Project Number: FC- PHL1

Mapping City Potholes

A Major Qualifying Project Report: submitted to the Faculty of
WORCESTER POLYTECHNIC INSTITUTE
in partial fulfillment of the requirements for
the Degree of Bachelor of Science of
Electrical and Computer Engineering
By:

Nicholas Angelini
na1303@wpi.edu _____

Matt Gdula
mgdula@wpi.edu _____

Craig Shevlin
cshvelin@wpi.edu _____

Jose Brache
jbrache@wpi.edu _____

April 27, 2006

Fred Looft
fjlooft@wpi.edu _____

Fabio Carrera
carrera@wpi.edu _____

Michael Ciaraldi
ciaraldi@,wpi.edu _____

100 Institute Road Worcester, MA 01609

Abstract

The goal of this capstone design project was to design a fully automated data collection system that could be installed in any automotive vehicle to monitor road or highway pavement conditions. The system is based on recording and analyzing the output from an accelerometer mounted near a front wheel shock absorber. The data from the accelerometer is time and position tagged with GPS data from a small, low power, OEM GPS unit and stored to a DOS compatible file on a Compact Flash card interfaced to the internal embedded processor system. The prototype system is powered from a power outlet for up to a week of data collection. Once data is collected, the Compact Flash card is downloaded into a desktop system after which Geographical Information System (GIS) software is used to remap the GPS surface roughness data to a physical location on a user viewable city map. The prototype is intended to be placed in city vehicles that roam about the city in order to map these potholes. This system can aid cities in the efficient release of road repair crew and lower the costs to the city due to civilian automobile damages.

Table of Contents

Table of Figures

Acknowledgments

Fred Looft – For his guidance on the project and patience

Fabio Carrera – For his creative ideas

Michael Ciaraldi – For his input in the initial stages of our project

Tyler Benoit – For his advanced knowledge for writing in the C++ language and help writing our embedded programs.

Tom Angelotti – For helping us with our system enclosure and parts

1. Introduction

City streets, particularly in New England, have been a major concern for local roadwork crews, as well as everyday drivers. When drivers head out on the streets the last thing they should be worried about is damaging their vehicle from a destructive flaw in the road. Ideally, roads should be smooth and pothole-free. New technology may assist city roadwork resources to maintain city roadways to reduce pothole related vehicle damages.

There are few new technologically advanced methods to detect potholes in city roadways. For example, using lasers to scan road smoothness is one particularly new method that has been used in New York City. However, this method is costly and requires a special vehicle in which the device has been installed.

To assist city maintenance resources, our goal was to design and construct a device that will in fact improve the infrastructure of a city. Further, this new, innovative instrument will systematically utilize a GPS unit and an accelerometer to determine the condition of roadways. To accomplish this task our device must be able to determine the smoothness of a given surface. It will record data on the severity if bumps it may encounter in the road. Secondly, this data must be processed and our GPS unit must be able to determine the coordinates of our data collection system. This will allow us to produce a map of the smoothness of city roadways.

The remainder of this document details the design process undertaken by the team to design and build this prototypical device.

2. Background

Potholes have been known to be a major nuisance in roadside transportation, especially in New England streets. Due to the significant weather fluctuations we experience, these potholes evidently are more severe. Something must be done to prevent vehicles from being damaged by these potholes.

Potholes are created when moisture or rainwater seeps into the pavement and becomes absorbed by the mixture of rock, gravel, and sand that supports the road. This moisture then freezes and causes the pavement to expand. The frozen moisture eventually melts and evaporates in warmer temperatures. Generally, this will weaken the pavement and cause air pockets that will eventually cause pavement to break up. Traffic loosens the pavement even more, and it eventually crumbles, creating a pothole. A winter of heavy snow or rain and several freeze-thaw cycles can mean a big pothole season ahead. Roads with high traffic volumes typically have more potholes than others due to the sheer amount of use causing erosion.[1] Bridges and ramps, which receive heavy doses of snow removal chemicals all winter, are also considered to be more prone to potholes as well. Roads today are being built to reduce their moisture capacity, and researchers are working to develop a better, more durable pavement. Researchers also have improved the cold-patch asphalt so that the repaired pothole patches last longer.[2]

Fixing the holes is a concern in its own when considering the effectiveness of traditional methods. Workers say they have four options on how to deal with the pothole, or an area of asphalt that has stripped away, leaving an uneven road. Outside of tearing up the road and resurfacing it, which is rarely done, public works directors use a "cold patch" or "hot patch" to fix the holes. A cold patch is sand and rock mixed with asphalt and can be applied to the hole in three to five minutes. A hot patch, or hot asphalt, takes up to 20 minutes to apply, but if used correctly, can last until the road is replaced. However, while these methods may be fast and simple, the most effective method requires time and labor. This method is a process that involves using a jackhammer to create a rectangle around the hole, removing the existing asphalt, filling the hole with 21 to 24 inches of gravel base, and covering the base with 4 to 6 inches of binder asphalt and another 2 inches of resurfacing hot asphalt. Although this method is the most effective, the problem may be that there are too many potholes and not enough time to fix them all. In some cases, a city will reimburse the driver for damages done to their car from hitting a pothole, but will not reimburse for pothole damage unless it was negligent in filling the pothole in a timely

[1] http://www.lacity.org/boss/Resurfacing/potholes.htm
[2] http://www.virginiadot.org/infoservice/faq-potholes.asp

manner. Once notified of a pothole, the Public Works Department is responsible for repairing it in a reasonable period of time (usually 48 hours).[3]

Substandard road conditions are dangerous. Outdated and substandard road and bridge design, pavement conditions, and safety features are factors in 30% of all fatal highway accidents, according to the Federal Highway Administration. The FHWA also mentions, on average, more than 43,000 fatalities occur on the nation's roadways every year. Motor vehicle crashes cost U.S. citizens $230 billion per year, or $819 for each resident for medical costs; lost productivity; travel delay; and workplace, insurance and legal costs.

According to the Massachusetts Infrastructure Report Card of 2005(ASCE), 71% of Massachusetts' major roads are in poor or mediocre condition. Also, driving on roads in need of repair costs Massachusetts motorists $2.3 billion a year in extra vehicle repairs and operating costs --- $501 per motorist. FHWA ranks "poor" roads as those in need of immediate improvement. In other words, these roads are a safety hazard and require immediate reparations. "Mediocre" roads need improvement in the near future to preserve usability.[4]

The nation is failing to maintain even the current substandard conditions, a dangerous trend that is affecting road and highway safety and also the health of the economy. The ASCE Infrastructure report notes that while passenger and commercial travel on our highways has increased dramatically in the past 10 years, America has been seriously under-investing in needed road and bridge repairs. For these reasons, we have designed this pothole detection device which recognizes the condition of roads and provides GPS coordinates. This will allow workers to properly determine exactly where the potholes are that need to be repaired immediately. In turn, the device will save drivers the money, the anger, and the metaphorical pain and suffering that a pothole endows us with.

[3] http://www.cityoflewiston.org/publwrks/Pages/Street%20Pages/potholes.htm
[4] http://www.asce.org/reportcard/2005/index.cfm

3. Problem Statement and Project Goals

This section clearly defines the goal of our project work. From our goal statement, one will be able to understand the end-product for which we were striving. Our goal will then be broken-down into the smaller objectives that comprised the project, and then into the smaller tasks taken to fulfill these objectives. The schedule that was followed in order to complete all of the tasks, objectives, and the goal of our project will be presented and explained. Finally, we present our overall project requirements to further elucidate our device. With all of this information presented, one will be able to understand the vision of the group, the requirements of the group, and the steps taken in order to achieve our goal.

3.1. Goals and Objectives

The goal of our project work was to develop an automated data collection system that can be installed in any automotive vehicle to monitor road or highway pavement conditions. In order to reach this goal, we had to meet all of our objectives by our set deadlines. Meeting these deadlines will assure that we are where we need to be to successfully achieve our goal. Within the context of our overall goal, we developed the following objectives.

The first goal of our project was to research potholes, GPS, accelerometers, and hardware and software solutions. We began with analyzing the effects of poor road conditions on a vehicle and also how this could be measured. Once having an understanding of each, the project group began to consider viable design solutions and created two main objectives:

- How to design a working prototype for an automated data collection system that can monitor road conditions
- How to process data with Geographical Information System software to map surface roughness data from GPS coordinates on a user-viewable city map.

The next goal of our project was to implement our two objectives and successfully create a working device. The implementation of our design can be seen in the methods chapter of this report.

3.2. Tasks

This project initially began C term of 2005 and was scheduled to be completed by the end of the 2005-2006 school year. The project group set deadlines for each of our objectives, along with milestones throughout the project. This will allow us to keep on track and effectively manage our time as needed. As shown in our Gantt chart in **Figure 1**, we have designated specific dates to each individual module of our entire device.

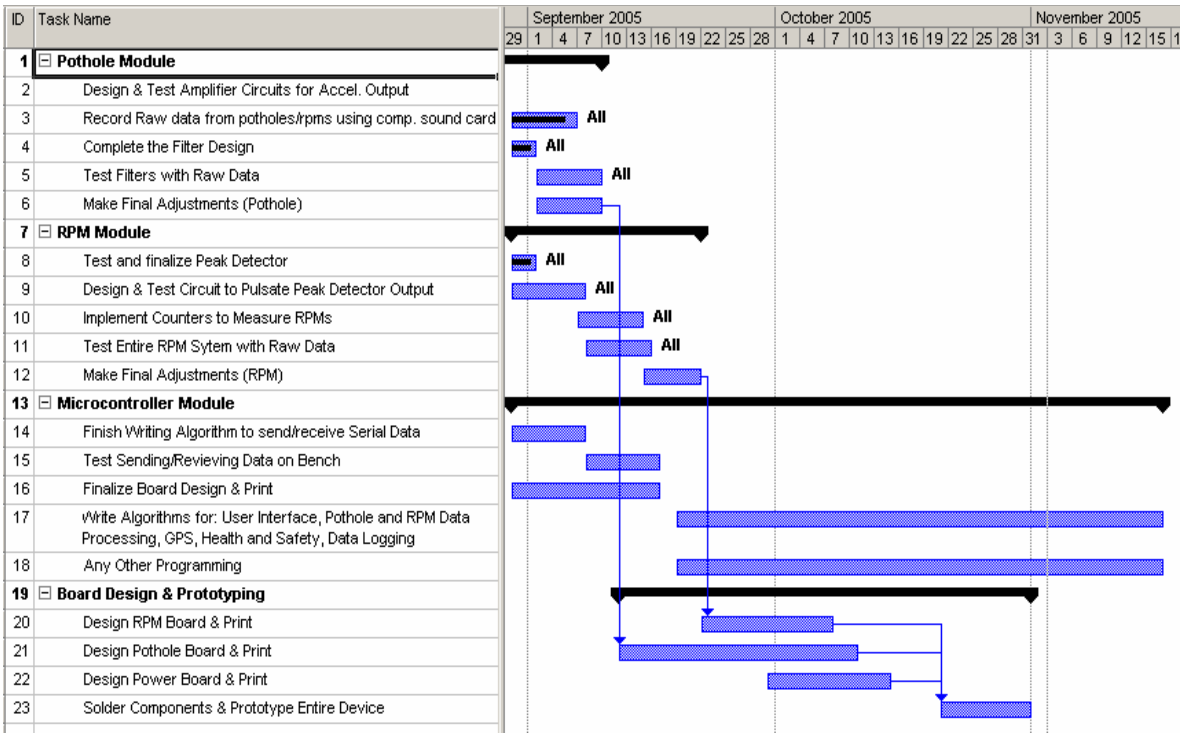

Figure 1: Gantt chart

We have generally met most of our milestones, but it may be of our best interest to add a week or two onto our Gantt chart if needed.

3.3. Project Requirements

As a group, we constructed our product requirements based on our initial research and design. These are the product requirements for the entire device.

Product Requirements

- Rugged Design
- Device must be able to display if the device is correctly working
- Device must log the location of road conditions
- The device must be able to operate for a weeks worth of data
- Get Power from the Cigarette Lighter
- Display if memory is almost full
- Display if device is writing to memory
- Have a standby button
- Display if system is in standby mode

- Must be portable
- Must be easy to mount

System Requirements

Microcontroller

- At least 4 Serial Ports
- At least 8 8-bit Analog to Digital Converters
- At least 18 Digital I/O ports
- External Flash memory

Accelerometer Module

- Single axis accelerometer
- Respond to frequencies below 20 Hz

Pothole Module

- Fifth order low-pass filter
- Operational Amplifier with a non-inverting gain
- Peak Detector Circuit

For the microcontroller, there should be at least four serial ports; one is designated for the LCD display, one for GPS, one for programming, and a final one for debugging.

A 1% accuracy of the actual analog value should be more than enough for signal processing and for measurements/logging. To represent values with this accuracy, 100 bits must be used; using 128 bits allows us to represent values with a little better accuracy than 1%. An 8-bit representation of values yields approximately a 0.78% accuracy of analog values.

$$Accuracy = \frac{1}{Bits} * 100 = \frac{1}{2^7} * 100 = \frac{1}{128} * 100 = 0.0078125 * 100 = 0.78\%$$

Equation 1: Microcontroller Accuracy

The microcontroller must handle eight ADC needed for monitoring road conditions as well as voltage and temperature readings from the health and safety (H & S) module. The additional ADC channels can be used for additional accelerometers if needed and other unforeseen uses. Also, external memory is the best option for data logging since it allows for quick removal and download ability to a personal computer.

The GPS device should have a pulse rate lower than approximately 130ms in order to provide a worst-case accuracy of 10 ft from a pothole if a vehicle is traveling at 50 mph.

The sensor in the accelerometer should respond for a frequency range between 0 Hz and 20 Hz. This is the typical frequency range of a pothole. Any frequencies over 20Hz will be ignored.

3.4. Summary

This chapter broadly described the major goal for this capstone design project as constructing a prototype for an automated data collection system that can monitor road conditions and store GPS related positioning data. The objectives and tasks required in achieving this goal were described along with some major difficulties present in the attempt to complete the main project goal.

4. Methods

This chapter outlines the methods used by our project team to accomplish our project goals. Briefly, we had to determine our system requirements when choosing a microcontroller to fit our needs. The team also had to research different means of measuring road conditions. As a result, the approach used to develop our system was as follows:

1. Reviewed different methods to collect potholes and road conditions
2. Reviewed System Requirements
3. Implemented the design from system requirements
4. Implemented packing requirements (Size of case, user interface)
5. Reviewed data processing requirements (How data from unit was going to be used to produce maps)

4.1. Reviewing Data Collection Methods

The first task of our design project was investigating the different methods for obtaining pothole and road condition information. After researching previous work, we found a couple options to determine our retrieval method, which included the use of an accelerometer and the use of a microphone to determine the vibrations of the car. We determined that the best way to distinguish movement of a vehicle would be the use of an accelerometer's voltage signal.

4.2. Reviewing System Requirements

The second task of our design project was to outline the system requirements that we were going to use in the design of the automated data collection system. To complete this task we specified and reviewed:

1. Reviewed project requirements (What exactly needed to be recorded)
2. Reviewed previous work done on the problem
3. Specified length of device operation
4. Specified amount of memory needed
5. Specified power requirements
6. Specified input and outputs on microcontroller
7. Specified number of serial ports on microcontroller
8. Specified number of ADC channels

Once we listed these system requirements we conducted value analysis on a few different microcontrollers on the market. Choosing the right microcontroller was essential in implementing our system design.

4.3. Implementing Design from System Requirements

Once we reviewed the system requirements of our system the next task was implementing our design. Implementing our design involved multiple steps and procedures. These procedures are outlined below:

1. Constructed analog circuitry to condition the accelerometer signal
2. Constructed circuit on a breadboard and tested in lab
3. Designed motherboard on printed circuit board layout software
4. Designed "Pothole Module" board and "Accelerometer Module" board on printed circuit board layout software
5. Interconnected system with GPS, LCD, buttons, etc under program control

As seen from the list above, the first procedure in implementing our design was constructing circuitry for the conditioning of the accelerometer signal. Printed circuit board layout software was then used to design multiple boards that brought functionality to the system once connected. The system was finally interfaced with other subsystems, for functionality, under software control.

4.4. Implementing Packaging Requirements

Once the multiple modules had been implemented, the next task was building its packaging. From our system requirements we identified the different considerations that had to be taken into account when packaging the device to one standalone system. These considerations included its size, compactness, durability, etc.

4.5. Reviewing Data Processing Requirements

The final task of our project was to develop a way of processing data and outputting it. The task of processing data was completed by using the software given to us by the manufacturer of our chosen microcontroller and creating algorithms that successfully recorded the data. The task of outputting the data was completed by using chosen GIS software and inputting the recorded data from the microcontroller to the Graphical Information System (GIS) software.

4.6. Summary

This chapter presented the methods used by our team to achieve our project goal. These methods included reviewing different requirements we were attempting to satisfy. These requirements included having functionality to determine potholes and road conditions, and other system requirements that were decided early on by our project team and project advisors. Another prerequisite was in data processing where we had to find methods of displaying the recorded data. This chapter also outlines the methods used for implementing our design.

5. System Design

In the following sections, the overall system design of the project will be presented. The overall design can be broken down into seven subsystems which include the accelerometer module, road conditioning module, GPS module, LCD module, the microcontroller module, health and safety module, and motherboard module. **Figure 2** shows the basic interaction between the subsystems and how the supply voltage is used.

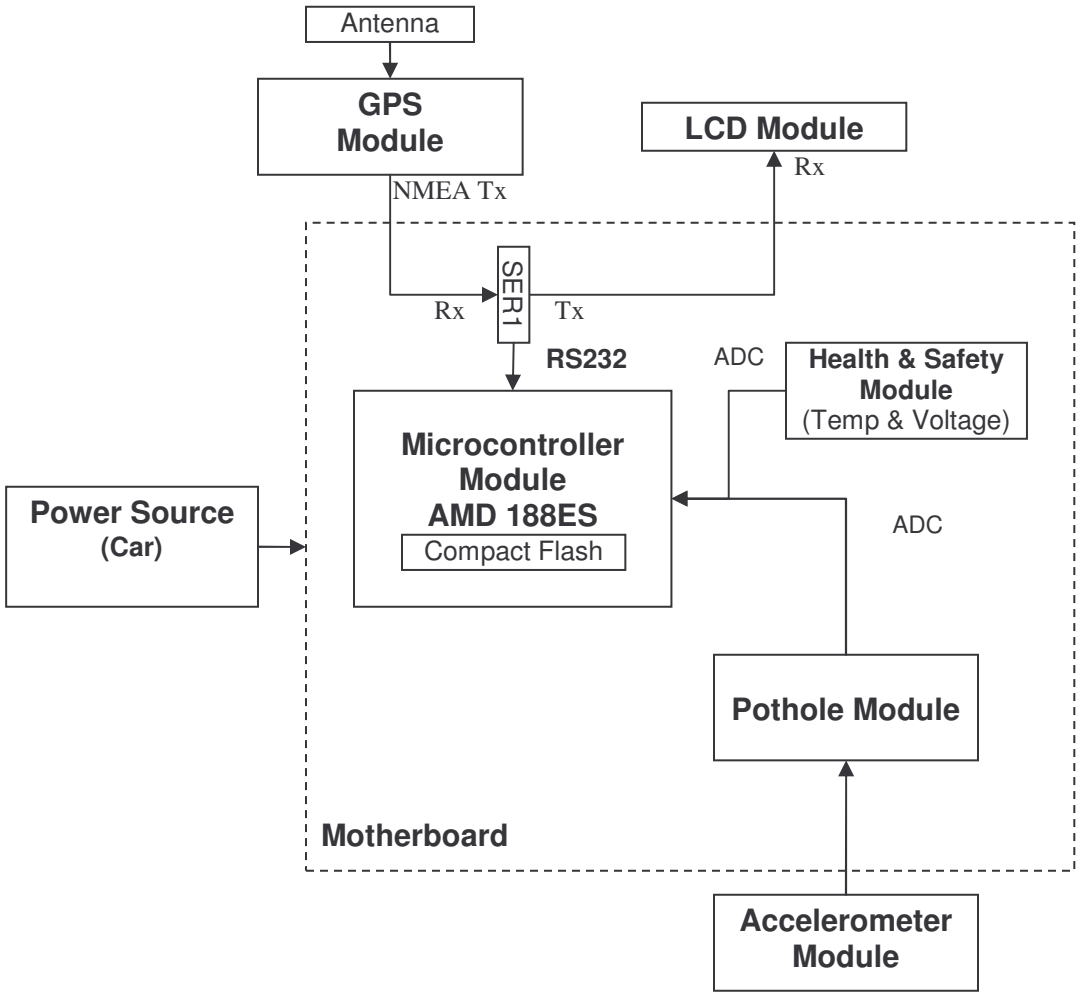

Figure 2: System Block Diagram

5.1. Accelerometer Module

The accelerometer module is the first module of our system design. This module is designed so that when a vehicle comes in contact with a pothole, the accelerometer will move, creating a voltage signal that is relative to the severity of the pothole (the more severe the pothole, the higher the voltage signal). The entire module is powered from two voltage regulators located on the Road Conditioning Module and a 5 V regulator on the module which powers the accelerometer. The signal from the accelerometer is conditioned on this module by the DC offset, the low-pass filter, and the voltage amplifier. First, due to the orientation of the accelerometer, the voltage signal that it sends out will rest on +2.5 V or -2.5 V. Using an operational amplifier and a voltage divider, consisting of a potentiometer, an adjustable DC offset is created to lower the signal back to 0 V. Once on 0V, the signal passes through a low-pass filter to remove the noise from the accelerometer. Finally, using another operation amplifier, the signal is amplified with a gain of 50. This is done because the accelerometer signal's output sensitivity is only 18 mV per g. The signal is then sent to the Road Conditioning Module. The complete module can be seen in **Figure 3.**

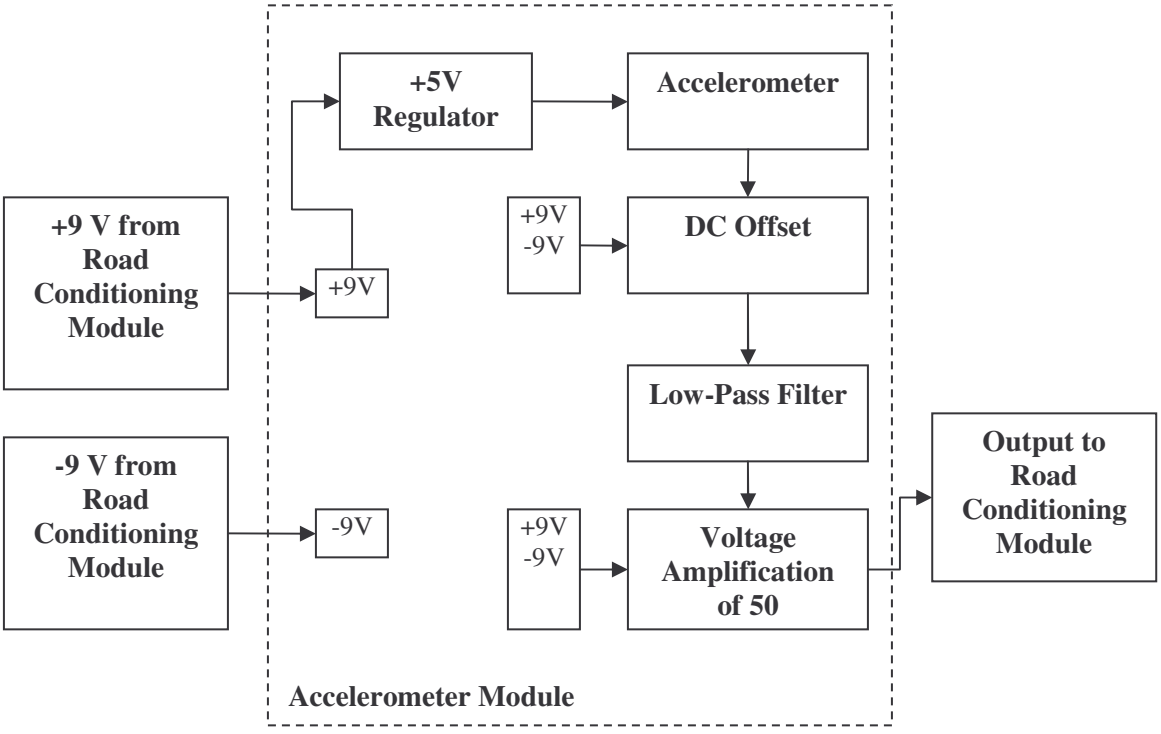

Figure 3: Accelerometer Module

5.2. Road Conditioning Module

The next module of the system design is the Road Conditioning Module. As seen in **Figure 4**, the signal from the accelerometer is received and then conditioned through the 5th order low-pass filter, voltage amplifier, and voltage peak detector. Also on this module are a dc-dc converter which converts the +12 V to ±9, and two voltage regulators which are used to power the different integrated circuits (ICs) of this module and the accelerometer module. The first circuit that is used to condition the signal is the 5th order low-pass filter. This is used to eliminate any frequency over twenty hertz, which was chosen to eliminate the normal vibrations of an automobile. An operational amplifier with a gain of ten is then used to compensate for the voltage lost from the low-pass filter IC. Once amplified, the signal is sent to the voltage peak detector. The peak-detector is made up of an operational amplifier which acts as a half-wave rectifier and a RC circuit which captures the peak of the signal. The information from this part of the module is then sent to the A/D converter of the microcontroller which can be used to determine if a pothole has been hit.

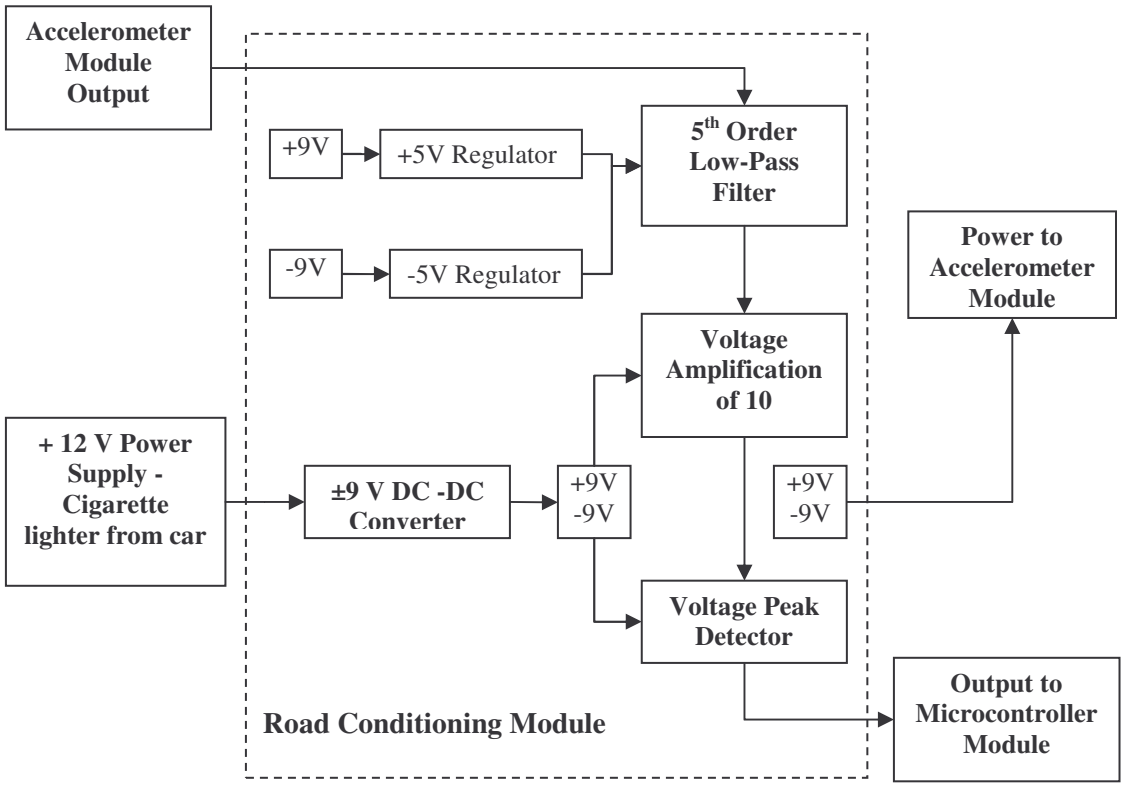

Figure 4: Road Conditioning Module

5.3. GPS Module

The GPS module of our system, seen in **Figure 5**, consists primarily of a GPS receiver that connects, through serial communication, to the microcontroller and is powered by a +5 V voltage regulator which is located on an external PCB board mounted on the case. The GPS receiver we used in our design was a Garmin[5] GPS receiver, the GPS25-LVS[6], seen in **Figure 6**. This module contains an external antenna that can be mounted onto a glass surface inside the vehicle. The antenna is connected to the GPS receiver through coaxial cable. Once the antenna is placed correctly, the GPS receiver acquires satellites and outputs National Marine Electronics Association (NMEA) strings. These NMEA strings contain useful information such as date, time, speed, location, and other data. The GPS receiver can be programmed to output multiple NMEA strings, each containing different pieces of information. Also, note the connections of the GPS receiver to the microcontroller: the NMEA serial transmit port (Tx) on the GPS receiver is connected to the serial receive port (Rx) of the microcontroller. The signal ground of the GPS receiver is connected to the system ground. The GPS receiver also has a serial receive (Rx) port that can be used to program the types of NMEA strings output. These pin connections can be seen in **Figure 7**.

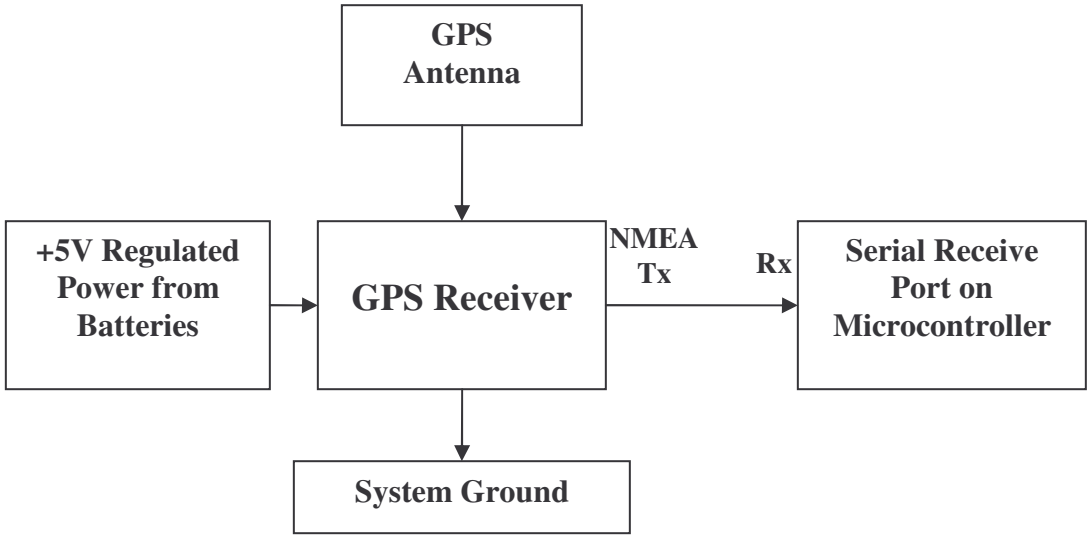

Figure 5: GPS Module Block Diagram

[5] http://www.garmin.com
[6] http://www.garmin.com/products/gps25/

The actual GPS receiver used is shown in **Figure 6**. A PIN-OUT description of the GPS receiver is shown in **Figure 7**. The pin out descriptions shows the connections that were made when connecting power (+5V) to the unit (Pin 10/11), ground (Pin 8), as well as the NMEA Output (Pin 12). The serial NMEA transmit (NMEA Tx) or NMEA Output was connected to the Rx pin on the microcontroller SER1 serial port.

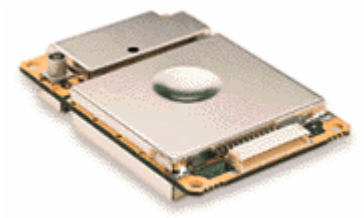

Figure 6: Garmin GPS25-LVS[7]

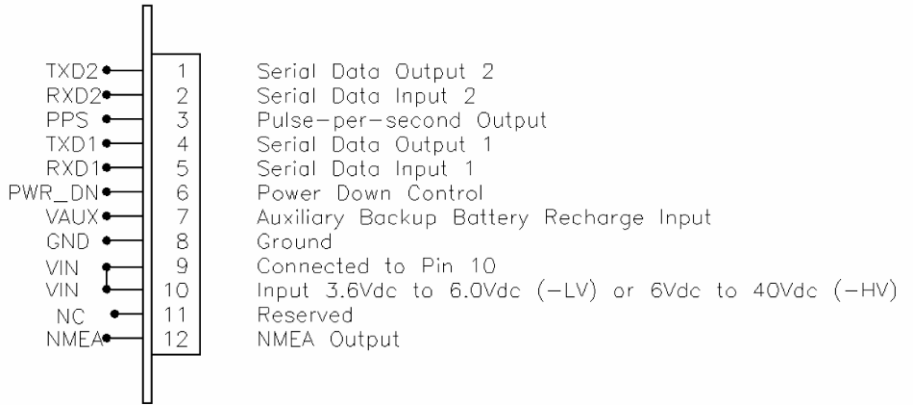

Figure 7: Garmin GPS25-LVS Pin Out Description[8]

5.4. LCD Module

The LCD module of our system, seen in **Figure 8**, consists primarily of a LCD unit that connects, via a RS-232 serial port, to the microcontroller and is powered by a +5 V voltage regulator which is located on an external PCB board mounted on the case. The actual LCD used in our design was a Crystalfontz[9] serial LCD module, model number CFA-632[10], and is displayed in **Figure 9** along with the pin out descriptions in **Figure 10**. This unit is stand alone and

[7] http://www.garmin.com/products/gps25/
[8] http://www.garmin.com/manuals/GPS25LPSeries_TechnicalSpecification.pdf
[9] http://www.crystalfontz.com
[10] http://www.crystalfontz.com/products/632/index.html

displays American Standard Code for Information Interchange (ASCII) characters once ASCII characters are serially transmitted to the LCD display. This device allows our system to display functional information about the device, to the user, while the rest of our system is operating in real time. Also, note the connections of the LCD unit to the microcontroller: the serial receive port (Rx) on the LCD unit is connected to the serial transmit port (Tx) of the microcontroller. The signal ground of the LCD unit is connected to the system ground.

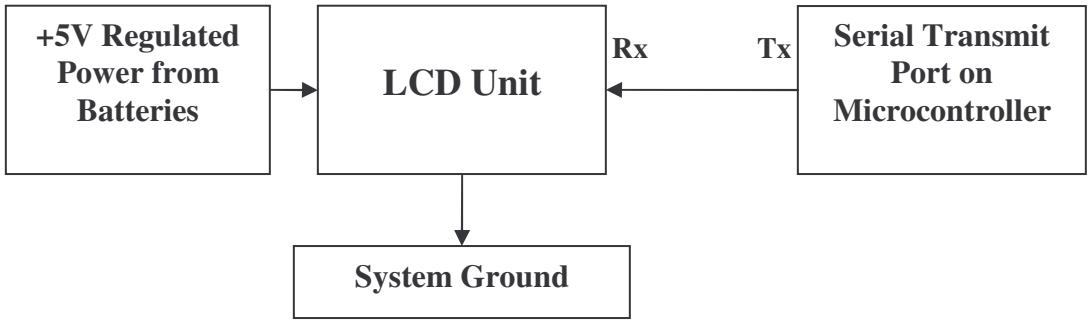

Figure 8: LCD Module Block Diagram

A picture of the LCD unit is shown in **Figure 9**. A PIN-OUT description of the LCD unit is shown in **Figure 10**. The pin out descriptions shows the connections that were made when connecting power to the unit (+5V LCD), ground, as well as the serial receive pin (DATA_IN). The serial receive (Rx) or DATA_IN was connected to the Tx pin on the microcontroller SER1 serial port.

Figure 9: LCD Screen[11]

[11] http://www.crystalfontz.com/products/632/index.html

PIN	PIN NAME	CRYSTALFONTZ DISPLAY FUNCTION
1	GROUND	Ground (backlight and controller)
2	+5V(LCD)	Controller and LCD power (+5 volts only)
3	+5V(LED)	LED Backlight power (+5 volts only)
4	DATA_IN	SPI or RS-232 data in (input)
5	$\overline{\text{SPI\_CS}}$	SPI Chip Select (active low input)
6	SPI_CLK	SPI Clock (input)
7	SPI_BUSY	SPI Busy (output)

Figure 10: LCD Pin out Description[12]

5.5. Health and Safety Module

The health and safety module's function is to provide information about the voltage levels of the power source and ambient temperature. A block diagram of this system is shown in **Figure 11**. We determined that measuring these conditions is important for the overall operation of our device. Whenever these conditions fail, we configured our microcontroller to act accordingly under software control.

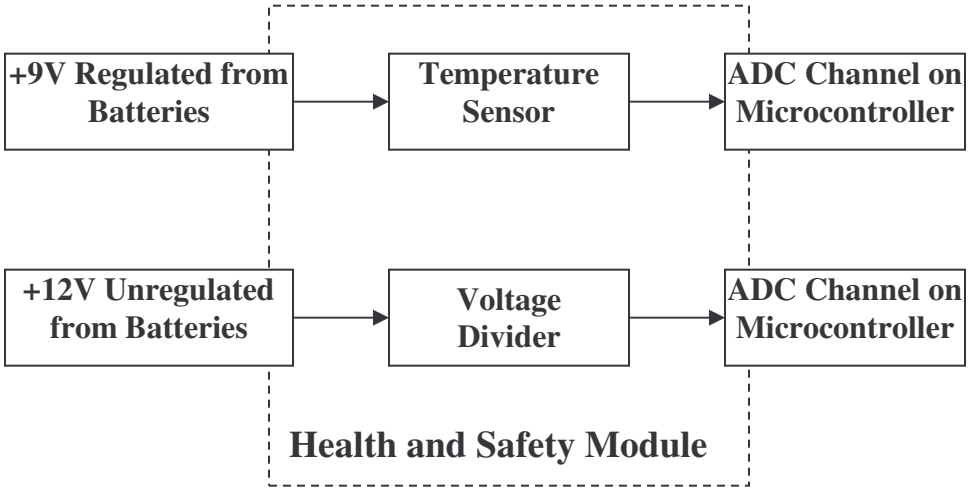

Figure 11: Health and Safety Module Block Diagram

[12] http://www.crystalfontz.com/products/632/data_sheets/CFA-632_v2.0.pdf

As is seen in **Figure 11**, the temperature sensor is connected to a +9V regulated power supply from the motherboard. The temperature sensor used was an analog devices AD590KF two terminal IC temperature transducer that outputs a current proportional to temperature. The voltage divider (Voltage meter) is connected directly to the power source, to monitor the voltage. Both the temperature sensor and the voltage divider connect to their respective ADC channels on the microcontroller.

5.6. Microcontroller Module

The microcontroller module is a microcontroller we purchased after performing a value analysis on different microcontrollers that could function in our system. The value analysis we conducted in choosing the TERN FlashCore-B is discussed in **Appendix: D**. The functional description of the TERN FlashCore-B is explained in detail in the FlashCore-B (FB)™ Technical Manual found on the Tern Inc. website[13]. A functional block diagram of the FlashCore-B microcontroller can be found in **Figure 13**. A brief description of the FlashCore-B from the technical manual is quoted below[14]:

> "The ***FB*** is a complete standalone C/C++ programmable embedded controller including a 188 CPU,512KB ACTF Flash, 128KB or 512KB SRAM, 512-byte EEPROM, 2 channel RS-232 driver, 5Vregulator, with optional real-time clock, battery, 8 channel 16-bit ADC, and/or 4 channel 12-bit DAC."

A picture of the FlashCore-B can be seen in **Figure 12**.

[13] http://www.tern.com
[14] FlashCore-B(FB)™ Technical Manual. Tern Inc. http://www.tern.com

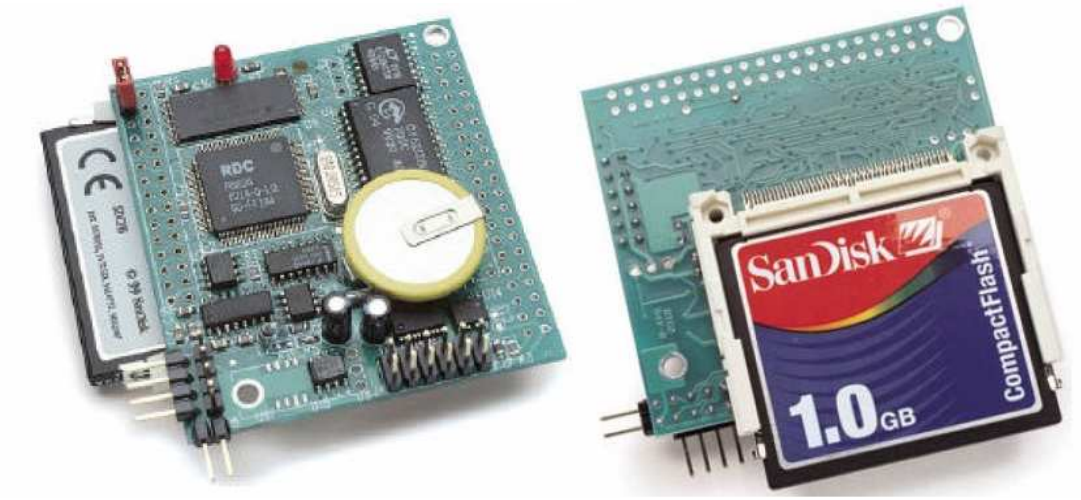

Figure 12: TERN FlashCore-B[15]

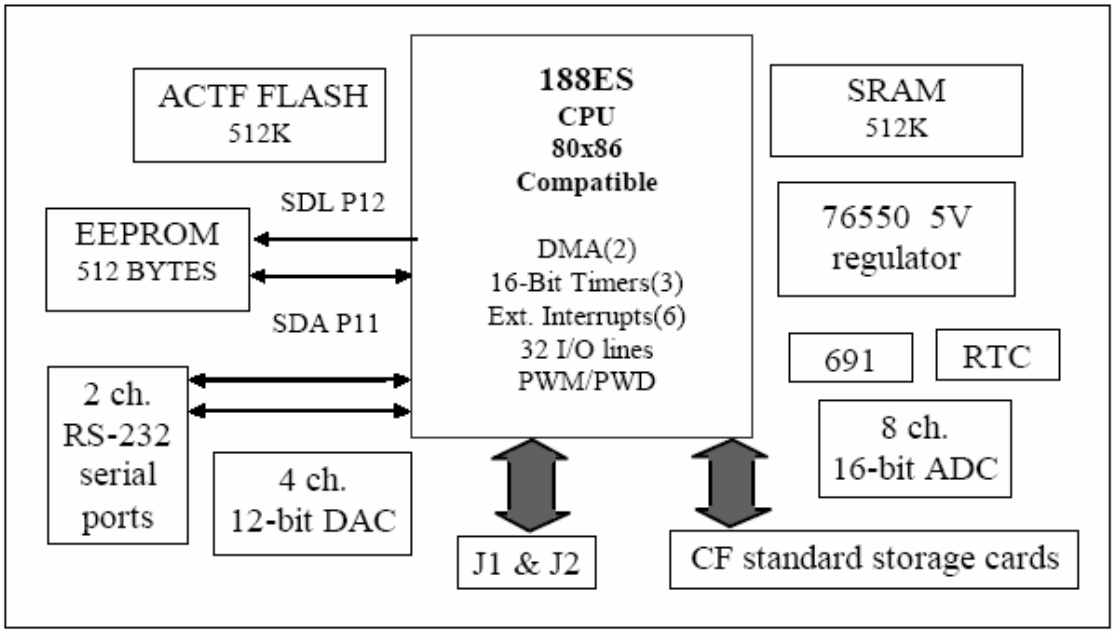

Figure 13: Tern FlashCore-B Functional Block Diagram[16]

We had to interconnect our microcontroller with the other modules explained in this section (Pothole, Health and Safety, GPS, LCD Module's). A more detailed block diagram showing the connections between the microcontroller module and the other modules described in this section is shown in **Figure 14**. The block diagram from **Figure 14** shows how the elements

[15] FlashCore-B(FB)™ Technical Manual. Tern Inc. http://www.tern.com
[16] FlashCore-B(FB)™ Technical Manual. Tern Inc. http://www.tern.com

of the microcontroller module connect to the other modules. As you can see the GPS and LCD module connect to one of the serial ports on the microcontroller. The health and safety module connects to two of the ADC channels on the microcontroller to provide ambient temperature and voltage readings. The conditioned signal from the Pothole module to measure road conditions is connected to one of the ADC channels on the microcontroller. Power to the microcontroller module is provided by an external +9V regulator that is placed on the motherboard. The motherboard is explained in the next subsection.

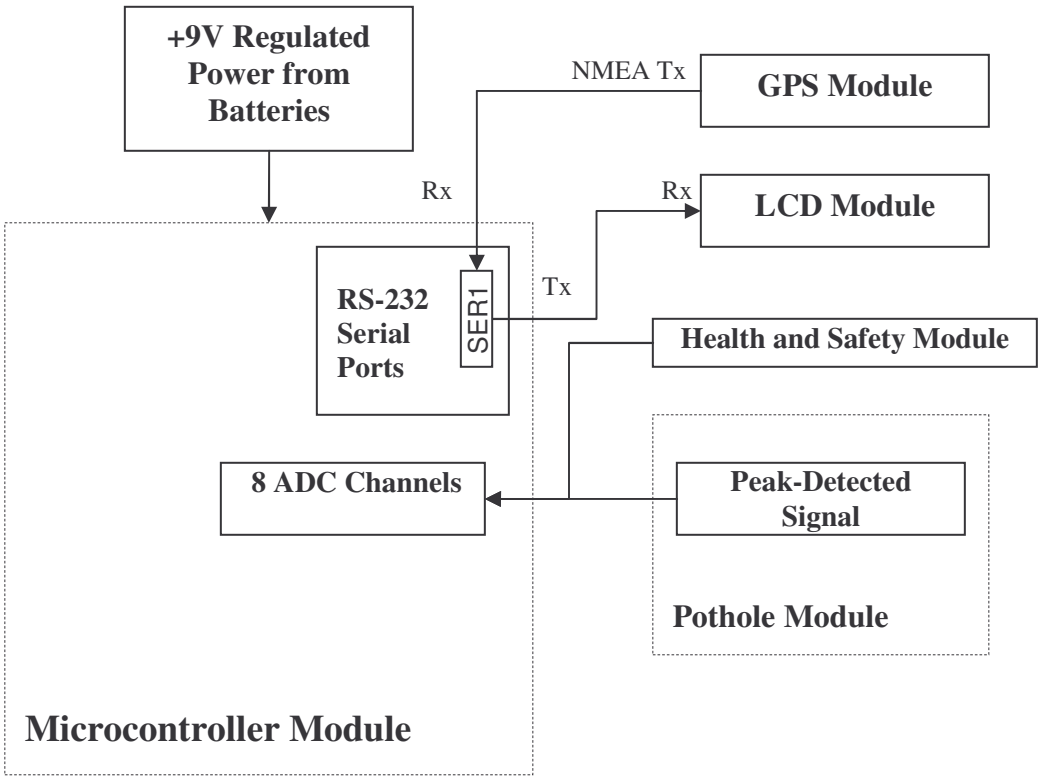

Figure 14: Microcontroller Module Connections

5.7 Motherboard

The motherboard connects all the different modules together. In addition to interconnecting the microcontroller with the GPS, LCD, Health and safety, and Pothole Module, the motherboard provides power for all the other modules. Please refer to **Figure 2** to see how the motherboard brings the system together. Everything within the dotted square represents the interconnections made within the motherboard. The microcontroller and the Pothole module are

the only modules which actually mount, via headers, on the motherboard, shown in **Figure 15**. The GPS module and the LCD module connect to the motherboard using a male header which interfaces the microcontroller SER1 serial port.

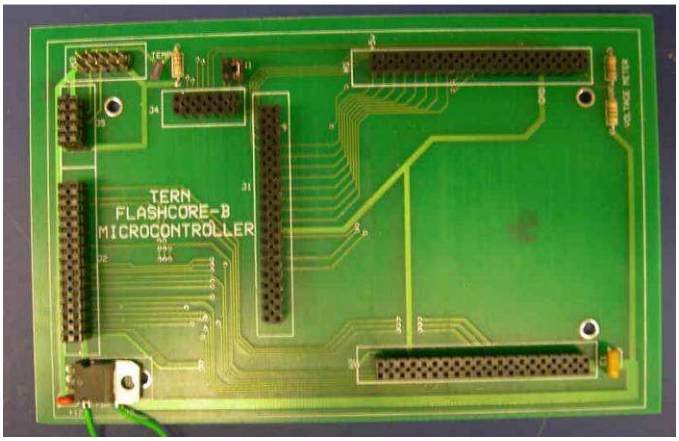

Figure 15: Motherboard Module

5.8 Summary

This chapter presented various structural block diagrams showing the operation of different modules. The different modules in our system design include: the GPS, LCD, Health and Safety, Pothole, Accelerometer, Microcontroller, and Motherboard module. A system block diagram that illustrates how all the different modules connect to bring functionality to our entire design was also presented in this chapter.

6. Results

Following the problem statement, methods, and system design, the group initialized a plan to develop a working system which fulfilled the project goals. The following section presents the results of the project.

6.1. Hardware Results

This section describes the circuitry design, simulation, and testing done to complete a device to fulfill our project statement. All schematics described in this section can be located in Appendix E.

6.1.1. Accelerometer Module

One goal of our project was to design a system in which a voltage signal would be created when a pothole has been struck. We decided that the easiest solution to this problem would be if an accelerometer would be used. An accelerometer is an integrated circuit where, depending on the orientation to the ground and the motion of it, a small voltage will be outputted; the more the accelerometer moves the higher the voltage signal will be. After researching and testing several accelerometers, we decided on the Analog Devices ADXL193. This accelerometer was chosen because it offers a low power, single axis solution. Once the accelerometer was chosen, field testing was done to see what kind of voltage signal the accelerometer would output inside a vehicle. The signal can be seen in **Figure 16**.

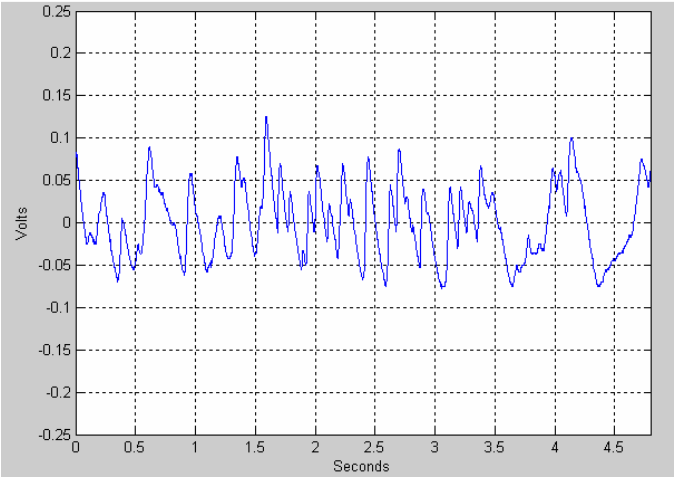

Figure 16: Accelerometer Signal

With the use of MATLAB software, a discrete Fourier transform (DFT) of the signal was taken and can be seen in **Figure 17.** By analyzing the DFT, it is visible that any frequency over 20 Hertz will not be needed.

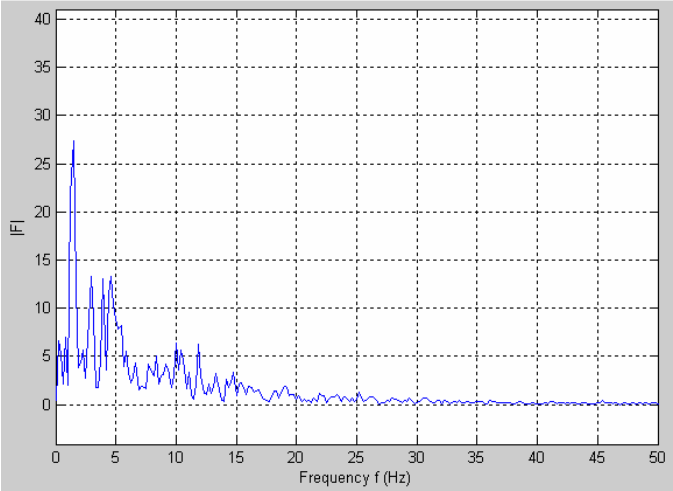

Figure 17: DFT of Accelerometer Voltage Signal

Once the initial testing of the accelerometer was complete, the design of the accelerometer module began. The first task of the module was to create a DC offset. This was created because the accelerometer will normally operate at ±2.5 V, depending on its orientation. Using a TL081CP operational amplifier (op-amp) and a voltage divider with the precision of 100K potentiometer, the DC offset was created. The accelerometer voltage signal is inputted into the positive terminal of the op-amp and +5V going through the voltage divider is inputted into the negative terminal of the op-amp. Using the potentiometer, the amount of voltage inputted into the negative terminal is controlled, making it manageable to set the accelerometer signal at 0 V. Also, using two 100K resistors, the op-amp has a gain of one.

After the DC offset was complete, the next task was to create a simple low-pass filter. Knowing that 20 Hz was an acceptable cutoff frequency, a RC circuit was designed. With the use of Equation 2, we were able to calculate the needed values (R =7k and C=1μF).

$$f = \frac{1}{2\pi RC} = \frac{1}{2\pi(7*10^{3})(10^{-6})} = \frac{1}{.04398} = 22.7Hz$$

Equation 2: Cutoff Frequency

Following the low-pass filter, a non-inverting op-amp configuration is used to increase the amplitude of the voltage signal. With Equation 3, the output voltage can be changed into a

measurable voltage size to transmit to our pothole module. The needed amplification is due to the low output voltage of the accelerometer and the length of wire connecting each module. The accelerometer only outputs 18 mV per g and by using our design, the voltage obtained is now around 1 V per g.

$$v_O = v_I(1+\frac{R_2}{R_1}) = .0018(1+\frac{100k}{2k}) = .0918\frac{V}{g}$$

Equation 3: Non-inverting Gain

Once the design was complete, the next step was to create the schematic in the Electronics Workbench software. With this software, we were able to simulate the results and determine if our design would work to suit our goal. The signal used to simulate the accelerometer signal was a 10 Hz, one volt sine wave. The results of the simulation can be seen in **Figure 18**. With the phase shift and amplification in the output signal seen in channel two, we determined that the signal was working correctly.

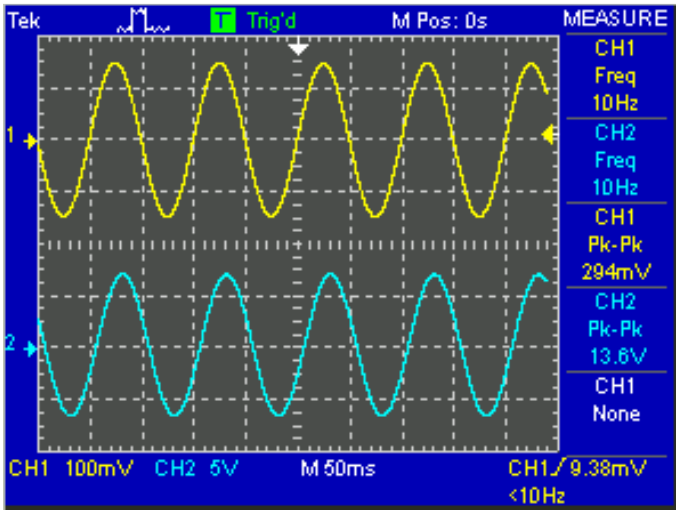

Figure 18: Simulated Results of the Accelerometer Module

After having successful simulations, the circuit was constructed on a bread board, tested, and was then put onto PCB board. The completed module can be seen in **Figure 19**.

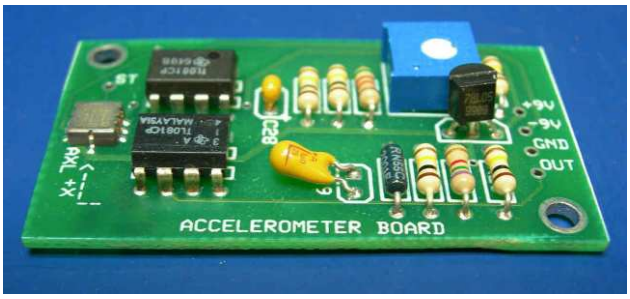

Figure 19: Accelerometer Module

The module was then coated in an epoxy to protect it from environmental hazards such as water and sand. Using this coated module, field testing was able to be done. Final test results are located in **Figure 20** along with the DFT in **Figure 21**.

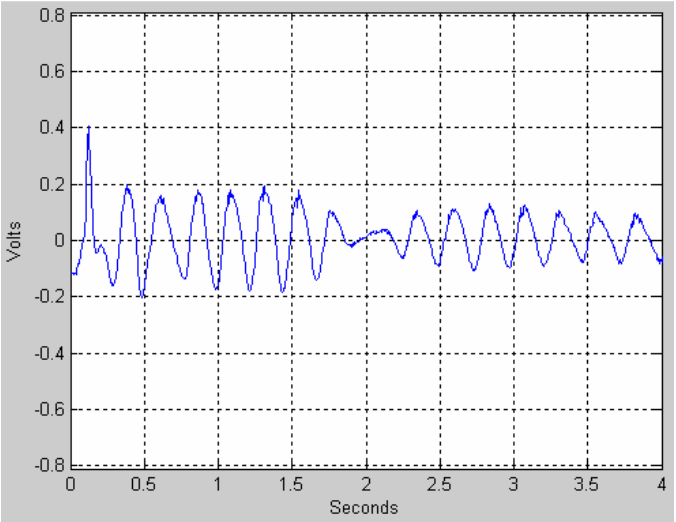

Figure 20: Accelerometer Module Test Results

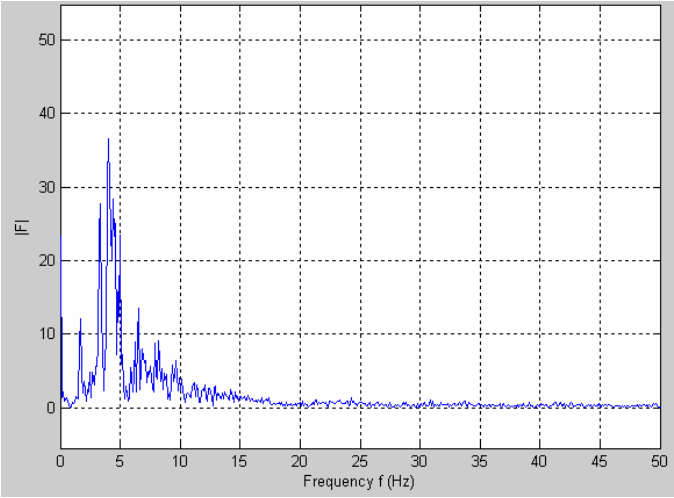

Figure 21: DFT of Accelerometer Module Test Results

6.1.2. Pothole Module

The next step of our design was to create a circuit which would take our accelerometer module output and alter it into something our microcontroller could process. The microcontroller can process zero to five volts so a half-wave rectifier would be needed to eliminate any negative voltage. Also, with the use of a peak detector circuit, the accelerometer peaks would be more attainable than just the half-wave signal. This is useful in the post processing to determine how severe the road condition is.

The first step of this circuit design was to eliminate the noise caused by the cable that is needed to transmit the signal from the accelerometer module to this module and also to eliminate any frequency over 20 Hz. This task was completed by using a 5^{th} Order Butterworth low-pass filter. The next step was to use an op-amp with a gain of 10 to compensate for the voltage loss through the filter. Once completed, the half-wave rectifier circuit was constructed followed by the peak detector. This design was completed in the Electronics Workbench software and then simulated. **Figure 22** shows the simulation with the use of a 10 Hz, 100 mV sine wave input signal.

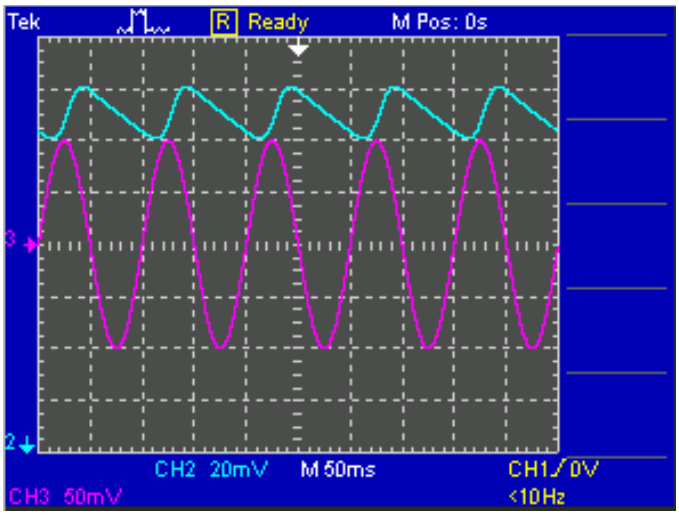

Figure 22: Simulated Pothole Module

To successfully construct the entire circuit on the bread, each piece was individually built and tested on a bread board. This first step of the construction was the 5th Order Butterworth low-pass filter. A LTC1062 IC was used to accomplish this task and is shown in **Figure 23**.

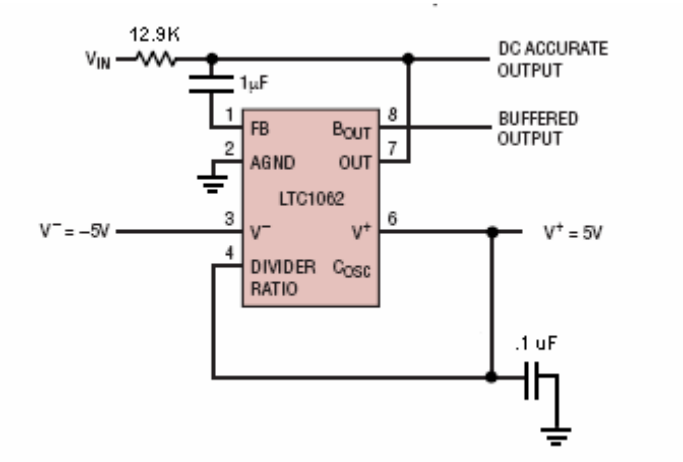

Figure 23: 5th Order Butterworth Low-Pass Filter

The equation for the cutoff frequency of the IC, from **Figure 23**, is different from a normal RC filter; this can be seen from **Equation 4**. Using 13k and 1µF, the cutoff frequency of 20 Hz is attainable.

$$\frac{1}{2\pi RC} = \frac{f_c}{1.62}$$

Equation 4: IC Cutoff Frequency

After several tests using different frequencies, a correlation between the magnitude of the signal and the frequency was established. We can see that with a loss of around -4 dB around 20 Hz our filter is working correctly.

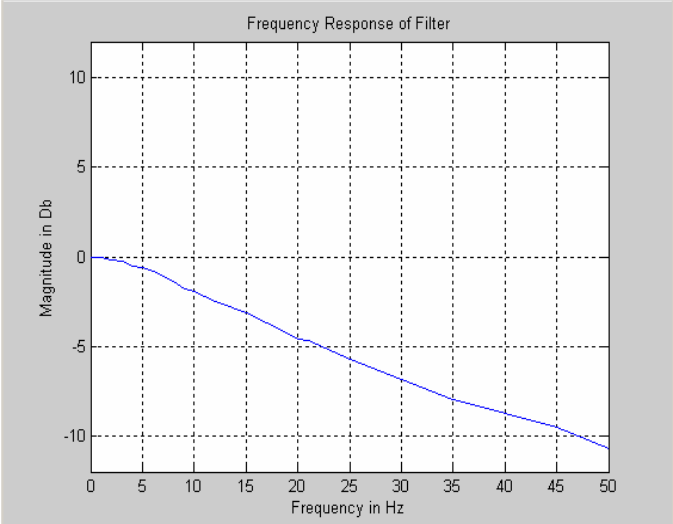

Figure 24: Frequency Response of Filter

Upon completion of the filter, a non-inverting op-amp with a variable gain was implemented. When using a resistor of 10K and a potentiometer of 100k, as seen in Equation 3, the gain will be centered at 10. The use of the potentiometer is for us to have the ability to control the output of the signal. If we are receiving unreadable voltages as an output, increasing the resistance of the potentiometer will increase the voltage amplitude of the output.

After the non-inverting op-amp, the voltage signal undergoes a half-wave rectifier circuit. This circuit consists of a TL081CP op-amp along with two 1N914 diodes, and allows us to eliminate any negative voltage the accelerometer may output. Once rectified the signal undergoes the peak detector. Using another diode, this circuit captures the high peak of the accelerometer and then decays until the next peak is hit. Using this circuit, we are able to distinguish movement of the accelerometer. The final component of this circuit is a five volt zener diode. This zener diode is used to clamp the voltage at five volts so the microcontroller will not be ruined.

After successful testing on the bread board, a PCB board was constructed and assembled. This board can be seen in **Figure 25**.

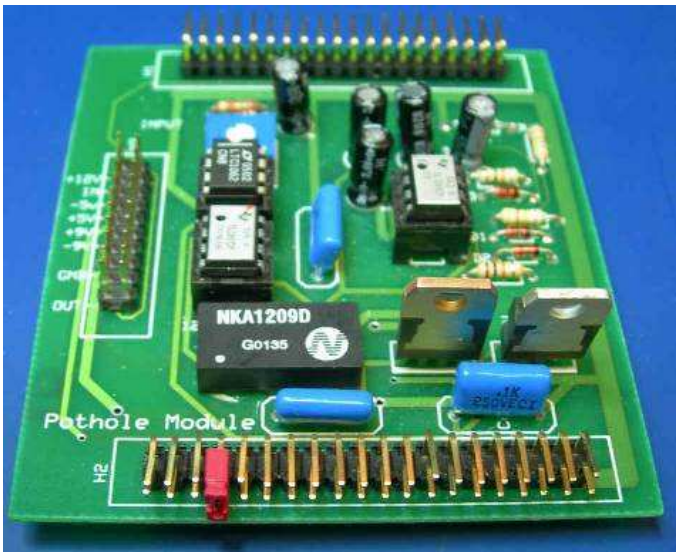

Figure 25: Pothole Module

Using both the accelerometer PCB module and the pothole PCB module, final testing was completed. The final signal can be seen in **Figure 26**.

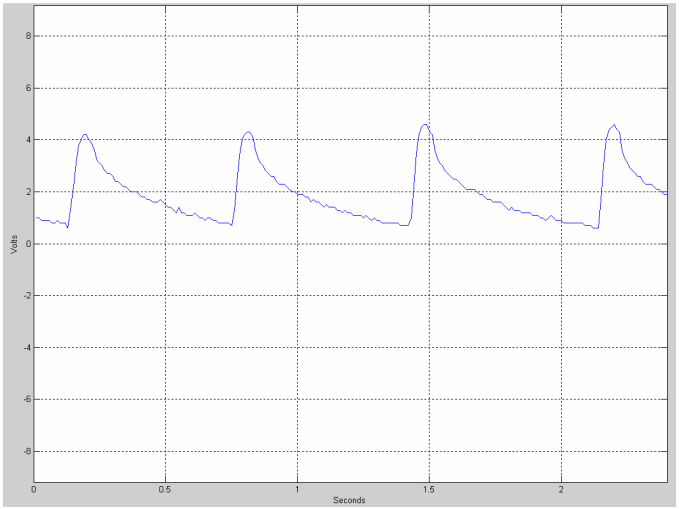

Figure 26: Final Output Signal

6.1.3. Motherboard Module

The final step of the hardware design was to incorporate each module and the microcontroller together. This was accomplished by the motherboard module. On this module are headers for the microcontroller, the pothole module, the GPS, and the LCD display. The GPS and the LCD display connect to the microcontroller through the RS232 serial input/outputs. Also on this module is the health and safety module. The basic function of the health and safety module is

to provide temperature readings and measure the voltage output from the device's power source. Our real-time embedded program on the microcontroller then acts accordingly depending on the voltage and temperature readings it gets (i.e. closes file on compact flash). The program explained in the **Software Results** section. The motherboard module with the microcontroller and pothole module attached can be seen in **Figure 27**.

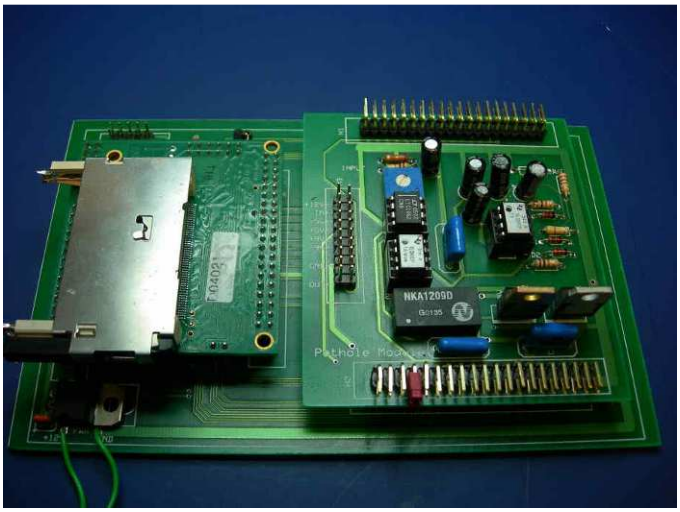

Figure 27: Motherboard Module

6.2. Software Results

This section presents the results, from a software standpoint, when the microcontroller was programmed to bring functionality to the entire system. In addition, this section describes the functions of our program by means of a software flow chart. A complete version of our C++ program source code is included in **Appendix C: C++ Embedded Program Source Code**.

The software flow chart of our embedded program is shown in **Figure 28**. The basic operation of our code, from looking at this software flow chart, can be followed from the initialization of variables down to the "Main Loop". Our functions check to see which GPS string type (From the GPS receiver) was received from the SER1 serial port. The program executes different procedures depending on the GPS String.

If the GPS string is type "GPGGA" the program first parses for time and stores the most recent time. It also parses for the number of satellites "in view" from the "GPGGA" string.

If the GPS string is type "GPRMC" the program stores it as the most recent GPS coordinates on onboard memory.

If the GPS string is type “GPVTG” the program parses and stores the most recent velocity.

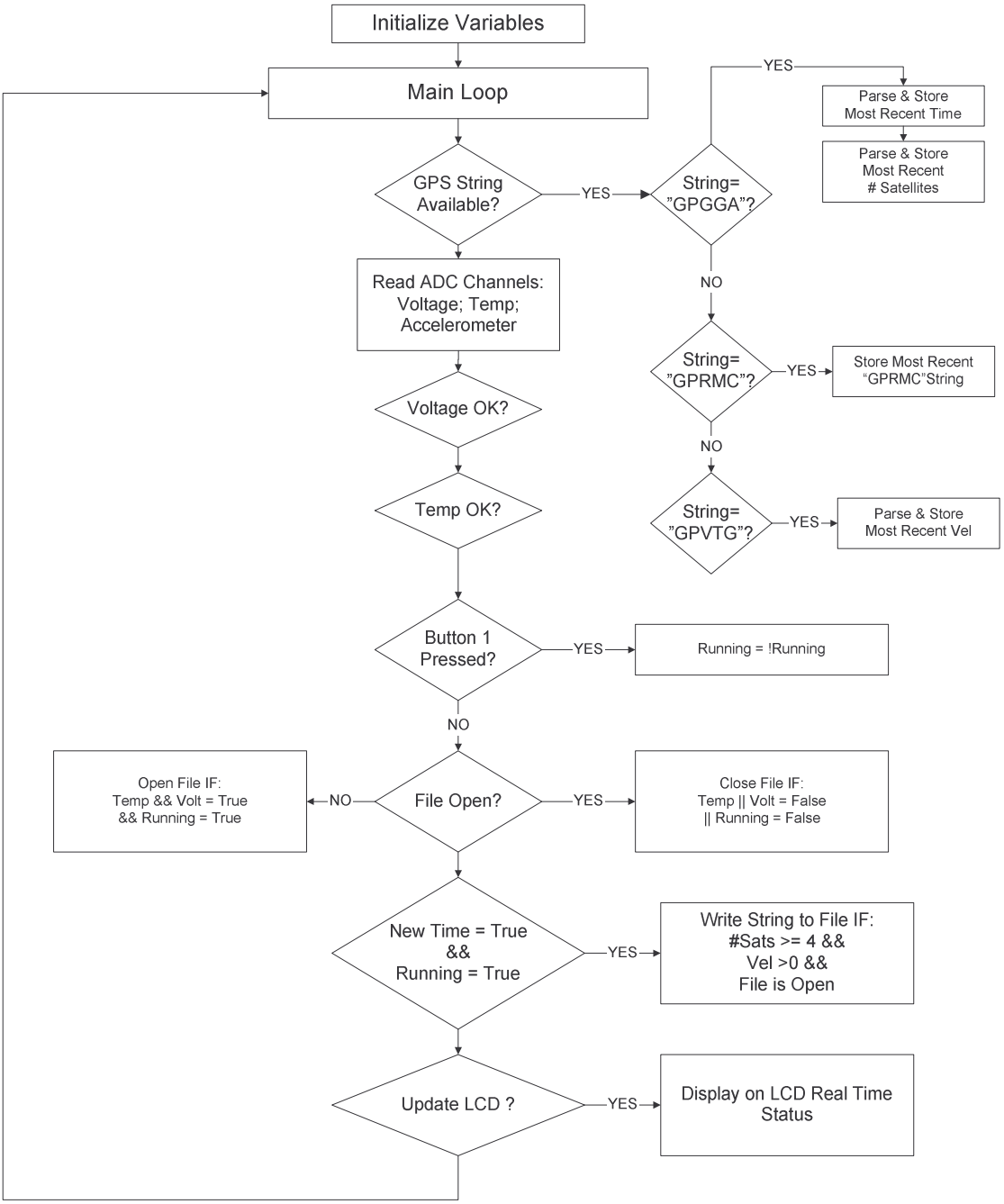

Figure 28: Road Condition Recording Software Flow Chart

Once the program processes the current GPS string, it reads the ADC channels to read the voltage (From Voltage Meter), the temperature (From Temperature Sensor), and the intensity of the accelerometer (From Pothole Module). It sets a Boolean variable to represent if the voltage

and temperature readings are within a correct range. If the temperature and voltage readings are outside the given range, either the temperature or voltage "statistic" is set to false signifying that one of them is invalid for device operation.

Then the program goes on to check to see if a button was pressed. Again it also sets a Boolean function to represent if the button was pressed (Running is set to !Running if pressed).

The program then goes on to check to see if a file accessing the Compact Flash card is open. If the file is not open, then the file is opened only if the voltage and temperature statistics are true and if the button was not pressed (Running=True). If a file is currently open, the file is only closed if the voltage or temperature statistics are false and if the button was pressed (Running=False).

The program then checks if the time has changed (NewTime=True); this depends on strings from GPS, and if the button was not pressed (Running=True). If these conditions are true then the program enters the procedure that writes to compact flash.

The procedure that writes to compact flash first converts the binary value read from the ADC channel, which is connected to the Pothole Module output, to its decimal equivalent. Then the program writes to compact flash only if: the number of satellites is greater than 3, velocity is greater than 0 Knots, and a file is currently open.

Next the program updates the LCD screen depending real time data like: voltage, temperature statistics, the value of the accelerometer [peak detected] signal, and the number of satellites "in view".

The actual data recorded onto the Compact Flash card resembles the text in the rows of **Table 1**:

$GPRMC,063403,V,4216.5799,N,07148.4762,W,000.0,000.0,300306 Axl=0.123456V #SATS=04
$GPRMC,063404,V,4216.5799,N,07148.4762,W,000.0,000.0,300306 Axl=0.123456V #SATS=04
$GPRMC,063405,V,4216.5799,N,07148.4762,W,000.0,000.0,300306 Axl=0.123456V #SATS=04
$GPRMC,063406,V,4216.5799,N,07148.4762,W,000.0,000.0,300306 Axl=0.123456V #SATS=04

Table 1: Compact Flash Stored Data

The data is stored as a text file on compact flash and then GIS software can be used to interpret the data graphically on city map that has road layers. A description of the GIS software used to process this data and source code is included in **Appendix B** of this document.

6.3. Test Results

Upon completion of the hardware and software, in car testing with the finished device was conducted. The system enclosure, containing the motherboard module, pothole module, GPS, and LCD display was placed inside the vehicle in the passenger seat. The system was powered by a cable connecting to the cigarette lighter of the vehicle. The system enclosure is seen in **Figure 29**.

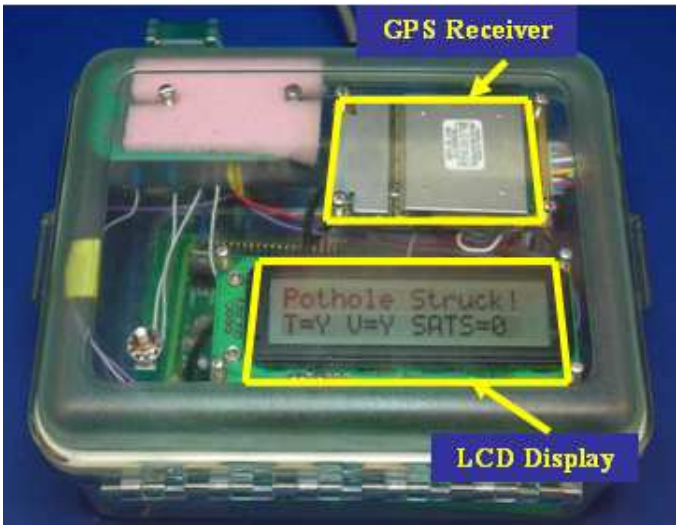

Figure 29: System Enclosure

A cable from the system enclosure was ran to the epoxy coated accelerometer module which was placed in the right wheel well of the car. The accelerometer module was held onto the wheel well by industrial strength Velcro. **Figure 30** shows the accelerometer placement. Once the setup was complete the car was driven around the Worcester area and data points were collected.

Figure 30: Accelerometer Placement

After collecting data points, the device was taken back to the lab so the post-processing could be completed. The data is taken off of the flash card which is plugged into the microcontroller. **Figure 31** shows a sample of data results found on the flash card.

```
$GPRMC,200912,V,4216.5333,N,07148.4155,W,000.0,000.0,240306 Axl=2.992142V #SATS=04
$GPRMC,201019,V,4216.5333,N,07148.4155,W,000.0,000.0,240306 Axl=2.807660V #SATS=04
$GPRMC,201021,V,4216.5333,N,07148.4155,W,000.0,000.0,240306 Axl=2.772717V #SATS=04
$GPRMC,201023,V,4216.5333,N,07148.4155,W,000.0,000.0,240306 Axl=2.795377V #SATS=04
$GPRMC,201025,V,4216.5333,N,07148.4155,W,000.0,000.0,240306 Axl=2.788968V #SATS=04
$GPRMC,201028,V,4216.5333,N,07148.4155,W,000.0,000.0,240306 Axl=3.044022V #SATS=04
$GPRMC,201030,V,4216.5333,N,07148.4155,W,000.0,000.0,240306 Axl=3.039139V #SATS=04
$GPRMC,201032,V,4216.5333,N,07148.4155,W,000.0,000.0,240306 Axl=2.581979V #SATS=04
$GPRMC,201034,V,4216.5333,N,07148.4155,W,000.0,000.0,240306 Axl=3.014725V #SATS=04
$GPRMC,201036,V,4216.5333,N,07148.4155,W,000.0,000.0,240306 Axl=3.029068V #SATS=04
```

Figure 31: Data Results from Microcontroller

With the use of GIS software, this data is transformed into a visual map with points of the road conditions. Appendix B explains how the GIS software works and how the final display is created. The final map of which displays the results is shown in **Figure 32**. A point, which consists of the location and accelerometer voltage, is recorded every second. With the GIS software, we were able to display these points in different colors; a low voltage is associated with the color green and means a low voltage and that the road condition is good, the color yellow means mediocre, and the color red means a bad road condition and a high voltage.

Figure 32: Final Map

6.4. Summary

This chapter presented the results of our system design. The design of the Accelerometer module and Pothole Module were discussed in detail; detailing the conditioning of the accelerometer signal. The design of hardware such as the motherboard module was also illustrated. In addition to the hardware results, our team presented the software implemented in bringing functionality to the entire system; software flow charts and source code were included as reference. The tests we performed on our system were all done in the Worcester area. We verified that the tests would determine potholes and other road conditions.

7. Conclusions and Recommendations

This chapter summarizes the work completed by our project team at Worcester Polytechnic Institute. The project objectives that were accomplished, with the intent of meeting the project goal, are also reviewed. The overall goal of this project was to develop an automated data collection system that can be installed in any automotive vehicle to monitor road or highway pavement conditions.

7.1. Summary of Project Design, System Design and Results

To complete our project design we followed a methodology that consisted of researching how to measure engine potholes and road conditions, establishing our system requirements, building our system from those system requirements, and finally processing the data produced by our device to provide analysis.

Our system design consisted of various modules, these modules are listed below:

1. GPS Module
2. LCD Module
3. Health and Safety Module
4. Accelerometer Module
5. Pothole Module
6. Microcontroller Module
7. Motherboard

The GPS module consisted of a GPS receiver that output NMEA string serially to the microcontroller. The GPS receiver transmitted information like: date, time, number of satellites in view, velocity, and coordinates among other things.

The LCD module was another stand alone unit, like the GPS receiver, that received serial data from the microcontroller to display information for the user. The data that was displayed on the LCD display consisted of real time engine RPM readings, temperature and power source voltage levels, as well as the number of satellites the device was currently using to store GPS related data.

The health and safety module allowed the device to monitor ambient temperature's as well as power source voltage levels. This module provides a means of letting a user know of a

fault that is causing the system function incorrectly. It also allows the microcontroller to stop writing to compact flash so that data does not become corrupted.

The Accelerometer module's function is to condition the signal from the accelerometer. The conditioning included filtering and amplification. The signal is conditioned so that it can be transferred to through a long cable and be processed by the Pothole module.

The Pothole module's function is to condition the signal from the Accelerometer module so that it can be interfaced to the microcontroller for processing. This is done by filtering, amplification, and peak detected.

The microcontroller's function is to provide data processing, displaying data, and for data storage on compact flash. In essence the microcontroller is interfaced with all the other modules for complete system integration by way of the motherboard.

The motherboard provides a means to interconnect all the different modules to the microcontroller. It also allows the device to function in modules and different modules could be added to provide for other functions.

Our project team was able to successfully implement all these modules in hardware. In addition we also wrote the software that allowed the system to properly function as an automated data collection system for monitoring road conditions. The final system was tested in field and we were able to see it working correctly.

7.2. Overall Assessment and Future work

There are several improvements that we would recommend in the continued development of this project. The next generation might include the use a wireless accelerometer module and also the use of multiple accelerometers (1 per each wheel). Also, a complete compact unit which could include the accelerometers, microcontroller, memory, and GPS receiver all in once case could be designed. Finally, software algorithms which incorporate the speed of the vehicle should be developed to better map road conditions.

7.3. Conclusions

Finally, we were able to successfully implement a GPS-GIS pothole mapping system which can be easily redesigned to fit smaller enclosures. We were also able to create a device in which the user interface can be easily updated for other functions and applications. Overall, we feel that our system could potentially lower the percentage of damaged roads by properly allocating road repair resources.

8. References

Introduction

This chapter lists the references used for this report. The list ordered by the sequence each reference appears in the main text. Each reference number corresponds to the appropriate footnote in the main text.

1. http://www.lacity.org/boss/Resurfacing/potholes.htm2.
2. http://www.virginiadot.org/infoservice/faq-potholes.asp
3. http://www.cityoflewiston.org/publwrks/Pages/Street%20Pages/potholes.htm
4. http://www.asce.org/reportcard/2005/index.cfm
5. http://www.garmin.com
6. http://www.garmin.com/products/gps25/
7. http://www.garmin.com/products/gps25/
8. http://www.garmin.com/manuals/GPS25LPSeries_TechnicalSpecification.pdf
9. http://www.crystalfontz.com
10. http://www.crystalfontz.com/products/632/index.html
11. http://www.crystalfontz.com/products/632/index.html
12. http://www.crystalfontz.com/products/632/data_sheets/CFA-632_v2.0.pdf
13. http://www.tern.com
14. FlashCore-B(FB)™ Technical Manual. Tern Inc. http://www.tern.com
15. FlashCore-B(FB)™ Technical Manual. Tern Inc. http://www.tern.com
16. FlashCore-B(FB)™ Technical Manual. Tern Inc. http://www.tern.com
17. FlashCore-B(FB)™ Technical Manual. Tern Inc. http://www.tern.com

Appendix A: Executive Summary

Project Code: FC- PHL1

GPS Coordinated Pothole Mapping

Executive Summary

Project Team:

Nicholas Angelini
na1303@wpi.edu

Jose Brache
jbrache@wpi.edu

Matthew Gdula
mgdula@wpi.edu

Craig Shevlin
cshvelin@wpi.edu

Advisors:

Fabio Carrera
carrera@wpi.edu

Michael Ciaraldi
ciaraldi@wpi.edu

Fred Looft
fjlooft@wpi.edu

Date: March 15, 2006

gps-05@wpi.edu

100 Institute Road Worcester, MA 01609

1. Introduction

The conditions of city streets, particularly in New England, are a major concern for local roadwork crews, as well as everyday drivers.

To assist in road maintenance, this project team designed a device to successfully identify a pothole. Furthermore, this device systematically used GPS to locate the position of a given pothole. To accomplish this task the device must be able to determine if an encountered shock from striking a pothole is great enough to be deemed as a significant road deformity and not just a bumpy surface. Secondly, this pothole data must be processed with the use of a microcontroller that is interfaced with a GPS receiver to determine the coordinates multiple potholes encountered. The remainder of this document details the design process undertaken by this project team to design and build a working prototype.

2. Background

The occurrences of potholes, especially in New England streets, have become a major nuisance for both road repair crew and daily commuters. Roads are constantly being subjected to weather fluctuations, extreme or not these fluctuations worsen the conditions of city streets and makes road repair an ongoing concern. According to the American Society of Civil Engineers' "Report Card for America's Infrastructure", "poor road conditions cost US motorists $54 billion in vehicle repairs and operating costs," $2.3 billion coming from Massachusetts' motorists. The report also states that, "71% of Massachusetts' major roads are in poor or mediocre condition." Massachusetts has made strides to repair the infrastructure, i.e. the "Fix-it-First" policy which was passed in 2003 that gives "priority to the repair of existing streets, roads and bridges," but still faces many difficulties in preventing damages.

3. Project Statement

This design project was conceptualized in an initial effort to develop an information system that could potentially lower the percentage of damaged roads by properly allocating road repair resources. The goal of this capstone design project was to design a fully automated data collection system that can be installed in any automotive vehicle to monitor road or highway pavement conditions. In order to monitor road or highway pavement conditions, Geographical Information System software was used to remap the GPS surface roughness data on a user viewable city map.

4. Methods

During the design process there were two main tasks needed to be successfully completed to develop a functional device, retrieval of data and processing of data. The first task, a way to

retrieve useful data that we would be able to use to effectively map potholes, was completed by following these steps:

1. Reviewed project requirements (what exactly needed to be recorded).
2. Reviewed previous work done on the problem.
3. Researched different approaches (used reference material).
4. Constructed and tested circuit using simulation software.
5. Constructed and tested circuit on a breadboard.
6. Completed circuit and final testing using a PCB board.

The final task of our project was to develop a way of processing data and outputting it. The task of processing data was completed by using the software given to us by the manufacturer of our chosen microcontroller and creating algorithms that successfully recorded the data. The task of outputting the data was completed by using chosen GIS software and inputting the recorded data from the microcontroller to the GIS software.

5. Results

Towards the initial stages of this project, we selected and tested multiple accelerometers in vehicles. We chose a specific accelerometer to which we based the design of the rest of our system. The raw signal from the type of accelerometer used in this project is shown below in **Figure 1**.

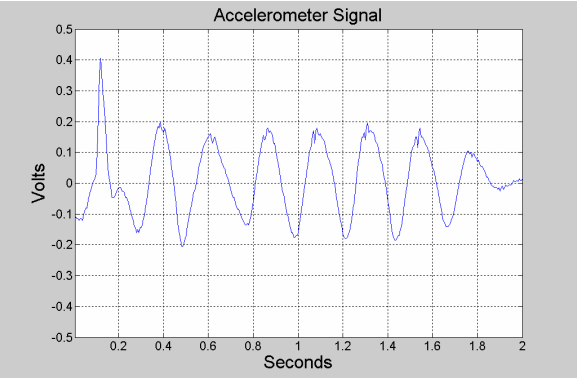

Figure 1: Raw Signal from Accelerometer

Using the voltage signal from the accelerometer shown in **Figure 1**, we designed and simulated the pothole detector circuit under software control. After successful simulation trials, we implemented and tested the design in the field. The signal from the accelerometer is wired to the designed "peak detector" circuit. The output of the peak detector circuit is shown in **Figure 2**.

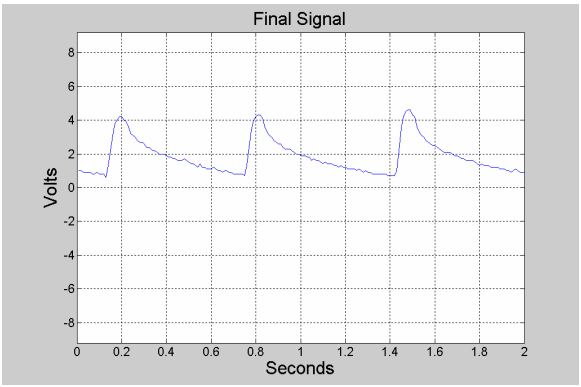

Figure 2: Output of "Peak Detector"

Comparing the waveform shown in **Figure 2** to the waveform in **Figure 1,** it can be seen that the output of the "peak detector" circuit holds the peaks from the accelerometer signal. Holding these peaks for a short amount of time is necessary because the microcontroller requires time to process these signals.

Once the pothole detection circuitry (Peak Detector), the accelerometer circuitry, and the microcontroller were all operating up to design specifications, the prototype was designed on printed circuit board (PCB) layout software. The Global Positioning System (GPS) receiver, liquid crystal display (LCD), and all the individual boards, were then integrated to create a functional device. The system integration is shown in **Figure 3.**

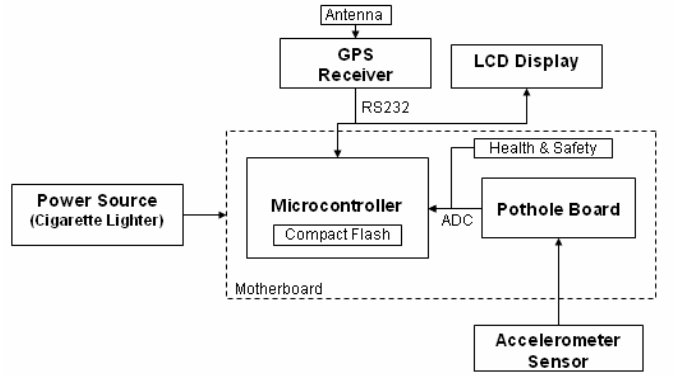

Figure 3: System integration

Once the system was integrated and fully operational, field testing was completed. After being coated with epoxy, to avoid environmental damage, the accelerometer board was placed in the wheel well and cables ran to the data logging system, located inside the of the car. Once successfully setup, test runs were made around Worcester, MA. With the use of Graphical Information System (GIS) software, we were able to display our results on a user friendly map. These results can be seen in **Figure 4** shown below.

Figure 4: GIS Results

6. Summary and Conclusions

Our project group was able to successfully implement a GPS-GIS pothole mapping system which can be used in any vehicle. Our system can be easily redesigned to fit smaller enclosures, and also, the user interface can be easily updated for other functions and applications, making our system very useful for other projects. There are some improvements that could be looked into such as wireless accelerometers and the use of multiple accelerometers (1 per each wheel). The next generation system should also use constant logging to determine road smoothness, and use algorithms that would help map these road conditions. Overall, our current system could potentially lower the percentage of damaged roads by properly allocating road repair resources.

Key References

1. http://www.fcny.org/cmgp/streets/pages/indexb.htm
2. http://www.asce.org/reportcard/2005/index.cfm
3. Sedra, Adel, and Kenneth Smith. Microelectronic Circuits. New York: Oxford University Press, 2003.

Appendix B: MapInfo/MapBasic

MapBasic Pothole Code

This supplement is for use of the MapBasic code written to document potholes for City Lab MQP. The code has been designed to implement a GIS map of Worcester with orthophotos (taken from: http://maps.massgis.state.ma.us/massgis_viewer/index) and map GPS coordinates from a text file as an additional layer. The following is a guide explaining the code.

First, functions are declared at the beginning of the file. The three functions of this program are "main", "inputpotholes", and "map." Every function begins with "sub 'function name'" and ends with "end sub." Once the functions are declared, the main function is executed. The first step of the main function is to create a unique menu. The menu's name is "Add Potholes" which has two options: "Draw map" which calls the "map" function and "Read potholes" which calls the "inputpotholes" function. Creating this menu for MapInfo is done by using the "create menu" statement. The next step in the main function is to add the newly created menu to MapInfo's menu bar. This is executed by the "alter menu bar" statement. Once this statement is executed the program will terminate.

When called to in the main function, the "inputpotholes" function will prompt the user the for the pothole text file (which can be located on the flash card). Once the text file is inputted, MapInfo will then create a table of GPS coordinates including the accelerometer value, number of satellites, and Coordinated Universal Time (UTC date and time). The first step in this function is to create the variables that are going to be needed in this function. The variables in this function are tablename, yNorS, xEorW, readline, temp1, temp2, temp3, temp4, temp5, temp6, P, x, y, v, s, day, month, year, date, hour, min, sec, and time. Declaring these variables is done by using the "dim" statement. Once the variables are declared the program prompts the user to open the text file which contains the pothole information. This is done by the "open file FileOpenDlg" statement. Next, the table that contains the pothole information is created which is executed by "create table." Once created, a while loop begins to read all the needed information off of the text file. Each piece of needed information is located on the same spot of each line of text which makes the "mid$" statement very valuable in this process. Each line of text is stored into the "readline" variable and the "mid$" statement can take information from a given location on the stored line of text. Once this text is located MapBasic uses a statement called "val" which takes the string and converts it into a numeric value. The first numeric value is stored in temp1 and the following in temp2. Temp2 is then divided by 60 and then stored into temp3. This process is done because the GPS information needs to be converted from decimal minutes to decimal degrees.

Once the correct GPS value is calculated the program checks if it is in the northern or southern hemisphere. This is done by using an "if…then" statement. After the latitude is finished being calculated the longitude goes under the same process. Next, the voltage of the accelerometer is read in along with the number of satellites and this information is stored into variables "v" and "s". After the location, voltage, and satellites are stored, MapBasic reads in the date and time of each point. This text is then formatted for easy use. The next step is to set each point into a specific color. If the accelerometer voltage is between 0V and 1.5V, the point will be green, if between 1.51V and 3.0V, the point will be yellow, and if between 3.1V and 5V, the point will be red. This is done to distinguish between a large pothole and smooth pavement. Once the value has been calculated, MapBasic creates a point using the "create point into variable" statement. Each point is put into the table and is ready to be plotted onto a map. When the text file is ended, the program will close the text file and return to the main function.

After the table of pothole points has been created, the user is able to use the "map" function (This function is mainly for visual purposes and if the user has another map than Worcester this step can be removed or the code can be changed to add the needed map). When executed, the user is prompted to open the Worcester workspace. This previously created workspace contains the orthophotos and layers needed to create the map. Once the workspace has been opened, the program prompts the user to open the newly created pothole table. The next step in the code maps all the files together using the "map from" statement. Next, all the zoom layering constrictions are removed from each layer using the "Set Map Layer" statement. Next, the function centers the map on Atwater Kent at -71.809746°, 42.27447°. Finally, a "browser" window is opened containing all the pothole information so the user can see the created table. The map will appear with every layer and the potholes marked on the map.

MapInfo Software

This part of the supplement describes how to use the created executable MapBasic file. First, double click on the "mqp.mbx" file to open it. On the menu bar there will be an "Add Potholes" menu as seen in **Figure 33**.

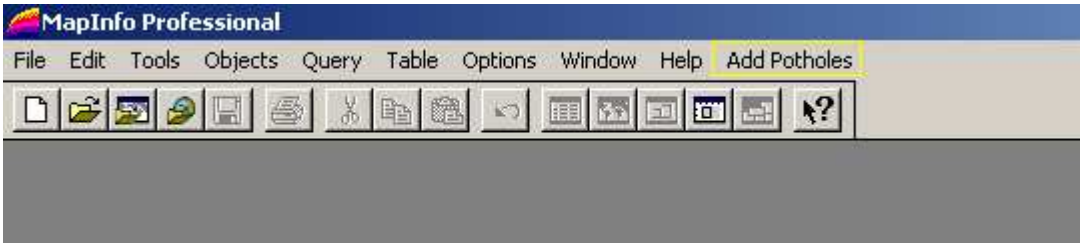

Figure 33: Add Potholes menu

Click on this menu to open it and two options will be given, “Read Potholes” and “Create Map”. Click on the “Read Potholes” option first which can be seen in **Figure 34**.

Figure 34: Read Potholes

This will open a dialog box asking to open the potholes.txt file which can be seen in **Figure 35**. Move to the directory where the pothole file is located and open it.

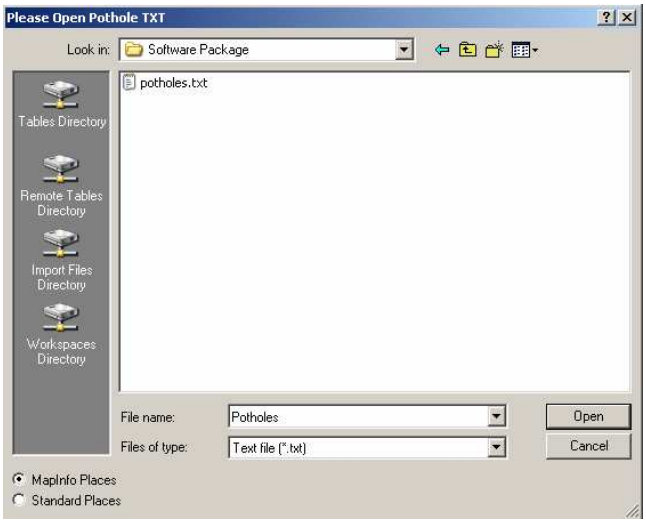

Figure 35: Pothole Dialog Box

Next, follow **Figure 36** and click on the Create Map option in the menu.

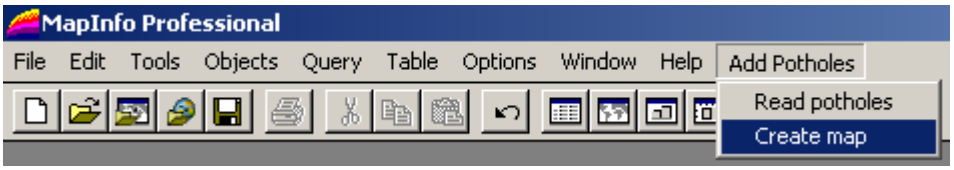

Figure 36: Create Map

This will open a dialog box that asks the user the open the Worcester workspace. This can be seen in **Figure 37**.

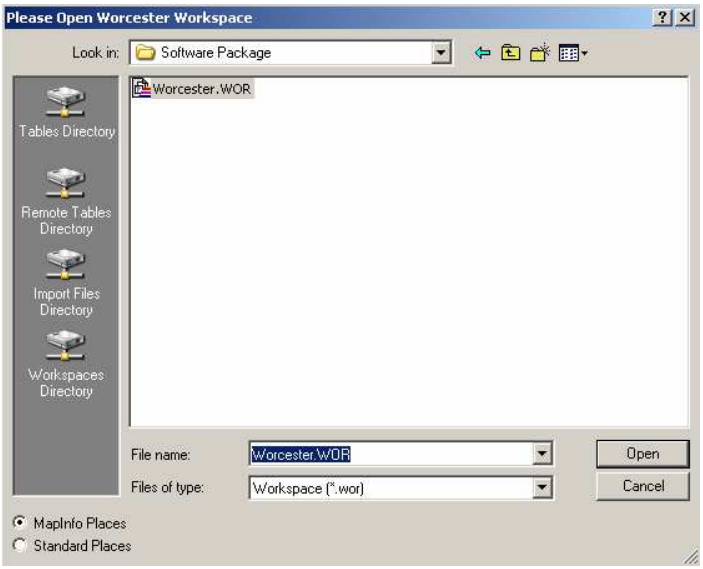

Figure 37: Worcester Workspace

After the workspace has been open, another dialog box appears and prompts the user to open the potholes table that was created when "Add Potholes" was executed. This dialog box is shown in **Figure 38**. After being opened a map with the points will be displayed. This is the file product of this program. A sample map can be seen in **Figure 39** and a sample browser window can be seen in **Figure 40**. When exiting MapInfo you will be asked to save the file, the option is up to the user to save the results.

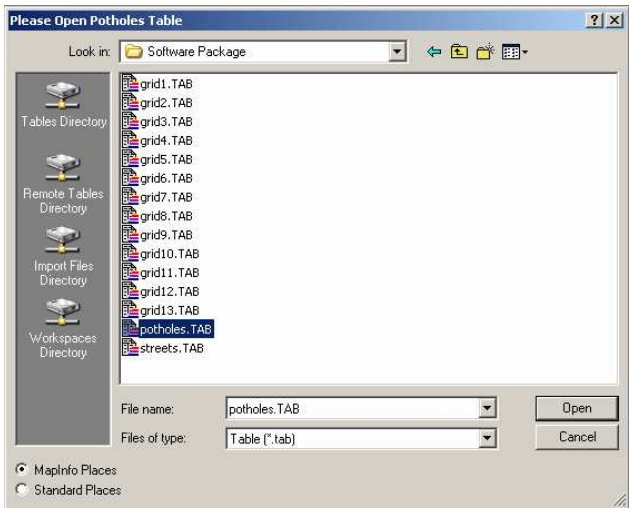

Figure 38: Potholes Table

Figure 39: Sample Map

potholes Browser

Latitude	Longitude	Voltage	Sats	UTC_Date	UTC_Time
42.2756	-71.8069	1.29214	9	3/24/6	20:9:12
42.2712	-71.799	2.80766	12	3/24/6	20:10:19
42.268	-71.7984	3.77272	1	3/24/6	20:10:21
42.2642	-71.798	0.795377	5	3/24/6	20:10:23
42.2642	-71.798	0.795377	5	3/24/6	20:10:23

Figure 40: Browser Window

Map Basic Code

```
' City Lab MQP
' Adding potholes to the Worcester Map

' declare functions
declare sub main
declare sub inputpotholes
declare sub map

sub main

' Create a menu for MapInfo to execute program
create menu "Add Potholes" as "Read potholes" calling inputpotholes, "Create map" calling map

' Add the menu to MapInfo Menu Bar
alter menu bar add "Add Potholes"

end sub
```

```
' plot the worcester table and pothole table
sub map

' Prompt user to open worcester workspace
Run Application FileOpenDlg("C:\","Worcester","WOR" ,"Please Open Worcester Workspace")

' Prompt user to open the newly created potholes table
open table FileOpenDlg("C:\","potholes","TAB" ,"Please Open Potholes Table")Interactive

' map all files together
map from potholes,streets, grid1, grid2, grid3, grid4,grid5,grid6, grid7,
grid8, grid9, grid10, grid11, grid12, grid13

' remove zoom layering constriction to all grids

Set Map Layer 3  Zoom (0, 20) Units "mi"  Off
Set Map Layer 4  Zoom (0, 20) Units "mi"  Off
Set Map Layer 5  Zoom (0, 20) Units "mi"  Off
Set Map Layer 6  Zoom (0, 20) Units "mi"  Off
Set Map Layer 7  Zoom (0, 20) Units "mi"  Off
Set Map Layer 8  Zoom (0, 20) Units "mi"  Off
Set Map Layer 9  Zoom (0, 20) Units "mi"  Off
Set Map Layer 10  Zoom (0, 20) Units "mi"  Off
Set Map Layer 11  Zoom (0, 20) Units "mi"  Off
Set Map Layer 12  Zoom (0, 20) Units "mi"  Off
Set Map Layer 13  Zoom (0, 20) Units "mi"  Off
Set Map Layer 14  Zoom (0, 20) Units "mi"  Off
Set Map Layer 15  Zoom (0, 20) Units "mi"  Off

' Center map to Atwater Kent
Set Map  Center (-71.809746, 42.27447)

'open pothole table
Browse * from potholes
end sub

' input the potholes text file and convert to latitude and longitutde, making a table
sub inputpotholes

' declaring all variables
dim tablename, yNorS, xEorW, readline  as string
dim temp1, temp2, temp3 as float
dim temp4, temp5, temp6 as float
dim P as object
dim x, y, v as float
dim s as integer
dim day,month,year,date as string
dim hour, min, sec, time as string

' Prompt user to open Potholes output (From GPS)
open file FileOpenDlg("C:\","Potholes","TXT" ,"Please Open Pothole TXT") for input as 1
```

```
' give the table's name "potholes"
tablename = "potholes"

' declare the columns latitude and longitude
create table tablename (Latitude float, Longitude float, Voltage float,
Sats integer,UTC_Date char(8), UTC_Time char(8))

' create a map for potholes
create map for tablename

' read the file until nothing is left
while not eof(1)

' read entire line
line input #1, readline

' change GPS string to decimal degrees from decimal minutes
' negate if value is S
temp1 = val (mid$ (readline,17,2))
temp2 = val (mid$ (readline,19,7))
temp3 = temp2/60
y = temp1 + temp3
yNorS = mid$(readline,27,1)
' if South make latitude negative
if (yNorS = "S") then
 y=-y
end if

' change GPS string to demical degrees from decimal minutes
' negate if value is W
temp1 = val (mid$(readline,30,2))
temp2 = val (mid$(readline,32,7))
temp3 = temp2/60
x = temp1 + temp3
xEorW = mid$(readline, 40,1)
if (xEorW = "W") then
 x=-x
end if

v = val (mid$(readline,65,8))
s = val (mid$(readline,81,2))


'Create Date and Time
hour = val (mid$(readline,8,2))
min = val (mid$(readline,10,2))
sec = val (mid$(readline,12,2))
time = hour + ":" + min + ":" + sec

day = val (mid$(readline,54,2))
```

```
month = val (mid$(readline,56,2))
year = val (mid$(readline,58,2))
date = month + "/" + day + "/" + year

' set symbol for potholes
Include "mapbasic.def"

if(v <= 1.5) then
Set style Symbol MakeSymbol (33,GREEN, 15)
elseif (v>1.5 AND v<=3 ) then
Set style Symbol MakeSymbol (33, (RGB(255,238,21)), 15)
else
Set style Symbol MakeSymbol (33,RED, 15)
end if

'make a point
create point into variable P
(x, y)
insert into tablename (Longitude, Latitude, Obj, Voltage, Sats,UTC_Date, UTC_Time)
values (x, y, P, v,s,date,time)

wend

' close pothole.txt file
close file 1

end sub
```

Appendix C: C++ Embedded Program Source Code

The source code provided in the **Program Source Code** section of this appendix comprises the entire operation of the device from a software standpoint. A software flow chart is included in **Figure 41** to provide a functional top level view of our program. Some other functions defined in other header files are not included in this document. Please explore the files included in the [Paradigm C++ Light – Tern Edition IDE] CITYLAB project file. In addition to these files please see the TERN FlashCore-FB technical manual[17] for the description of the functions controlling the FlashCore-B provided by TERN. This code was specifically written for the use of measuring road conditions.

[17] FlashCore-B(FB)™ Technical Manual. Tern Inc. http://www.tern.com

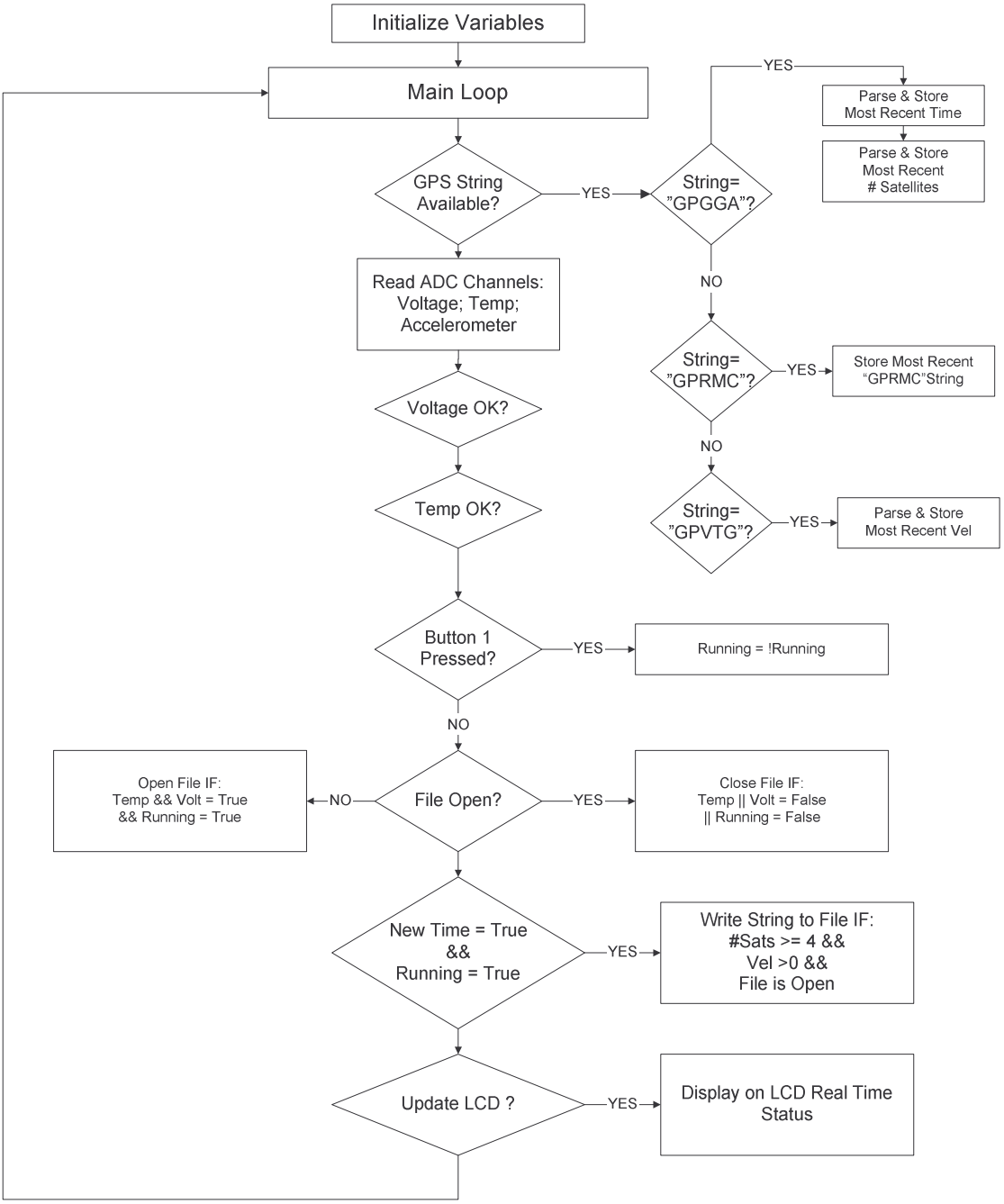

Figure 41: Pothole Software Flow Chart

1. Program Source Code

```
/*********************************************************************
        For C++ object files (.cpp), TERN header files must be
  declared as "C" objects.

  extern "C" {
        #include " .h"
```

```
        #include " .h"
        #include " .h"
  }
  3-10-03
  - Also, if you're using C++ classes (even if you don't instantiate
  objects dynamically), you will need to define a non-zero heap.
  Take a look at the 'readme.txt' to see how to define a heap using
  heapsize.c
*****************************************************************/
extern "C" {
        #include "ae.h"   // AE88 initialization */
        #include "ser1.h"
        #include "fileio.h"
        #include "string.h"
        #include "stdio.h"
        #include "adc.h"
        #include "math.h"
}
#include <dos.h>
#include <string>
#include <stdlib.h>
#include "gpsstringreader.h"

#define BUFFSIZE 1024

#ifndef NULL
#define NULL 0
#endif

unsigned char inBuff[BUFFSIZE];
unsigned char outBuff[BUFFSIZE];
extern COM ser1_com;
COM* com1 = &ser1_com;

unsigned int adat[8];
unsigned int fb_ad16(unsigned char k);

// Control bytes for single ended, internal clock mode of ADS8344
// S A2 A1 A0 - SGL/DIF PD1 PD0, 1xxx 0110
unsigned char c_byte[] = {0x86,0xc6,0x96,0xd6,0xa6,0xe6,0xb6,0xf6};
enum {TEMP, VOLT, POT, DIESEL, GAS};
//ad0 temp
//ad1 volts
//ad2 pot holes
//ad3 diesel
//ad4 gas

/*****************************************************
*       Function: double ConvertVolts(unsigned int Axl)
*                 Output: type double representing value in Volts
*
```

```
*                   where:  int Axl is value read from ADC (Decimal)
**************************************************/
double ConvertVolts(unsigned int Axl)
{
        double retVal=Axl;
        retVal /= 0xFFFF;
        retVal *= 5;
        return retVal;
}

/**************************************************
*       Function: int FindNumSats(const char* gpsStr)
*               Output: type int the number of satellites according to
*                                       GPS string
*
*               where:  gpsStr is the GPS string type '$GPGGA'
**************************************************/
int FindNumSats(const char* gpsStr)
{
        const char* curTok = gpsStr+41;
        int retVal = ((curTok[0]-0x30)*10)+(curTok[1]-0x30);
        return retVal;
}

/**************************************************
*       Function: int FindVelocity(const char* gpsStr)
*               Output: type int the velocity in knots (without decimal point) as parsed
*                       from the GPS string
*
*               where:  gpsStr is the GPS string type '$GPVTG'
**************************************************/
int FindVelocity(const char* gpsStr)
{
        const char* curTok = gpsStr+19;
        int retVal = ((curTok[0]-0x30)*1000)+((curTok[1]-0x30)*100)+((curTok[2]-
                0x30)*10)+((curTok[4]-0x30));
        return retVal;
}

/**************************************************
*       Function: void UpdateLcd(char* topLine, bool tStat, bool vStat, int numSats)
*               INPUT: char* topLine - String holding the characters to be disp. On the
*                                       Top Line
*                       bool tStat - Temperature Status
*                       bool vStat - Voltage Status
*                       int numSats - Number of Satellites
*
*               The topLine is outputted to the LCD display serially. The bottomline
*               is then outputted after tStat, vStat, and numSats are put together.
**************************************************/
void UpdateLcd(char* topLine, bool tStat, bool vStat, int numSats)
```

```
{
        char bottomLine[17];
        char* tempStr;
        char* voltStr;

        if(tStat)                       // If temp status is true, temp is OK, else NOT OK
                tempStr = "Y";
        else
                tempStr = "N";
        if(vStat)                       // If voltage status is true, voltage is OK, else NOT OK
                voltStr = "Y";
        else
                voltStr = "N";

        sprintf(bottomLine, "T=%s V=%s SATS=%i", tempStr, voltStr, numSats);

        putser1('\f', com1);            // Clear LCD
        putsers1(topLine, com1);        // Place topLine on first line
        putser1('\n', com1);            // New Line
        putser1('\r', com1);            // Carriage Return
        putsers1(bottomLine, com1);     // Output bottomLine
}

#define MAX_TEMP 0xCCCC
#define MIN_TEMP 0x3333

#define MAX_VOLT 0x80F0
#define MIN_VOLT 0x5300

#define MIN_SATS 4
#define MIN_VEL 0

void main(void)
{
        ae_init();                      // A-Engine initialization

        for(int i=0; i<=5; i++)         // Delay for about 5 Sec.; waiting for GPS
                delay_ms(999);

/******ADC INITIALIZATION******************/
        pio_init(9, 1);                         // A19=P9 as input for ADC data output DOUT
        pio_init(2, 0);                         // P2 as PCS6
        for(int i=0; i<8; i++)
                outportb(0x600+i, 0xff);        // All control pins high
/*******************************************/

        pio_init(26, 3);                        //button init as an input

        //intialize the serial port
        unsigned char baud = 7;         //baud rate for the serial port
        s1_init(baud, inBuff, BUFFSIZE,  outBuff, BUFFSIZE, com1);
```

```
        //initialize flash
        fs_descrip* file = NULL;

        if(fs_initPCFlash()!=0)          // Compact Flash Card is missing
        {
                putsers1("\fRestart with\n\rCompact Flash", com1);
                while(1)
                {
                        delay_ms(999);          // Infinite Loop
                }
        }

        char* topLine = NULL;           // topline of the display

        bool tStat = false;             // temp status
        bool vStat = false;             // voltage status

        bool needUpdate = true;         // need to update the lcd?

        bool running = true;
        bool butDown = false;
        bool record = false;

        GPSStringReader sr(com1);       //Object to read GPS strings
        char* gpsStr = NULL;            //the string being read in from GPS
        char curGPS[100] = "";          //the last gps string we care about
        int numSats = -1;               //the current number of satellites in view

        led(0);
        char strType[6] = {0,};
        char strnewTime[7] = "";
        char strTime[7] = "";
        int Vel = 0;

        double Axl;

while(true)     //MAIN LOOP
{
/******GPS STRING HANDLING**********************/
        if(gpsStr = sr.GetString())
        {
                for(int i=0;i<5;i++)
                strType[i] = gpsStr[i+1];

                if(!strcmp(strType, "GPGGA"))
                {
                        for(int i=0;i<6;i++)            // For extracting Time
                        strnewTime[i]= gpsStr[i+7];

                        int newNumSats = FindNumSats(gpsStr);
```

```
                        if(newNumSats!=numSats)                 // need to change display
                        {
                                numSats = newNumSats;
                                needUpdate = true;
                        }
                        else if(strcmp(strnewTime, strTime))     // Need to record
                        {
                                strcpy(strTime, strnewTime);
                                record = true;
                        }
                }
        else if(!strcmp(strType, "GPRMC")&&(numSats>=MIN_SATS))
                strncpy (curGPS, gpsStr, 59);

        else if(!strcmp(strType, "GPVTG")&&(numSats>=MIN_SATS))
        {
                int newVel = FindVelocity(gpsStr);
                if(newVel!=Vel)                                 //need to change display
                {
                        Vel = newVel;
                }
        }

        }
/***********************************************/

        //Update values from ADC
        for(int i=0; i<5; i++)
                adat[i] = fb_ad16(c_byte[i]);


/*******TEMPERATURE MONITORING******************/
        if(tStat)
        {
                if((adat[TEMP]<MIN_TEMP)||(adat[TEMP]>MAX_TEMP))
                {
                        tStat = false;
                        needUpdate = true;
                }
        }
        else
        {
                if((adat[TEMP]>MIN_TEMP)&&(adat[TEMP]<MAX_TEMP))
                {
                        tStat = true;
                        needUpdate = true;
                }
        }
/***********************************************/

/********VOLTAGE MONITORING**********************/
```

```
		if(vStat)
		{
			if((adat[VOLT]<MIN_VOLT)||(adat[VOLT]>MAX_VOLT))
			{
				vStat = false;
				needUpdate = true;
			}
		}
		else
		{
			if((adat[VOLT]>MIN_VOLT)&&(adat[VOLT]<MAX_VOLT))
			{
				vStat = true;
				needUpdate = true;
			}
		}

/*****BUTTON MONITORING*************************/
		if(pio_rd(1)&0x400)
		{
			if(!butDown)
			{
				running = !running;
				needUpdate = true;
				butDown = true;
			}
		}
		else
			butDown = false;

/*****FILE OPEN/CLOSE***************************/
		if(file)
		{
			if(!vStat||!tStat||!running)
			{
				fs_fclose(file);
				file = NULL;
			}
		}
		else
		{
			if(vStat&&tStat&&running)
			{
				file = fs_fopen("potholes.txt", O_WRITE|O_APPEND);
				if(file&&(file->ff_status!=fOK))		//Make sure it opened ok
				{
					fs_fclose(file);
					file = NULL;
				}
			}
		}
```

```
/******Recording Every Second************************/
        if(record&&running)
        {
                Axl = ConvertVolts(adat[POT]);
                needUpdate = true;
                record = false;
                if((numSats>=MIN_SATS)&&(Vel>MIN_VEL)&&file)
                {
                        char strSats[] = {((numSats/10)+0x30),((numSats%10)+0x30), '\0'};
                        fs_fprintf(file, "%s Axl=%lfV #SATS=%s\r\n", curGPS, Axl, strSats);
                }
        }
/***********************************************/

                //Update the display if it needs it
        if(needUpdate)
        {
                if(!vStat||!tStat)
                        sprintf(topLine, "Env Unsafe!");
                else if(!running)
                        sprintf(topLine, "Paused");
                else
                        sprintf(topLine, "Axl=%lf V", Axl);

                UpdateLcd(topLine, tStat, vStat, numSats);
                needUpdate = false;
        }
}
}
```

Appendix D: Value Analysis

We decided that it would be best if we completed a thorough value analysis between microcontrollers to decide which option best fit our needs. We began our analysis by doing research on microcontrollers, specifically looking for ones which had the capability of having a CompactFlash Card. After deciding on the TERN FlashCore-B and the BASIC Tiger we listed the requirements we felt were necessary and gave each of them a value. The following list shows each criteria and its value:

- Size – 3
- Temperature Range – 3
- Ports – 8
- Expandability – 5
- Development – 6
- Language – 4
- I/O – 7
- Power – 7
- Speed – 7
- Price – 10

With this list we were able to build a value matrix which helped us decide on what microcontroller would work best for us. We broke down the microcontroller into three different categories quality, specs, and overall price. Each criterion was given a value, five being the best and one being the worst. The following shows how each criterion is broken down:

Quality

Size: How large is the microcontroller

Very Small – 5

Small – 4

Medium – 3

Large – 2

Very Large – 1

Temperature Range: How severe of conditions can the microcontroller withstand

Extremely Well – 5

Good – 4

Fair – 3

Bad -2

Terrible – 1

Specs

Ports: Amount of Serial Ports

Five or more – 5

Four to Five – 4

Two to Three – 3

One – 2

Zero – 1

Expandability: The ease to expand the microcontroller

Very Easy – 5

Easy – 4

Moderate – 3

Poor – 2

Incapable – 1

Development: The software and peripherals included

Everything needed – 5

Complete Software and Microcontroller – 4

Partial Software and Microcontroller – 3

Just Software – 2

Just Microcontroller – 1

Language: The ease of programming it

Very Easy – 5

Easy – 4

Moderate – 3

Hard – 2

Very Hard – 1

I/O: The amount of ADC/DAC and other devices

Ten or more ADC and Twenty Digital I/O – 5

Seven to Nine ADC and Fifteen to Nineteen Digital I/O – 4

Four to Six ADC and Ten to Fifteen Digital I/O – 3

Two or Three ADC and Five to Nine Digital I/O – 2

One ADC and One to Four Digital I/O – 1

Power: The amount of power it takes to run

Very Small – 5

Small – 4

Medium – 3

Large – 2

Very Large – 1

Speed: How fast it runs

Extremely Fast – 5

Fast – 4

Moderate – 3

Slow – 2

Very Slow – 1

Price: The Cost of the Development Kit

Expensive – 5

Costly – 4

Reasonable – 3

Economical – 2

Inexpensive – 1

Once going through each individual criterion and giving the microcontroller their values, we received the table below:

		Market	TERN		Tiger	
Quality		Value point	Value point	Total	Value point	Total
1	Size	3	3	9	2	6
2	Temp. Range	3	3	9	3	9
	Total			18		15
		Market	TERN		Tiger	
Specs		Value point	Value point	Total	Value point	Total
1	Ports	8	4	32	3	24
2	Expandability	5	2	10	5	25
3	Development	6	3	18	4	24
4	Language	4	4	16	3	12
5	I/O	7	3	21	4	28
6	Power	7	3	21	4	28
7	Speed	7	4	28	3	21
	Total			146		162
		Market	TERN		Tiger	
Cost		Value point	Value point	Total	Value point	Total
1	Price	10	2	20	3	30
	Total			20		30
Customer Value: (Quality*Specs/Cost)				131.4		81

After the value analysis, the TERN is a better choice for our design.

Appendix E: Motherboard Schematics

This appendix illustrates the interconnections made on the motherboard to connect the TERN FlashCore-B microcontroller to a temperature sensor, a voltage meter, power, as well as different headers. The different headers on the motherboard connect to the FlashCore-B pins for expansion. **Figure 42** details all the wirings on the motherboard. The PCB layout shown in **Figure 43** displays the layout of the motherboard, and **Figure 44** shows the PCB wiring schematic. The blue wire layer on the wiring schematic shown in **Figure 44** is the top copper layer and the green is the bottom copper layer. Please see source Schematic and PCB files on our project CD for more details.

IMPORTANT NOTE:

When programming the TERN FlashCore-B microcontroller, a jumper must be placed or removed between J2.1 and J2.3 header (On the actual FlashCore-B board). If the microcontroller is mounted on the motherboard, these two pins can be accessed from H2 on the motherboard. Placing a jumper between H2.7 and H2.8 allows setting up the microcontroller for debug mode or for standalone mode. Likewise if a module is placed on H1 and H2 (On the motherboard), these two pins (H2.7 and H2.8) still have to be accessible for the placement or removal of a jumper between them. Please see the TERN FlashCore-B(FB)^{TM} Technical Manual for more information on the FlashCore-B Programing Overview. The red square, shown in **Figure 43**, shows where a jumper should be placed on the Motherboard PCB schematic.

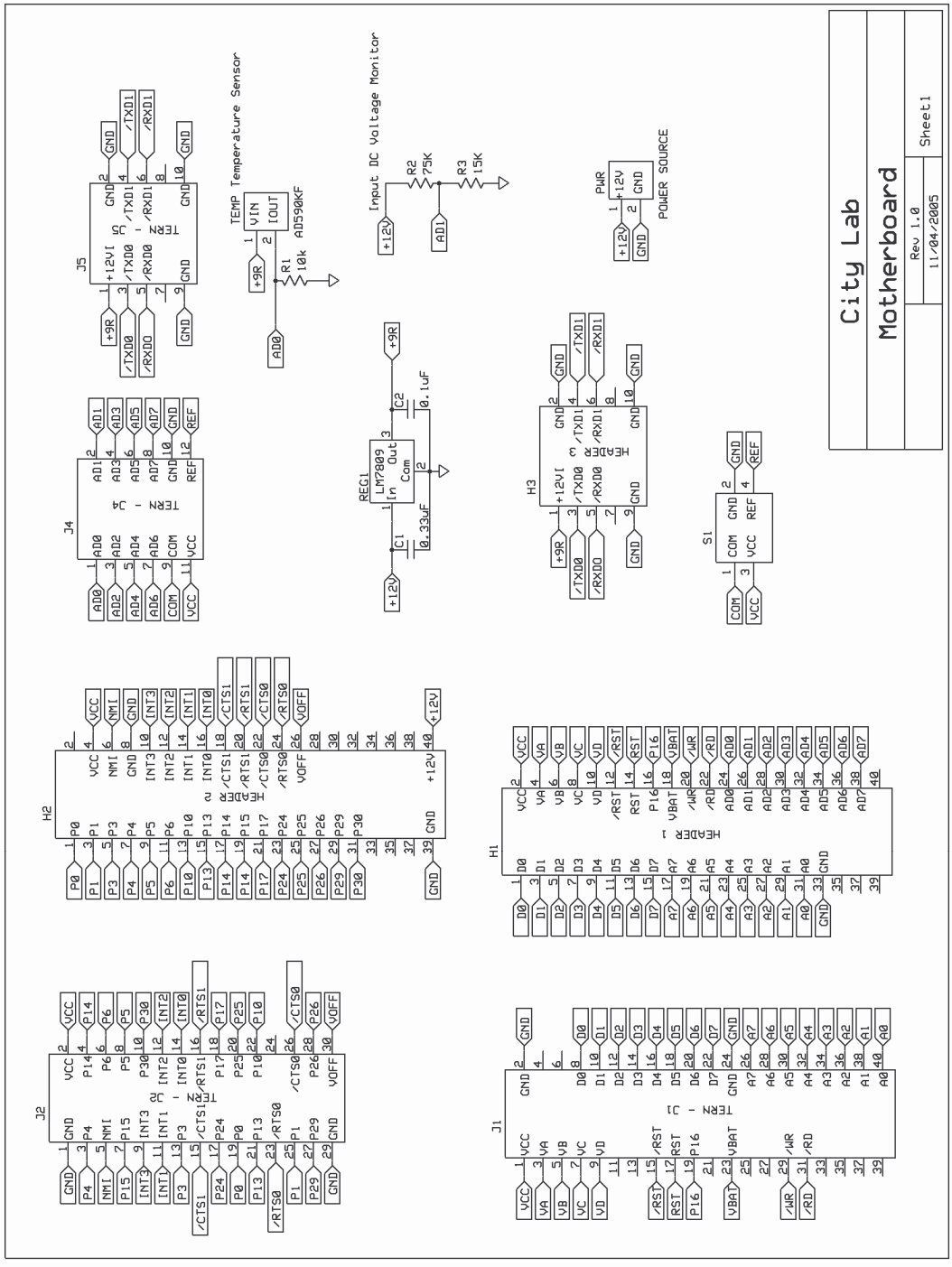

Figure 42: Motherboard Inter Connections

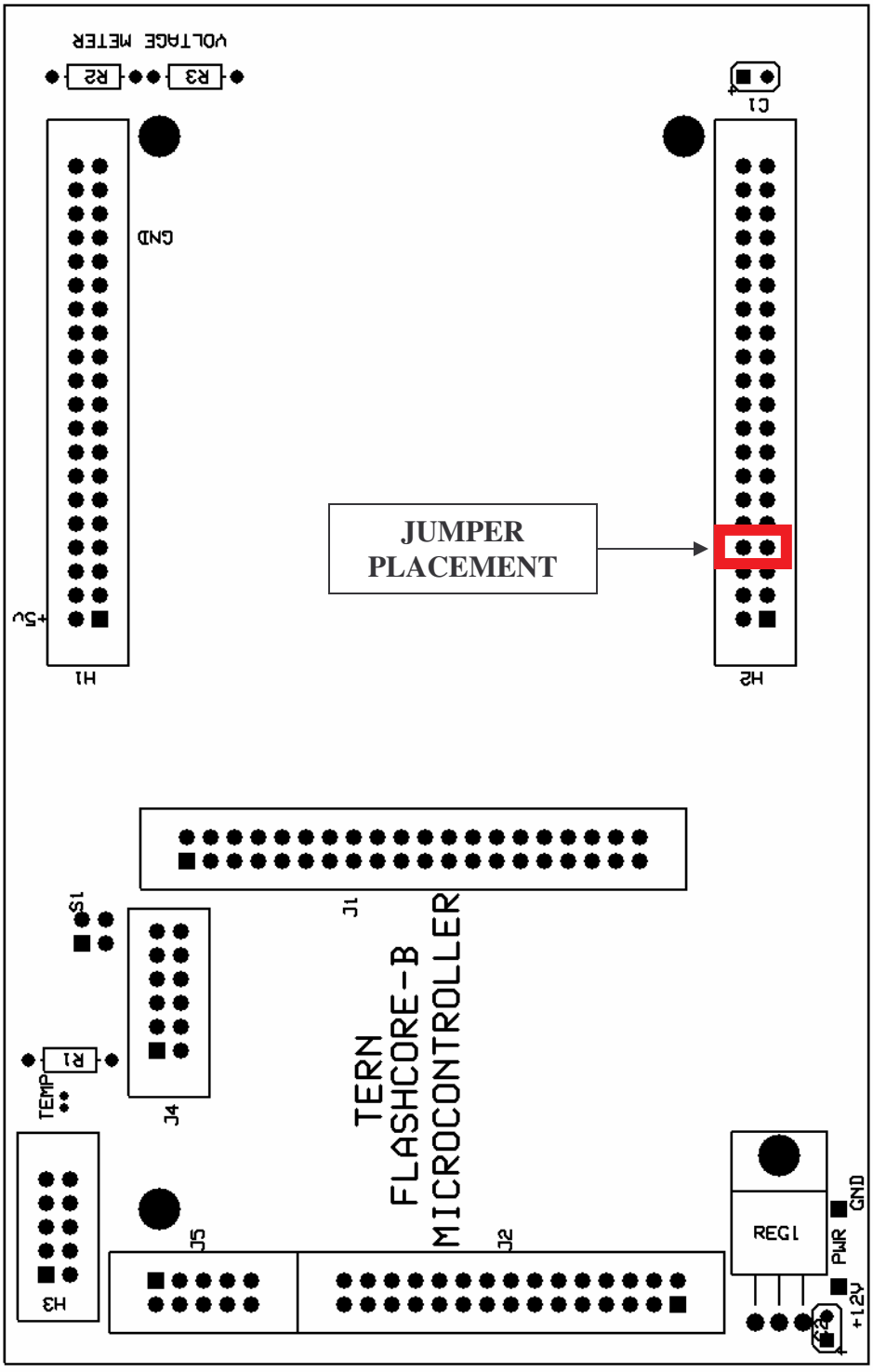

Figure 43: Motherboard PCB Silkscreen Schematic

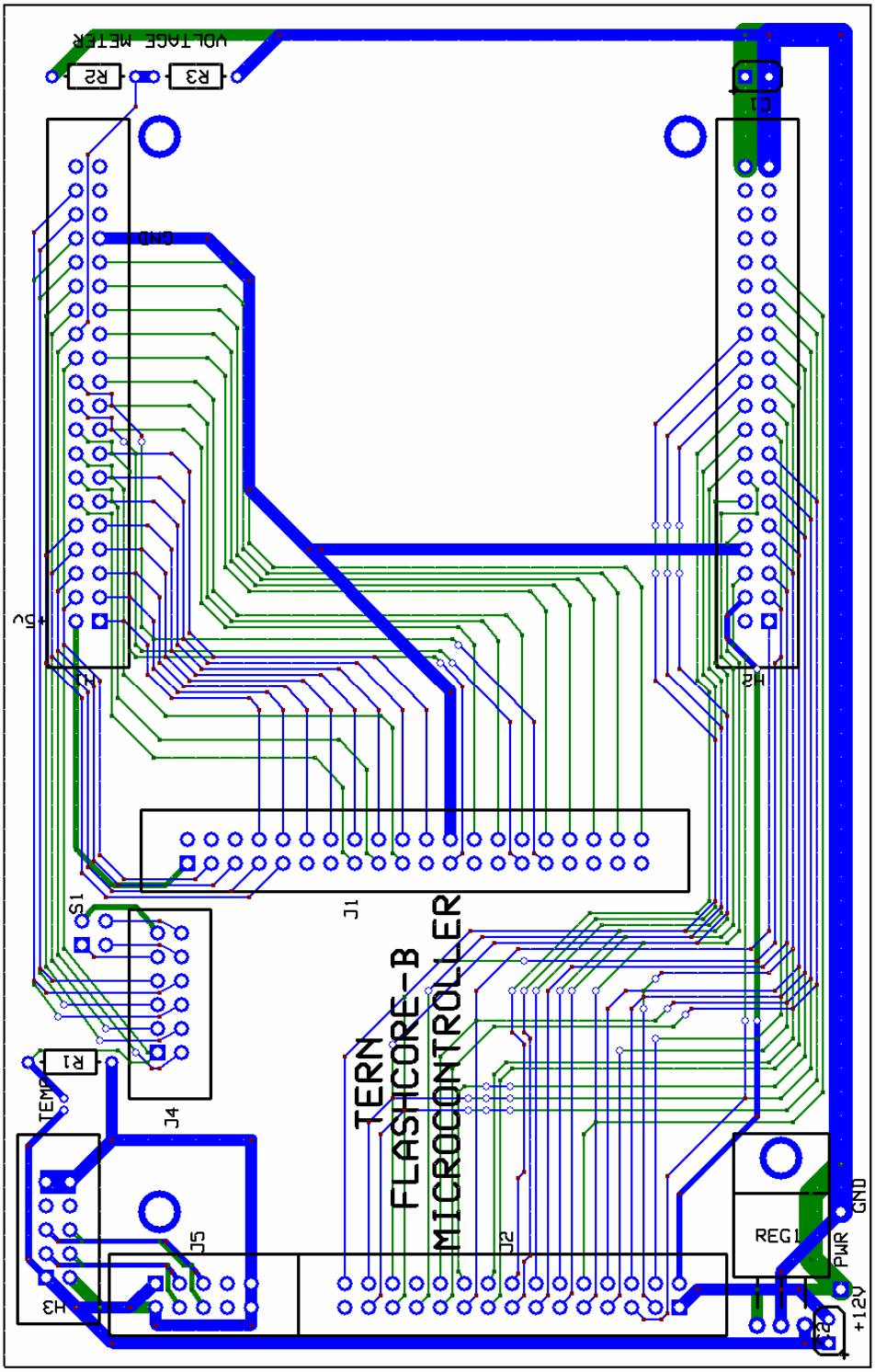

Figure 44: Motherboard PCB Wiring Schematic

Appendix F: Schematics

This appendix illustrates the schematics of the RPM Module circuits designed by our project team. The H1 and H2 headers shown in **Figure 48** connect the outputs of the signal conditioning circuits to the same headers (H1 and H2) on motherboard; the motherboard allows the RPM module to interface with the FlashCore-B. The H3 and H4 headers, also shown in **Figure 49**, provide the pins to connect the input signals to their respective signal conditioning circuitry. The PCB layout shown in **Figure 49** displays the layout of the motherboard, and **Figure 50** shows the PCB wiring schematic. The blue wire layer on the wiring schematic shown in **Figure 50** is the top copper layer and the green is the bottom copper layer. Please see source Schematic and PCB files on our project CD for more details.

IMPORTANT NOTE:

When programming the TERN FlashCore-B microcontroller, a jumper must be placed or removed between J2.1 and J2.3 header (On the actual FlashCore-B board). If the microcontroller is mounted on the motherboard, these two pins can be accessed from H2 on the motherboard. Placing a jumper between H2.7 and H2.8 allows setting up the microcontroller for debug mode or for standalone mode. Likewise if a module is placed on H1 and H2 (On the motherboard), these two pins (H2.7 and H2.8) still have to be accessible for the placement or removal of a jumper between them. Please see the TERN FlashCore-B(FB)^{TM} Technical Manual for more information on the FlashCore-B Programing Overview. The red square, shown in **Figure 49**, shows where a jumper should be placed on the Pothole Module PCB.

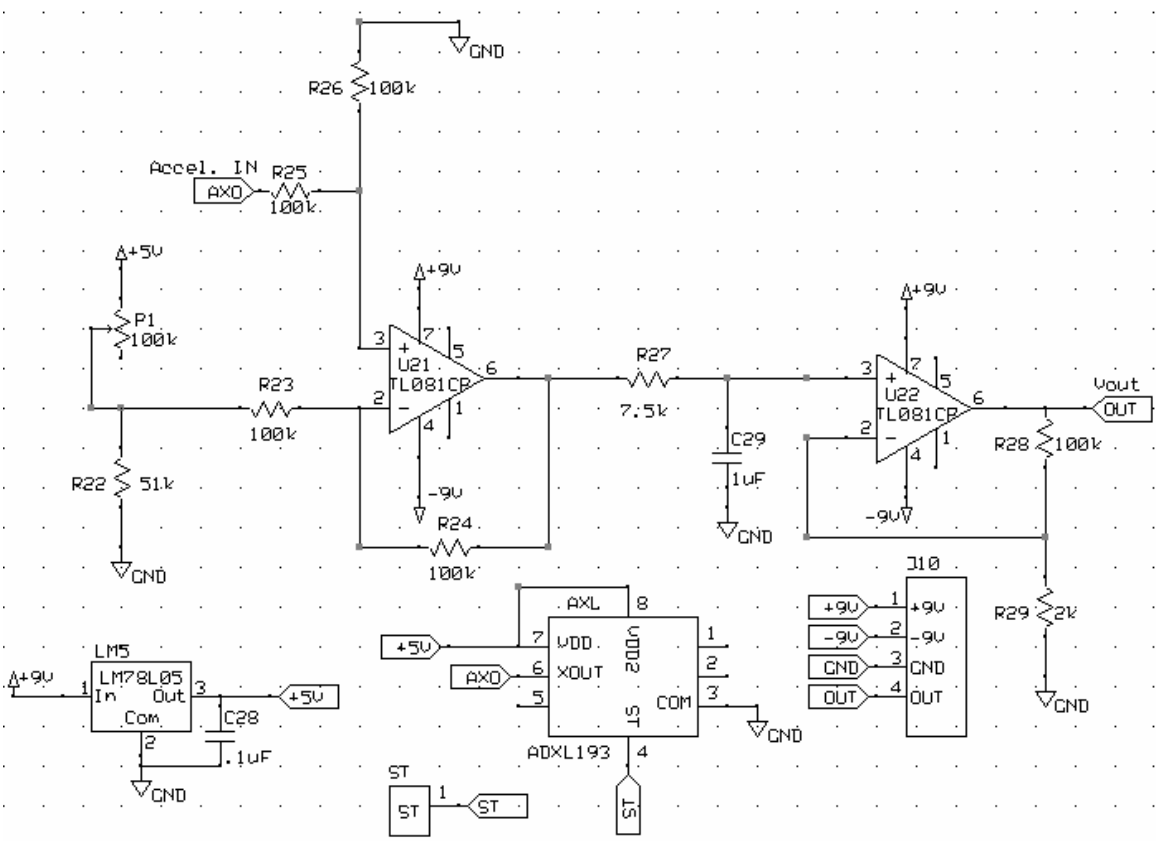

Figure 45: Accelerometer Module Schematic

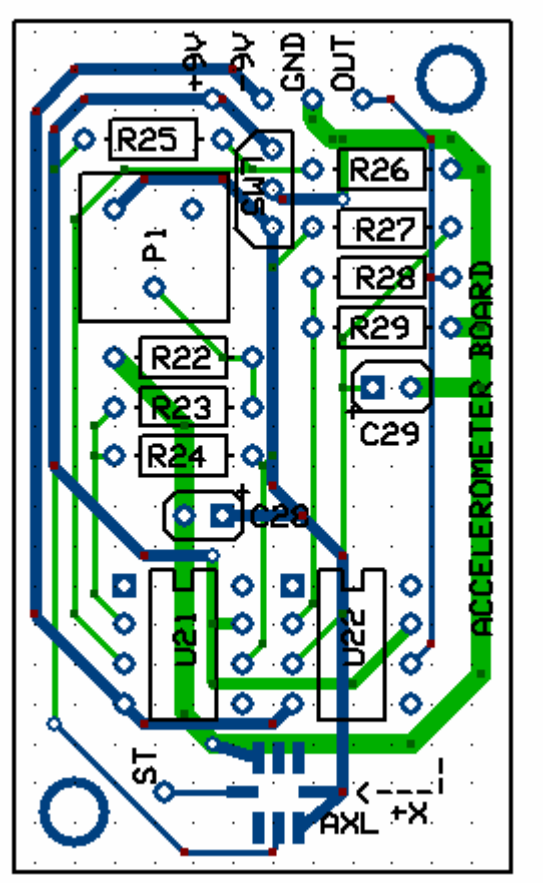

Figure 46: Accelerometer Module PCB Board

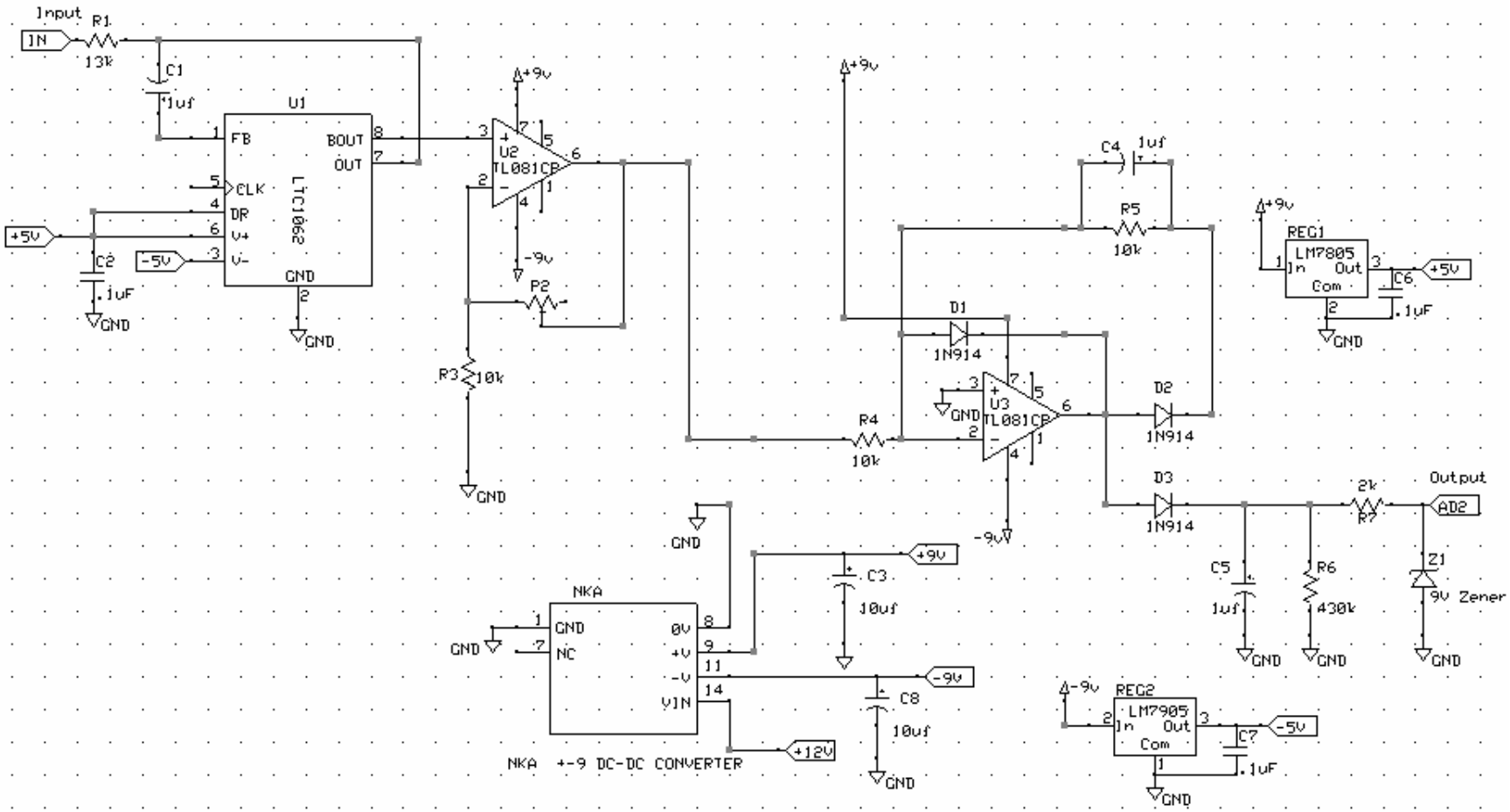

Figure 47: Pothole Module Schematic

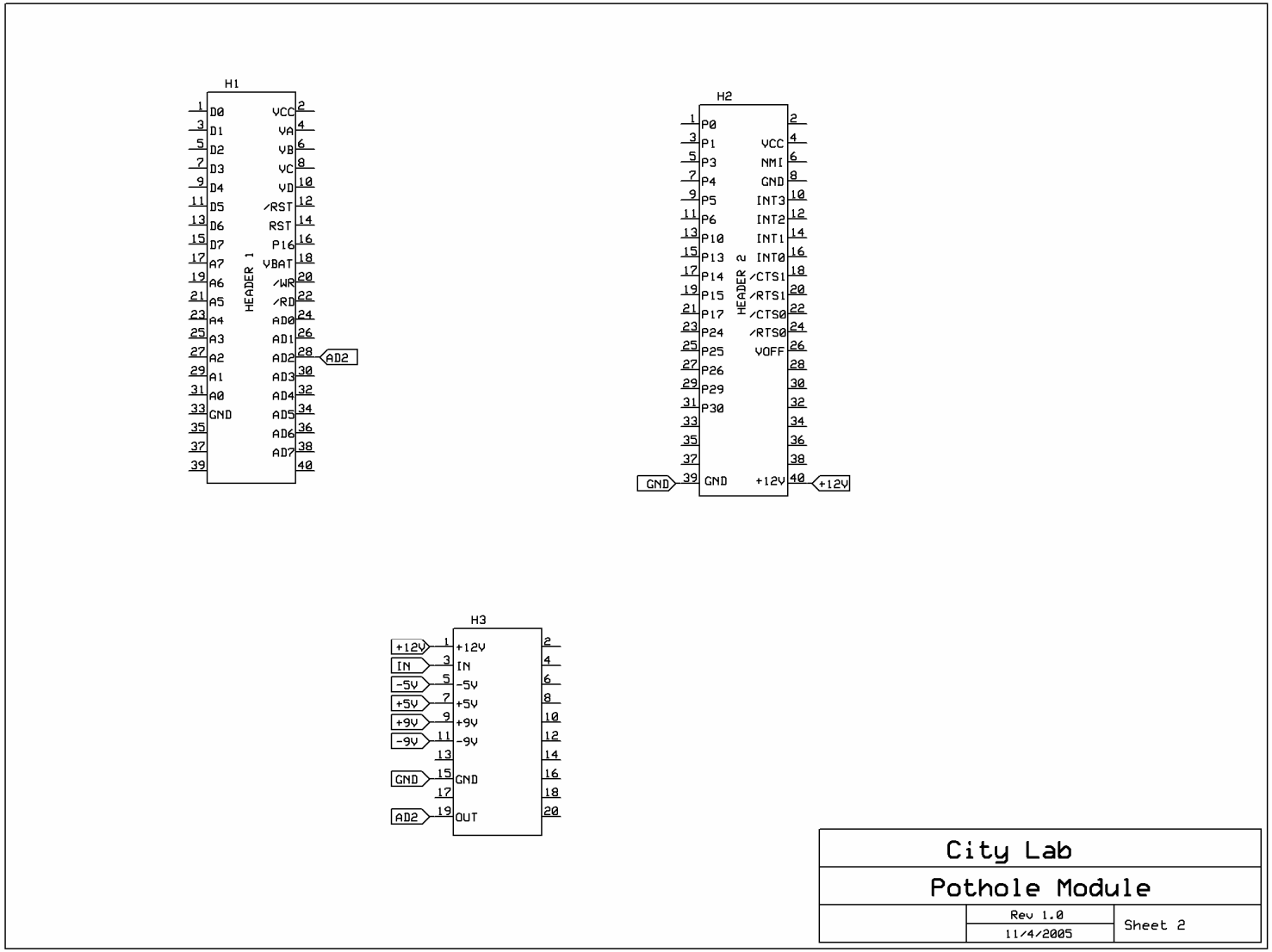

Figure 48: Pothole Module Header Connections

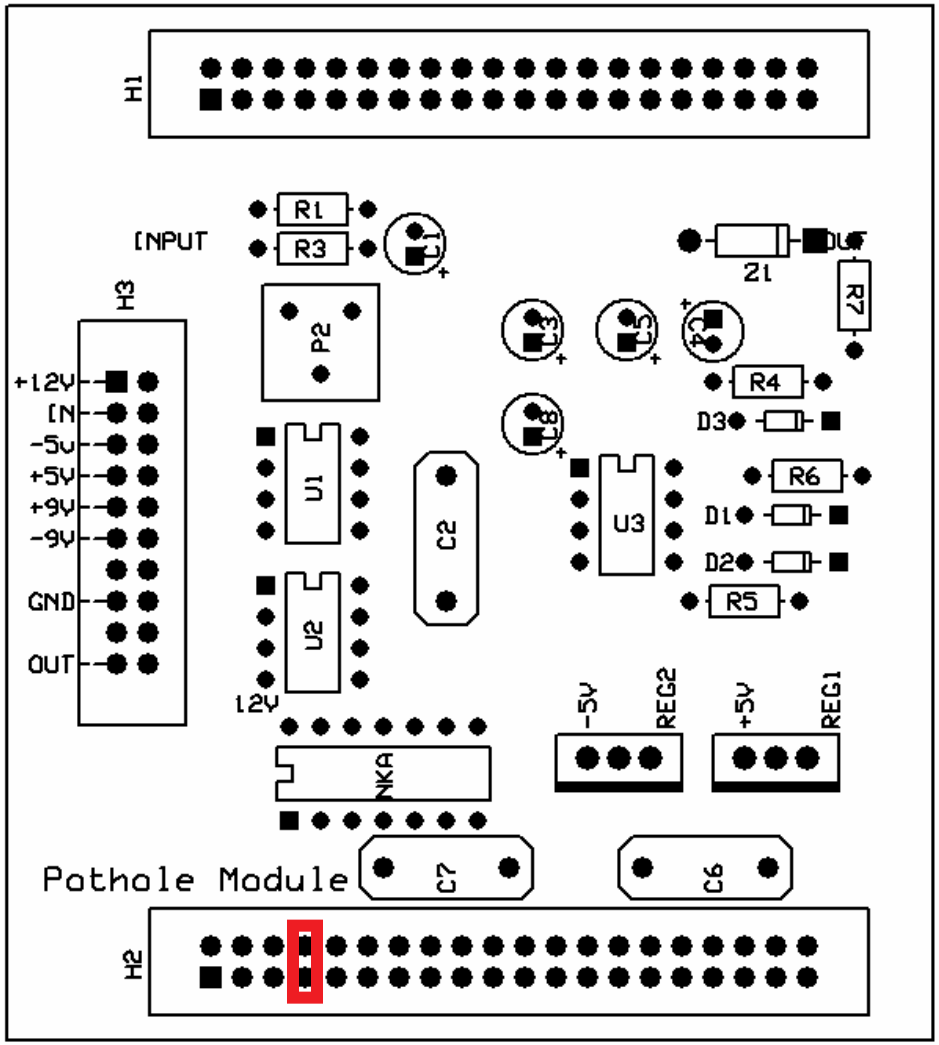

Figure 49: Pothole Module PCB Silkscreen Schematic

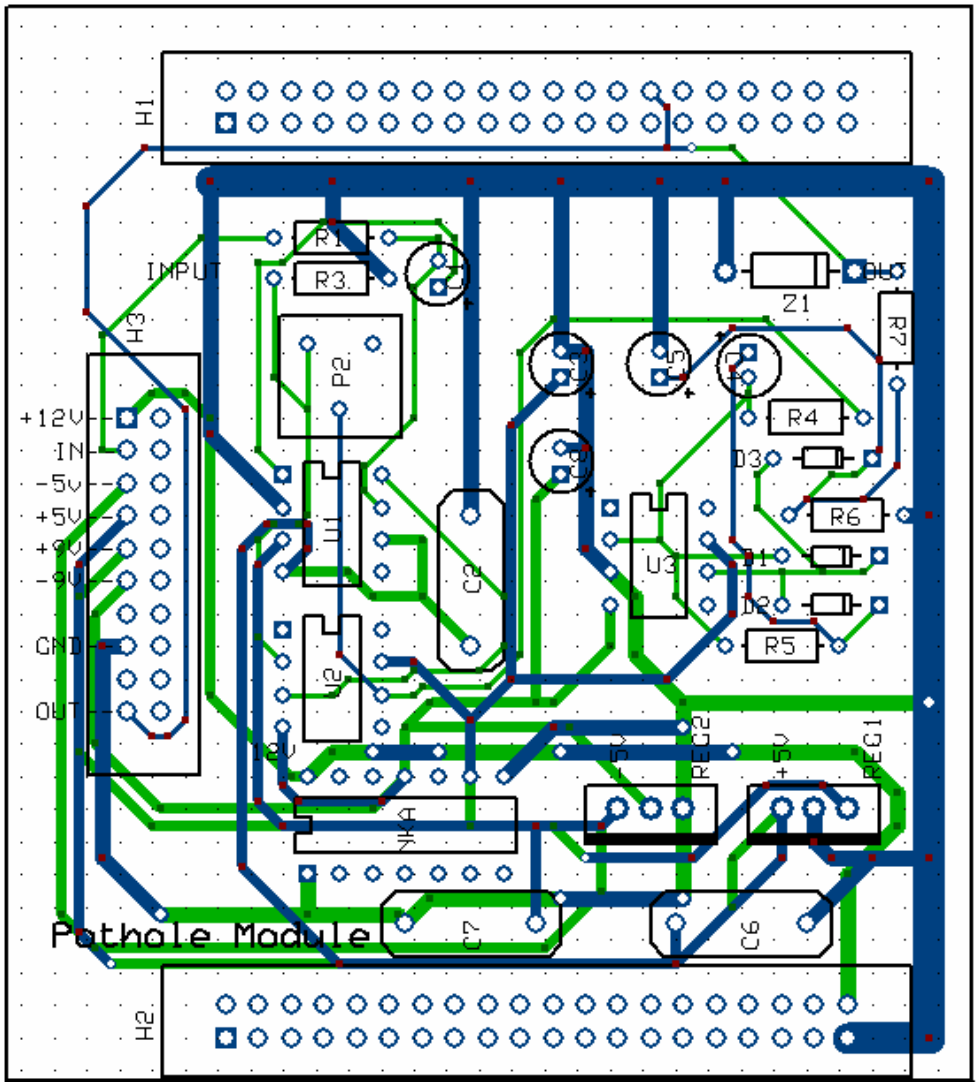

Figure 50: Pothole Module PCB Board

Appendix G: DGSCOPE MATLAB Decoder Code

The MATLAB code provided in this appendix allows for the decoding of files downloaded from the Metex Instruments DG SCOPE 20MHz Oscilloscope. To download files from the oscilloscope it requires use of their proprietary software and saving those files to a directory. Once a file has been downloaded onto a PC, place the ".DSO" file under the same directory as the ".M" file containing the code from the **MATLAB Code** section. Be sure to name the MATLAB file containing the code: "decode.m" and set the current working directory to the folder where "decode.m" is placed. The decoding function can be called in MATLAB by typing the following in the command window:

decode('FILENAME.DSO');

Once the code executes, both the signal waveform and the frequency spectrum of the waveform is shown. A Screen capture showing the command window is shown in **Figure 51**. Changing the views on the output waveforms can be modified using the plot parameters in the end of the MATLAB code.

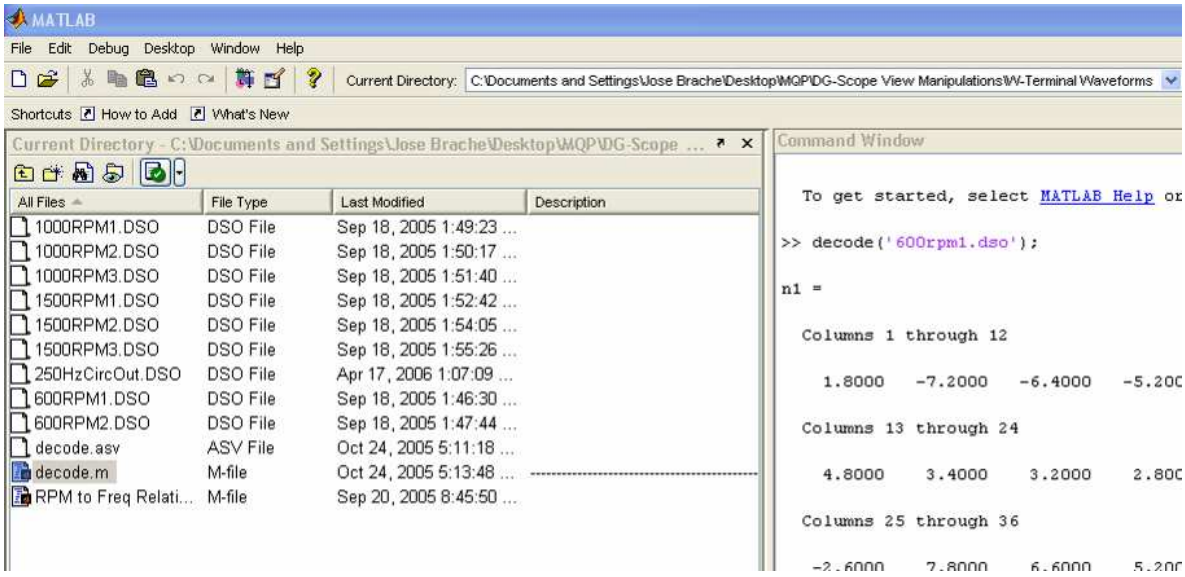

Figure 51: MATLAB Command Window

1. MATLAB Code

```
function [n1, n2, fs] = decode(file)
fid = fopen(file, 'r');
c = fread(fid, '*uint8', 'b');
fclose(fid);

%--------------------------------------------------------------------
% IDENTIFY AND SET THE SCALE PER DIVISION BEING USED
        % After playing around with all the possible voltage division
         % combinations, the HEX value for all the possible scales was
        % identified. The HEX value is stored in c(6) for CH1 and c(12) for CH2.
        % These two variables: ampopt and ampscale are used to set up the correct
        % scale in the loops where the array's n1 (CH1) and n2 (CH2) are created.
```

```
ampopt = [00 01 02 03 04 05 06 07 08];
ampscale = [1/4000 1/2000 1/1000 1/400 1/200 1/100 1/50 1/20 1/10];

        % 00    5mV/Div scalar = 1/4000;
        % 01    10mV/Div scalar = 1/2000;
        % 02    20mV/Div scalar = 1/1000;
        % 03    50mV/Div scalar = 1/400;
        % 04    100mV/Div scalar = 1/200;
        % 05    200mV/Div scalar = 1/100;
        % 06    500mV/Div scalar = 1/50;
        % 07    1V/Div scalar = 1/20;
        % 08    2V/Div scalar = 1/10;

%-------------------------------------------------------------------------
% SET UP ARRAY n WITH CORRECT DATA POINTS FROM THE HEX FILE
        % First checks if the mode being used includes channel 1 or channel 2 or
        % both and then checks the timing scale being used for both channels
        % accordingly to fit the array n with the correct data points. See notes
        % for time scale to undrestand how number of samples is different for
        % certain sampling rates.

%---------- EXCLUSIVELY FOR CH1
if (c(4) ~= 01);            % Check if Channel 1 is used
   i = 46;
   x = 1;
   if (03 < c(18)) & (c(18) < 20);          % Check what timing scale is used
      while ((i) <= 3884);                  % If timing scale is between values
         n1(x) = c(i);                      % in the scale check then all the
         n1 = double(n1);                   % data points are in this segment of
         if (n1(x) <= 80);                  % the file (CH1)
            n1(x) = abs(n1(x)-128);
         else
            n1(x) = -1*(n1(x)-128);
         end
         i = i+2;
         x = x+1;
      end
   else
      while ((i) <= 524);                   % If the timing scale is out of the
         n1(x) = c(i);                      % bounds set above then this is
         n1 = double(n1);                   % the segment where CH1 is stored
         if (n1(x) <= 80);
            n1(x) = abs(n1(x)-128);
         else
            n1(x) = -1*(n1(x)-128);
         end
         i = i+2;
         x = x+1;
      end
   end
```

```
% This while loop identifies the scale used for the channel and applies
% it to the respective array

   i = 1;
   while i <= 9;
      if (c(6) == ampopt(i));
         n1 = n1 .* ampscale(i);
         i = 9;
      end
      i = i+1;
   end
else
   n1 = 0
end

%---------- EXCLUSIVELY FOR CH2
if (c(4) ~= 00);            % Check if Channel 2 is used
   i = 3888;
   x = 1;
   if (03 < c(18)) & (c(18) < 20);                  % Check what timing scale is used
      while ((i) <= 7726);                          % If timing scale is between values
         n2(x) = c(i);                              % in the scale check then all the
         n2 = double(n2);                           % data points are in this segment of
         if (n2(x) <= 80);                          % the file (CH2)
            n2(x) = abs(n2(x)-128);
         else
            n2(x) = -1*(n2(x)-128);
         end
         i = i+2;
         x = x+1;
      end
   else
      while ((i) <= 4366);                  % If the timing scale is out of the
         n2(x) = c(i);                      % bounds set above then this is
         n2 = double(n2);                   % the segment where CH2 is stored
         if (n2(x) <= 80);
            n2(x) = abs(n2(x)-128);
         else
            n2(x) = -1*(n2(x)-128);
         end
         i = i+2;
         x = x+1;
      end
   end

% This while loop identifies the scale used for the channel and applies
% it to the respective array
   i = 1;
   while i <= 9;
      if (c(12) == ampopt(i));
         n2 = n2 .* ampscale(i);
```

```
            i = 9;
        end
        i = i+1;
    end
else
    n2 = 0;
end

%-----------------------------------------------------------------------
% IDENTIFY AND SET THE TIMING SCALE
        % An explanation for the numbers in timescale below:
        % DG-Scope uses 1920 samples to hold data sampled at a specific rate, I
        % figured out that, for the most part, it uses 96 time divisions to
        % display the entire signal of 1920 samples. The total amount of time
        % that the scope gathers data is defined as: (Number of
        % Divisions)*(SweepTime/Division)=Total Time. I used the logic that
        % after 1920 samples there would be 96 divisions, 96 divisions
        % multiplied by the SweepTime/Division yields the total SweepTime. As
        % you can see from the table below, this method is used to determine the
        % respective timing scale from the waveform. Since the file is in HEX it
        % took some time to determine where this timing scale is saved. That is
        % where timeopt comes in, timeopt holds the values for all the possible
        % timing scales DG-Scope uses. I use a while loop below to first
        % identify what is the timing scale and then link it with the actual
        % value of the timing scale.

% IMPORTANT NOTE
        % The time divisions below that have an asterisk were noted not to
        % follow this characteristic. These only use the first 240 samples to
        % display useful data.

timeopt = [00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17 18 19 20 21 22 23];
timescale = [
    12/240*50*10^(-9)           %00  *   50ns/Div timediv = 50*10^(-9);
    12/240*0.1*10^(-6)          %01  *   0.1us/Div timediv = 0.1*10^(-6);
    12/240*0.2*10^(-6)          %02  *   0.2us/Div timediv = 0.2*10^(-6);
    12/240*0.5*10^(-6)          %03  *   0.5us/Div timediv = 0.5*10^(-6);
    96/1920*1*10^(-6)           %04      1us/Div timediv = 1*10^(-6);
    96/1920*2*10^(-6)           %05      2us/Div timediv = 2*10^(-6);
    96/1920*5*10^(-6)           %06      5us/Div timediv = 5*10^(-6);
    96/1920*10*10^(-6)          %07      10us/Div timediv = 10*10^(-6);
    96/1920*20*10^(-6)          %08      20us/Div timediv = 20*10^(-6);
    96/1920*50*10^(-6)          %09      50us/Div timediv = 50*10^(-6);
    96/1920*0.1*10^(-3)         %10      0.1ms/Div timediv = 0.1*10^(-3);
    96/1920*0.2*10^(-3)         %11      0.2ms/Div timediv = 0.2*10^(-3);
    96/1920*0.5*10^(-3)         %12      0.5ms/Div timediv = 0.5*10^(-3);
    96/1920*1*10^(-3)           %13      1ms/Div timediv = 1*10^(-3);
    96/1920*2*10^(-3)           %14      2ms/Div timediv = 2*10^(-3);
    96/1920*5*10^(-3)           %15      5ms/Div timediv = 5*10^(-3);
    96/1920*10*10^(-3)          %16      10ms/Div timediv = 10*10^(-3);
    96/1920*20*10^(-3)          %17      20ms/Div timediv = 20*10^(-3);
```

```
    96/1920*50*10^(-3)          %18    50ms/Div timediv = 50*10^(-3);
    96/1920*0.1                 %19    0.1s/Div timediv = 0.1;
    12/240*0.2                  %20  *   0.2s/Div timediv = 0.2;
    12/240*0.5                  %21  *   0.5s/Div timediv = 0.5;
    12/240*1                    %22  *   1s/Div timediv = 1;
    12/240*2                    %23  *   2s/Div timediv = 2;
    ];

i = 1;
while i <= 24;
    if (c(18) == timeopt(i));
        timescalar = timescale(i);
        i = 24;
    end
    i = i+1;
end

format short;

fs = 1/timescalar;

%--------------------------------------------------------------------------
% FOR DISPLAYING THE SIGNAL AND FREQUENCY SPECTRUM FOR
% DIFFERENT MODES

%---------------- MODE: CH1
if (c(4) == 00)

    x = (1:length(n1)) .* timescalar;
    samples = length(n1);
    t1 = 0*samples * (1/fs);
    t2 = samples * (1/fs);

    % Sets the amplitude scale to about +-3/2 of the actual max amplitude for
    % CH1
    if (max(n1) >= abs(min(n1)));
        amp = max(n1) * 3/2;
    else
        amp = abs(min(n1)) * 3/2;
    end

    % ORIGINAL SIGNAL PLOT
    figure()
    plot(x, n1)
    grid on

    title(['Aquired signal in CH1 from file: ', file]);
    xlabel('Seconds')
    ylabel('Volts')
    axis([t1 t2 -amp amp])
```

```
    % FREQUENCY SPECTRUM PLOT
    fF1 = abs(fft(n1));
    df1 = fs/length(fF1);
    f1 = 0 : df1 :(length(fF1)-1) * df1;

    figure()
    plot(f1, fF1)
    grid on

    famp = max(fF1) * 3/2;
    axis([0 max(f1)/2 -(famp)/10 famp])
    title(['Magnitude Spectra of Signal in CH1 from file: ', file]);
    ylabel('|F|');
    xlabel('Frequency f (Hz)');

%---------------- MODE: CH2
elseif (c(4) == 01)

    x = (1:length(n2)) .* timescalar;
    samples = length(n2);
    t1 = 0*samples * (1/fs);
    t2 = samples * (1/fs);

    % Sets the amplitude scale to about +-3/2 of the actual max amplitude for
    % CH2
    if (max(n2) >= abs(min(n2)))
       amp = max(n2) * 3/2;
    else
       amp = abs(min(n2)) * 3/2;
    end

    % ORIGINAL SIGNAL PLOT
    figure()
    plot(x, n2)
    grid on

    title(['Aquired signal in CH2 from file: ', file]);
    xlabel('Seconds')
    ylabel('Volts')
    axis([t1 t2 -amp amp])

    % FREQUENCY SPECTRUM PLOT
    fF2 = abs(fft(n2));
    df2 = fs/length(fF2);
    f2 = 0 : df2 : (length(fF2)-1) * df2;

    figure()
    plot(f2, fF2)
    grid on

    famp = max(fF2) * 3/2;
```

```
    axis([0 max(f2)/2 -(famp)/10 famp])
    title(['Magnitude Spectra of Signal in CH2 from file: ', file]);
    ylabel('|F|');
    xlabel('Frequency f (Hz)');

%---------------- MODE: DUAL
elseif (c(4) == 02)

    x = (1:length(n1)) .* timescalar;

    % PLOT
    figure()
    plot(x, n1, x, n2)
    grid on

    title(['Aquired signal in DUAL Mode from file: ', file]);
    xlabel('Seconds')
    ylabel('Volts')

%---------------- MODE: ADD
elseif (c(4) == 03)

    addmode = n1 + n2;
    x = (1:length(addmode)) .* timescalar;
    samples = length(addmode);
    t1 = 0*samples * (1/fs);
    t2 = samples * (1/fs);

    % Sets the amplitude scale to about +-3/2 of the actual max amplitude for
    % CH1 + CH2
    if (max(addmode) >= abs(min(addmode)))
        amp = max(addmode) * 3/2;
    else
        amp = abs(min(addmode)) * 3/2
    end

    % PLOT
    figure()
    plot(x, addmode)
    grid on

    title(['Aquired signal in ADD Mode from file: ', file]);
    xlabel('Seconds')
    ylabel('Volts')
    axis([t1 t2 -amp amp])

%---------------- MODE: SUB
elseif (c(4) == 04)

    submode = n1 - n2;
    x = (1:length(submode)) .* timescalar;
```

```
    samples = length(submode);
    t1 = 0*samples * (1/fs);
    t2 = samples * (1/fs);

    % Sets the amplitude scale to about +-3/2 of the actual max amplitude for
    % the CH1 - CH2
    if (max(submode) >= abs(min(submode)))
        amp = max(submode) * 3/2;
    else
        amp = abs(min(submode)) * 3/2
    end

    % PLOT
    figure()
    plot(x, submode)
    grid on

    title(['Aquired signal in SUB Mode from file: ', file]);
    xlabel('Seconds')
    ylabel('Volts')
    axis([t1 t2 -amp amp])

%---------------- MODE: X-Y
else
    % Sets the time scale to about +-3/2 of the actual max amplitude for
    % CH1 (X)
    if (max(n1) >= abs(min(n1)));
        amp1 = max(n1) * 3/2;
    else
        amp1 = abs(min(n1)) * 3/2;
    end
    % Sets the amplitude scale to about +-3/2 of the actual max amplitude for
    % CH2 (Y)
    if (max(n2)>= abs(min(n2)))
        amp2 = max(n2) * 3/2;
    else
        amp2 = abs(min(n2)) * 3/2;
    end

    % PLOT
    figure()
    plot(n1, n2, '.')
    grid on

    title(['Aquired signal in X-Y Mode from file: ', file]);
    xlabel('CH1 (Volts)')
    ylabel('CH2 (Volts)')
    axis([-amp1 amp1 -amp2 amp2])
end
return
```