameOrNil bundle:nibBundleOrNil];
    if (self) {
        // Custom initialization
    }
    return self;
}
-(void)viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the view.
    // Set the title of the view
    self.title = vehicleName;
}

-(void)didReceiveMemoryWarning
{
    [super didReceiveMemoryWarning];
    // Dispose of any resources that can be recreated.
}
@end

Tabbed View Header File
//@interface TabbedViewController : UITabBarController
//@property NSInteger vehicleNumber;
//@property (strong, nonatomic) NSString * vehicleName;
@end