RYERSON UNIVERSITY

Department of Electrical & Computer Engineering
Program: Electrical/Computer Engineering

Course Number	**ELE/COE700**
Course Title	**Engineering Design Project**
Semester/Year	Fall 2012
Instructor	Dr. Xavier Fernando

Final Report 1

Report Title	**Project Design Report**

Submission Date	**November 27, 2012**
Due Date	**November 27, 2012**

Name	Student ID	Signature*
Ariel Fertman	*xxxx4543*	
Daniel Balilti	*xxxx5259*	
Yudister Narine	*xxxx2737*	

(Note: remove the first 4 digits from your student ID)

**By signing above you attest that you have contributed to this submission and confirm that all work you have contributed to this submission is your own work. Any suspicion of copying or plagiarism in this work will result in an investigation of Academic Misconduct and may result in a "0" on the work, an "F" in the course, or possibly more severe penalties, as well as a Disciplinary Notice on your academic record under the Student Code of Academic Conduct, which can be found online at: www.ryerson.ca/senate/current/pol60.pdf.*

Introduction

Electric vehicle production is a growing industry and relatively new to the automotive market. This project will create a software monitoring system for an electric car provided by Toronto Electric. The car has an integrated battery monitoring system (BMS) which consists of an array of microcontrollers monitoring 24 four-cell modules. The BMS is able to transmit its information to a processor board designed by Archronix Inc. The processor board has the ability to transmit the battery information to a local web page and to a GSM modem. The battery is the most expensive product within the electric vehicle. With a monitoring system that keeps track of history and instances of results, much information can be graphed versus specific daily conditions to further improve the batteries.

Objective

The goal of this project will be to integrate the GSM modem to wirelessly transmit JSON information to a PHP cloud server provided by Archronix. The PHP server will then store the information into a database where battery analysis can be performed. Using the statistics stored in the database, this project will look to explore the non-linear behavior of an electric car battery using modern software architecture. The effects to be explored are battery state of charge versus weather conditions and outside temperature. This analysis will allow users to get a more detailed look into battery performance.

Alongside the software objective, hardware emulation will be created to demonstrate a simplified view of the battery management system. A four-cell array of lithium-ion batteries will be connected up to a differential circuit that will access individual cell voltages. When a battery is observed to be at a lower charge then its neighboring batteries, a flag will alert the system of an imbalance. A microcontroller will be used to consistently read the information using digital input/output. The microcontroller will also send information from a serial bus to be used as debug information. Finally, a bonus CAN controller daughter board will be used, if time permits, to send CAN protocol information to the Archronix Morpheus processor board. From the board a web page can be developed to visually broadcast the voltage information of the individual cells.

Design Specifications

Software

- Receive real-time battery information from E-Car over GSM
- Receive location information to view current location status on Google Maps
- Analyze battery information versus weather conditions/temperature
- Application should be used on a smart phone (Android)
- Application will notify user when battery charge is low and any faults occur

Hardware

- Create a differential voltage circuit that can tap 4 cells in series
- Send this information to a microcontroller via digital I/O
- Display the information on the LCD of the microcontroller
- Send debug information via a serial bus
- Send information via CAN **(bonus)**
- Display real-time information from the Morpheus board **(bonus)**

Design Methodology

The tools used in the design process:

- PHP (Server side scripting language)
- JavaScript (along with jQuery & other libraries)
- HTML/CSS
- Java (Android SDK)
- C in MPLabs
- MicroChip Pic Explorer Board/Can Daughter Board
- ArcX Morpheus Processor Board

The design methodology is to abstract the hardware emulation from the actual software integration with the real e-car. The software will be broken up into server side and client side development. This allows for parallel programming of the entire application without an overload of dependency of workloads being finished. The hardware emulation will be tackled by the third student as a side project to demonstrate the functionality and internals of the BMS and its communication flow. Each of the team members will help in all of the individual projects but will be assigned priority of their own.

This report will be broken down into the software and hardware design process.

PEV Battery Manager Software Outline

1.0 Server

The server side code is designed in PHP with a SQL database storing all relative information. The PHP allows for information to be transmitted and received from the cloud server to the mobile application. Below the organization of the server is illustrated.

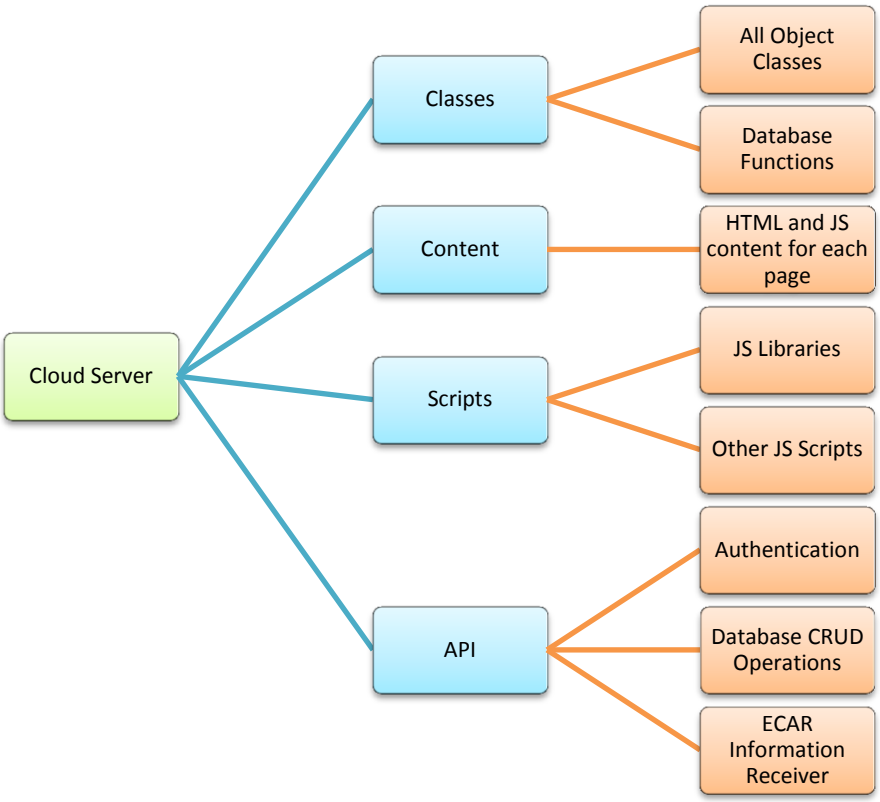


Figure 1.0: Server Organization

1.1 Database Design

The EVE-Monitor will have a database with four tables to store all required information. This design will help for future scalability and incorporation of many car entities to help preserve the intention of fleet operations.

1.1.1 Overview

The data base consists of five tables that will organize and store all information relative to the project. The tables included are user, e-car, batteries, modules and locations. The database employs a hierarchal system in which every table has a foreign key associated with the table above it. The table hierarchy is in the same order as listed above. This creates a dependency amongst data flow which will allow the server to follow the same systematic path to obtain information relative to the e-car.

1.1.2 Tables

Users: the main referencing table in which all queries will address first.

Column	Data Type	Notes
USER_ID	Integer(11)	Primary key
username	varchar(100)	
password_encypted	varchar(150)	MD5 hash encryption
user_type	varchar(100)	Fleet Operator, Fleet Admin and Super
date_created	datetime	

Cars: the list of cars within a fleet. Each car has a fleet operator (user from users table with user_type of "FLEET_OPERATOR") and a fleet admin (user from users table with user_type of "FLEET_ADMIN").

Column	Data Type	Notes
ECAR_ID	Integer(11)	Primary key
ecar_name	varchar(100)	
license_plate	varchar(50)	License plate of relative car.
ecar_address	varchar(50)	IP address obtained from e-car's UDP packet.
USER_ID	Integer(11)	FK-Users to USER_ID (Fleet Operator)
ADMIN_ID	Integer(11)	FK-Users to USER_ID (Fleet Admin)

Batteries: the battery of the car. Each battery contains 24 modules with 4 cells each. The battery table uses the ECAR_ID as a foreign key to for data requests.

Column	Data Type	Notes
BATTERY_ID	Integer(11)	Primary key
ECAR_ID	Integer(11)	FK-Cars to ECAR_ID
soc	double	
mode	INT	
state	INT	
faultmap	INT	
Vstate	INT	
voltage	double	
vCellMin	double	
vCellMax	double	
ctMap	INT	
Balancing	Bool	
Current	double	
maxCurrentOut	double	
maxCurrentIn	double	
tState	INT	
cTempMin	double	
cTempMax	double	
pTempMin	double	
pTempMax	double	
discharge	INT	
sensors	INT	
post_time	datatime	

Module: The modules are the individual cell arrays within the e-car battery. Each module consists of 4 cells.

Column	Data Type	Notes
Module_ID	Integer(11)	
BATTERY_ID	Integer(11)	FK-Batteries to ECAR_ID
soc	double	
voltage	double	
current	double	
temp	double	
pcb_temp	double	
flags	bool	
cell_0	double	
cell_1	double	
cell_2	double	
cell_3	double	

Locations: Car locations at different times.

Column	Data Type	Notes
ECAR_ID	Integer(11)	FK-Locations to ECAR_ID
longitude	double	
latitude	double	
outdoor_temp	Int	
w_condition	varchar(50)	Weather condition as a string
speed	Int	Speed at which car was going
work_location	bool	
home_location	bool	
post_time	datatime	

1.1.3 Designer View

Below in figure 1.1, the database heirarchy dependency is shown. In order to retreive information on an ecar, first the user must return the ecar corresponding to that user. Then to retreive battery, module and location information, the ecar ID is used to retreive relevent information. To narrow down the results, the post times of the information can be used as a constraint. This database design will allow for easy querying structures to obtain required information for both analytical and visual purposes.

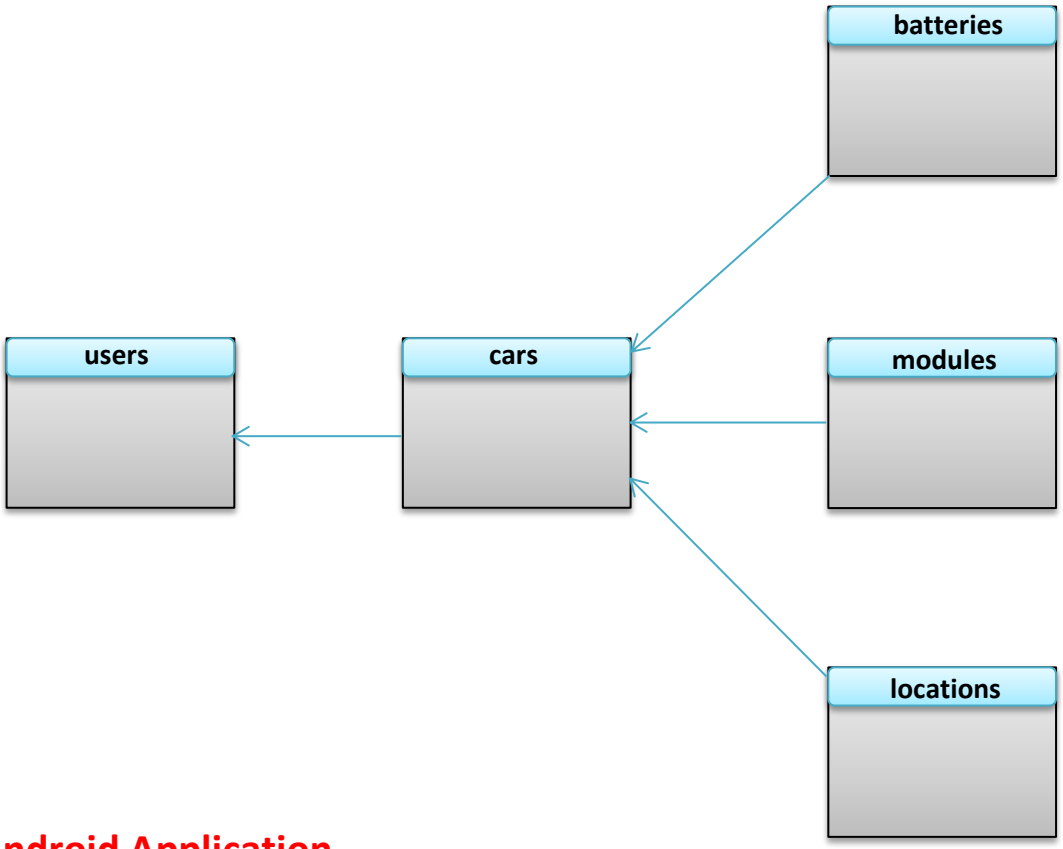


Figure 1.1: Database Hierarchy View

2.0 Android Application

The android application incorporates the ideology of fragments within the android development kit. This takes advantage of the web view to incorporate cloud architecture within the application without compromising the inability to communicate with the device. Below in *figure 2.0* is a demonstration of the device layout within the MEVE application.

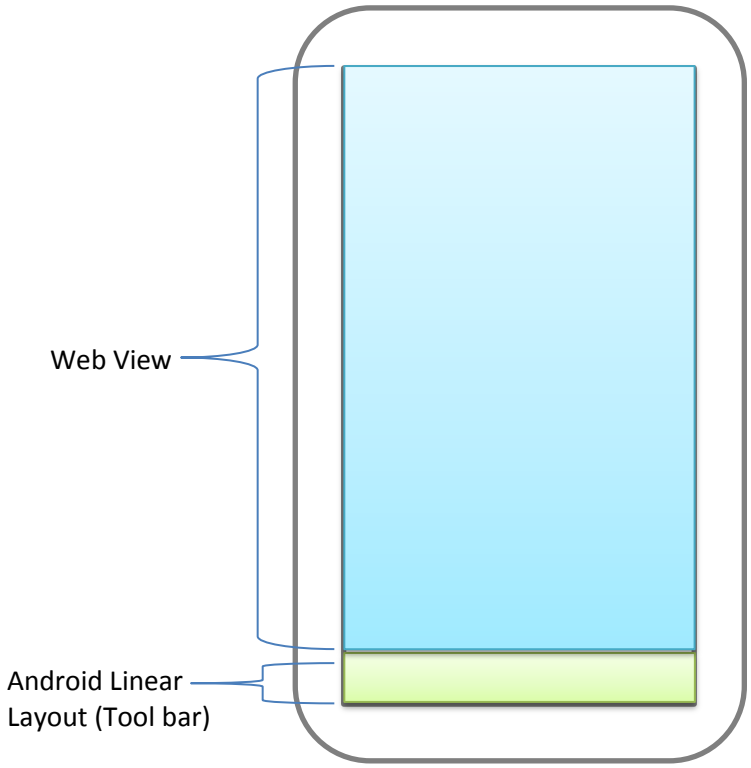


Figure 2.0: MEVE Android Outline

2.1 The Web View

The web view allows the application to connect to the cloud server via HTTP. This also allows for all UI design to refrain from being constructed within the application. This way the storage resides in the server and the android application simply loads the information as requested.

The web view also incorporates a JavaScript interface allowed by the chrome web vie client. This interface will allow for internal client-side communication between the device and the web application. By using this interface, the phone will be able to receive broadcasts using JSON strings. The broadcasts will then be interrupted within logic coded into the interface. The interface, for example, can request current state of charge information. If the state of charge is below a specified margin, the phone can then broadcast a notification to the user that state of charge is low and the e-car will soon be in need of charging. Using this design metrology, this will abstract the phone itself from synchronously storing information alongside the main cloud server database.

2.2 The Tool Bar

The tool bar will illustrate a text view of the current server status (whether the server is online or not), the current user logged in and the current e-car selected for information viewing. This small set of data will be used for authentication requests to the server to view information regarding the user's e-car. If the user is a multi-car owner (a fleet administrator of super user) the phone will allow the user to select between lists of cars from an android list view. The tool bar will also include a refresh button to refresh the web view in case of errors.

2.3 Web View Interaction with the Server

When the application first loads up a login screen appears to authenticate the user. The login screen is built in within the web app and stores the user information in an android XML file. Before the application can start the session, it must send a user object that contains the username and password to the server. The client will then authenticate the login information and return with the appropriate webpage or an error screen indicating the wrong user information was sent.

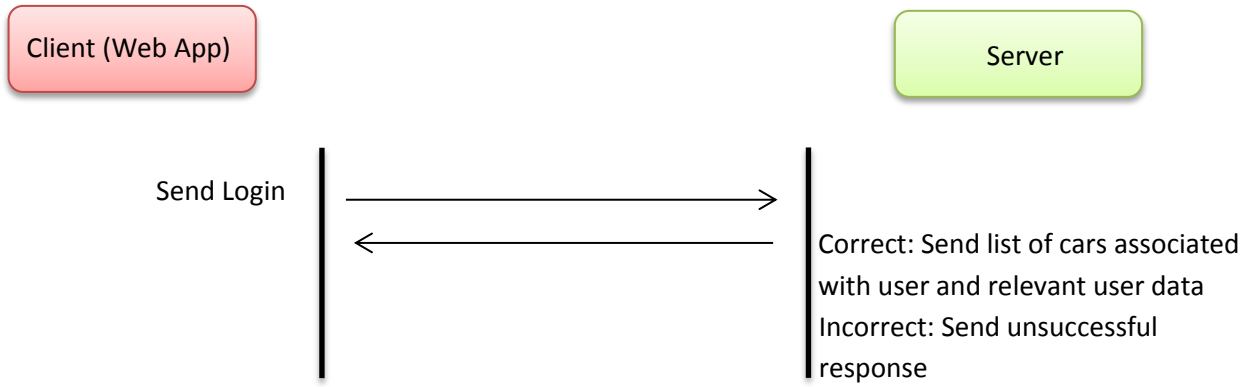


Figure 2.3: Client – Server Data Flow

2.4 Web View Interaction with the Client

In order for the android application to establish a communication channel with the web view, a JavaScript interface must be deployed. This JavaScript interface will allow the web application to send JSON information the android phone and vice versa. The interface will have functions to enable the client to notify the android phone if the login was successful, send state information and the value of the state of charge in real-time. The JavaScript interface will also allow the web application to interface with the functionality android provides within its API such as notifications.

Figure 2.4: Client – Android Data Flow

3.0 Mobile Web Application

The mobile web application is the final piece within the software design. It is the page that the web view loads and interacts directly with the servers API to retrieve information.

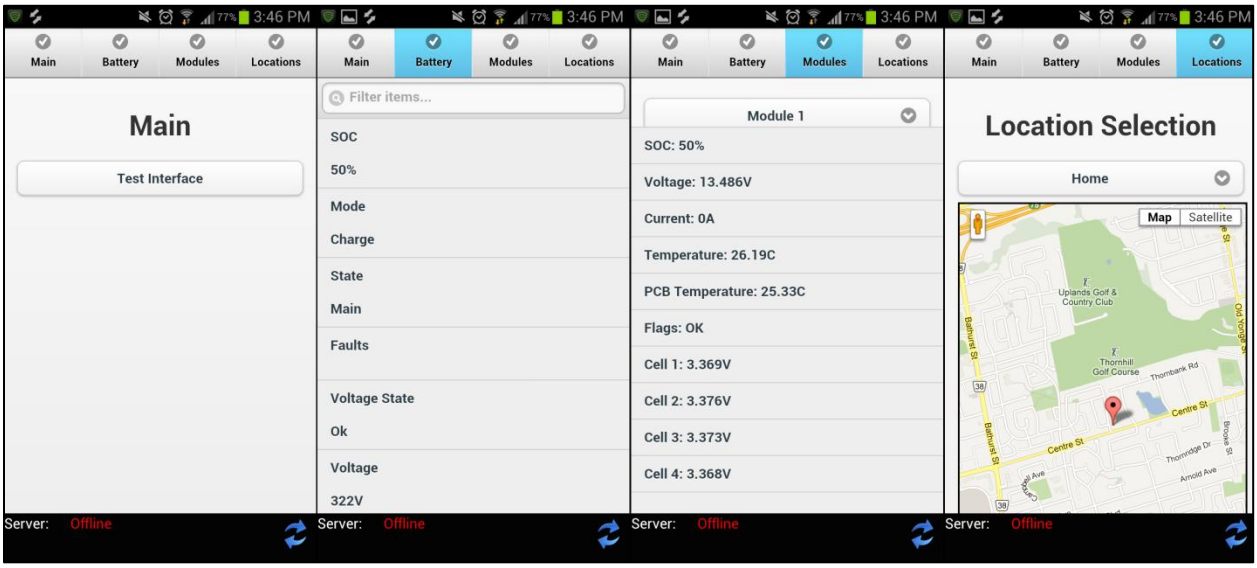


Figure 3.0: Current Mock Web App View

3.1 Overview of Template

The mobile web application used the jQuery Mobile framework to create smooth, modern looking user interfaces. This framework also allows for use of swipes and native touch interfaces within the android phone.

3.2 Content Loading

The content of the web application will make use of the jQuery load function that will allow for asynchronous loading of html information into the content div. This will eliminate the need of refreshing the page and reloading the application every time the user wishes to change information. This also allows for a cleaner set of code that is more scalable and does not require individual pages per context.

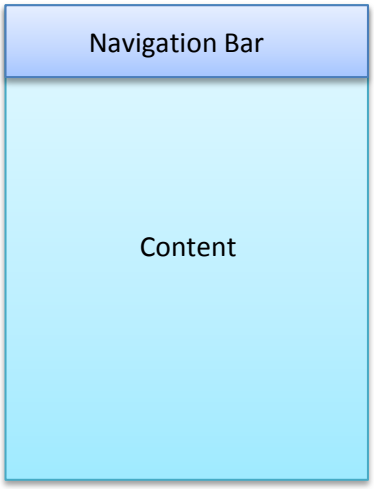


Figure 3.1: WebApp Layout

3.3 Periodic Database Querying

The information querying of the ecar is a real time system which will require a periodic request function to update the database with the latest information. PHP is a server side scripting language which cannot periodically run itself. To create the ability to run a script periodically, a cron (time-based job scheduler) will be used to run a PHP script. The server is based on a windows (Azure) server so the cron functionality is already built into the operating system.

As seen in figure 3.2, a basic diagram is used to show the abstraction of the cron function from the cloud server. This design will enable the ability for the PHP script to handle requesting ecar information and give it the ability to run periodically.

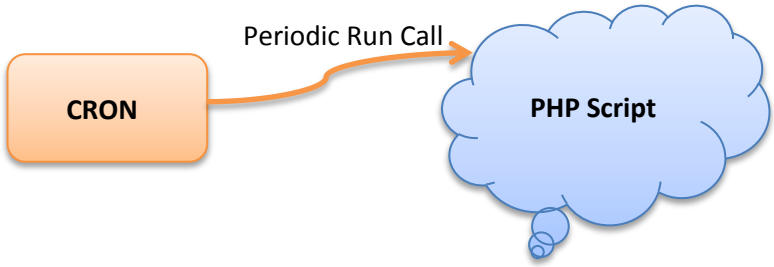


Figure 3.2: Cron Function

3.4 Real-time functions to deal with drops in SOC and other flags

The web application will be able to employ a real time environment by periodically updating its information from the database. The main targets will be flags and charge states. Below in figure 3.3 is an example of the process of real time notifications. When the web app first loads the page it performs an asynchronous call to the server to get the latest information. When this call is preformed the JavaScript will parse through the information and determine if any flags are set off and if the state of charge is below a specific value. If nothing flags, the call will repeat itself in a specific interval and preform the checks again. Once a flag is set, the web app will notify the android device by using the JavaScript interface provided by the android application. It will send a notification message as well as the flag triggered and its value. The android application will then be able to use its API to the phone and send a notification out the user. These events will trigger as long as the application session does not close.

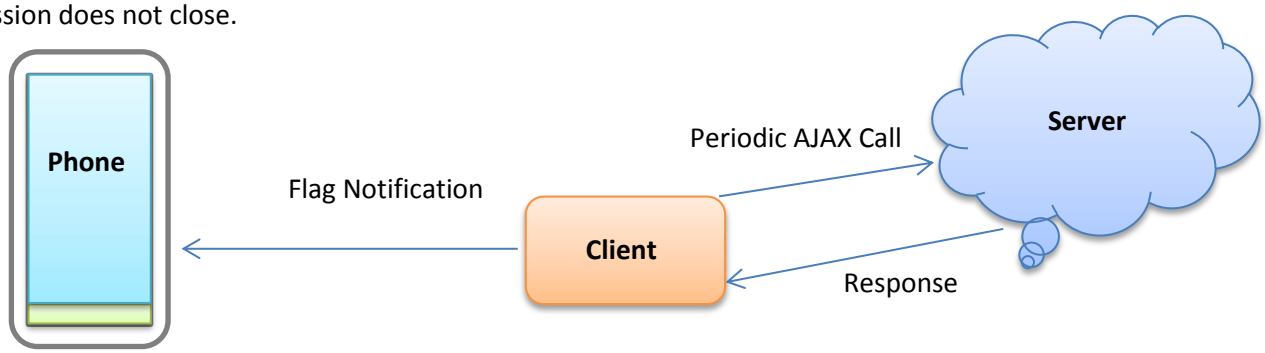


Figure 3.3: Real Time Notifications

3.5 Web App Content

The web application consists of 5 content pages. This section will go into detail of the functionality of each of the pages.

Main

The main page will be a quick guide to display all general information. The page will show the latest state of charge, mode of operation and temperature state. An animated battery will be used to visually represent the state of charge.

Battery

The battery page will display all information held within the battery table listed in the 1.1.2. The information will be listed in a simple block list format.

Modules

The modules page will use a dropdown box to select individual modules. Once the module is selected the web app will perform an asynchronous call and request the information relative to the module selected.

Locations

The locations page will use the Google maps API to load a Google maps within the page. The page will query the latest location information and drop a marker accordingly.

History

The history page will load up a list of possible maps to generate. These maps will only generate on request due to the amount of information required. The user will be able to select between plots listed in section 4. The system will be very similar to the module page as seen in figure 3.5.

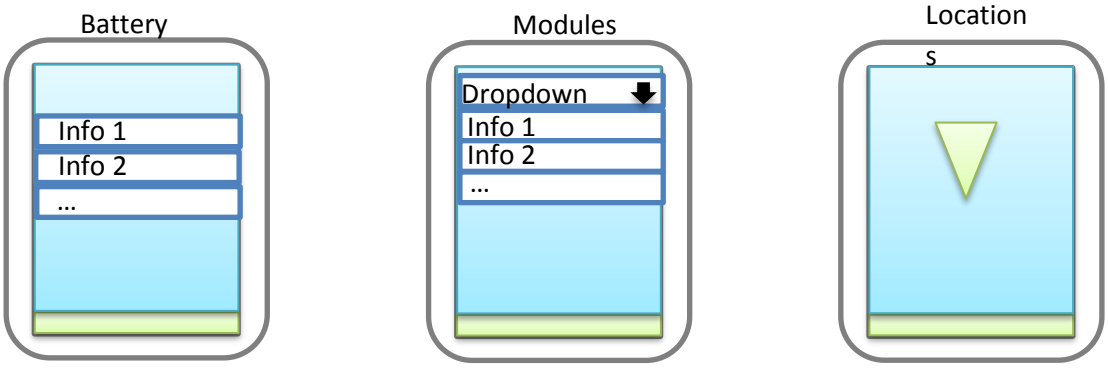


Figure 3.4: Various Page Templates

4.0 Statistical Analysis

4.1 Common Route Calculations

Within society the main method of transportation is from some sort of vehicle, mostly from people driving their cars to some destination. Given a long period of using a vehicle, there will be many paths that become clear-cut and an everyday option. Whether, the route taken is from a day-to-day basis or week-to-week basis, just about everyone has a common route that is used. Now, when talking about a normal gasoline-fueled car, this may not be significant, however in terms of the E-Car, the knowledge of this common route is vital. Since the E-Car is fueled by electricity, and has a limited amount of driving capability which is dependent upon elapsed charging time, knowing the common route can save a lot of time and money. If someone was to drive to work from Monday to Friday every week, they would take the same route (most likely the least distance to work) to get there. With the common route calculated, the least amount of time to charge the car daily such that it can get the user to work and back can be determined; this would save a lot of time, money and worries. Also, if the user was planning on using the vehicle to travel further distances, an approximation of charge time could be given based upon the charge time required for the common route.

4.2 Weather Effects on Battery

As is common in most electronic devices (especially vehicles) with power (or fuel) as a performance indicator, weather has a huge impact on the E-Car battery. Having a battery charge or discharge during the summer season will not yield the same charge or discharge results during the winter season. Therefore, if the E-Car would want to be the primary vehicle of the majority of the population, this must be taken into account. For example, if it was determined that the battery needed to charge for a longer period of time in the winter season, and the vehicle owner would charge the car for the normal amount of time, this could lead to drastic situations such as the car losing power in the middle of traffic. By having a detailed description of how the weather affects the battery, users will be aware of the required charge time to operate at satisfactory or optimum operational level.

4.2.1 Temperature versus State of charge

The temperature can directly affect the charge time on the battery. For example, take a normal winter day; the average length of time for the car's engine to be fully function for a gasoline-fueled car is significantly longer than on a normal summer day. This can lead to different charge times based upon the surrounding temperature of the battery. Therefore, in order to have the car charge its battery optimally, there would be an ideal temperature at which the battery would charge the fastest.

4.2.2 Weather Condition versus State of charge

The weather condition is probably the most important factor in the charging of the battery. Each of the different weather conditions has a different effect on the battery charge. For instance, in the event that there is a thunderstorm in the area that the E-Car would be charging, it is possible that any thunder that would hit the car or its surroundings could create a disruption with the charging process either by stopping it, increasing the charge time or even decreasing charge quality (as in some of the charge is being lost somewhere in the process). An example of how the weather would play an impact upon the battery usage is shown in Figure 4.0 and Figure 4.1 below.

Figure 4.0: Power consumption in weather conditions

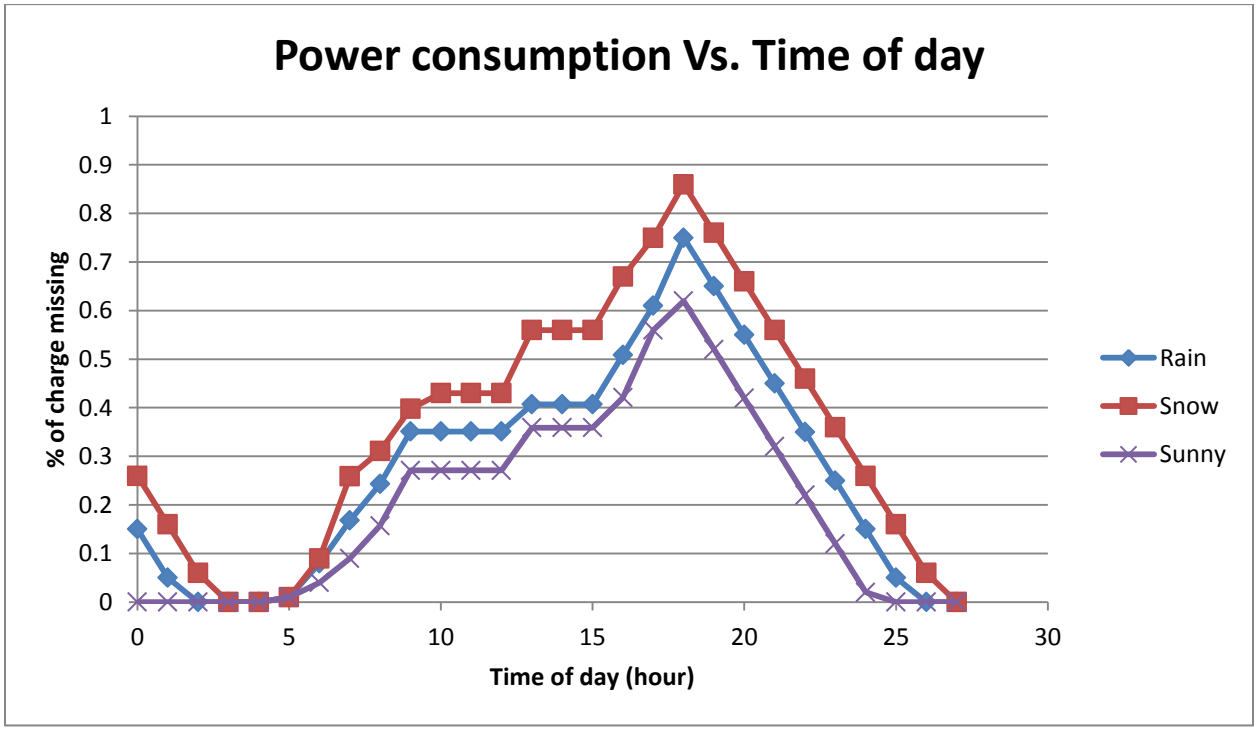
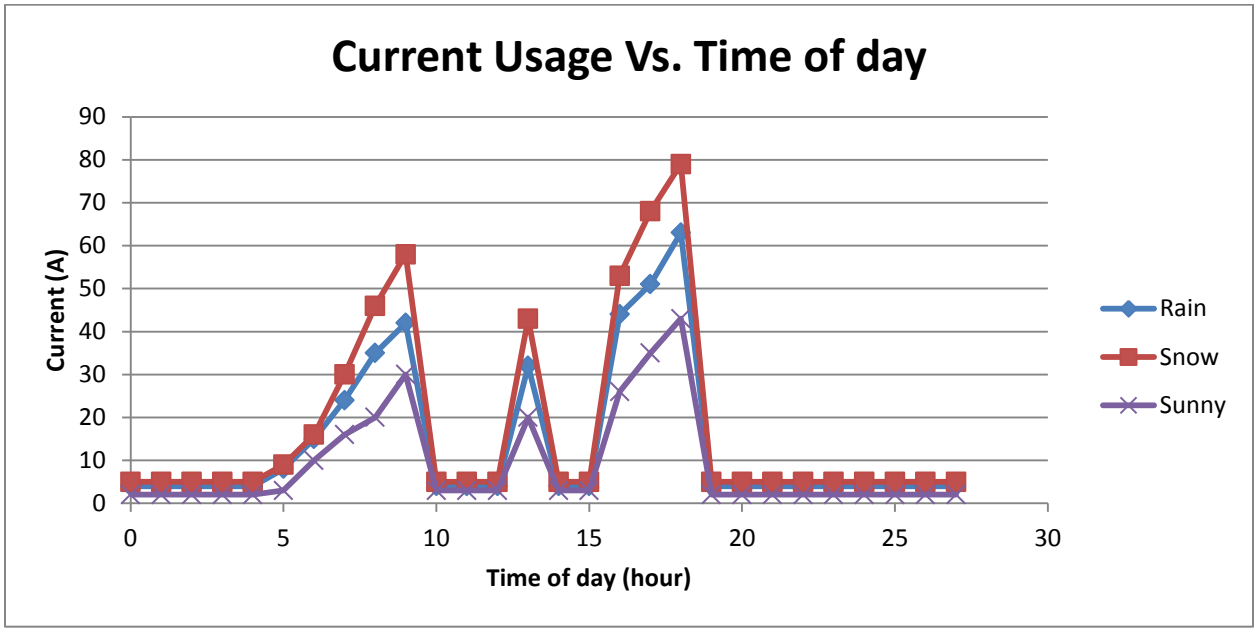


Figure 4.1: Current usage in weather conditions



The diagrams above use fabricated data; however the curves are similar to the actual E-Car power consumption and current usage. This is because of the idle state of the car, when it is idle, the % of charge missing will be constant, and the current usage will drop down to its lowest point. Also, the weather effects on both power and current are things that can be pre-determined because of the conditions a car must endure to be operational. One thing to note is the slope at the end of Figure 4.0, this would be equal to the charge rate of the E-Car, and the actual value would be dependent upon a number of different things such as: battery, voltage used to charge, amount of time charged and quality of charge.

4.3 Database tables

Inside the cloud server will rest a database containing all sorts of available information about the electric car. There will be five tables in the database: users, cars, batteries, modules and locations.

To the person driving the car daily (Fleet Operator), the “users” and “cars” database will be unavailable because it shows all the cars and people driving the cars, this data will be obtained upon a registration basis. The other two users Fleet Admin and Super are administrators that are in charge of knowing the condition of each car and driver.

The second table is specifically for electric cars and their identification parameters. It is important to note that this table is designed for multiple electric cars, as in there would be a company that manufactures or sells the electric cars that would need to have access to each car. The data in this car would also be upon a registration basis, although the license plate data would be dynamic and

dependent upon the owner of the vehicle. The car address is a piece of data that would be obtained through the server and client program that is implemented on the cloud server hosting the database. Upon sending a request for data to the car, the car will send back an acknowledgement with the IP address of the car and all other relevant data for data transmission.

The next table is the “Batteries” table containing all the information about every battery inside of the specified electric car. All of design specification data would be inside of this table, it contains information on: state of charge, operation mode, the state of the battery, voltage, current, maximum and minimum cell voltage, temperature, sensor toggling, discharge toggling and post time of data. All of this data will be obtained through the UDP data transmission process from the car to the cloud server. The car will send packets of data, and the program will separate the packet into each of these variables.

The Modules table contains data of each module inside each battery (twenty four modules per battery). There are also four cells within each module, which is the method of how the battery will be charged. The modulation of these cells in each module will determine whether the car will be charging its battery or not. Each module contains data that is obtained by the UDP program on: total voltage per module, current, temperature, and the individual cell voltage of each cell.

The last table is the location table, containing specific data that would correspond to satellite and global co-ordinates of where the car is currently located. This table is primarily used for GPS or tracking purposes; this will be the main table that a fleet administrator would be interested in. This data will be obtained through the use of yahoo location service, this data will be obtained in the form of JSON (JavaScript Object Notation) after which it will be parsed and separated in a similar fashion to the “Batteries” table. The work and house location variables will be determined by having the user enter the work and home locations and comparing with the current location. The speed and outdoor temperature are just parameters that will be obtained through the UDP packet transmission program.

PEV Battery Manager Hardware Outline

Hardware Simulation

The hardware simulation will give the project a visual demonstration of how the E-Car reads and transmits data to the database. Three major steps need to be taken in order to achieve the total hardware simulation in this project. The first step is to simulate the data capture through sensors. In the demonstration, voltage will be read from a series of four cells (single module) which will be simulated through Lithium Ion batteries. These batteries are ideal to simulate the actual batteries used in the actual E-Car. Once the data is captured, the information needs to be processed by means of a microcontroller. The microcontroller must have the necessary components in order to read, display and transmit the data. The final step will be to display the data through SPI and CAN interface to software interfaces and eventually to a server. In the figure below shows a hardware diagram displaying the entire hardware demonstration of the project.

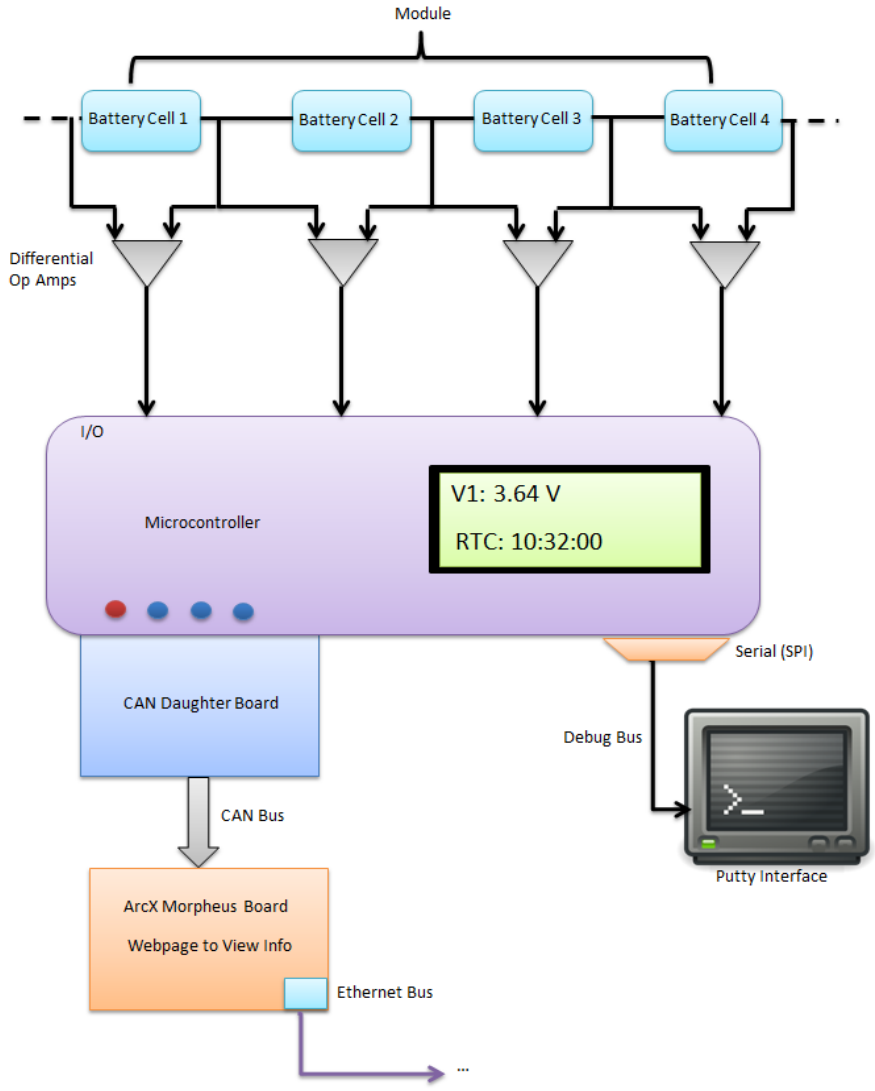


Figure 5: Hardware Diagram

2.0 Battery Simulation

The simulation of the PEV battery will allow for a visually clear understanding of how the voltages are read and flagged as unbalanced through the use of a microcontroller. Using separate battery cells allows for a realistic analysis of the actual batteries being used in the E-Car. Four cells should be chosen in order to simulate a single module. The figure below shows a diagram of the battery-microcontroller relationship.

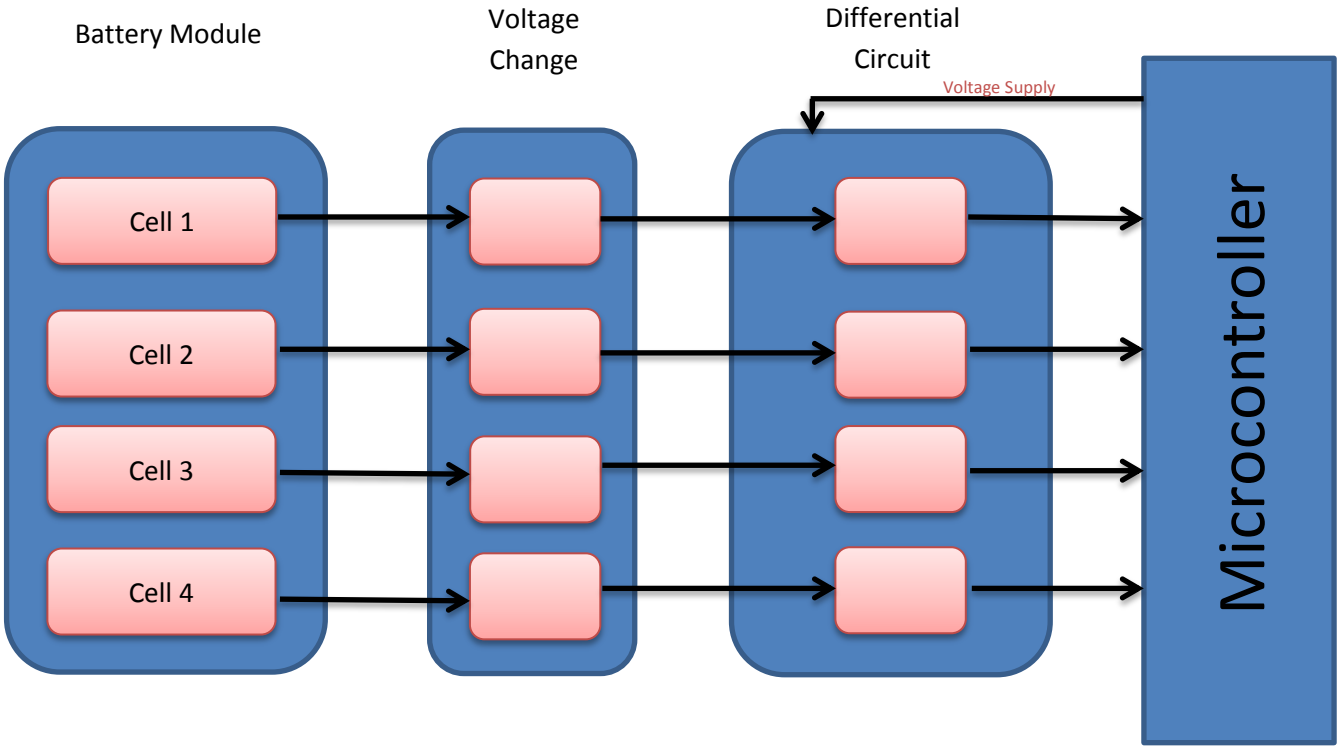


Figure 5.0: Battery Simulation

5.1 Battery simulation design

The electric vehicle uses a Lithium Iron Magnesium Phosphate battery. To best simulate a module from the actual module of the EVE project, Li-polymer battery cells are chosen. The battery cells chosen for this project is the 063048 LI-Polymer Battery. This battery provides a 3.7 V, 850 mAh and a two-pin JST-PH connector [1]. It will simulate how the actual module runs as it is a similar type of battery. Four Li-polymer batteries will be used to represent a single module. The 850 mAh indicates that if 850 mA are drawn, the battery will last for a single hour, thus the demonstration needs to take current being pulled into consideration. The figure below shows a photo of the battery cell mentioned.

Figure 5.1: Single Cell Battery Simulation

5.2 Differential Circuit

In order to properly access the battery voltage from a series of battery cells, a differential circuit needs to be set up. The four battery cells used will be set up in series, thus reading voltage from each cell needs to be done by taping the positive and negative terminal of the cell and using a differential amplifier to show the proper voltage. A LM741 is efficient enough to use as it can handle the DC values being used. The 741 op-amp can handle ±22V supply and gains of up to 10^5[2]. Both these properties can handle the requirements needed for the battery simulation demonstration. In order to achieve unity gain, matching resistors need to be chosen which are capable of handing the voltage without giving off too much current. The following diagram and equations are used in order to achieve differential voltage and unity gain:

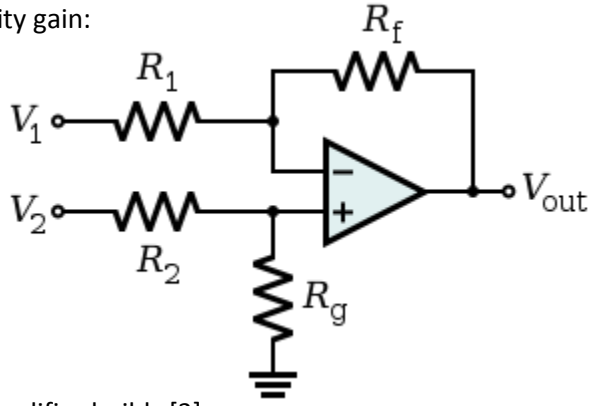


Figure 5.2.1: Differential amplifier builds [3]

$$V_{out} = \frac{(R_F + R_1)R\_G}{(R_G + R_2)R_1}V_2 - \frac{R_F}{R_1}V_1 = \frac{R_F}{R_1}(V_2 - V_1)\ [3]$$

Thus, if R_F & R_1 are equal, a unity gain will be achieved. Four 100 kΩ resistors are used at both ends of the amplifier terminal in a differential amplifier set up. Below is the LM741 pin diagram. Pins 2 and 3 are the inputs, pin 6 is the output, and pins 7 and 4 are the positive and negative source terminals respectively [2]. The voltage supply will be coming from the 9V supply of the microcontroller and negative terminal of the module itself. By applying the 9V supply from the microcontroller instead of the batteries themselves reduces the amount of current being drawn which allows the batteries to last longer during the project demonstration. In order for the Op-amp to continue operating, the voltage being fed in must be a factor of 0.8 or less. The highest voltage being fed into the amplifier is about 7.4 volts of the first amplifier which can be handled by a 9 volt supply. Below is a pin diagram of the LM741 amplifier.

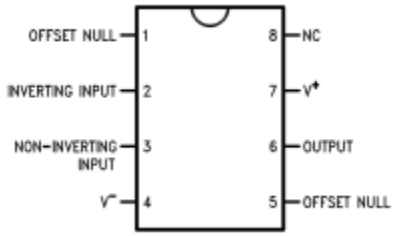


Figure 5.2.2: LM741 Pin diagram [2]

5.3 Voltage Change

By using potentiometers in the battery simulation, unbalanced modules can be manually created by turning the potentiometers. A 100kΩ potentiometer is applied to every cell in order to be able manually change each battery cell. The potentiometer works a lot like a voltage divider.

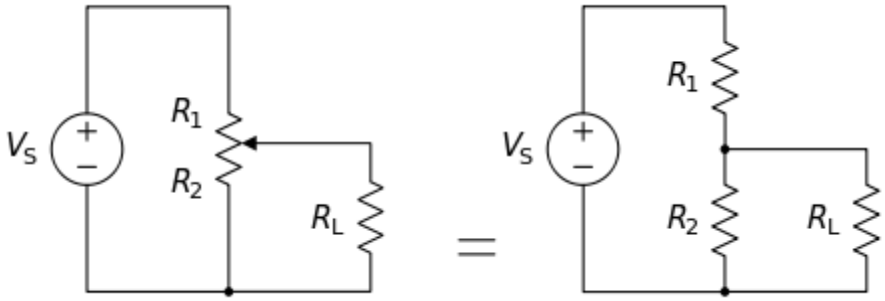


Figure 5.3.1: Potentiometer equivalent circuit [9]

$$V_l = \frac{R_2}{R_1 + R_2} * V_S$$

In order to achieve 100% output, a very small R_1 needs to be applied which will result in $V_l \approx V_S$. Thus, a potentiometer provides a simple way to show an unbalanced module by manually creating battery voltage loss. A 100 kΩ potentiometer is used because the high resistance minimizes current and will allow for the battery to last longer.

5.4 Multisim Design

Multisim was used to create the battery simulation for the PEV hardware simulation. Four 3.7 V DC voltage source components are used to act as the polymer lithium ion batteries used. 100kΩ potentiometers, 100 kΩ resistors, and 741 operational amplifiers are used as well the high resistors are necessary in order to lower the current being drawn which will not use the batteries too quickly. The figure below shows the battery schematic designed in Multisim with the output voltages that will be fed into the microcontroller ADC ports. Also note that the potentiometers are set to 100% in the figure.

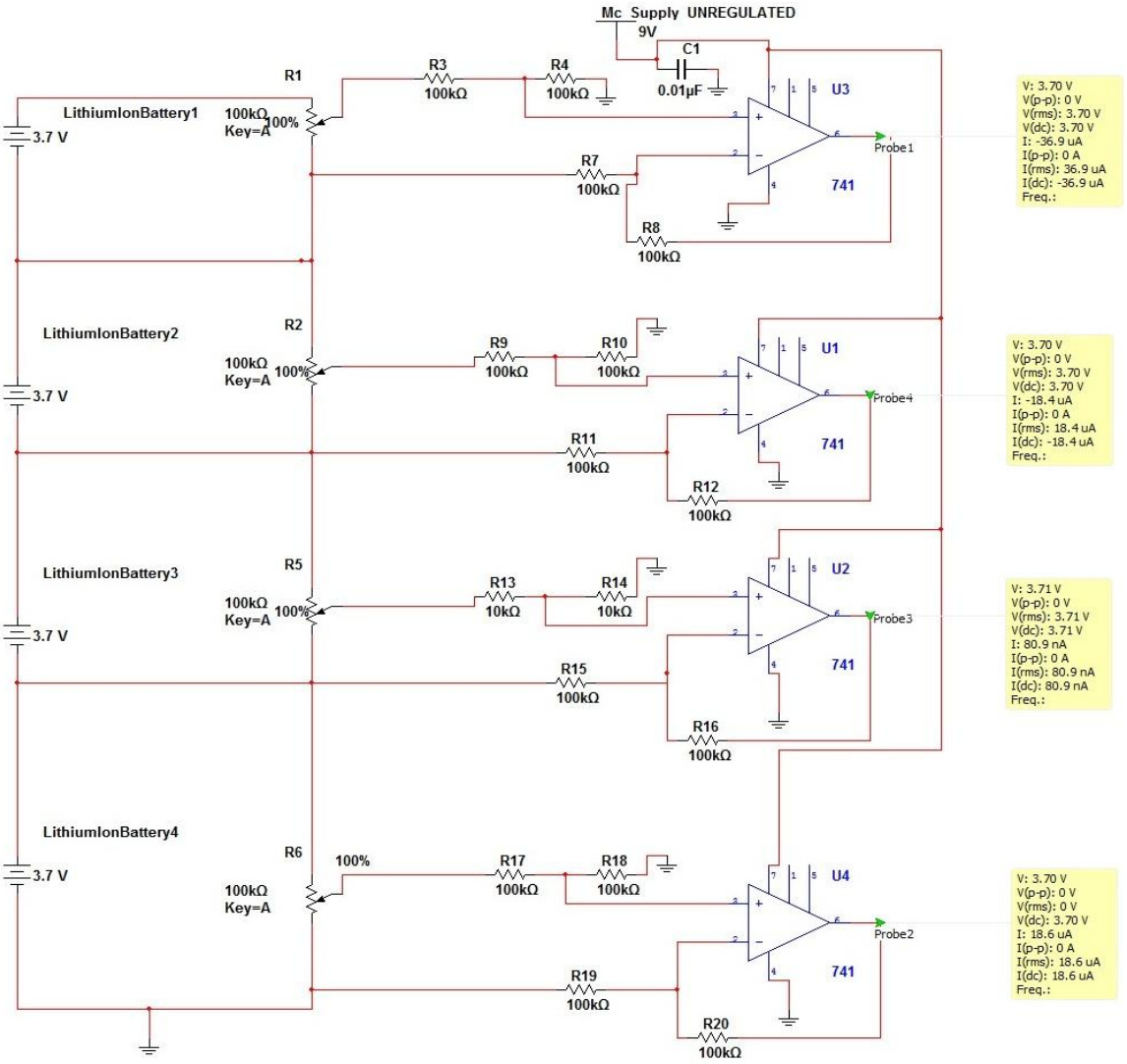


Figure 5.4.1: Multisim Schematic

The Multisim showed the expected results. The output pins gave the voltage drop across each lithium ion battery and the output current was very negligible, thus the microcontroller can handle the outputs. It was also found that the battery module will only pull approximately, 10 mA which results in battery life of around 80 hours on a single charge. This time is more than acceptable for our demonstration.

6.0 Microcontroller interface

The microcontroller is used to grab data, process it, and send the data through SPI and possibly CAN for further analysis. The microcontroller must be able to support basic components like ADC and DAC conversion and RTC (real time clock) capability. An LCD should be included as well to show real time results being grabbed from the battery simulation and current time. The microcontroller should also have SPI and CAN capabilities. The microchip MPLAB ICD 3 & Explorer 16 kit provides the software and hardware needed to perform the tasks needed for this project. The explorer 16 demonstration board gives for easy microchip use and provides all necessary components. The kit also includes a DsPIC33FJ256GP710A microchip which provides the appropriate pins and registers in order to achieve what is needed. MPLAB IDE software and MPLAB ICD 3 is also included and allows us to program the microchip.

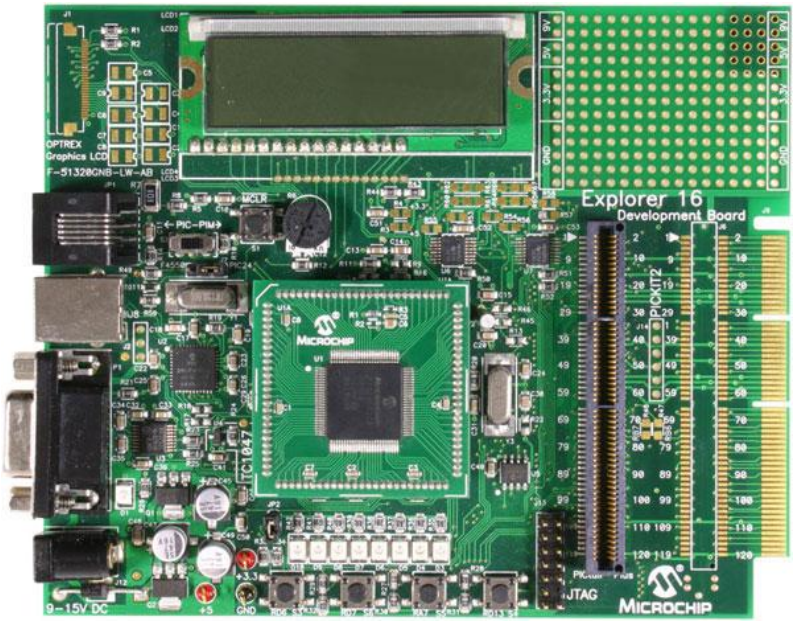


Figure 6.0: Explorer 16 board with DsPIC33FJ256GP710A [4]

Figure 6.0.1: ICD3 Debugger [5]

6.1 Explorer 16 Demonstration board.

The explorer 16 demonstration board allows for appropriate interaction with the microcontroller and the communication interfaces. The board has many features yet the relevant features included are the following:

- 100-pin PIM riser, compatible with the PIM versions of all Microchip PIC24F/24H/dsPIC33F devices
- Direct 9 VDC power input that provides +3.3V and +5V (regulated) to the entire board
- RS-232 serial port and associated hardware
- USB connectivity for communications and device programming/debugging
- Push button switches for device Reset and user-defined inputs
- Eight indicator LEDs
- 2-line by 16-character LCD
- Independent crystals for precision microcontroller clocking (8 MHz)
- RTCC operation (32.768 kHz)
- Socket and edge connector for PICtail™ Plus card compatibility

6.1.1: 100 pin PIM riser

As the explorer 16 boards can be used for multiple microcontroller applications, the explorer 16 board includes a PIM riser for easy microchip changes. In our lab, we will be using a dsPIC33FJ256GP710A microchip which was included in the kit [6].

6.1.2: Direct 9 VD Power input with regulated supplies

A 9 volt VDC power supply input is included on the board. This is used to power the board and allow for all the components to run accordingly. The 9 volt power provides 3.3V and 5V supplies which power everything on the board from the LCD to the switch buttons and LEDS [6].

6.1.3: RS232 Serial port

This component will allow for communication between the microcontroller and the computer. The will be used later for data transfer to the computer interface and eventually the software database.

6.1.4: USB connectivity

The ICD 3 debugger communicates through the USB interface. The Debugger will allow the computer to interact and program the microcontroller.

6.1.5: Push Button switches

The switch buttons located on the board will be one of the two components that will allow for user interaction. There are 5 switches available for use. S1 switch is generally used as a reset button. It allows the user to return to the top of the program. Switch buttons SW3,SW6 and SW4 ports are located in register port D as D6,D7 and D13 respectively, SW5 switch is located in register port A under port A7.

Switch SW5 is also connected to LED 10(JP2). The switches will be used to show the voltage for each cell on the LCD [6].

6.1.6: LEDS

8 LEDs are included on the explorer 16 demonstration board. These LEDs are located in port A register under ports A0-A7. These LEDs provide easy user indication that the task is being done. The LEDs will flash every time data transfer occurs through SPI and it will show which battery is being shown on the LCD. As well, the LEDs will indicate which batteries are unbalanced by simply turning on when voltage difference is greater than 10 % compared to average of the voltages [6].

6.1.7 LCD

The LCD is the most important component of the explorer 16 demonstration board as it provides the most user friendly interface. The LCD will be used to show the voltages and time being read by the batteries. The LCD will also indicate to the user that the module is unbalanced. The LCD installed on the explorer 16 board is a TRULY 16x2 3.3v (TSBG7000) GENERIC LCD module. The LCD primarily uses ports C, D and E in order to initialize it. It requires a 3.3 input voltage and is a parallel interface type LCD. The figure below shows the picture of the LCD being used [6].

Figure 6.1.7: LCD

6.1.8 Independent crystal & RTCC operations

The independent crystal is used for the internal timers. These timers are very important as they for the microcontroller to accomplish its tasks within its appropriate time. The timers will be used in the LCD display and in the data transfer of the project. The RTCC clock component is used to show the real time [6].

6.1.9 PicTail plus connector

The PicTail plus connector will allow the microcontroller to connect to the CAN bus daughter board. The daughter board will be used to transmit data via CAN interface [6].

6.2 dsPIC33FJ256GP710A microchip

The dsPIC33FJ microchip comes in the microchip development kit. Unlike the PIC24F chip which also comes in the kit, PIC33 specifically includes CAN bus capability. It also includes the basics capabilities like SPI and I2C communication peripherals, 5V tolerant input pins, and ADC. Below is a figure of all the available ports available on the microchip [7].

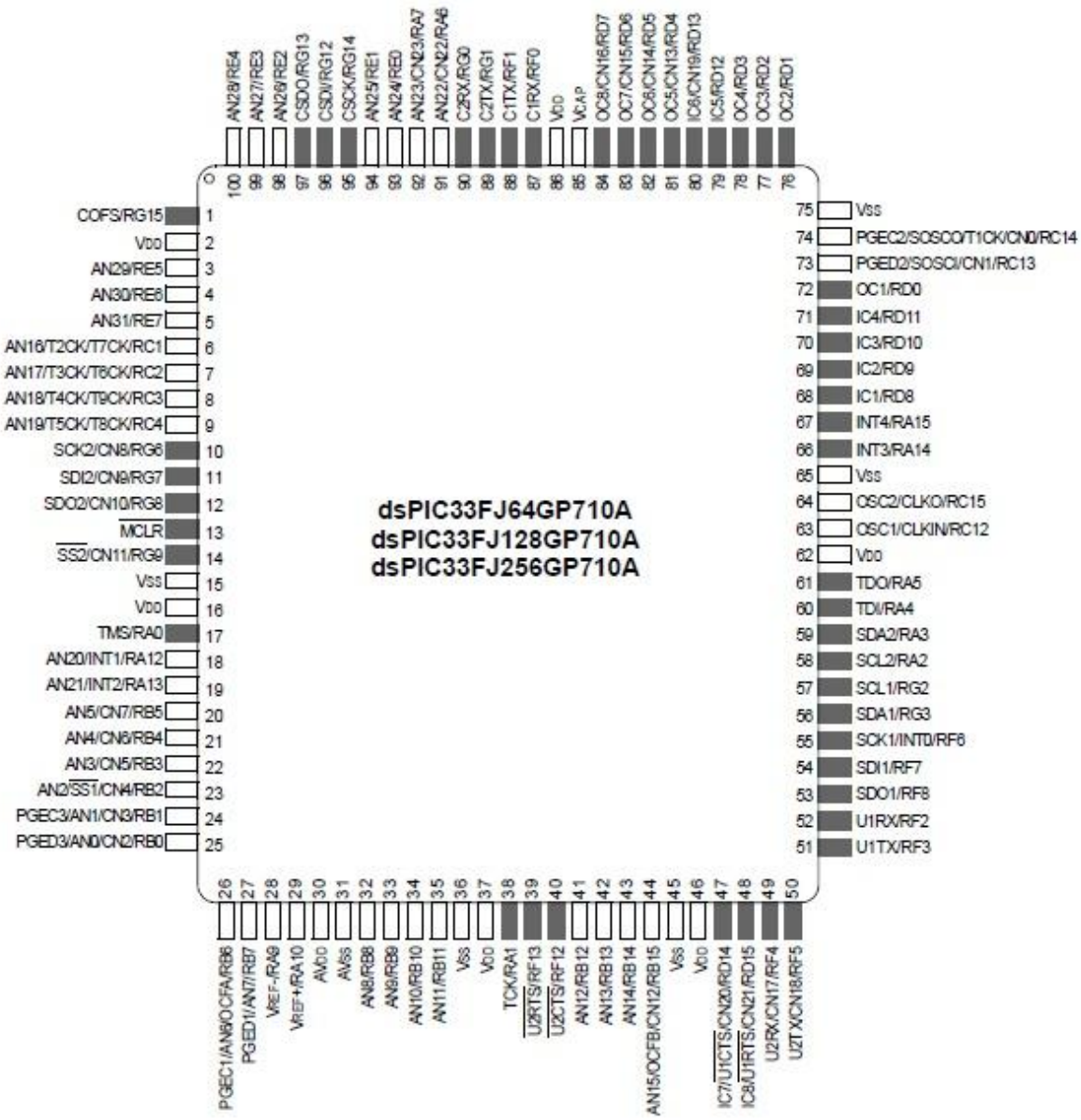


Figure 6.2: DsPIC33FJ256GP710A Pin diagram

6.2.1 Input ports

The input ports of the microchip are able to handle up to 5 V and the current must be less than 5 mA. In our case, the voltage will range between 0 and 3.7 V while the current was found to be approximately 37 µA in the simulation. With the values seen from the simulation, the microchip will be able to handle the values being read [7].

6.2.2: CAN Ports

The DsPIC33F has an ECAN (Enhanced controller area network) serial interface for communication with CAN modules. The module was designed such that communication can be done even in noisy environments. This is a very important aspect of the ECAN module as the vehicle being used will be driven in an open environment and external noise will present yet it will have no effect on CAN communication. The ECAN has standard and extended data frames and can send data lengths of 0-8 bytes. All these features are very important for our purposes and simulate the battery as realistically as possible to the actual battery modules being used in the electric car [7].

6.3 Microchip code

In order for the microcontroller to process and send the data correctly, the microcontroller needs to be designed in a flow chart form. As the voltages are sampled, there needs to be a constant check to see if the voltages are the same, if so the microcontroller will continue to wait for the switches to be pressed in order to present the values of voltages being read. If the voltages are not the same, a flag is set and shown on the LED's. A flow chart of this process can be seen below.

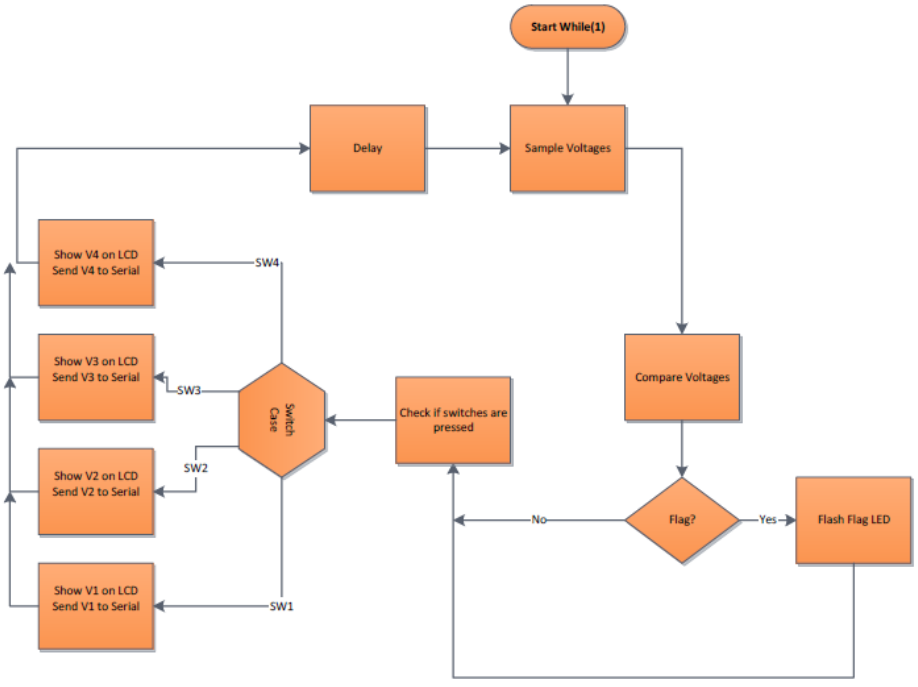


Figure 6.3.1: Code flow chart

7.0 SPI Interface

The SPI interface will allow for instant communication between the microcontroller and the software interface. In order to use the SPI interface, a few registers need to be set by the microcontroller. SPIxSTAT, SPIxCON1, and SPIxCON2 need to be set [7]. The STAT register indicates various status conditions like transmit buffer full, receive buffer full and receive overflow. It also enables and disables the module and specifies what operation to accomplish during idle mode. The CON2 register specifies the clock prescaler, master/slave mode, byte communication, clock polarity and clock/data pin operation. The CON2 register will enables/disables the framed SPI operation, specifies the frame synchronization pulse direction, polarity and the edge selection. One other register is related to the SPI interface and that is the SPIxBUF register. The BUF register is two separate internal registers, the transmit buffer (SPIxTXB) and the receive buffer (SPIxRXB) [7]. A user uses these buffers to write data to be transmitted to the SPIxBUF address and read the receiving data from the SPIxBUF register. The receiving and transmitting ports are tied to the RX and TX lines of the RS-232 which is located on the explorer 16 demonstration board. The software interface used in the project will be PuTTY, which is an open source serial console client. Using the SPI interface will allow for a better method of debugging due to the ability to see in real time what files are being transmitted to a from the CAN bus [7].

8.0 CAN bus Interface

The CAN bus interface will be done through the CAN/LIN PicTail (Plus) Daughter Board. The PicTail board features two ECAN nodes with MCP2551 transceivers. This board will provide us with the ability to communicate through CAN which is unavailable with the explorer 16 boards on its own. In the software code, the CAN module needs to be set to normal operation mode. In normal operation mode, REQOP<2:0> is set to zero. This will activate the module and the I/O pins will transmit and receive can bus messages via CiTX and CiRX pins. The figure below shows a photo of the daughter board being used in this project [8].

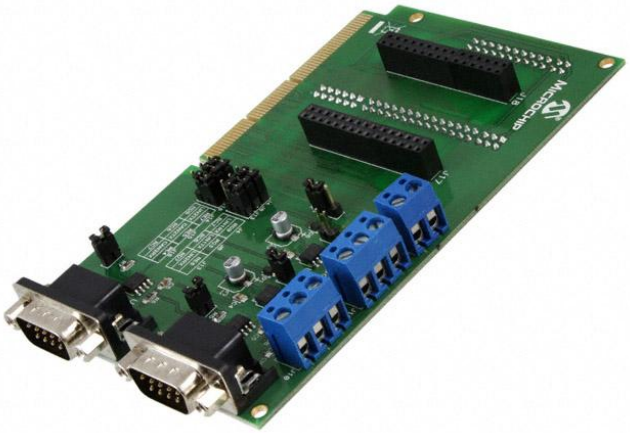


Figure 8.0: CAN/LIN Daughter board

9.0 Morpheus board interface

The final portion of the project will involve the Morpheus board which will be provided by ArcX. The Morpheus board that runs embedded Linux. It has a CAN port known as device net in which it can translate the CAN protocol received from the daughter board. Using the libraries already developed by ArcX, a webpage can be created that will read the CAN bus data using JavaScript. The information can then be displayed on a webpage that can be accessed using the Morpheus board local IP.

REFERENCES

1. 063048 LI-POLYMER BATTERY Specifications. 2006. Unionfortune. 16 Mar. 2006 <http://www.sparkfun.com/datasheets/Batteries/063048%20Li-polymer.pdf>
2. LM741 Operational Amplifier. 2004. Texas Instruments. May. 2004 <http://www.ti.com/lit/ds/symlink/lm741.pdf>
3. OP-Amp Application. I*ambiomed.com/*. I Am Biomed, n.d. Web. 23 Nov. 2012. <http://www.iambiomed.com/dac/opamp.htm>.
4. Explorer 16 Microchip. Digital image. *Http://www.machinegrid.com*. N.p., n.d. Web. 23 Nov. 2012. <http://www.machinegrid.com/machinepress/wp-content/uploads/2010/06/Explorer_16_Microchip_Development_Board2.jpg>.
5. ICD 3. Digital image. *www.mal.jp*. N.p., n.d. Web. 23 Nov. 2012. <http://www.mal.jp/open/products/icd3/icd3_l.jpg>.
6. Explorer 16 Development Board User's Guide. 2005. Microchip Technology Inc. 31 Oct. 2005. <http://ww1.microchip.com/downloads/en/DeviceDoc/Explorer%2016%20User%20Guide%205 1589a.pdf>
7. dsPIC33FJXXXGPX06A/X08A/X10A. 2011. Microchip Technology Inc. 29 Nov. 2011. < http://ww1.microchip.com/downloads/en/DeviceDoc/70593d.pdf>
8. CAN/LIN/J2602 PICtail™ (Plus) Daughter Board User's Guide 2011. Microchip Technology Inc. 11 Feb. 2011 <http://ww1.microchip.com/downloads/en/DeviceDoc/70319B.pdf>
9. Potentiometer with Load. Digital image. *Www.wikipedia.org*. N.p., n.d. Web. 25 Nov. 2012. <http://upload.wikimedia.org/wikipedia/commons/c/c9/Potentiometer_with_load.svg>.