

Project Number: ECE-FJL-CLDA

MAPPING TURBULENT DISCHARGES IN THE VENETIAN CANALS

A major Qualifying Project Report:

Submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

In partial fulfillment of the requirements for the

Degree of Bachelor of Science

By

---

Steven Christopher Marshall

---

Francis John Carino

Date: June 1, 2007

Approved:

---

Professor Fred J. Looft, Major Advisor

1. mapping
2. turbulent
3. Venetian

---

Professor Fabio Carrera, Co-Advisor

This report represents the work of one or more WPI students  
submitted to the faculty as evidence of completion of a degree requirement.  
WPI routinely publishes these reports on its web site without editorial or peer review.

# Table of Contents

<b>1 INTRODUCTION .....</b>	<b>9</b>
1.1 PROBLEM STATEMENT .....	9
1.2 SUMMARY .....	10
<b>2 BACKGROUND.....</b>	<b>11</b>
2.1 TRANSPORTATION IN VENICE .....	11
2.2 CANAL WALL DAMAGE .....	12
2.2.1 Causes of Canal Wall Damage .....	12
2.2.2 Effects of Canal Wall Damage [1].....	13
2.2.3 Methods of Preservation of Canal Walls .....	15
2.3 THE AGE OLD DEBATE .....	16
2.4 THE MOTO ONDOSO INDEX .....	16
2.5 MAPPING TURBULENCE IN THE CANALS OF VENICE .....	16
2.6 MAPPING UNDERWATER TURBULENCE IN VENICE .....	17
2.5 GPS.....	17
2.5.1.1 WAAS.....	19
2.5.2 GIS.....	19
2.6 SUMMARY .....	20
<b>3 PROBLEM STATEMENT AND METHODS.....</b>	<b>21</b>
3.1 PROJECT GOAL .....	21
3.2 OBJECTIVES .....	21
3.3 SYSTEM REQUIREMENTS .....	21
3.4 METHODS .....	22
<b>4 SYSTEM DESIGN AND TESTING.....</b>	<b>23</b>
4.1 OVERALL SYSTEM DESIGN .....	23
4.1.1 Overall Hardware Design.....	23
4.1.2 Hardware Design and Testing .....	24
4.1.2.1 Power module.....	24
4.1.2.2 Microcontroller Module .....	25
4.1.2.3 GPS Module.....	30
4.1.2.4 RPM Module.....	35
4.1.2.5 Display Module.....	39
4.1.2.6 Data Storage Module.....	42
4.1.2.7 Accelerometer .....	44
4.1.3 OVERALL SOFTWARE DESIGN AND TESTING .....	47
4.1.3.1 GIS .....	48
4.2 SUMMARY .....	49
<b>5 SYSTEM INTEGRATION AND TESTING.....</b>	<b>50</b>
5.1 PROTOBOARD TESTING .....	50
5.2 PCB.....	52
5.3 SUMMARY .....	53
<b>6 SUMMARY AND CONCLUSIONS.....</b>	<b>54</b>
6.1 SUMMARY OF DESIGN, AND RESULTS .....	54
6.2 FUTURE RECOMMENDATIONS .....	55
6.3 CONCLUSIONS.....	55

## Table of Figures

FIGURE 2.1 TRAFFIC DISTRIBUTION .....	12
FIGURE 2.2: RESIDENTIAL CANAL DAMAGE.....	13
FIGURE 2.3: MAP OF VENICE, ITALY .....	14
FIGURE 2.4: CA FOSCARI AFTER REPAIRS .....	15
FIGURE 2.5: A DIAGRAM OF TRILATERATION.....	18
FIGURE 2.6: SAMPLE GIS .....	20
FIGURE 4.1 OVERALL SYSTEM BLOCK DIAGRAM.....	23
FIGURE 4.2: POWER MODULE SCHEMATIC .....	24
FIGURE 4.3: POWER MODULE TESTING .....	25
FIGURE 4.4: MSP430F169 BLOCK DIAGRAM.....	28
FIGURE 4.5: MICROCONTROLLER MODULE CONNECTIONS .....	29
FIGURE 4.6: MSP430F169 W/HEADER BOARD.....	30
FIGURE 4.7: GPS MODULE BLOCK DIAGRAM .....	33
FIGURE 4.8: PG-31 .....	33
FIGURE 4.9: PG-31 PIN-OUT TABLE .....	34
FIGURE 4.10: GPS MODULE TESTING .....	35
FIGURE 4.11: INDUCTIVE PICKUP OUTPUT. ....	36
FIGURE 4.12: W-TERMINAL SIGNAL .....	37
FIGURE 4.13: RPM MODULE BLOCK DIAGRAM .....	38
FIGURE 4.14: RPM MODULE SCHEMATIC .....	38
FIGURE 4.15: RPM CONDITIONING CIRCUIT TESTING USING FUNCTION GENERATOR.....	39
FIGURE 4.16: LCD MODULE BLOCK DIAGRAM.....	40
FIGURE 4.17: CFAH1602CYHJP FRONT VIEW .....	40
FIGURE 4.18: LCD PIN-OUT TABLE .....	41
FIGURE 4.19: LCD TESTING.....	42
FIGURE 4.20: M25P32.....	42
FIGURE 4.21: SPI MEMORY DEVICE PIN-OUT .....	43
FIGURE 4.22: MEMORY MODULE BLOCK DIAGRAM.....	43
FIGURE 4.23: SPI MEMORY TESTING .....	44
FIGURE 4.24: ADXL330 W/DEVELOPMENT BOARD .....	45
FIGURE 4.25: AXL PIN-OUT .....	45
FIGURE 4.26: ACCELEROMETER CIRCUIT SCHEMATIC.....	46
FIGURE 4.27: ACCELEROMETER TESTING 1 .....	46
FIGURE 4.28: ACCELEROMETER TESTING 2 .....	47
FIGURE 4.29: ACCELEROMETER TESTING 3 .....	47
FIGURE 4.30: BASIC SOFTWARE FLOWCHART .....	48
FIGURE 4.31: GIS TESTING .....	49
FIGURE 5.1: SYSTEM TESTING(GASOLINE ENGINE) 1 .....	50
FIGURE 5.2: SYSTEM TESTING(GASOLINE ENGINE) 2.....	51

## **Table of Tables**

TABLE 1: MICROCONTROLLER VALUE ANALYSIS. ....	28
TABLE 2: GPS VALUE ANALYSIS. ....	32

## Table of Equations

EQUATION 1: PULLEY RATIO CALCULATION[4] .....	37
EQUATION 2: DIESEL ENGINE RPM CALCULATION[4] .....	37

## Appendices

APPENDIX A: EMBEDDED C CODE .....	58
APPENDIX B: GOOGLE EARTH CODE.....	110
APPENDIX C: PCB SCHEMATICS .....	118

## **Acknowledgements**

Fred Looft – for his guidance and technical expertise

Fabio Carrera – for his knowledge of Venice and creative input

Carlton Stedman III and Andrew Demarco (Pothole Team) – for their help in the design process

## **Abstract**

The goal of this project was to create a device to automatically measure and record turbulence discharged from motorboats in Venetian canals, and create a map of those discharges over time. The system was based on processing signals from: the w-terminal on a diesel engines alternator or an inductive pickup placed on a gasoline engines spark plug. These signals are processed, under software control, to calculate actual engine RPM. Engine RPM readings are 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 can be powered from the boat's battery 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 engine RPM data to a physical location on a user viewable city map. The prototype is intended to be placed in motorboats that roam about the Venetian lagoon in order to map the location and magnitude of underwater turbulence. This system can aid the city of Venice, Italy in identifying areas that suffer from constant underwater turbulence and by correctly correlating canal wall damage to underwater turbulence, canal repair crews can be released more efficiently.

# 1 Introduction

Venice, Italy, was founded in 421 by Roman refugees<sup>1</sup>. Between the ninth and twelfth centuries, Venice flourished, becoming a center of trade between Western Europe and the rest of the known world. During the next few centuries, Venice was the most prosperous city in the world. After a decline following the Renaissance, Venice became an extremely influential city in trade, art, architecture, and literature during a period known as the Settecento(1700s), when it became one of the most elegant cities of Europe. In 1797, Venice finally lost its independence for the last time, when it was conquered by Napoleon Bonaparte. In 1866, it finally became part of Italy.

One of the most unique cities in the world, Venice is world famous for its canals. The city is actually an archipelago of 118 islands containing roughly 150 canals and 400 bridges, built on a lagoon. Although there is transportation to and from Venice on land, its internal infrastructure still remains mostly aquatic. The only land based transportation is by foot.

Traditionally, the Venetian mode of transportation in the water was the gondola. However, these boats are now only used for tourism and special occasions such as wedding, funerals, and other ceremonies. Most Venetians currently travel by motorized waterbuses, or vaporetti, which have set routes along the major canals and between the city's major islands, similar to the way a public bus route works. In addition to the many vaporetti, there are many private boats in use in the canals. With the population of 271,000<sup>2</sup>, and roughly seven million tourists visiting Venice per year<sup>3</sup>, the public transportation system in the canals is heavily used.

The advent of the motorboat in the late 19<sup>th</sup> century allowed for faster transportation through water, but at the same time introduced increased canal damage in Venice. Motorboats are unable to simply brake in the water, and therefore in order to stop, the engine must drive the propeller in the reverse direction. When a boat does so, a tremendous amount of energy is produced and transferred to the water in the form of surface and underwater pressure waves. These waves eventually crash into the walls of the canals and cause significant damage over a long period of time. Methods of controlling such damage are currently being researched.

## 1.1 Problem Statement

The purpose of this project was to design a device to measure the energy a

---

1 <http://www.veniceword.com/vehistory.html>

2 <http://en.wikipedia.org/wiki/Venice>

3 [www.american.edu/TED/VENICE.HTM](http://www.american.edu/TED/VENICE.HTM)

motorboat puts into the water, and record both the amount of energy and the geographic coordinates of the turbulence. This device needed to be able to record for a long period of time, and therefore its data recording and power use had to be minimized. A module was also designed to transfer this stored data to a data post processing system for display and analysis.

## **1.2 Summary**

The purpose of this report is to document our senior design project, “Mapping Turbulent Discharges in the Venetian Canals”. The remainder of the document is broken into five parts, with a full bibliography and set of appendices.

The second section of the report is devoted to the background of the project. This section will give the reader an understanding of the history of the project. It will also present relevant information and technologies, and discuss the previous attempts to solve the problem, as well as the improvements we have made to a previous project’s overall design.

The third section of the report will be the problem statement and system specifications. This will open with a global statement of the problem to address, as well as the goals the project has met. It includes tasks for assessing the objectives of the project, as well as the requirements of each objective, task, and goal. The methods by which the project was designed will also be included in this chapter. This details the approach used to develop the system, and its design requirements. The methods of section three lay the foundation for sections four and five, the system design and results.

Section four, the system design and testing section presents the overall design of the system, and the overall software flow of the system. It also presents the system detailed design presents each block of the system. It contains extensive and detailed block diagrams, and the specific functionality of each module, as well as notes and design trade studies.

Section five, the system integration and testing, presents system tests, as well as analysis.

The sixth section contains the summary and conclusion, where the system is analyzed compared to the original objectives and goals. It also presents ideas for future work to be completed based on our results, and the conclusions of our work.

## 2 Background

With the introduction of the motorboat, transportation became easier in Venice, Italy. Today, motorboats are used in almost every aspect of transportation involving the canals. Unfortunately, due to this enormous increase in boat traffic, the canal walls are suffering major damage<sup>4</sup>. In this section, we will discuss the causes and effects of canal wall damage, as well as some previous attempts to solve these issues.

### ***2.1 Transportation in Venice***

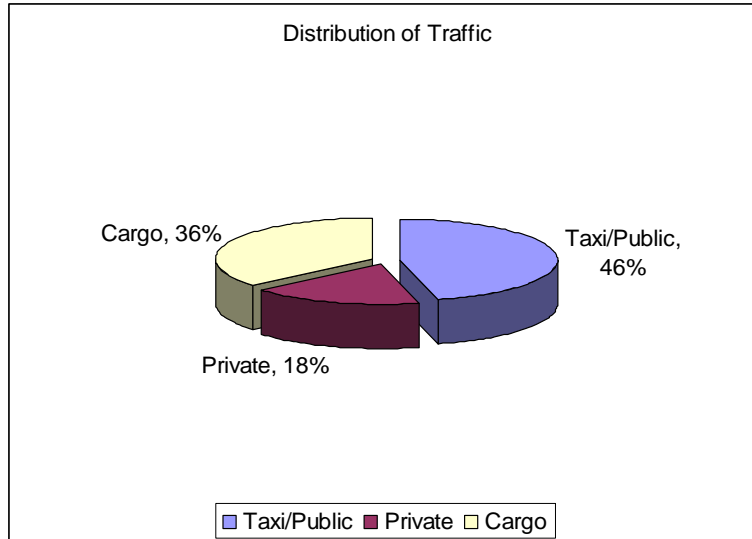
With little land based infrastructure, Venice is almost entirely dependent on the canals for transportation. Being an archipelago, the use of boats in transportation has been necessary since the founding of the city, many centuries ago. The canals are used for transporting goods, public services, and public transportation since long before the advent of the motorboat. With the canals dividing the city into more than 100 small islands, there is no possible alternative means for transportation<sup>5</sup>. The city will continue to make use of the waterways, especially with an ever increasing tourism industry placing heavy demands on public transportation.

Around the 1950's, motorboats took over as the primary mode of transportation in Venice. Tasks that used to be done by rowboat have now been motorized. Only a very small number of gondolas, the traditional rowboat for the Venetian canals, exist, and are only used for scenic tours, and in limited sections of the city. The traffic distribution of Venice can be seen in Figure 2.1. The majority of the traffic is from cargo boats, taxis, and public transportation.

---

4 <http://paxinaqua.provincia.venezia.it/chisiamo.html>

5 <http://en.wikipedia.org/wiki/Venice>



**Figure 2.1 Traffic Distribution<sup>6</sup>**

## **2.2 Canal Wall Damage**

There are several factors that contribute to canal wall damage. These factors are evident by the deterioration of the walls themselves, and in the many restorative efforts that have taken place over the years.

### **2.2.1 Causes of Canal Wall Damage**

There are several factors that cause canal wall damage. These factors include damage done to the walls by natural factors, external events, and boats.

The rising sea levels and sinking of the Venetian land mass is causing damage to the canal walls. The rising sea level, as well as extraction of water from underneath the city, is causing the land on which Venice sits, to sink into the lagoon that contains the city. The water levels have risen approximately 23 centimeters since 1897<sup>7</sup>. Scientists say that increased sea levels can lead to flooding and increased damage<sup>8</sup>.

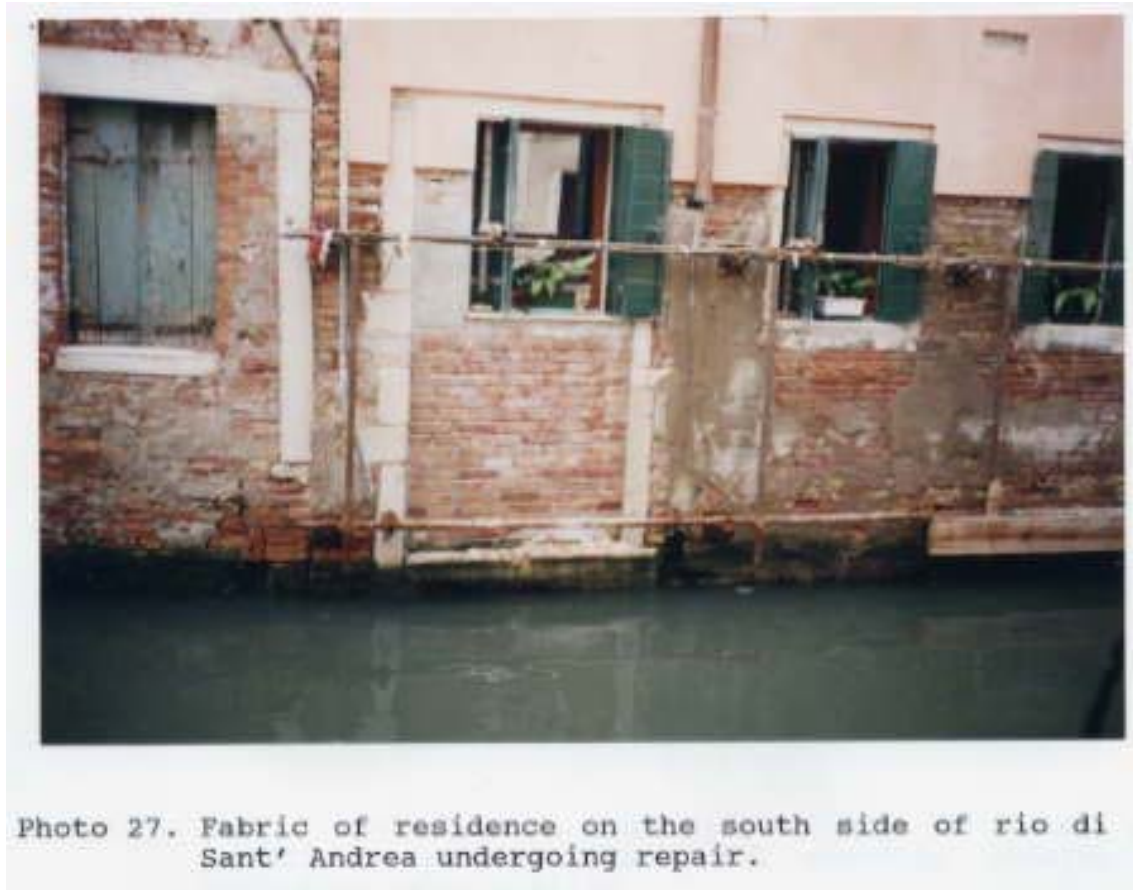
The sewage system in Venice can also cause damage to the canals. The sewage enters the canals underwater, through a system of pipes. Silt deposits in front of sewage pipes can cause clogs in the sewage system. When clogging occurs, the sewage backs up and causes the pipes to burst. The sewerage then seeps into the surrounding mortar. This can contribute to the weakening integrity of the canal walls, as seen in figure 2.2. In the bottom of the picture, the canal walls have taken large amounts of damage due to a

6 Camera, Fabio and Caniato, Giovanni, "Venezia la Citta Dei Rii"

7 <http://www.agiweb.org/geotimes/mar04/geophen.html>

8 [http://news.mongabay.com/2007/0201-sea\\_level.html](http://news.mongabay.com/2007/0201-sea_level.html)

sewerage system problem.



**Figure 2.2: Residential Canal Damage<sup>9</sup>**

Another significant cause of damage, however, comes from motorboats. There are several methods by which boats cause damage. Boat wakes can cause weakening in the mortar that holds the bricks together. In addition to the damage above water, the underwater turbulence caused by engine propellers can weaken the canal wall foundations. The vibrations caused by underwater turbulence take any damage that has already occurred and amplify it, widening cracks, and eroding the walls beneath the surface. Furthermore, boats colliding with walls can cause cracks in the wall. When these damaged foundations are exposed to wakes the damage done to the wall will increase. If no repairs are made the rate at which damage is done increases as time goes on.

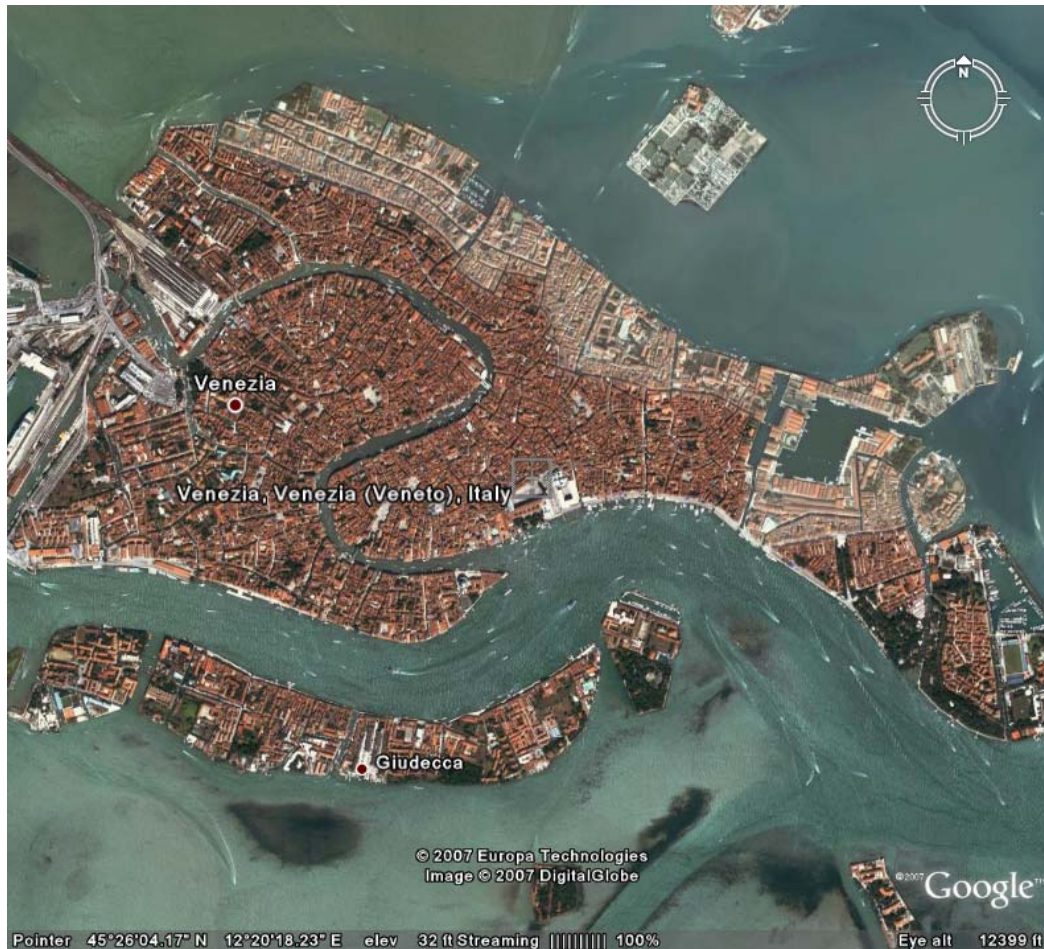
### **2.2.2 Effects of Canal Wall Damage [1]**

The following is a map of the canals of Venice, Italy. This is to give the reader an overall idea of where the canals are in relationship to the city. Smaller maps are provided

---

<sup>9</sup> <http://maya.csuhayward.edu/archaeoplanet/Venice/Ph27.html>

to show the reader where each specific canal is.



**Figure 2.3: Map of Venice, Italy<sup>10</sup>**

In 1990, a large hole was found behind one of the walls of the Rio Novo canal. This seemingly small hole on the surface was actually the outward manifestation of structural damage inside the wall. With its supporting structures on the verge of collapse, the canal was closed and boats were rerouted to the Rio Cerris canal. Due to the increased traffic, irreparable damage was done to the Rio Cerris, and within a short span of only two years, a building near the Ponte Rosso collapsed, due to the waves caused by motorboats. After an astonishing 7 billion euros of work, the Rio Novo reopened in 1997.

Later, in 1995, the Ca' Foscari building, shown in figure 2.4 was in risk of collapse and closed for repairs. It is one of the many buildings on the waterfront of the Grand Canal that constantly are exposed to moto ondoso.

---

10 [http://www.veniceonline.it/Maps/Map2\\_VeniceOnLine.jpg](http://www.veniceonline.it/Maps/Map2_VeniceOnLine.jpg)



**Figure 2.4: Ca Foscari after repairs**

The Galeazze Canal has been plagued by erosion problems. In 1996 it was repaired for the first time. Within two years of these repairs, metal sheeting was installed along the canal walls to stop erosion. In 1999, the entire canal was closed from boat traffic because of the tremendous amount of repairs that were needed. In addition, the Rio della Maddalen, San Moise, San Lorenzo, and the Rio di Noal have all undergone repairs in the last 10 years.

The effects of moto ondosso can be seen in other parts of the Venetian lagoon. City Hall was once closed for fear of damage cause by moto ondosso. The island housing the cemetery is in an area that has a high volume of traffic, and therefore takes a great amount of damage. One billion euros were spent fixing and reinforcing the Codussi chapel, which has taken extensive damage due to erosion caused by moto ondosso.

### **2.2.3 Methods of Preservation of Canal Walls**

Currently, there are two methods being used to preserve the canal walls. Preventative methods are being used to stop ongoing damage, as well as restorative methods being used to fix damaged walls. Preventative methods include the enforcement of boat registration and speed limits. The combination of the two allows police to regulate traffic to permitted areas, and more fully enforce speeding laws. Some of the restoration techniques include reforming the bottom of the canal where damage has occurred, sealing off crevices, and rebuilding the walls themselves.

## **2.3 The Age Old Debate**

Two groups are currently debating the most significant cause of erosion in the Venetian canals. The group Pax in Aqua supports the theory that moto ondosso is the major cause of canal deterioration. Insula claims that numerous other factors are more significant, and that if canals were reconstructed using modern methods and better materials, moto ondosso would be an insignificant factor.

Pax in Aqua<sup>11</sup> considers the turbulence caused by engines, as well as boat wakes to be the leading cause of canal wall destruction. They are working to raise public awareness, and have succeeded in creating regulations governing boat wake generation in the canal.

Insula<sup>12</sup> believes that canal wall damage should be independent of moto ondosso. They claim that if certain construction methods were employed, moto ondosso would be negligible. By using concrete, and making walls that are slanted outward, they believe they can minimize the effects of turbulence.

## **2.4 The Moto Ondoso Index**

A team of three students from WPI studied the energy dispelled into the canal system using a shore based visual approximation [2]. Data was collected for boats that fit a certain class type, and the turbulence was indexed based on the correlation between boat type and annual canal traffic. However, this was only based on a visual approximation. The study only allowed for speculative data, and therefore the turbulence in each case cannot absolutely be stated. They measured the energy output by several different sized boats at different speeds and times of day. They concluded that several factors increase the amount of energy output by the boats, such as increased speed, and increased traffic. Their study showed that there was a need to get concrete measurements for the energy output by boats, and laid the foundation for the first study done in this area, by a group of seniors a few years later.

## **2.5 Mapping Turbulence in the Canals of Venice**

A WPI project team of three students created a device that could aid researchers in mapping turbulent discharges in the canals [3]. Their design was a prototype that was used to map underwater turbulence. Their device used a modular approach to meet their objectives.

---

11 <http://paxinaqua.provincia.venezia.it/chisiamo.htm>

12 <http://www.insula.org/>

- Design and prototype a non-invasive approach to determining energy discharge
- Design and prototype a system to collect and store energy and GPS data
- Integrate the two previous systems (energy discharge and data storage) into one unit that can be portable, durable, and can operate for a week without much human input
- Design and implement a software program to analyze the collected data

They took the device to Venice in the summer of 2003 to test the device, and found that there were problems with the design. The most major problem they found was that most boats in Venice use diesel engines and that meant that new features would have to be added to the project.

## ***2.6 Mapping Underwater Turbulence in Venice***

In 2005-2006, a WPI project team of four students completed their senior year capstone designed an improved version of the device and approach of the previous group [4]. This was an effort to further develop capabilities map underwater turbulence; the design of the system outlined in this report improves upon functionality, power consumption, and ease of use of this device. They were able to design a module to measure the RPM of a diesel engine, using a signal from the alternator as a way of making the measurement. This group also met its objectives which were to:

- Develop a method for determining engine RPM from both diesel and gasoline engines
- Develop a prototype for an embedded system that can store engine RPM readings and GPS coordinates on external memory, and provide the user with relevant information
- Write the software to bring all the components of the device together.

Some suggestions they had for improving upon their design included additional testing of the device's ability to measure RPM from gasoline engines, field testing for the device, and further improvement of the Geographical information systems (GIS) software for data post processing.

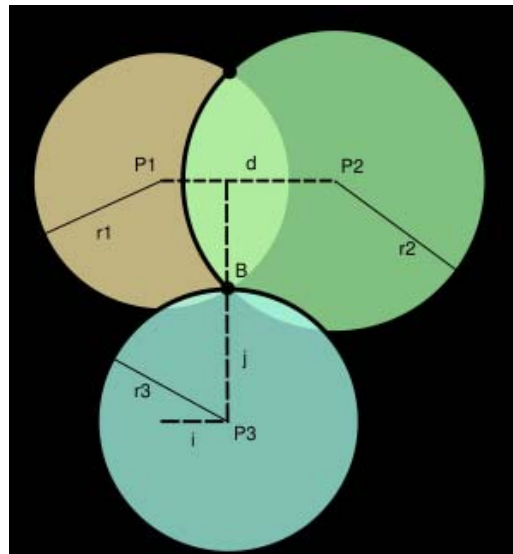
## **2.5 GPS<sup>13</sup>**

Determining the boat's position is an essential component of this project. Therefore, researching different navigation systems was imperative. Global Navigation Satellite System (GNSS) is a standard generic term for satellite systems that provide geospatial positioning. A GNSS allows for small receivers to determine location using time

---

<sup>13</sup> [http://en.wikipedia.org/wiki/Global\\_Positioning\\_System](http://en.wikipedia.org/wiki/Global_Positioning_System)

signals transmitted along a line of sight by radio from satellites. Currently, the United States NAVSTAR Global Positioning System (GPS) is the only fully operational GNSS. (GPS) is a fully-functional satellite navigation system. Utilizing more than two dozen satellites in medium earth orbit, this system allows users to determine their location, speed and direction. It measures the time delay between transmission and reception to determine the distance to the satellite. When the distance between it and 4 or more satellites is known, the device can determine its location using a process called trilateration<sup>14</sup>. Using the two dimensional model shown in figure 2.6 trilateration will be explained. If the receiver were at point B, the location of B could be determined using satellites placed at points P1, P2, and P3. Measuring the distance between P1 and B would give a circle of radius  $r_1$  that B could lie on. Next, by measuring the distance between P2 and B,  $r_2$ , the intersection of the two circles would yield two points that B could be located at. Finally, by adding a point at a third known location, P3, the final location of the receiver could be known. The distance between B and P3,  $r_3$ , could be calculated, and the intersection of these three circles would yield one point on a two dimensional plane. In three dimensions, this technique still works, only the circles are replaced by spheres, and one additional known point is needed to yield one exact point.



**Figure 2.5: A diagram of trilateration.**<sup>15</sup>

GPS is used in many practical applications, such as military targeting, automobile and aircraft navigation, weather systems, and surveying and mapping. GPS, however, has imperfections like any other system. Some sources of error include atmospheric affects, such as humidity, clock errors, such as drift, and electromagnetic radiation

14 <http://en.wikipedia.org/wiki/Trilateration>

15 <http://upload.wikimedia.org/wikipedia/en/thumb/6/6e/Trilateration.svg/300px-Trilateration.svg.png>

interference. Several techniques have been developed to increase accuracy, such as integrating external information into the calculation process.

### **2.5.1.1 WAAS**

One technique used to integrate external information into the GPS calculation process is the Wide Area Augmentation System (WAAS). WAAS is a network of fixed ground based reference stations. These stations broadcast the difference between their known fixed position, and the satellite systems in space. This system compensates for errors that occur due to atmospheric distortions, and is capable of bringing the accuracy of the receiver from 15m down to 1-5m<sup>16</sup>. The accuracy of this method degrades with distance from the reference station, but studies suggest that this degradation is minor, or about .22m per 100 km<sup>17</sup>. WAAS is not a perfect system, and does have limitations. Due to the location of most satellites, WAAS is not available for locations north of 71.4 N latitude or south of 71.4 S latitude. In this project however that will not be an issue. Most receivers are WAAS capable, and due to the density of canals in Venice, it is important that the receiver we choose is capable of this technology.

### **2.5.2 GIS**

A geographical information system (GIS) is a system used in capturing, storing, analyzing, and managing data which are spatially referenced to the earth. Using a GIS, a user is able to analyze spatial information, edit data, and present the results of all these operations. In the case of this project, GIS was used to track the location of turbulent discharges, and then add them to a map, to create a visual presentation of the collected data. Some examples of GIS are data modeling, topological modeling, networks, map overlay and geostatistics. Below is an example of a GIS that shows the cases of cholera in an 1854 epidemic in London. Each point represents an individual case of cholera. By analyzing the distribution, the source of the disease, a contaminated water pump, was discovered.

---

16 [http://en.wikipedia.org/wiki/Differential\\_GPS](http://en.wikipedia.org/wiki/Differential_GPS)

17 <http://www.navcen.uscg.gov/pubs/frp2001/FRS2001.pdf>



Figure 2.6: Sample GIS<sup>18</sup>

## 2.6 Summary

The background of this document has presented many issues related to transportation in Venice, destruction of canal walls, debates about the cause of said destruction, and previous studies made to find the cause of canal wall damage. This section presented information about the current state of Venice, and how the Venetian canals are affected by motorboat turbulence. It also has presented previous studies that try to correlate the effect motorboats have on the damage caused to the canals and their foundations. Finally, it gives background information on some of the relevant technologies associated with the project.

18 <http://upload.wikimedia.org/wikipedia/en/c/c7/Snow-cholera-map.jpg>

## **3 Problem Statement and Methods**

This chapter presents the specific goals of the project, and the project requirements which need to be met. Also, the purpose of this chapter is to outline the methods used by the group to accomplish our objectives. We had to determine our system requirements in order to choose several components.

### **3.1 Project Goal**

The goal of this project is to create a device to automatically measure and record turbulence discharged from motorboats in Venetian canals, and creates a map of those discharges over time.

### **3.2 Objectives**

In order to realize our project goal, our team addressed several objectives. Specific objectives for this project were as follows:

1. Familiarize ourselves with related previous design project.
2. Develop a prototype for an embedded system that can collect engine RPM readings, and GPS coordinates, and provide data post processing.
3. Write and debug software for controlling the system.
4. Demonstrate the use of GIS software for data post processing.

### **3.3 System Requirements**

Before developing a system to solve this problem, its requirements had to be determined. The system requirements were as follows:

1. The system must read RPM from both gasoline and diesel engines
2. The signal output by the engine must be conditioned in order to be sent to the microprocessor
3. The system must determine instances of changes in RPM that are greater than 25% per second
4. Instances must be marked with location and acceleration data for the boat
5. RPM, GPS, and accelerometer readings must be stored for at least one week
6. The system must be powered by the boat.
7. A GIS needs to be developed that can display the appropriate data
8. The system needs to have a method to upload data to the GIS

The system needs to be compatible with both gasoline and diesel engines.

Previous studies have shown that a great number of boats in the Venetian Canals have diesel engines. A method for determining RPM must be developed for both types of engine. The signal output by the engine must be conditioned by circuitry so that it can be understood by the microprocessor. The microprocessor unit will do all of the calculations for RPM and needs to be able to properly interpret the signal output by the boat. The data that we are interested in is instances of change in RPM. The code developed will poll for RPM every second. We decided that 25% shift in RPM would create enough turbulence. These instances must be marked with location and acceleration data. It is important to know where the turbulent discharges are occurring, and in what direction the boat is moving. This device should be automated to record these labeled spikes in RPM for at least a week. With enough memory to operate for a week, less maintenance will be needed, and the unit can simply be placed in the boat and left for a week. Rather than rely on batteries to power the device, the boat battery will be used to power the device. This will save the operator wasted time and money from depleted batteries. Lastly is the GIS system. The GIS needs to be developed to display the data taken from the device on a map. This will create an easy method for displaying and analyzing the data. The device needs to have a method to upload the data to the GIS, otherwise the data becomes useless.

### **3.4 Methods**

The process used by our team to accomplish the project can be seen in the flow of this report. First, we developed a problem statement, an overall goal to guide our project (section 1.1). Next, background research was conducted to familiarize ourselves with not only previous turbulent discharge monitoring methods, but also relevant technologies to this project (section 2). Based on our problem statement, and the work done by previous project groups, we were able to create a list of project objectives, as well as system requirements (section 3). These specifications acted as a guide for our entire project, affecting every decision we made. From here we worked to design the device. It was designed in a modular manner, and modeled using software (section 4). The design process was multifaceted. First, each module's specifications and requirements were created. Next, several options were chosen for each of the modules, and subjected to value analysis to determine the best choice for the given application. Next, each component was tested to verify functionality. Next, the system was put together on a breadboard for system testing (section 5). A printed circuit board was designed and laid out, but with time restrictions we were unable to populate and test the printed circuit board.

## 4 System Design and Testing

This section presents the overall design approach implemented in order to complete our project goal. Both the hardware and software are shown at the highest level of operation. This section also presents the overall design approach implemented in order to complete our project goal. The system is broken down in a modular fashion in order to represent how the system functions as a whole. The system is broken down into the following modules: the microcontroller module, the GPS module, the RPM module, the display module, data storage, and the accelerometer. The function of each module will be discussed in detail throughout this section.

### 4.1 Overall System Design

This section presents the overall system design, from both a hardware and software perspective.

#### 4.1.1 Overall Hardware Design

The diagram shown below is the overall system design. The microcontroller sends commands to the GPS, display and data storage modules. Additionally, it sends data to the data storage module. It also receives data from the RPM and accelerometer modules to be processed.

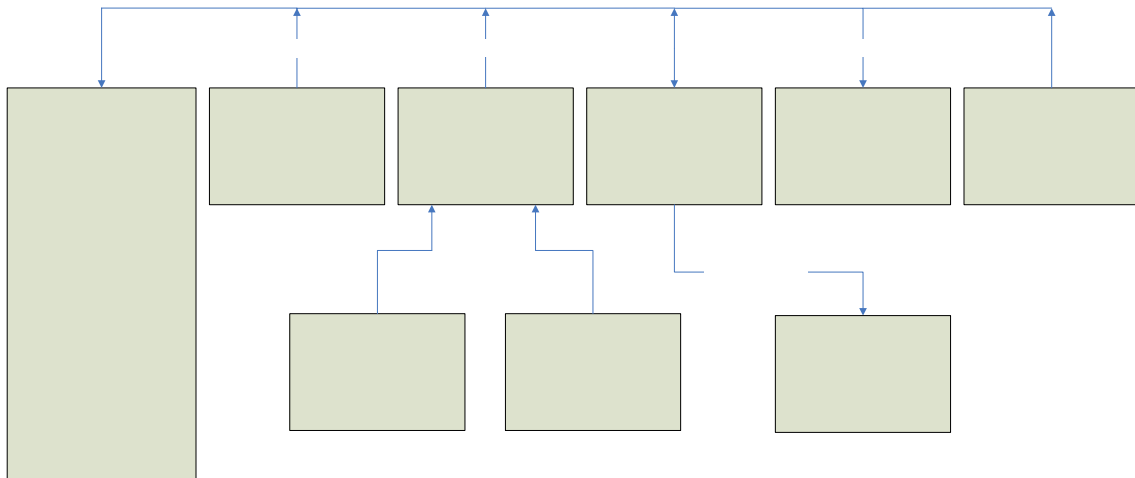
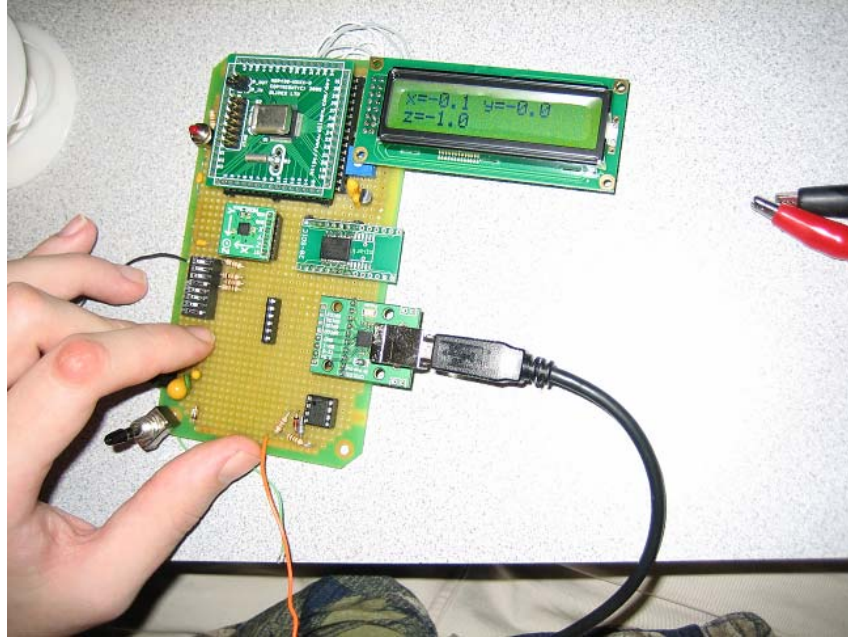


Figure 4.1 Overall System Block Diagram





**Figure 4.3: Power Module Testing**

#### **4.1.2.2 Microcontroller Module**

The microcontroller module is a microcontroller that we purchased after performing a value analysis on different microcontrollers that could perform the function. We decided that it would be best if we completed a thorough value analysis of microcontrollers, as it is the central processing unit of the device. We began our analysis by doing research on microcontrollers, using the previous MQP as a starting point. We decided on the TERN FlashCore-B and the MSP430. A brief description of the TERN FlashCore\_B from the technical manual is quoted below,

“Measuring only 2.1" by 2.35", the FlashCore-B (\$69, qty 100) is a variant of the 186-generation programmable core, now improved to also support data-logging applications. With the integrated CompactFlash drive (using cards up to 2 GB in size), the FlashCore-B can add an entire new dimension to your embedded application. With the FB, your equipment becomes truly stand-alone. With FAT filesystem support (and 16-bit ADC/DAC), accessing your data is as easy as plugging the card directly into a CompactFlash drive on your PC/PDA.”

A brief description of the MSP430F169 from the technical manual is quoted below;

“The Texas Instruments MSP430 family of ultralow power microcontrollers consist of several devices featuring different sets of peripherals targeted for various applications. The architecture, combined with five low power modes is optimized to achieve extended battery life in portable measurement applications. The device features a powerful 16-bit RISC CPU, 16-bit registers, and constant generators that attribute to maximum code efficiency. The digitally controlled oscillator (DCO) allows wake-up from low-power modes to active mode in less than 6 $\mu$ s. The MSP430x15x/16x/161x series are microcontroller configurations with two built-in 16-bit timers, a fast 12-bit A/D converter, dual 12-bit D/A converter, one or two universal serial synchronous/asynchronous communication interfaces (USART), I2C, DMA, and 48 I/O

pins. In addition, the MSP430x161x series offers extended RAM addressing for memory-intensive applications and large C-stack requirements.“

We created a list of requirements, which were each assigned a value. The following list shows each criteria and its weight:

- Size – 3
- Ports – 7
- Expandability – 5
- Development – 6
- Language – 4
- I/O – 9

Using this list we created a table, critiquing each microcontroller based on the requirements. Each criterion was given a value, five being the best and one being the worst. The following shows how each criterion is broken down:

Size: How large is the microcontroller

- Very Small – 5
- Small – 4
- Medium – 3
- Large – 2
- Very Large – 1

Ports: # 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 # 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

After creating criteria for value analysis, we judged each of the two microcontrollers accordingly. The technical specifications of the microcontrollers are as follows:

**Flashcore B:**

- 2.1x2.35x0.7 inches
- 2 RS-232 serial ports
- 8 ch. 16-bit ADC (ADS8344, 20 KHz)

**MSP430f169:**

- 1.9x2.15x0.5 inches
- 2 SPI ports

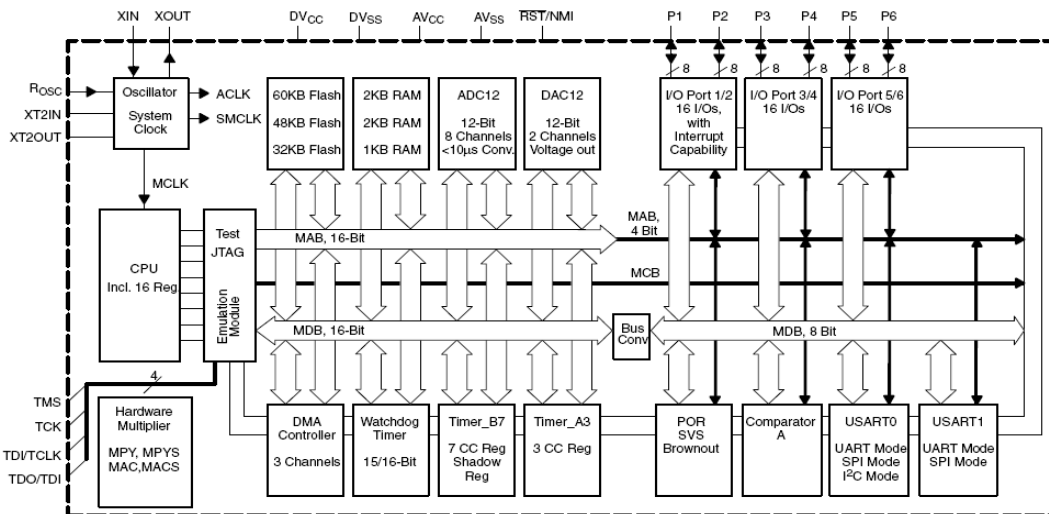
- Real time clock, battery
- 8 ch. 12-bit ADC

Once going through each individual criterion and giving the microcontroller their values, we came up with Table 1 **Error! Reference source not found.**

			TERN		MSP	
		Weight	Value point	Total	Value point	Total
1	Size	3	3	9	5	15
2	Ports	8	4	32	3	24
3	Expandability	5	2	10	5	25
4	Development	6	3	18	4	24
5	Language	4	4	16	5	20
6	I/O	9	4	36	5	45
	Total			121		153

**Table 1: Microcontroller Value Analysis.**

After the value analysis, the MSP430 is a better choice for our design. The functional description can be found in detail at the Texas Instruments website<sup>19</sup>. A functional block diagram of the microcontroller can be found below in figure 6.1.



**Figure 4.4: MSP430f169 Block Diagram<sup>20</sup>**

A brief description of the MSP430F169 from the technical manual is quoted below;

<sup>19</sup> <http://www.ti.com>

<sup>20</sup> <http://focus.ti.com/lit/ds/symlink/msp430f169.pdf>

“The Texas Instruments MSP430 family of ultralow power microcontrollers consist of several devices featuring different sets of peripherals targeted for various applications. The architecture, combined with five low power modes is optimized to achieve extended battery life in portable measurement applications. The device features a powerful 16-bit RISC CPU, 16-bit registers, and constant generators that attribute to maximum code efficiency. The digitally controlled oscillator (DCO) allows wake-up from low-power modes to active mode in less than 6 $\mu$ s. The MSP430x15x/16x/161x series are microcontroller configurations with two built-in 16-bit timers, a fast 12-bit A/D converter, dual 12-bit D/A converter, one or two universal serial synchronous/asynchronous communication interfaces (USART), I2C, DMA, and 48 I/O pins. In addition, the MSP430x161x series offers extended RAM addressing for memory-intensive applications and large C-stack requirements.”

A picture of the MSP430F169, set on a header board, is shown in Figure 6.2.

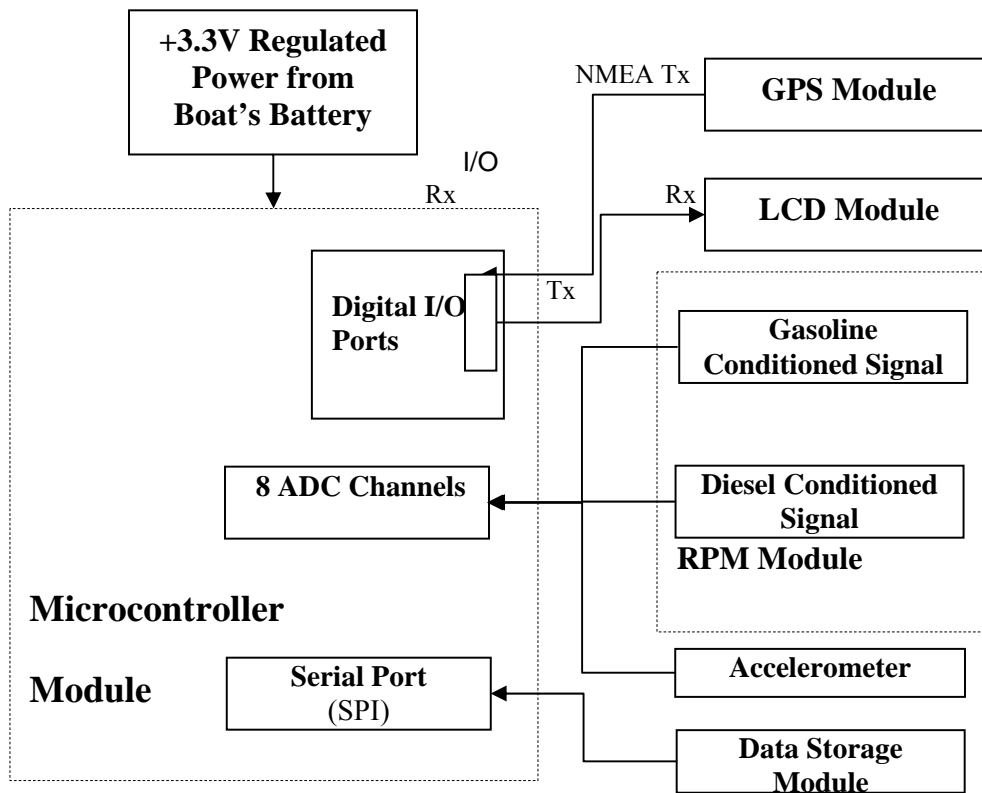


Figure 4.5: Microcontroller Module Connections



**Figure 4.6: MSP430F169 w/header board<sup>21</sup>**

The microcontroller is connected to every other module explained in this section (GPS, RPM, display, data storage, accelerometer). A more detailed diagram showing the connections between the microcontroller module and the other modules can be seen below in figure 6.1. The diagram shows how the various modules connect to the elements of the microcontroller. The LCD, display, RPM, GPS and accelerometer all connect to digital I/O ports on the microcontroller. The data storage module connects using serial peripheral interface (SPI). Power is provided to the microcontroller by an external +3.3V regulator placed on the printed circuit board (PCB).

#### **4.1.2.3 GPS Module**

The GPS module of our system, seen in figure 4.7, consists of a GPS receiver that connects to the microcontroller through digital I/O and is powered by an external +3.3V regulator on the PCB. Similar to the microprocessor module, we conducted value analysis on GPS units. We chose the Lassen IQ GPS receiver and the PG-31. We again created a list of requirements, which were each assigned a weight. The following list shows each criteria and its value:

- Size – 3
- Acquisition Time – 7
- Power consumption – 3
- Output Strings – 8
- WAAS capability – 9

Using this list we created a table, critiquing each microcontroller based on the

---

<sup>21</sup> [http://www.taylec.co.uk/acatalog/taylec\\_MSP149HBoth.jpg](http://www.taylec.co.uk/acatalog/taylec_MSP149HBoth.jpg)

requirements. Each criterion was given a value, five being the best and one being the worst. The following shows how each criterion is broken down:

Size: How large is the GPS unit

Very Small – 5

Small – 4

Medium – 3

Large – 2

Very Large – 1

Acquisition Time(Cold): Time to acquire a signal(seconds)

35 – 5

40 – 4

45 – 3

50 – 2

55 – 1

Power Consumption: How much power is consumed at typical operation voltage(3.3V)(mA)

25 – 5

50 – 4

75 – 3

100 – 2

125 or higher – 1

Output strings: Outputs strings with appropriate data

Everything needed plus additional information – 5

Everything needed – 4

Partial Information – 3

Little Information – 2

No relevant information – 1

WAAS capability: The ease of programming it

Yes – 5

No – 1

**Lassen IQ:**

- 42 mm x 50.5 mm x 13.8 mm
- 42 second acquisition time for warm start
- Less than 27mA @ 3.3V during typical operation
- NMEA compatible
- No WAAS

**PG-31:**

- 50 mm x 55.5 mm x 15.6 mm
- 35 second acquisition time for warm start
- Less than 35 mA @ 3.3V during typical operation
- NMEA compatible
- WAAS capable

Once going through each individual criterion and giving the GPS modules their values, we came up with **Error! Reference source not found.2**.

			Lassen		PG-31	
		Weight	Value point	Total	Value point	Total
1	Size	3	5	15	4	12
2	Acquisition Time	7	3	21	4	28
3	Power	3	5	15	4	12
4	Strings	8	5	40	5	40
5	WAAS	9	1	9	5	45
	Total			100		137

**Table 2: GPS Value Analysis.**

The GPS receiver chosen after value analysis was the Laipec Technology Inc<sup>22</sup>. PG-31, shown in figure 4.8. An external antenna can be mounted on the vehicle. Once correctly set up, the GPS receiver acquires satellites and outputs strings that are compliant with the National Marine Electronics Association (NMEA) data protocol. These strings contain information such as the date, time, speed, longitude, and latitude. The GPS receiver can be given commands to output certain specific strings.

---

22 <http://www.laipec.com>

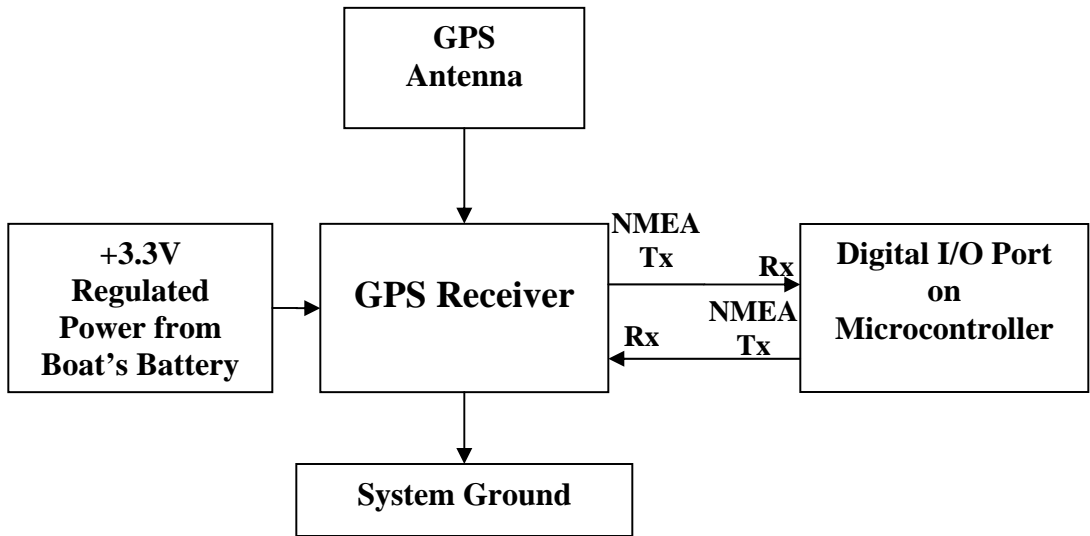


Figure 4.7: GPS Module Block Diagram

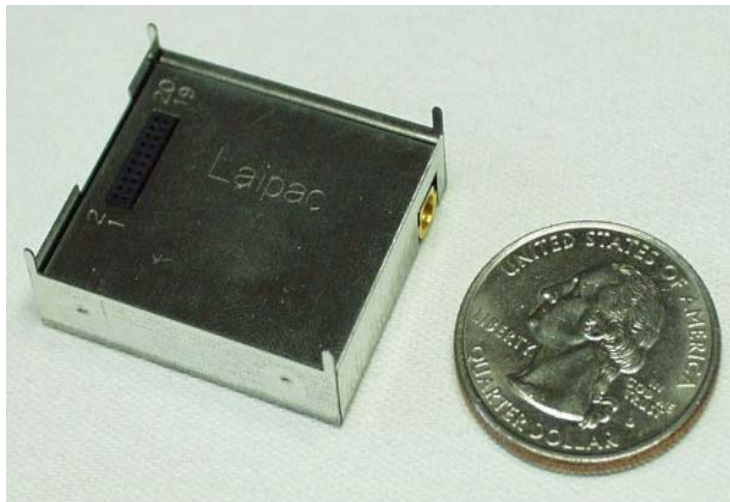


Figure 4.8: PG-31<sup>23</sup>

There are four communication lines, three of which are connected to digital I/O ports on the microcontroller. RXA, the main receiver line, is used to send commands to the GPS from the microcontroller. RXB is used to enable the WAAS functionality, which increases precision from approximately twenty five meters down to one to five

<sup>23</sup> <http://www.sparkfun.com/commerce/images/PG-31-1.jpg>

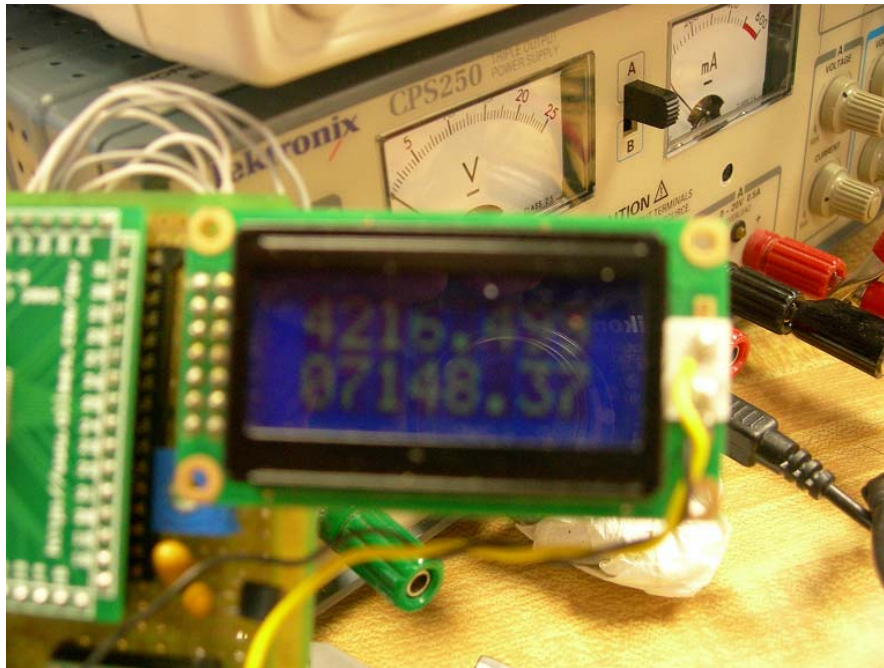
meters. TXA, the main output line, sends data out at an adjustable baud rate, in this case 4800. The remaining pins are unused. The pin-out description of the receiver is shown below, in Figure 4.9.

PIN	Name	Description
1	VCC	3.3V +/- 10% DC Power Input
2	TXA	Host Serial Data Output A
3	RXA	Host Serial Data Input A
4	TXB	Aux. Serial Data Output B
5	RXB	Aux. Serial Data Input B (DGPS)
6	TIMEMARK	1PPS Time Mark Output
7	BAT	Battery Backup Power Input
8	GPIOA	General Purpose Input/Output
9	RESET	Reset, Active Low
10	RESERVED	Reserved
11	GND	Ground
12	BOOTSEL	Internal/External Boot selective
13	GPIOB	General Purpose Input/Output
14	GPIOC	General Purpose Input/Output
15	GPIOD	General Purpose Input/Output
16	GPIOE	General Purpose Input/Output
17	GPIOF	General Purpose Input/Output
18	GPIOG	General Purpose Input/Output
19	GPIOH	General Purpose Input/Output
20	GND	Ground

**Figure 4.9: PG-31 Pin-out Table<sup>24</sup>**

The GPS unit was tested for functionality. The GPS module is given the initialization command, and then told to take a reading. The reading is then displayed on the LCD. As shown in figure 4.10, the GPS readings taken are accurate.

<sup>24</sup> <http://www.sparkfun.com/datasheets/GPS/PG31-Spec.pdf>



**Figure 4.10: GPS Module Testing**

#### **4.1.2.4 RPM Module**

The purpose of the RPM module is to condition the signals from either a diesel engine or a gasoline engine. The RPM module is powered by the +3.3V regulator. For diesel engines, the signal can be obtained by placing leads on the alternator w-terminal. For gasoline engines, a signal from an inductive pickup is used. The basic operation of this module for both diesel and gasoline engines is to condition the signal produced by the engine, so that a clean signal can be sent to the microcontroller. The RPM module is connected to the microcontroller using Digital I/O. Using software interrupts, the frequency of the incoming signal can be calculated.

For gasoline engines, the method chosen for measuring engine RPM was measuring gasoline engine RPM directly from spark plugs using inductive pickups[4]. The research conducted showed the output waveform of a gasoline engine to resemble the waveform shown below, in figure 3.1.

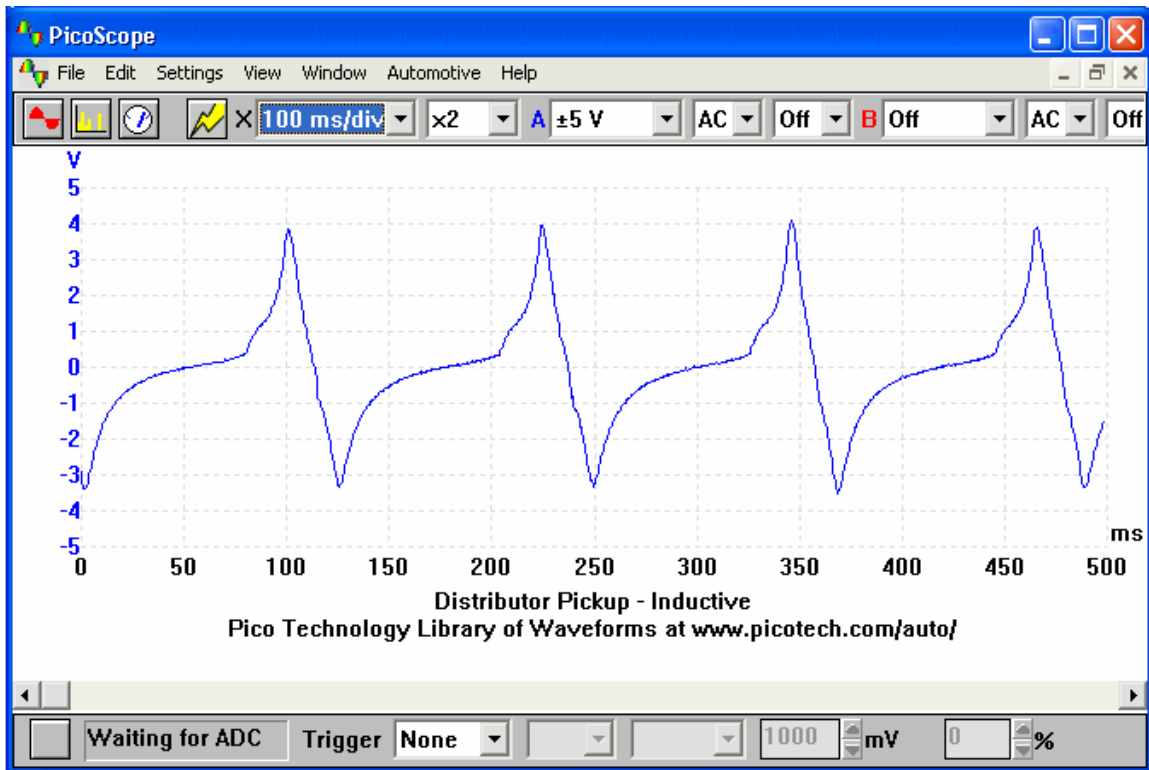
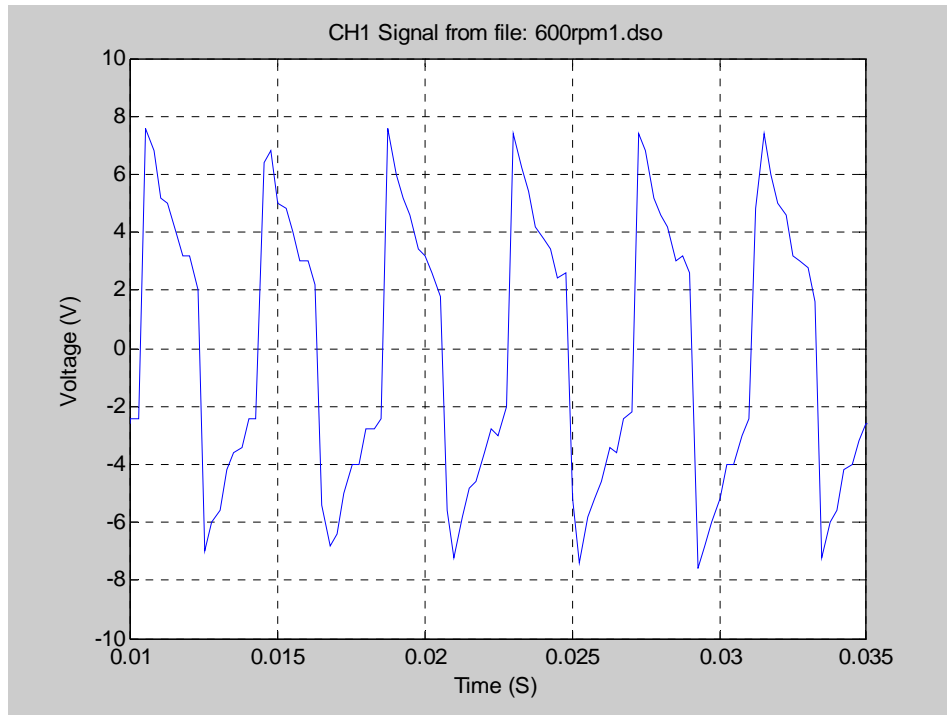


Figure 4.11: Inductive pickup output.<sup>25</sup>

Diesel engines do not have spark plugs, and therefore the engine RPM must be measured in a different manner. Diesel engine RPM must be measured using signals from the alternator, the part of the engine that charges the batteries that start the engine. The signal from the W-terminal of the alternator is a common way to calculate engine RPM. In order to design our system we had to do two things, understand the tachometer signal from an alternator, and design circuitry to condition the signal to be read by the microcontroller. Using previous research we were able to design a circuit to condition the RPM signal of a diesel engine[4]. The signal output of a w-terminal for a diesel engine operating at 600 RPM is shown below.

<sup>25</sup> [http://www.picotech.com/auto/waveforms/graphics/distributor\\_inductive\\_cranking.png](http://www.picotech.com/auto/waveforms/graphics/distributor_inductive_cranking.png)



**Figure 4.12: W-Terminal Signal**

The design of the diesel engine RPM conditioning circuit was based on the measurements made in the previous design project. The previous project directly measured the alternator w-terminal signal on a diesel boat. The signal in the figure below is the measurement taken by the group with the engine at 600 RPM. By transforming the signal into the frequency domain, and plotting frequency vs. engine RPM in a plot, the previous group was able to create two equations used in our software. The two equations are as follows:

$$PulleyRatio = \frac{DrivePulleyDiameter}{AlternatorPulleyDiameter}$$

**Equation 1: Pulley Ratio Calculation[4]**

$$RPM = W - TERMINAL(Sig\_Freq\_in\_Hz) * \frac{60Sec}{Min} * \frac{1}{AlternatorPoles} * \frac{1}{PulleyRatio}$$

**Equation 2: Diesel Engine RPM Calculation[4]**

These two equations use defined engine parameters, along with the frequency of the measured pulses to determine engine RPM. Based on these outputs, a circuit was created to condition the signal to be interpreted by a processing unit. A functional block diagram of the circuit can be seen below in figure 4.13.

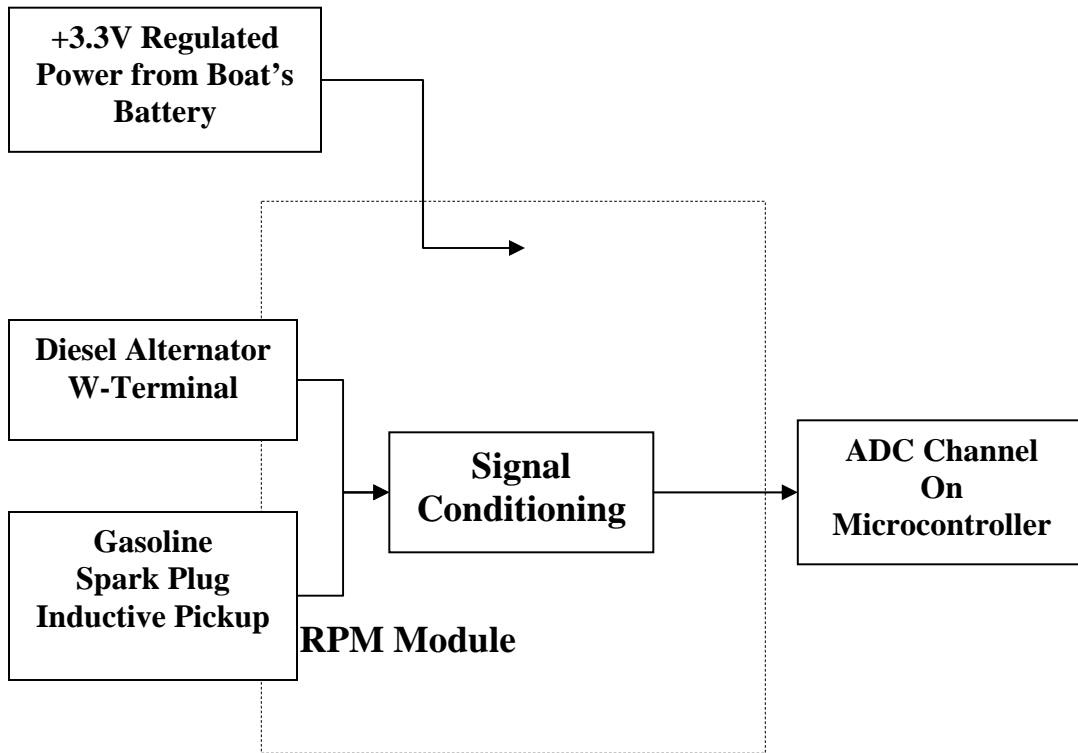


Figure 4.13: RPM Module Block Diagram

The output was made to be a square wave that oscillates between approximately 1.3 and +2.9V. This signal is sent into one of the analog to digital converters on the microprocessor. Using software interrupts, the signal frequency is calculated.

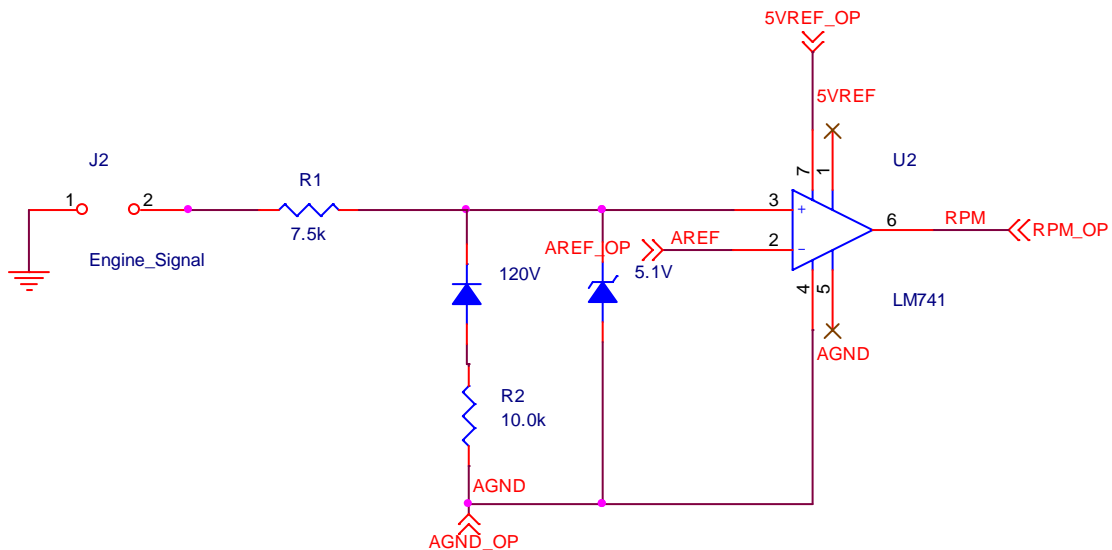


Figure 4.14: RPM Module Schematic

The schematic shown above, depicts the conditioning circuit we designed to take the signal from both the w-terminal input or the inductive pickup input and convert it to a cleaner signal that can be read using the ADC of the microprocessor. The resistor R1 limits the current of the engine's output, protecting the 5.1V Zener diode and the operational amplifier U2, which acts as a comparator. The zener diode regulates the voltage of the input waveform, to a waveform that ranges from about 0V to +5V. This wave form is not as clean as a perfect square wave. Therefore, a comparator is used to produce a signal compatible with the ADC.

An input signal similar to that of a w-terminal was modeled, and sent to the input of the circuit, as seen on channel 1 in figure 4.15. The output signal can be seen below in figure 4.15, on the oscilloscope's channel 2. The function of this circuit is to produce a signal that is compatible with the ADC. The output of the circuit in figure 4.14 is shown in figure 4.15. The output signal is within the voltage range of the ADC, and has severely reduced noise.

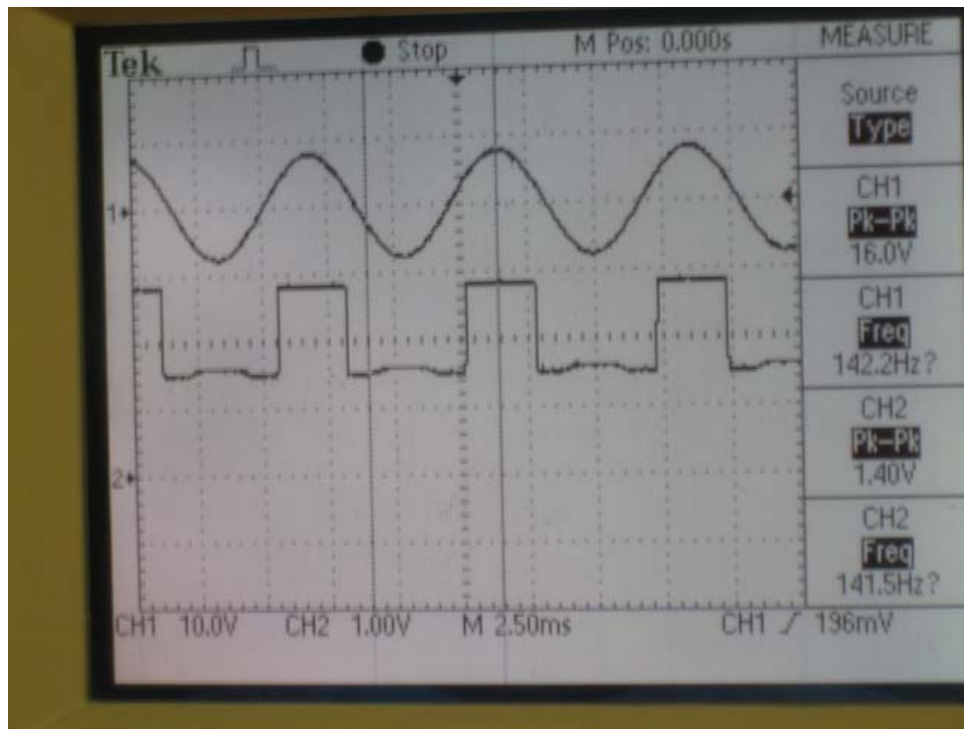


Figure 4.15: RPM Conditioning Circuit Testing Using Function Generator

#### 4.1.2.5 Display Module

The display module of our system, shown in figure 4.16, consists primarily of an

LCD that connects to the microcontroller via digital I/O, and is powered by a +5V voltage regulator. The LCD unit is the CFAH1602CYYHJP from Crystalfontz<sup>26</sup>, seen in Figure 4.17.

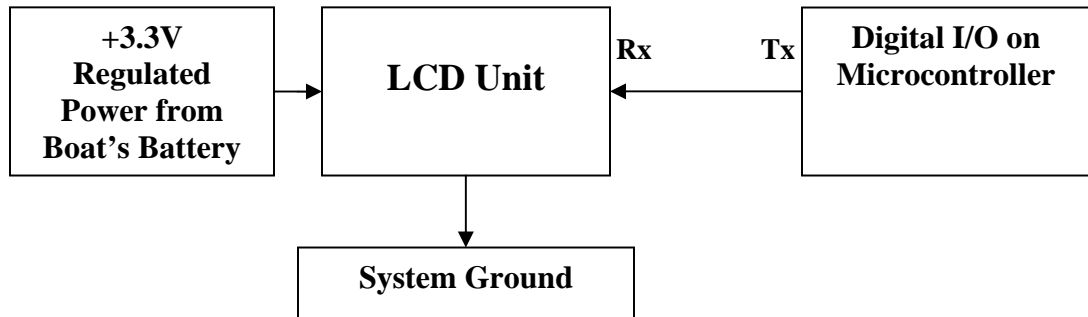


Figure 4.16: LCD Module Block Diagram



Figure 4.17: CFAH1602CYYHJP Front View<sup>27</sup>

It is a simple 16x2 character LCD display that runs on only 11 pins, a diagram of which can be seen in figure 4.19. It runs on the extremely common and well documented Hitachi HD44780 driver. By giving the unit 10-bit commands, you can write any of the 204 characters that the driver supports to any of the 32 display positions. This device is utilized heavily in the debugging of the system, as well as initial programming of the

<sup>26</sup> <http://www.crystalfontz.com/>

<sup>27</sup> [http://www.crystalfontz.com/products/1602c-wt/CFAH1602C-YYH-JP\\_front\\_bl\\_on.jpg](http://www.crystalfontz.com/products/1602c-wt/CFAH1602C-YYH-JP_front_bl_on.jpg)

unit. For our device, Data bits 4-7 are connected to the microcontroller, as are the RS, R/W, and Enable pins. This allows the LCD to read and write data, as well as instructions. The signal ground of the LCD unit is connected to the system ground.

Pin No.	Symbol	Level	Description
1	V <sub>SS</sub>	0V	Ground
2	V <sub>DD</sub>	5.0V	Supply Voltage for logic
3	V <sub>O</sub>	(Variable)	Operating voltage for LCD
4	RS	H/L	H: DATA. L: Instruction code
5	R/W	H/L	H: Read(MPU→ Module) L: Write(MPU→ Module)
6	E	H,H→ L	Chip enable signal
7	DB0	H/L	Data bit 0
8	DB1	H/L	Data bit 1
9	DB2	H/L	Data bit 2
10	DB3	H/L	Data bit 3
11	DB4	H/L	Data bit 4
12	DB5	H/L	Data bit 5
13	DB6	H/L	Data bit 6
14	DB7	H/L	Data bit 7
15	A	-	LED +
16	K	-	LED -

**Figure 4.18: LCD Pin-out table<sup>28</sup>**

A simple test of the LCD was created by simply reading a set word to the display. Using the microprocessor, the word “Turbulance” is written to the LCD, in the first ten characters.

<sup>28</sup> <http://www.crystalfontz.com/products/1602c-wt/CFAH1602CYYHJP.pdf>



**Figure 4.19: LCD Testing**

#### **4.1.2.6 Data Storage Module**

The data storage module is a flash memory device. The device chosen was made by STMicroelectronics<sup>29</sup>, model number M25P32, a 32 Mbit device. The device is shown in the figure below, figure 4.20.



**Figure 4.20: M25P32<sup>30</sup>**

The device receives one-byte commands from the microcontroller. Once commands are sent to the unit it is capable of writing information to memory, sent in three byte chunks by the microcontroller. Additionally, a command can be issued to read data to the data out pin. The device interfaces with the microcontroller using Serial Peripheral Interface(SPI). A pin-out diagram is shown below.

---

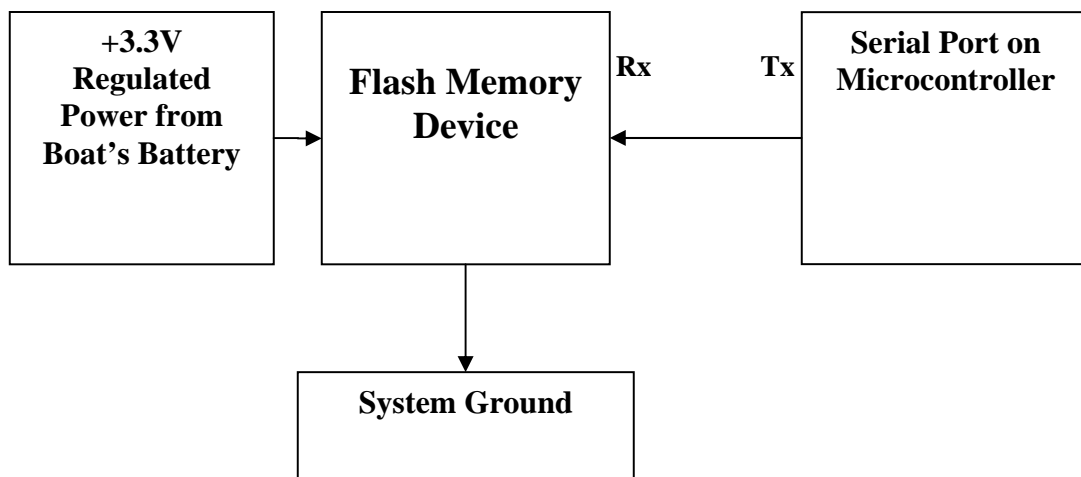
<sup>29</sup> [Http://www.st.com](http://www.st.com)

<sup>30</sup> <http://www.w-field.com/DSCF07261.jpg>

Signal name	Function	Direction
C	Serial Clock	Input
D	Serial Data Input	Input
Q	Serial Data Output	Output
$\bar{S}$	Chip Select	Input
$\bar{W}/V_{PP}$	Write Protect/Enhanced Program supply voltage	Input
$\bar{HOLD}$	Hold	Input
$V_{CC}$	Supply Voltage	Input
$V_{SS}$	Ground	

**Figure 4.21: SPI memory device pin-out<sup>31</sup>**

The input(D),output(Q), hold, chip select(S), and write protect(W) pins are connected to the microcontroller using Digital I/O. In order to synchronize the device with the microcontroller, the clock signal from the microcontroller is used as the input to the serial clock(C) pin on the SPI device. The device is powered by the 3.3V regulator on the PCB and the signal ground is connected to the system ground.



**Figure 4.22: Memory Module Block Diagram**

The device meets the requirement that the system should be able to collect data for one week. The data we are storing takes up a total of 67 bits per reading, worst case. If the spikes in rpm occur every second, also worst case, then the device can operate for 16025997 seconds, or 185 days. For our application this more than satisfies our needs.

As was the case with the other modules, the memory block was tested independently. A sample set of data was written to the device using the microcontroller. That data was then read, and written to the LCD. The data given was “Steven Marshall Frank Carino”. As shown in figure 4.23 the data was correctly written to, and read from

31 <http://www.st.com/stonline/books/pdf/docs/10366.pdf>

the SPI device.



**Figure 4.23: SPI Memory Testing**

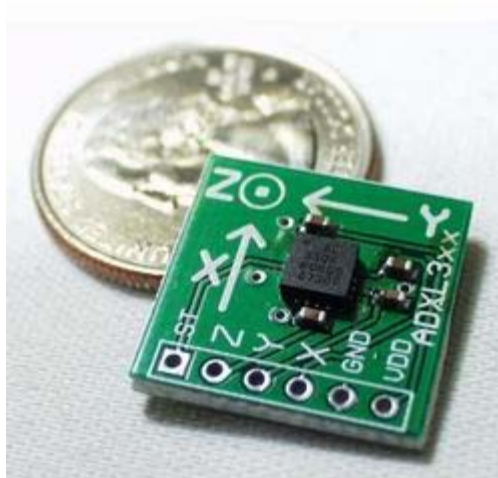
#### **4.1.2.7 Accelerometer**

The accelerometer chosen for this design is an Analog Devices<sup>32</sup> unit, model number ADXL330<sup>33</sup>, a three axis accelerometer with signal conditioned voltage outputs. It can be seen below.

---

<sup>32</sup> <http://www.analog.com/>

<sup>33</sup> [http://www.sparkfun.com/datasheets/Components/ADXL330\\_0.pdf](http://www.sparkfun.com/datasheets/Components/ADXL330_0.pdf)



**Figure 4.24: ADXL330 w/development board<sup>34</sup>**

This unit is powered by the 3.3V regulator on the PCB. Based on the acceleration in a given direction, the unit outputs an analog voltage, to one of its three outputs. A pin-out diagram of the unit is shown below, in figure 4.25.

Table 4. Pin Function Descriptions

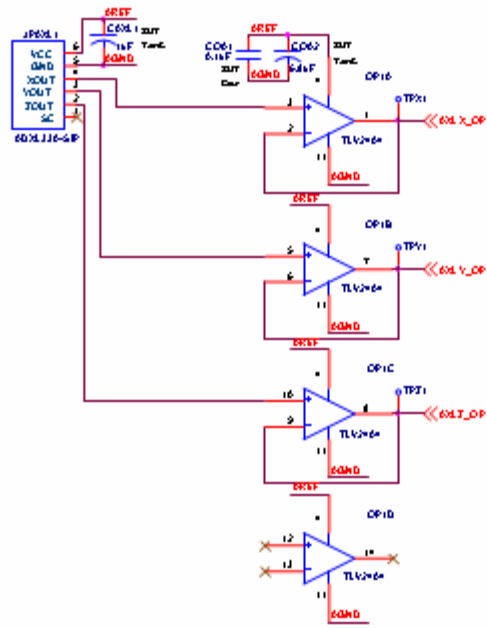
Pin No.	Mnemonic	Description
1	NC	No Connect
2	ST	Self-Test
3	COM	Common
4	NC	No Connect
5	COM	Common
6	COM	Common
7	COM	Common
8	Z <sub>out</sub>	Z Channel Output
9	NC	No Connect
10	Y <sub>out</sub>	Y Channel Output
11	NC	No Connect
12	X <sub>out</sub>	X Channel Output
13	NC	No Connect
14	V <sub>s</sub>	Supply Voltage (2.0 V to 3.6 V)
15	V <sub>s</sub>	Supply Voltage (2.0 V to 3.6 V)
16	NC	No Connect

**Figure 4.25: AXL pin-out<sup>35</sup>**

The accelerometer outputs are connected to the accelerometer using the on board Analog to Digital converter(ADC). Each output takes up one channel of the ADC. The signal ground is connected to the system ground on this unit as well. As a precaution, the accelerometer is fed through a buffer, OP1, as shown in figure 4.26, so as to avoid timing issues with the ADC.

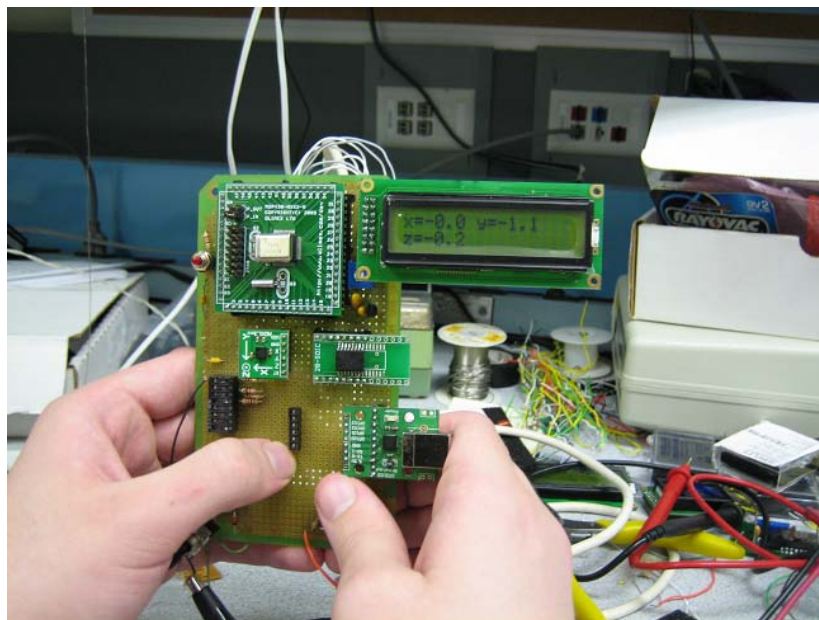
<sup>34</sup> [http://www.sparkfun.com/commerce/product\\_info.php?products\\_id=692#](http://www.sparkfun.com/commerce/product_info.php?products_id=692#)

<sup>35</sup> [http://www.sparkfun.com/datasheets/Components/ADXL330\\_0.pdf](http://www.sparkfun.com/datasheets/Components/ADXL330_0.pdf)



**Figure 4.26: Accelerometer Circuit Schematic**

The accelerometer was tested using a program to indicate the position of the board relative to the three axes. The accelerometer is used to determine the new position of the board, which is read to the LCD. As shown in figures 4.27-4.29, the LCD correctly displays the orientation of the board.



**Figure 4.27: Accelerometer Testing 1**



Figure 4.28: Accelerometer Testing 2

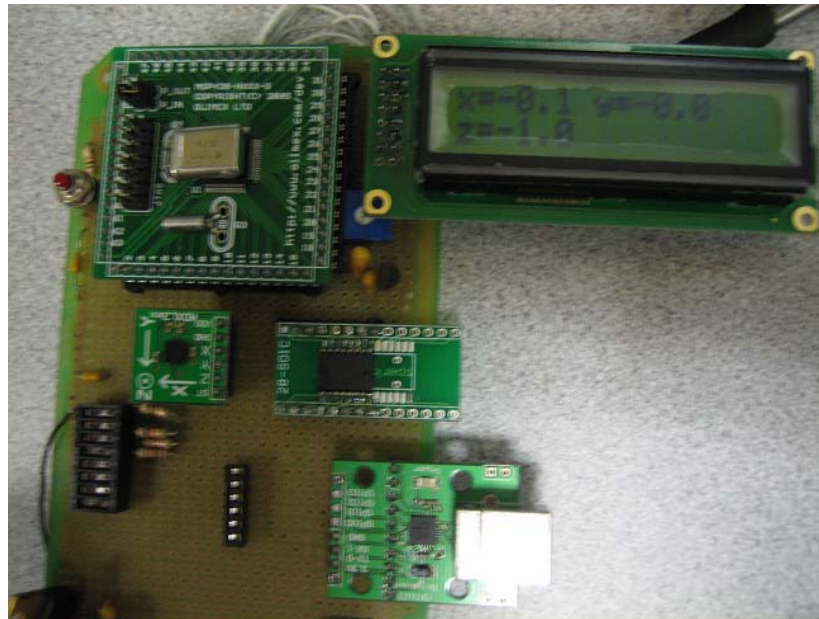
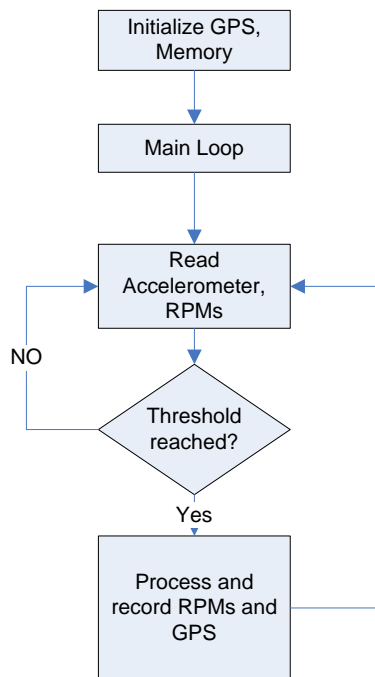


Figure 4.29: Accelerometer Testing 3

### ***4.1.3 Overall Software Design and Testing***

The flow chart below shows the overall software design of the main program. When the device is turned on, the GPS and the memory device are initialized. Readings are taken from the accelerometer and RPM modules. If the threshold is reached, the GPS data is taken, and the GPS data, as well as the RPM reading is stored to memory. If not,

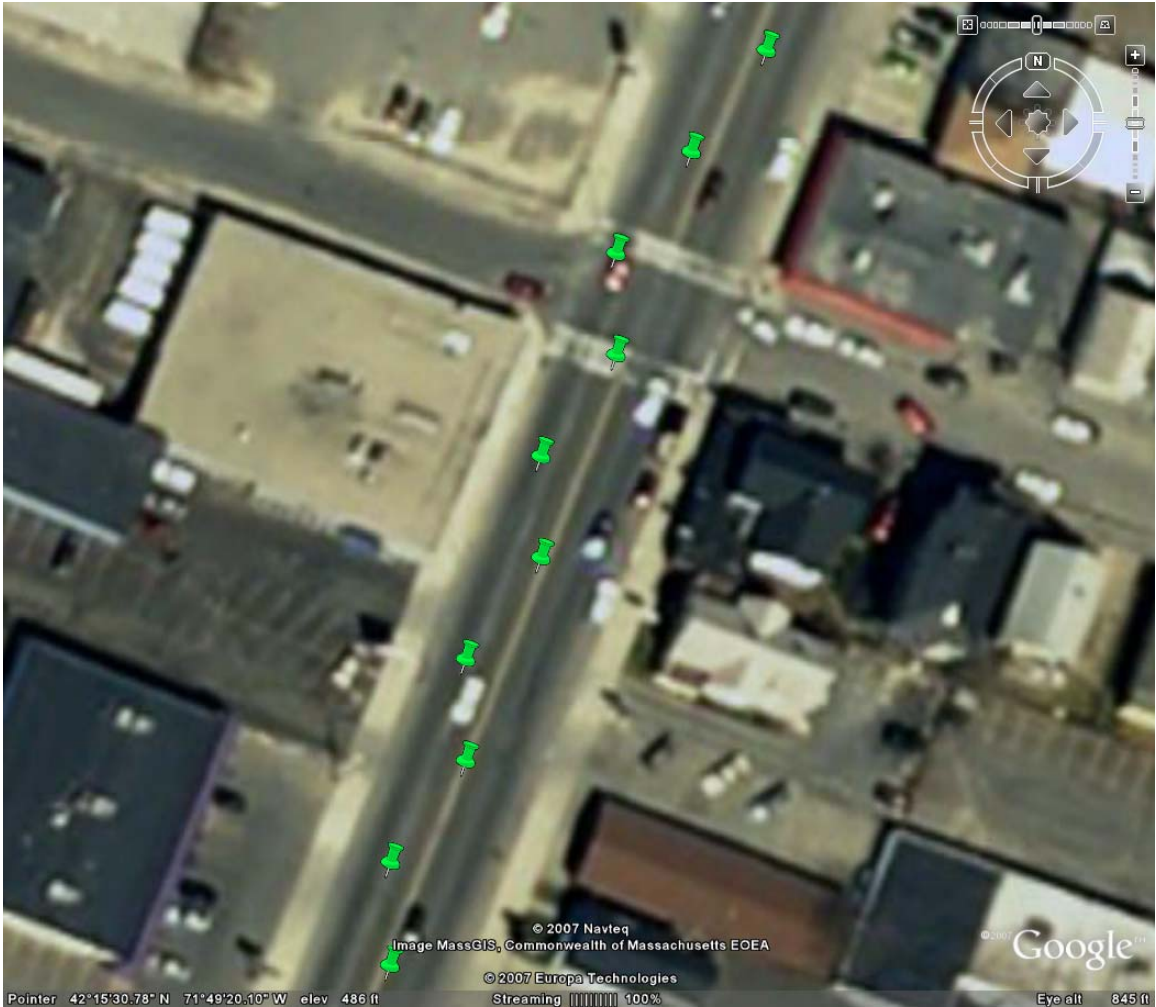
the loop continues repeating until the threshold is reached.



**Figure 4.30: Basic Software Flowchart**

#### **4.1.3.1 GIS**

The GIS that we used is Google Earth. Using the MATLAB code given in appendix B, the output file from the device can be read and converted into a format that Google Earth understands. The program, `turbulence_final_test.m` asks for an input `.txt` file. It takes this file and parses it for time, latitude, hemisphere(latitude), longitude, hemisphere(longitude), accelerometer readings(x,y,z), and rpm readings. It converts these characters to numbers, and then creates an output file. The output file places pushpins on the map corresponding to each instance of change in rpm that was recorded. The pushpins contain data for latitude, longitude, and magnitude of rpm reading. Additionally, the pins are color coded, green representing forward acceleration, and red representing backward acceleration. Below is a set of dummy data created to test the GIS. The data set modeled a car driving down the road, and correctly interpreted the data in the text file. When the output file was loaded, the pushpins were correctly placed, as seen in figure 4.31



**Figure 4.31: GIS Testing**

## **4.2 Summary**

This chapter presented an overall view of the system, from both a hardware and software perspective. The overall system block diagram illustrates the hardware connections, and the software flow chart indicates the basic software operation of the microcontroller, both were presented in this chapter. This chapter additionally presented various depictions of the operation of different modules and methods of testing them. The different modules include: the microcontroller, GPS, RPM, display, data storage, and accelerometer. Pin-out diagrams for each module were also presented in this chapter, as well as block diagrams.

## 5 System Integration and Testing

This section presents the integration and testing of the modules as one functional device. It details the building and testing of the device as a functional prototype. Additionally it details the design and layout of a printed circuit board.

### 5.1 Protoboard testing

Once each of the modules was proven functional, the entire system was wired together on a breadboard for system testing. To test the capabilities of the device, it was hooked up to a 2005 Honda Accord. An inductive pickup is attached to one of the spark plugs of the car, and output to the conditioning circuit. The LCD was set to display the current RPM in real time. As shown in figures 5.1 and 5.2, an accurate measurement of the engine's RPM is displayed.



Figure 5.1: System Testing(gasoline engine) 1



**Figure 5.2: System Testing(gasoline engine) 2**

The system was tested at many varying levels of RPM, from 700-5000 RPM. The system correctly interpreted the engine signal every time. The software for system testing was also written to save instances of change in RPM that is greater than 25% of the current RPM per second. We gave the system some spikes in RPM and it saved them to the flash memory device. The flash memory device upload, however, stopped working after the units were placed onto the protoboard. Communication with the device is still occurring, as shown in figure 5.3.



**Figure 5.3 SPI memory communications**

By debugging, we have found that the rest of the device functions properly. All of the tests described in section 4 were repeated, and all of them worked with the exception of the upload capabilities of the device. Without the ability to upload, the system is not fully operational. However, the device still meets the following requirements of the system:

1. The system can read RPM from both gasoline and diesel engines
2. The signal output by the engine is properly conditioned in order to be sent to the microprocessor
3. The system can determine instances of changes in RPM that are greater than 25% per second
4. Instances are marked with location and acceleration data for the boat
5. RPM, GPS, and accelerometer readings can be stored for at least one week
6. The system can be powered by the boat.
7. A GIS has been developed to display the appropriate data

## **5.2 PCB**

A printed circuit board was designed for this project, but we were unable to populate and test it due to time constraints. The layout can be seen in Appendix C: PCB Schematics.

### **5.3 Summary**

This chapter presented our system integration and testing. The system was tested for gasoline engines, and met almost all of its requirements. It was constructed and soldered into a protoboard, and a design for a PCB was laid out.

## 6 Summary and Conclusions

This chapter summarizes the work completed by our project team at Worcester Polytechnic Institute. The project objectives that were accomplished, are also reviewed. Develop an automated data collection system that can be installed in motorboats propelled by either diesel or gasoline engine(s) to monitor and store engine RPM as well as GPS positional data, for boats that navigate the Venetian canals.

### 6.1 Summary of Design, and results

To create our project design we followed a design process that involved researching how to measure engine RPM for both diesel and gasoline engines, establishing system requirements, implementing the design from the system requirements, and finally processing requirements for the data in post.

Our system took on the form of a modular design. The different modules are listed below:

1. Power Module
2. Microcontroller Module
3. RPM Module
4. GPS Module
5. Accelerometer Module
6. LCD Module
7. Data Storage Module

The power module consisted of 2 voltage regulators, connected to the boat's battery, which provide power to all of the other modules. The +5V regulator is used to power the LCD, while the +3.3V regulator is used to power the microprocessor, RPM, GPS, accelerometer, and data storage modules.

The microcontroller module provides data processing, as well as being the central processing unit of the device. It communicates with all the various modules, sending commands and receiving data. It interconnects all the different modules, for system integration.

The RPM module's function is to condition signals from the W-terminal on a diesel engine, and from an inductive pickup placed on a gasoline engines spark plug wires. The circuit conditions the signal so that it can be interfaced with the ADC on the microcontroller for processing and engine RPM calculation.

The GPS module consists of a GPS receiver, and its antenna. The receiver is initialized by the microcontroller, and then transmits an ASCII formatted string to the microcontroller.

The accelerometer module measures the acceleration of the boat. This is important in determining if the boat was accelerating or decelerating at the time of the turbulent discharge.

The LCD module is used in programming and debugging the device.

The data storage module receives location data, as well as RPM readings, and stores them using Flash memory. When the device is hooked up to a computer the readings can be taken and put into the GIS.

## **6.2 Future Recommendations**

While we successfully modeled, developed, and implemented portions of our system design, there are still some concerns that need to be addressed. We successfully:

1. Developed a method for determining engine RPM from both diesel and gasoline engines.
2. Developed a prototype for an embedded system that can store engine RPM readings and GPS coordinates on flash memory, and provide the user with relevant information.
3. Wrote software to interface each of the modules of the design.

During our research, we found the W-terminal solution to diesel tachometers to be incredibly difficult to implement. The system was based on previous research, and if in fact the signals from the diesel alternator are as the previous report suggested, then the design works as intended. However, research has shown that not all diesel engines are equipped with a w-terminal, and future research into this alternator technology will need to be made. Once the true nature of this terminal is discovered, then the method for reading RPM can be implemented and field tested.

Additionally, a PCB was designed and laid out, but not populated or tested. Using this design, a PCB can be created and housed in some form of packaging.

The device currently does not have the ability to upload stored data from the SPI memory device to a PC. The code had worked at one point, and the device is still receiving communications, so we hypothesize that the problem is in the software related to uploading data. Future debugging is necessary, but we believe there is no need to generate brand new code.

## **6.3 Conclusions**

We were able to successfully satisfy all of the system requirements individually. When the system was brought together, it met almost all of the system requirements, except for the ability to load data from the SPI device to a PC. The research conducted

shows that this system does have potential to help locate instances of canal damage, but it needs further testing before it is a fully viable solution. If the future recommendations were implemented, the system would then be ready for field tests in Venice, where the device is designed to operate.

## **Bibliography**

1. Chiu, Jagganath, and Nodine. "The Moto Ondoso Index" IQP. Worcester Polytechnic Institute, July 2002.
2. Chiu, Jagganath, and Nodine. "The Moto Ondoso Index" IQP. Worcester Polytechnic Institute, July 2002.
3. Chiu, Lacasse, and Menard. "Mapping Turbulence in the Canals of Venice" MQP. Worcester Polytechnic Institute, January 2004
4. Angelini, Brache, Gdula, Shevlin. "Mapping Underwater Turbulence in Venice" MQP. Worcester Polytechnic Institute, April 2006

## Appendix A: Embedded C Code

```
/* SPI MEMORY TEST PROGRAM */
```

```
#include "msp430x16x.h" // Definitions, constants, etc for msp430F169
```

```
#define DEBUG
#include "mqp.h"
#include "mqpdipsw.h"
```

```
#define LCD4BIT
#include "mqplcd.h"
```

```
/****** FUNCTION DECLARATIONS
******/
```

```
/* Modes */
#include "mqpdatalog.h"
#include "mqpdownload.h"
#include "mqpaxl.h"
#include "mqpdebug.h"
#include "mqpusb.h"
#include "mqpusart.h"
#include "mqpsfm.h"
```

```
/****** MAIN FUNCTION ******/
```

```
void main(void)
{
    WDTCTL = WDTPW + WDTHOLD; /* stop the watchdog

timer */
    lcdInit(); /* initialize the LCD */
    dipswInit(); /* initialize the DIP switch */
    taInit();
    axlInit();

    _EINT();

    /* print welcome message */
    lcdPrints( "Turbulence" );
    hwDelay( &timerA, 1, 0x7FFF ); /* set up timer A to wait for .5 second */
    while( timerA.status != DONE );
    lcdClear();

    /* main loop */
    while(1){
        /* run specified mode based on DIP switch */
        switch( dipswRead() ){
```

```

case 0x00:
    lcdPrints( "Datalog\nMode" );
    hwDelay( &timerA, 1, 0x7FFF ); /* set up timer A to wait for 1.5 second */
    while( timerA.status != DONE );
    datalogMode();
    break;
case 0x03:
    lcdPrints( "Download\nMode" );
    hwDelay( &timerA, 1, 0x7FFF ); /* set up timer A to wait for 1.5 second */
    while( timerA.status != DONE );
    downloadMode();
    break;
case 0x04:
    lcdPrints( "Debug\nMode" );
    hwDelay( &timerA, 1, 0x7FFF ); /* set up timer A to wait for 1.5 second */
    while( timerA.status != DONE );
    debugMode();
    break;
case 0x08:
    lcdPrints( "Delete\nMode" );
    hwDelay( &timerA, 1, 0x7FFF ); /* set up timer A to wait for 1.5 second */
    while( timerA.status != DONE );

    /* set up USART1 for SPI for SFM */
    sfmInit1();
    sfmPost();

    sfmErase();
    break;
default:
    lcdPrints( "No Mode?\n" );
    hwDelay( &timerA, 3, 0x7FFF ); /* set up timer A to wait for 1.5 second */
    while( timerA.status != DONE );
}
/* sleep a bit */
hwDelay( &timerA, 3, 0x7FFF ); /* set up timer A to wait for 1.5 second */
while( timerA.status != DONE );
lcdClear();
}
}

// init_spi0(); /* set up SPI memory on USART 0 */
// init_uart1(); /* enable the USB-UART on USART 1 */
// _EINT(); /* enable interrupts */
//
// usbPrint( "\n\n\rBegin USB Communication...\n\r" );

```

```

// while( !(tx1.status) );
// usbPrintMenu();
//
// rx1.stop='\r'; /* wait for carriage return */
// uAddrType addr=0x000000;

/* loop forever */
// while(1){
//     /* echo */
//     if( rx1.status ){
//         switch( rx1.buffer[rx1.length-1] ){
//             case 'i':
// while( rRetVal != Flash_Success ){
//     /* read manufacturer identification */
//     usbPrint( "\n\r-----");
//     rRetVal=Flash(ReadManufacturerIdentification, &fp );
//     foo = fp.ReadManufacturerIdentification.ucManufacturerIdentification;
//     usbPrint( "\n\r" );
//     usbPrint( FlashErrorStr( rRetVal ) );
//     usbPrint( " Manufacturer Identification: " );
//     TXBUF1 = '0'+(char)foo;
// }
//
//     /* read device identification */
//     rRetVal=Flash(ReadDeviceIdentification, &fp );
//     usbPrint( "\n\r" );
//     usbPrint( FlashErrorStr( rRetVal ) );
//     usbPrint( " Device Identification: " );
//     foo = fp.ReadDeviceIdentification.ucDeviceIdentification;
//     TXBUF1 = '0'+(char)(foo >> 8);
//     TXBUF1 = '0'+(char)foo;
//
//     /* read status register */
//     rRetVal = Flash( ReadStatusRegister, &fp );
//     usbPrint( "\n\r" );
//     usbPrint( FlashErrorStr( rRetVal ) );
//     usbPrint( " Status Register: " );
//     foo = fp.ReadStatusRegister.ucStatusRegister;
//     TXBUF1 = '0'+(char)foo;

/* write some data */
//     fp.Program.udAddr = 0;
//     fp.Program.udNrOfElementsInArray = 10;
//     fp.Program.pArray = (void*)wArray;
//     rRetVal = Flash( Program, &fp );
//     usbPrint( "\n\r" );

```

```

//      usbPrint( FlashErrorStr( rRetVal ) );
//      usbPrint( "  Wrote!\n\r" );

      /* read back data */
//      fp.Read.udAddr = 0;
//      fp.Read.udNrOfElementsToRead = READLEN;
//      fp.Read.pArray = (void*)pArray;
//      rRetVal=Flash(Read, &fp );
//      usbPrint( "\n\r" );
//      usbPrint( FlashErrorStr( rRetVal ) );
//      usbPrint( "  Memory Dump:\n\r" );
//      bar = (char)(fp.Read.pArray[addr]);

//      for( foo=0; foo<READLEN; foo++){
//          TXBUF1 = pArray[foo];
//          TXBUF1 = 55+(pArray[foo] >> 4);
//          TXBUF1 = pArray[foo]-185;
//      }

      /* erase a sector */
//      fp.SectorErase.ustSectorNr = 2;
//      rRetVal = Flash( SectorErase, &fp);
//      usbPrint( "\n\r" );
//      usbPrint( FlashErrorStr( rRetVal ) );
//      usbPrint( "  Erased Sector 2" );

//      addr++;

//ADCTEST
#include "msp430x16x.h"    // Definitions, constants, etc for msp430F169
#include <stdio.h>
// #include <in430.h>
#include "mqp.h"
#include "mqplcd.h"
#include "mqpadc.h"
#include "mqquart.h"

// ***** FUNCTION DECLARATIONS *****
void init_sys(void);    // MSP430 Initialization routine

void ledOn(void);    // turn on LED
void ledOff(void);    // turn off LED

/* GLOBALS */
signed long adc;

```

```

float fadc;

/*****
*/
/*
/*          */
/* main() variable declarations          */
/*          */
/*****
*/

void init_dipsw(void);
char dipswRead(void);

/***** MAIN FUNCTION *****/
void main(void)
{
    adc=0;
    fadc=0;

    WDTCTL = WDTPW + WDTHOLD; // Stop watchdog timer
    init_sys(); // Initialize the MSP430

    /* print "Hello World" */
    lcdPrints( "Hello World " );

    /* Loop forever */
    while(1)
    {
        adcRead(); // force a new ADC conversion */
        adc = ADC12MEM1; // read in channel one */
        fadc = adcNorm( adc ); // normalize ADC reading */

        fadc = adcNorm( ADC12MEM1 );

        if( fadc > 1 ){
            if( fadc > 2 ){
                lcdPrint( '2' );
                lcdPrint( 'V' );
            }
            else{
                lcdPrint( '1' );
                lcdPrint( 'V' );
            }
        }
        else{
            lcdPrint( '0' );
            lcdPrint( 'V' );
        }
    }
}

```

```

    }

    lcdCmd( 0xC6 );
}
}

/***** initSys() *****/
void init_sys(void)
{
    init_dipsw();
    init_lcd();
    init_adc();
    init_uart();
    ledOff(); // turn off LED
}

#define DIPSWDIR P1DIR
#define DIPSWSEL P1SEL
#define DIPSWIN P2IN
#define DIPSWBITS (BIT7|BIT6|BIT5|BIT4)

void init_dipsw(void)
{
    DIPSWDIR &= ~(DIPSWBITS); /* set dipswitch port to input direction */
    DIPSWSEL &= ~(DIPSWBITS); /* set dipswitch port I/O option */
}

char dipswRead(void)
{
    char input;

    /* read in from dipswitch, mask off unneeded lower nybble */
    input = DIPSWIN & 0xF0;

    return( input ); /* return what was read in */
}

/***** ledOn() *****/
void ledOn(void)
{
    P2DIR |= (BIT0);          // Set P2.0 to output direction
    P2SEL &= ~(BIT0);        // P2.0 I/O option
    P2OUT |= (BIT0);         // P2.0 output = 1 (LED on)
}

/***** ledOff() *****/

```

```

void ledOff(void)
{
    P2DIR |= (BIT0); // Set P2.0 to output direction
    P2SEL &= ~(BIT0); // P2.0 I/O option
    P2OUT &= ~(BIT0); // P2.0 output = 0 (LED off)
}

//Dipswitch
#include "msp430x16x.h" // Definitions, constants, etc for msp430F169
#include <stdio.h>
//#include <in430.h>
#include "mqp.h"
#include "mqpdipsw.h"
#include "mqplcd.h"
#include "mqpadc.h"
#include "mqpuart.h"

// ***** FUNCTION DECLARATIONS *****
void init_sys(void); // MSP430 Initialization routine

void ledOn(void); // turn on LED
void ledOff(void); // turn off LED

/* GLOBALS */
signed long adc;
float fadc;
char dipsw, olddipsw;

/*****
*/
/*
*/
/* main() variable declarations */
/*
*/
/*****
*/

/***** MAIN FUNCTION *****/
void main(void)
{
    adc=0;
    fadc=0;
    dipsw=0xFF;
    olddipsw=0xFF;
}

```

```

WDTCTL = WDTPW + WDTHOLD; // Stop watchdog timer
init_sys(); // Initialize the MSP430

/* print "Hello World" */
lcdPrints( "Hello World " );
swDelay( 5, DHSEC ); /* wait 2.5 seconds */

lcdClear(); // clear the LCD */

/* Loop forever */
while(1)
{
    dipsw = dipswRead();

    /* update LCD only if dipswitch changed position */
    if( dipsw != olddipsw ){
        lcdLine1(); // return LCD cursor to line 1, position 1 */
        switch( dipsw ){
            case 0x00: lcdPrints( "DIP:0000" ); break;
            case 0x01: lcdPrints( "DIP:0001" ); break;
            case 0x02: lcdPrints( "DIP:0010" ); break;
            case 0x03: lcdPrints( "DIP:0011" ); break;
            case 0x04: lcdPrints( "DIP:0100" ); break;
            case 0x05: lcdPrints( "DIP:0101" ); break;
            case 0x06: lcdPrints( "DIP:0110" ); break;
            case 0x07: lcdPrints( "DIP:0111" ); break;
            case 0x08: lcdPrints( "DIP:1000" ); break;
            case 0x09: lcdPrints( "DIP:1001" ); break;
            case 0x0A: lcdPrints( "DIP:1010" ); break;
            case 0x0B: lcdPrints( "DIP:1011" ); break;
            case 0x0C: lcdPrints( "DIP:1100" ); break;
            case 0x0D: lcdPrints( "DIP:1101" ); break;
            case 0x0E: lcdPrints( "DIP:1110" ); break;
            case 0x0F: lcdPrints( "DIP:1111" ); break;
            default: lcdPrints( "DIP:XXXX" );
        }

        olddipsw=dipsw;
    }

    // adcRead(); // force a new ADC conversion */
    // adc = ADC12MEM1; // read in channel one */
    // fadc = adcNorm( adc ); // normalize ADC reading */
    //
    // fadc = adcNorm( ADC12MEM1 );
    //

```

```

// if( fadc > 1 ){
//   if( fadc > 2 ){
//     lcdPrint( '2' );
//     lcdPrint( 'V' );
//   }
//   else{
//     lcdPrint( '1' );
//     lcdPrint( 'V' );
//   }
// } else{
//   lcdPrint( '0' );
//   lcdPrint( 'V' );
// }
//
// lcdCmd( 0xC6 );
//
}

/***** initSys() *****/
void init_sys(void)
{
  init_dipsw();
  init_lcd();
  init_adc();
  init_uart();
  ledOff(); // turn off LED
}

/***** ledOn() *****/
void ledOn(void)
{
  P2DIR |= (BIT0);           // Set P2.0 to output direction
  P2SEL &= ~(BIT0);         // P2.0 I/O option
  P2OUT |= (BIT0);          // P2.0 output = 1 (LED on)
}

/***** ledOff() *****/
void ledOff(void)
{
  P2DIR |= (BIT0); // Set P2.0 to output direction
  P2SEL &= ~(BIT0); // P2.0 I/O option
  P2OUT &= ~(BIT0); // P2.0 output = 0 (LED off)
}

```

```

/* MQP header */
/* some general constants and functions */

#ifndef MQP_H
#define MQP_H

char char256[256]; /* sfbuffer */
char char180[180]; /* GPS RX buffer, USB TX buffer */
char char125[125]; /* SFM RX buffer */
char char32[32]; /* SFM TX buffer */
char char1[1]; /* USB RX buffer */

int sfmreadpos;

#endif

/* mqpadc.h */
/***** header file for ADC interface *****/
#ifndef MQPADC_H
#define MQPADC_H

#include "mqp.h"

#define XAXIS ADC12MEM0
#define YAXIS ADC12MEM1
#define ZAXIS ADC12MEM2

/* accelerometer axis */
typedef struct _AXLaxis{
    int value; /* the latest value read from the ADC */
    int max; /* the maximum value read from the ADC */
    int min; /* the minimum value read from the ADC */
}AXLaxis;

/* triple-axis accelerometer */
typedef struct _AXL3{
    AXLaxis x; /* X axis */
    AXLaxis y; /* Y axis */
    AXLaxis z; /* Z axis */
}AXL3;

void axlInit(void);
void axlRead(void);

AXL3 axl;

```

```

/***** init_adc() *****/
* function to initialize the ADC channels
* channel setup:
* 0: Accelerometer X Axis
* 1: Accelerometer Y Axis
* 2: Accelerometer Z Axis
*/
void axlInit(void)
{
    /* set up conversion clocks, turn adc on, use multiple samples */
    ADC12CTL0 = SHT0_6 + ADC12ON + MSC;
    /* set up for smping signal select and single sequence sample */
    ADC12CTL1 = SHP + CONSEQ_1;

    /* set up channel zero for input, use AVcc with respect to Avss */
    ADC12MCTL0 = INCH_0 + SREF_0;
    /* set up channel one for input, use AVcc with respect to Avss */
    ADC12MCTL1 = INCH_1 + SREF_0;
    /* set up channel two for input, use AVcc with respect to Avss */
    ADC12MCTL2 = INCH_2 + SREF_0 + EOS;

    /* enable conversion */
    ADC12CTL0 |= ENC;
}

/***** axlRead() *****/
/* function to read a conversion from the ADC into ADC memory */
/* post: after being called, the ADC12MEM# has the new conversion in it */
void axlRead(void)
{
    ADC12CTL0 |= ADC12SC;          /* start up a conversion */
    while( ADC12CTL0 & ADC12SC ); /* hang until conversion is done */
}

#endif

/* mqpadc.h */
/***** header file for ADC interface *****/
#ifndef MQPADC_H
#define MQPADC_H

#define INTMAX 0xFFFF

```

```

#define XAXIS ADC12MEM0
#define YAXIS ADC12MEM1
#define ZAXIS ADC12MEM2
#define RPM ADC12MEM3

typedef unsigned int AXLint;

/* accelerometer axis */
typedef struct _AXLaxis{
    AXLint value; /* the latest value read from the ADC */
    AXLint max; /* the maximum value read from the ADC */
    AXLint min; /* the minimum value read from the ADC */
}AXLaxis;

/* triple-axis accelerometer */
typedef struct _AXL3{
    AXLaxis x; /* X axis */
    AXLaxis y; /* Y axis */
    AXLaxis z; /* Z axis */
}AXL3;

void axlInit(void);
void axlReset(void);
void axlConvert(void);
void axlRead(void);

AXL3 axl; /* the accelerometer */

/***** init_adc() *****/
* function to initialize the ADC channels
* channel setup:
* 0: Accelerometer X Axis
* 1: Accelerometer Y Axis
* 2: Accelerometer Z Axis
*/
void axlInit(void)
{
    /* set up conversion clocks, turn adc on, use multiple samples */
    ADC12CTL0 = SHT0_6 + ADC12ON + MSC;
    /* set up for smping signal select and single sequence sample */
    ADC12CTL1 = SHP + CONSEQ_1;

    /* set up channel zero for input, use AVcc with respect to Avss */
    ADC12MCTL0 = INCH_0 + SREF_0;
    /* set up channel one for input, use AVcc with respect to Avss */
    ADC12MCTL1 = INCH_1 + SREF_0;
}

```

```

    /* set up channel two for input, use AVcc with respect to Avss */
    ADC12MCTL2 = INCH_2 + SREF_0;

    ADC12MCTL3 = INCH_3 + SREF_0 + EOS;

    /* enable conversion */
    ADC12CTL0 |= ENC;

    /* clear accelerometer structure */
    axlReset();
}

/***** axlReset() *****/
/* function to clear accelerometer structure */
void axlReset(void)
{
    /* x axis */
    axl.x.value = 0;
    axl.x.max   = 0;
    axl.x.min   = INTMAX;

    /* y axis */
    axl.y.value = 0;
    axl.y.max   = 0;
    axl.y.min   = INTMAX;

    /* z axis */
    axl.z.value = 0;
    axl.z.max   = 0;
    axl.z.min   = INTMAX;
}

/***** axlConvert() *****/
/* function to read a conversion from the ADC into ADC memory */
/* post: after being called, the ADC12MEM# has the new conversion in it */
void axlConvert(void)
{
    ADC12CTL0 |= ADC12SC;          /* start up a conversion */
    while( ADC12CTL0 & ADC12SC ); /* hang until conversion is done */
}

/***** axlRead() *****/
/* function to get current ADC values into accelerometer structure */
void axlRead(void)
{
    /* do a conversion */

```

```

axlConvert();

/* store in structure */
axl.x.value = XAXIS;
axl.y.value = YAXIS;
axl.z.value = ZAXIS;

/* check for new max */
if( XAXIS > axl.x.max ){ axl.x.max = XAXIS; }
if( YAXIS > axl.y.max ){ axl.y.max = YAXIS; }
if( ZAXIS > axl.z.max ){ axl.z.max = ZAXIS; }

/* check for new min */
if( XAXIS < axl.x.min ){ axl.x.min = XAXIS; }
if( YAXIS < axl.y.min ){ axl.y.min = YAXIS; }
if( ZAXIS < axl.z.min ){ axl.z.min = ZAXIS; }
}
#endif

/* mqpdatalog.h */
/***** header file for datalogging mode *****/

#ifndef MQPDATALOG_H
#define MQPDATALOG_H

#include <stdio.h>

#include "mqptimer.h"
#include "mqpgps.h"
#include "mqpsfm.h"
#include "mqpaxl.h"

void datalogMode(void);
void datalogInit(void);
void datalogLoop(void);
void datalogKill(void);
char *pbuf;
char stringName[6];
char time[10];
char latitude[9];
char NorS[1];
char longitude[10];
char EorW[1];
char *token;
int digit;

```

```

int tens;
char bar;
char dips;
#define READLEN 50
char pArray[READLEN];
char wArray[] = "Turbulence";
char adcArray[] = "x= 0.0 y= 0.0\nz= 0.0";
float advolt;
int highorlow=0;
int counter=0;
char counternum[] = "rpm= \n";
int hundreds;
int tens;
int thousands;
int rpms[5] = {0,0,0,0,0};
int rpmcounter=0;
int avgrpm;
int i;
void datalogMode(void)
{
    _DINT();

    /* initialize */
    datalogInit();

    /* run loop */
    while( dipswRead() == DATALOGMODE ){
        datalogLoop();
    }

    /* kill */
    datalogKill();
}

void datalogInit(void)
{
    /* by default, each interrupt is disabled */
    /* set up USART0 for UART for GPS */
    gpsInit();
    gpsPost();

    /* set up USART1 for SPI for SFM */
    sfmInit1();
    sfmPost();

    /* set up ADC0-3 for AXL */

```

```

axlInit();

/* clear LCD */
lcdClear();
}

char buf8[9];
char *token;
char axls[30];

char utc[7];
char latitude[9];
char ns[2];
char longitude[10];
char ew[2];

int inumSats;

void datalogLoop(void)
{
    WDTCTL = WDTPW + WDTHOLD; // Stop watchdog timer
    taInit();
    _EINT(); // enable interrupts */
    while(1){
        hwDelay(&timerA,1,0);
        while(timerA.status!=DONE){
            axlRead();
            if (ADC12MEM3>3000){
                if (highorlow==0){
                    highorlow=1;
                    counter++;
                }
            }
            else{
                highorlow=0;
            }
        }
        counter/=2;
        counter*=120; //60*Cylinders/2
        rpms[rpmcounter]=counter;
        rpmcounter++;
        thousands=counter/1000;
        counternum[4]=thousands+'0';
        hundreds=counter/100-thousands*10;
        counternum[5]=hundreds+'0';
        tens=counter/10-hundreds*10-thousands*100;
    }
}

```

```

    counternum[6]=tens+'0';
    counternum[7]=counter-tens*10-hundreds*100-thousands*1000+'0';
    lcdClear();
    lcdPrints(counternum);
//    lcdLine1();
    if (rpmcounter==5)
        rpmcounter=0;
//    lcdClear();
    avgrpm=(rpms[0]+rpms[1]+rpms[2]+rpms[3]+rpms[4])/5;
    if (counter>avgrpm*1.25){
        lcdPrints("Change!");
        lcdLine1();
        /* set up timer A to wait for 2.5 seconds */
        hwDelay( &timerA, 3, 0x7FFF );
        gpsInit();
        _EINT();

        /*stuff.txt*/

        gpsEnable();

        while( timerA.status != DONE && (*hgps).rx.status != FREE ){
            axlRead();
        }
        gpsDisable();

        token = NULL;

        if( (*hgps).rx.length > 0 ){
            token = (*hgps).rx.buffer;
            strncpy ( latitude, token+18, 9);
            strncpy ( longitude, token+30, 9);
            latitude[9]='\n';
//            lcdClear();
//            lcdPrints(latitude);
//            lcdPrints(longitude);

        }

        sfmEnable();

        /* write some data */
        fp.Program.udAddr = sfmpos;
        fp.Program.udNrOfElementsInArray = (*hgps).rx.length;
        fp.Program.pArray = (void*)(*hgps).rx.buffer;

```

```

    rRetVal = Flash( Program, &fp );

    sfmpos += (*hgps).rx.length;
    sfmDisable();
for (i=0; i<5; i++)
    rpms[i]=counter;
    }
    counter=0;

}
}

void datalogKill(void)
{
/* flush SFM buffer to SFM */
/** sfmFlush() **/

/* free up buffers, nullify pointers */
dusart0.tx.buffer = NULL;
dusart0.rx.buffer = NULL;
dusart1.tx.buffer = NULL;
dusart1.rx.buffer = NULL;
}
#endif

/* mqpdebug.h */
/***** header file for debug mode *****/

#ifndef MQPDEBUG_H
#define MQPDEBUG_H

void debugMode(void);
void debugInit(void);
void debugLoop(void);
void debugKill(void);

void debugMode(void)
{
/* initialize */
debugInit();

/* run loop */
while(1){
    debugLoop();
}
}

```

```

    debugKill();
}

void debugInit(void)
{
    gpsInit();
    gpsPost();
}

void debugLoop(void)
{
    WDTCTL = WDTPW + WDTHOLD; // Stop watchdog timer
    taInit();
    _EINT(); // enable interrupts */
    while(1){
        hwDelay(&timerA,1,0);
        while(timerA.status!=DONE){
            axlRead();
            if (ADC12MEM3>3000){
                if (highorlow==0){
                    highorlow=1;
                    counter++;
                }
            }
            else{
                highorlow=0;
            }
        }
        counter/=2;
        counter*=120; //60*Cylinders/2
        rpms[rpmcounter]=counter;
        rpmcounter++;
        thousands=counter/1000;
        counternum[4]=thousands+'0';
        hundreds=counter/100-thousands*10;
        counternum[5]=hundreds+'0';
        tens=counter/10-hundreds*10-thousands*100;
        counternum[6]=tens+'0';
        counternum[7]=counter-tens*10-hundreds*100-thousands*1000+'0';
        lcdClear();
        lcdPrints(counternum);
//    lcdLine1();
        if (rpmcounter==5)
            rpmcounter=0;
//    lcdClear();
    }
}

```

```

    avgrpm=(rpms[0]+rpms[1]+rpms[2]+rpms[3]+rpms[4])/5;
    if (counter>avgrpm*1.25){
        lcdPrints("Change!");
        lcdLine1();
        /* set up timer A to wait for 2.5 seconds */
hwDelay( &timerA, 3, 0x7FFF );
gpsInit();
_EINT();

/*stuff.txt*/

gpsEnable();

while( timerA.status != DONE && (*hgps).rx.status != FREE ){
    axlRead();
}
gpsDisable();

token    = NULL;

if( (*hgps).rx.length > 0 ){
    token = (*hgps).rx.buffer;
    strncpy ( latitude, token+18, 9);
    strncpy ( longitude, token+30, 9);
    latitude[9]='\n';
//  lcdClear();
//  lcdPrints(latitude);
//  lcdPrints(longitude);

}

sfmEnable();

/* write some data */
fp.Program.udAddr = sfmpos;
fp.Program.udNrOfElementsInArray = (*hgps).rx.length;
fp.Program.pArray = (void*)(*hgps).rx.buffer;
rRetVal = Flash( Program, &fp );

sfmpos += (*hgps).rx.length;
sfmDisable();
for (i=0; i<5; i++)
    rpms[i]=counter;
}
counter=0;

```

```
}  
}
```

```
void debugKill(void)  
{  
// if( dusart0.tx.buffer != NULL ){  
// free( dusart0.tx.buffer );  
// dusart0.tx.buffer = NULL;  
// }  
// if( dusart0.rx.buffer != NULL ){  
// free( dusart0.rx.buffer );  
// dusart0.rx.buffer = NULL;  
// }  
// if( dusart1.tx.buffer != NULL ){  
// free( dusart1.tx.buffer );  
// dusart1.tx.buffer = NULL;  
// }  
// if( dusart1.rx.buffer != NULL ){  
// free( dusart1.rx.buffer );  
// dusart1.rx.buffer = NULL;  
// }  
}  
#endif
```

```
/* mqpdipsw.h */
```

```
/****** header file for DIPSW interface *****/
```

```
#ifndef MQPDIPSW_H  
#define MQPDIPSW_H
```

```
void dipswInit(void);  
char dipswRead(void);
```

```
#define DIPSWDIR P1DIR /* dipswitch directional port */  
#define DIPSWSEL P1SEL /* dipswitch control port */  
#define DIPSWIN P1IN /* dipswitch input port */  
#define DIPSWBITS (BIT3|BIT2|BIT1|BIT0) /* dipswitch bits used */  
#define DIPSWSHR 0 /* amount to shift read in bits to the right */
```

```
/* modes of operation */
```

```
##define DATALOGMODE 0x0F /* 1111 */  
##define UPLOADMODE 0x0E /* 1110 */  
##define DEBUGMODE 0x03 /* 0111 */  
#define DATALOGMODE 0x00 /* 0000 */
```

```

#define DOWNLOADMODE 0x01 /* 0001 */
#define DEBUGMODE 0x02 /* 0010 */
#define DELETEMODE 0x04 /* 0100 */

/***** init_dipsw() *****/
* function to initialize the dipswitch
*/
void dipswInit(void)
{
    DIPSWDIR &= ~(DIPSWBITS); /* set dipswitch port to input direction */
    DIPSWSEL &= ~(DIPSWBITS); /* set dipswitch port I/O option */
}

/***** dipswRead() *****/
* function to read from the dipswitch
*/
char dipswRead(void)
{
    char input;

    /* read in from dipswitch, mask off unneeded bits */
    input = DIPSWIN & DIPSWBITS;

    /* shift bits to the right to ignore unused bits */
    input >>= DIPSWSHR;

    return( input ); /* return what was read in */
}

#endif

/* mqpdownload.h */
/***** header file for download mode *****/

#ifndef MQPDOWNLOAD_H
#define MQPDOWNLOAD_H

#include "mqptimer.h"
#include "mqpusb.h"
#include "mqpsfm.h"

void downloadMode(void);
void downloadInit(void);
void downloadLoop(void);
void downloadKill(void);

```

```

void downloadMode(void)
{
    /* initialize */
    downloadInit();

    /* run loop */
    while( dipswRead() == 0x03 ){
        downloadLoop();
    }

    downloadKill();
}

void downloadInit(void)
{
    /* set up USB for UART1 */
    _DINT();
    usbDisable();
    usbInit();
    usbPost();

    /* print menu to USB */
    usbPrintMenu();

    /* set up SFM for SPI0 */
    sfmInit0();
    sfmPost();
    sfmreadpos = 0;

    // /* set up Timer A */
    // taInit();
}

void downloadLoop(void)
{
    /* set up timer A to wait for 2.5 seconds */
    hwDelay( &timerA, 3, 0x7FFF );
    (*husb).rx.status=WORK;
    _EINT();

    /* wait for USB command or time out */
    while( timerA.status != DONE && (*husb).rx.status != FREE );

    // /* if given command to download memory */
    // if( (*husb).rx.status == FREE )

```

```

    {
// /* look at latest character */
// switch( (*husb).rx.buffer[ (*husb).rx.length ] )
// {
// case 'C':
// case 'c': /* clear memory */
//   usbPrint( "Clearing memory.\n\r" );
//   sfmErase();
//   usbPrint( "Memory cleared.\n\r" );
//   break;
// case 'D':
// case 'd': /* download memory */
//   usbPrint( "Downloading memory.\n\r" );
//   usbDump();
//   usbPrint( "Memory downloaded.\n\r" );
//   break;
// case 'H':
// case 'h': /* help - print menu again */
//   usbPrintMenu();
//   break;
// default:
//   usbPrint( "Not a valid command.\n\r" );
// }
}

usbDisable();
_DINT();
}

void downloadKill(void)
{
/* free up buffers, nullify pointers */
dusart0.tx.buffer = NULL;
dusart0.rx.buffer = NULL;
dusart1.tx.buffer = NULL;
dusart1.rx.buffer = NULL;
}
#endif

/* mqpgps.h */
/***** header file for GPS *****/

#ifndef MQPGPS_H
#define MQPGPS_H

```

```
void gpsInit(void);
void gpsEnable(void);
void gpsDisable(void);
void gpsRecv(void);
void gpsSend(void);
void gpsPost(void);
```

```
char stringType[6];
char numSats[3];
```

```
#include "mqpusart.h"
#include <stdlib.h>
```

```
/****** gpsInit() *****/
```

```
 * function to initialize USART 0
 * post: USART 0 set up for 4800 baud 8-none-1 connection
 * with no flow control and interrupts are enabled
 */
```

```
void gpsInit(void){
    P3SEL |= (BIT4|BIT5); /* P3.4,5 = USART0 TXD/RXD */
    UCTL0 |= CHAR; /* 8-bit character, SWRST=1 */
    UTCTL0 |= SSEL0; /* UCLK = ACLK */
```

```
 /* 4800 baud */
    UBR0 = 0x06;
    UBR10 = 0x00;
    UMCTL0 = 0x77;
```

```
    UCTL0 &= ~SWRST; /* initialize USART0 state machine */
    IE1 |= URXIE0 + UTXIE0; /* enable USART0 RX/TX interrupts */
    IFG1 &= ~UTXIFG0; /* clear initial flag on POR */
```

```
    dusart0.tx.buffer = NULL;
    dusart0.tx.index = 0;
    dusart0.tx.length = 0;
    dusart0.tx.size = 0;
    dusart0.tx.status = FREE;
```

```
    dusart0.rx.buffer = char180;
    dusart0.rx.length = 0;
    dusart0.rx.size = 180;
    dusart0.rx.status = FREE;
    dusart0.rx.stopbyte=0x0A; /* end of sentence */
```

```
    dusart0.port = UART0;
    dusart0.recv = gpsRecv;
```

```

dusart0.send    = gpsSend;

hgps = &dusart0;

// numSats = NULL;
}

/***** gpsEnable() *****/
* function to enable interrupts for GPS
*/
void gpsEnable(void)
{
    ME1 |= URXE0;
    swDelay( 500, DMSEC );
}

/***** gpsDisable() *****/
* function to disable interrupts for GPS
*/
void gpsDisable(void)
{
    ME1 &= ~(URXE0);
}

/***** gpsRecv() *****/
* function to receive characters from UART and buffer them
* from PORT 2 of GPS using NMEA interface
*/
void gpsRecv(void)
{
    if( (*hgps).rx.length < (*hgps).rx.size ){
        (*hgps).rx.buffer[ (*hgps).rx.length ] = RXBUF0;
        (*hgps).rx.length++;
    }
    if( RXBUF0 == (*hgps).rx.stopbyte ){ /* stop byte reached? */
        (*hgps).rx.status=FREE;
    }

    // /* start over buffer when a '$' has been hit */
    // if( RXBUF0 == '$' ){
    //     rx0.length = 0;
    // }
    // /* if it's hit the end of the sentence, stop recording */
    // if( RXBUF0 == 0x0A ){
    //     rx0.length = 0;
    // }

```

```

// /* keep appending data to buffer unless it has overflowed */
// else if( rx0.index < MAXBUFFER ){
//   rx0.buffer[rx0.index++]=RXBUF0; /* store in buffer, increment index */
// }
//
// rx0.status=1;
}

```

```

/***** gpsSend() *****/

```

```

* function to transmit next character in buffer via UART
* to PORT 1 of GPS using TSIP interface
*/

```

```

void gpsSend(void)
{
}

```

```

void gpsPost(void)
{
}

```

```

#endif

```

```

/* mqpio.h */

```

```

/***** header file for IO structures *****/

```

```

#ifndef MQPIOBUFFER_H
#define MQPIOBUFFER_H

```

```

enum IOstate{
    FREE, WORK
};

```

```

typedef struct _IObuffer{
    char* buffer; /* the buffer */
    int length; /* length of used buffer */
    int size; /* max size of buffer */
    enum IOstate status; /* current status */
    union{
        int index; /* position in buffer for transmitting */
        char stopbyte; /* stop byte for receiving */
    };
} IObuffer;

```

```

enum IOport {
    UART0, UART1, SPI0, SPI1
}

```

```

};

typedef struct _IOdevice{
    IObuffer rx;        /* reception buffer */
    IObuffer tx;        /* transmission buffer */
    void (*recv)(void); /* receive function pointer */
    void (*send)(void); /* send function pointer */
    enum IOport port;   /* port this device is attached to */
} IOdevice, *IOhandle;

#endif

/* mqplcd.h */
/****** header file for LCD interface *****/
/*
    Author: Carlton C. Stedman II
    Last update: 2/19/2007
*/

#ifndef MQPLCD_H
#define MQPLCD_H

void lcdInit(void);
void lcdClock(void);
void lcdWrite(unsigned char);
void lcdCmd(unsigned char);
void lcdPrint(char);
void lcdPrints(char* );

#include "mqptimer.h"
#include <string.h>

/* LCD Control Port defines */
#define LCDCTRLDIR P4DIR
#define LCDCTRLSEL P4SEL
#define LCDCTRLOUT P4OUT
#define LCDEN     BIT1 /* LCD enable */
#define LCDRW     BIT2 /* LCD R/W */
#define LCDRS     BIT3 /* LCD RS */

/* LCD Data Port defines */
#define LCDDIR     P4DIR
#define LCDSEL     P4SEL
#define LCDOUT     P4OUT

```

```

#define LCDCOLS 16
#define LCDROWS 2

/*****
/
/***** LCD using 8-Bit Interface *****/
/*****
/
#ifdef LCD8BIT
#define lcdLine1() { lcdCmd( 0x80 ); }
#define lcdLine2() { lcdCmd( 0xC0 ); }
#define lcdClear() { lcdCmd( 0x01 ); lcdLine1(); }

/***** init_lcd() *****/
void lcdInit(void)
{
    /* wait for 20ms */
    swDelay( 20, DMSEC );

    /* function set: 8-bit interface */
    lcdCmd( 0x30 );

    /* wait for 5ms */
    swDelay( 5, DMSEC );

    /* function set: 8-bit interface */
    lcdCmd( 0x30 );

    /* wait for 200us */
    swDelay( 2, DHUSEC );

    /* function set: 8-bit interface */
    lcdCmd( 0x30 );

    /* function set: 8-bit interface, two-line display with 5x8 char font */
    lcdCmd( 0x38 );

    /* turn display off */
    lcdCmd( 0x08 );

    /* turn display clear */
    lcdClear();

    /* set for entry mode */
    lcdCmd( 0x06 );

```

```

/* turn on display, cursor off */
lcdCmd( 0x0C );
}

/***** lcdClock() *****/
/* function to flash the enable on the LCD */
void lcdClock(void)
{
    LCDCTRLDIR |= (LCDEN);
    LCDCTRLSEL &= ~(LCDEN);
    LCDCTRLLOUT &= ~(LCDEN);    /* clear enable */

    swDelay( 4, DMSEC );    /* wait 4 ms */
    LCDCTRLLOUT |= (LCDEN);    /* set enable */

    swDelay( 2, DMSEC );    /* wait 2 ms */
    LCDCTRLLOUT &= ~(LCDEN);    /* clear enable */
}

/***** lcdWrite() *****/
/* function to write data onto the data lines of the LCD */
void lcdWrite( unsigned char databus )
{
    /* set P4.0-4.7 to output direction */
    LCDDIR |= (BIT0|BIT1|BIT2|BIT3|BIT4|BIT5|BIT6|BIT7);
    /* set P4.0-4.7 I/O option */
    LCDSEL &= ~(BIT0|BIT1|BIT2|BIT3|BIT4|BIT5|BIT6|BIT7);
    /* P4.0-4.7 output = databus */
    LCDOUT = databus;

    lcdClock();
}

/***** lcdPrint() *****/
/* function to print a character at the cursor on the LCD */
void lcdPrint( char databus )
{
    LCDCTRLDIR |= (LCDRW|LCDRS);    /* set P2.2-2.3 to output direction */
    LCDCTRLSEL &= ~(LCDRW|LCDRS);    /* P2.2-2.3 I/O option */
    LCDCTRLLOUT &= ~(LCDRW);    /* clear R/W flag on P2.2 */
    LCDCTRLLOUT |= (LCDRS);    /* set RS flag on P2.3 */

    lcdWrite( databus );
}

#endif

```

```

/*****
/
/***** LCD using 4-Bit Interface *****/
/*****
/
#ifdef LCD4BIT
#define lcdLine1() { lcdCmd( 0x80 ); lcdCmd( 0x00 ); }
#define lcdLine2() { lcdCmd( 0xC0 ); lcdCmd( 0x00 ); }
#define lcdClear() { lcdCmd( 0x00 ); lcdCmd( 0x10 ); lcdLine1(); }

void lcdInit(void)
{
    /* wait for 20ms */
    swDelay( 20, DMSEC );

    /* function set: 8-bit interface */
    lcdCmd( 0x30 );

    /* wait for 5ms */
    swDelay( 5, DMSEC );

    /* function set: 8-bit interface */
    lcdCmd( 0x30 );

    /* wait for 200us */
    swDelay( 2, DHUSEC );

    /* function set: 8-bit interface */
    lcdCmd( 0x30 );

    /* function set: 4-bit interface */
    lcdCmd( 0x20 );

    /* function set: 4-bit interface, two-line display with 5x8 char font */
    lcdCmd( 0x20 );
    lcdCmd( 0x80 );

    /* turn display off */
    lcdCmd( 0x00 );
    lcdCmd( 0x80 );

    /* turn display on */
    lcdCmd( 0x00 );
    lcdCmd( 0x10 );

```

```

/* set for entry mode */
lcdCmd( 0x00 );
lcdCmd( 0x60 );

/* turn on display, cursor off */
lcdCmd( 0x00 );
lcdCmd( 0xC0 );
}

/***** lcdClock() *****/
/* function to flash the enable on the LCD */
void lcdClock(void)
{
    LCDCTRLDIR |= (LCDEN);
    LCDCTRLSEL &= ~(LCDEN);
    LCDCTRLLOUT &= ~(LCDEN);    /* clear enable */

    swDelay( 8, DMSEC );    /* wait 8 ms */
    LCDCTRLLOUT |= (LCDEN);    /* set enable */

    swDelay( 4, DMSEC );    /* wait 4 ms */
    LCDCTRLLOUT &= ~(LCDEN);    /* clear enable */
}

/***** lcdWrite() *****/
/* function to write data onto the data lines of the LCD */
void lcdWrite( unsigned char databus )
{
    /* set P4.4-4.7 to output direction */
    LCDDIR |= (BIT4|BIT5|BIT6|BIT7);
    /* set P4.4-4.7 I/O option */
    LCDSEL &= ~(BIT4|BIT5|BIT6|BIT7);
    /* P4.4-4.7 output = databus */
    LCDOUT &= 0x0F;
    LCDOUT += databus;

    lcdClock();
}

/***** lcdPrint() *****/
/* function to print a character at the cursor on the LCD */
void lcdPrint( char databus )
{
    LCDCTRLDIR |= (LCDRW|LCDRS);    /* set P4.2-4.3 to output direction */
    LCDCTRLSEL &= ~(LCDRW|LCDRS);    /* P4.2-4.3 I/O option */
    LCDCTRLLOUT &= ~(LCDRW);    /* clear R/W flag on P4.2 */
}

```

```

LCDCTRLLOUT |= (LCDRS);          /* set RS flag on P4.3 */

lcdWrite( databus & 0xF0 );      /* send high byte of data */
lcdWrite( databus << 4 );        /* send low byte of data */
}

#endif

/***** lcdCmd() *****/
/* function to send a command to the LCD */
void lcdCmd( unsigned char databus )
{
    LCDCTRLDIR |= (LCDRW|LCDRS);  /* set P4.2-4.3 to output direction */
    LCDCTRLSEL &= ~(LCDRW|LCDRS); /* P4.2-4.3 I/O option */
    LCDCTRLLOUT &= ~(LCDRW|LCDRS); /* P4.2-4.3 output = 00 (off) */

    lcdWrite( databus );
}

/***** lcdPrints() *****/
/* function to print a string at the cursor on the LCD */
void lcdPrints( char* buffer )
{
    char i;

    for( i=0; i<strlen(buffer); i++){
        /* check to see if we should skip down a line */
        if( buffer[i]=='\n' ){
            lcdLine2(); /* set DDRAM address to line 2, position 1 */
        } else{
            lcdPrint( buffer[i] );
        }
    }
}

#endif

/* mqpsfm.h */
/***** header file for SFM *****/

#ifndef MQPSFM_H
#define MQPSFM_H

void sfmInit0(void);
void sfmInit1(void);

```

```
void sfmEnable(void);
void sfmDisable(void);
void sfmRecv(void);
void sfmSend(void);
void sfmPost(void);
void sfmErase(void);
```

```
int sfmBuffer(char*, int);
int sfmFlush();
```

```
#define SFMFULL 0x0A
#define SFMNFULL 0xA0
#define SFMFLUSH 0x0C
#define SFMNFLUSH 0xC0
```

```
#include "mqpusart.h"
#include <stdlib.h>
```

```
#define SPIHOLD BIT4
#define SPIS BIT5
#define SPIW BIT6
```

```
/* SPI memory library */
#include "c2082.h"
#include "c2082.c"
#include "Serialize.h"
#include "Serialize.c"
```

```
/* GLOBALS */
ParameterType fp; /* contains all flash memory parameters */
ReturnType rRetVal; /* return type enum for flash memory */
char foobarbaz;
```

```
IObuffer sfmbuffer;
int sfmpos;
```

```
/****** init_spi0() *****/
* function to initialize USART 0 for SPI
* post: USART 0 set up for 3-wire SPI connection, 8-bit master
*/
```

```
void sfmInit0(void){
    P3SEL |= (BIT1|BIT2|BIT3); /* setup P3 for SPI mode */
    P3OUT = 0x20;
    P3DIR |= 0x30;
```

```
    U0ME |= UTXE0 + URXE0; /* enable USART0 transmit and receive modules */
```

```

UCTL0 = CHAR + SYNC + MM;    /* 8-bit, SPI, Master */
UTCTL0 = CKPL + SSEL1 - STC; /* Polarity, SMCLK, 3-wire */
U0BR0 = 0x02;                /* SPICLK = SMCLK/2 */
U0BR1 = 0x00;
UMCTL0 = 0x00;

IE1 |= URXIE0 + UTXIE0;     /* RX and TX interrupt enable */
UCTL0 &= ~SWRST;           /* enable SPI */
IFG1 &= ~UTXIFG0;          /* clear initial flag on POR */

P2DIR |= (SPIHOLD|SPIS|SPIW); /* setup for P2.4-6 for HOLD, S and W/VPP */
P2SEL &= ~(SPIHOLD|SPIS|SPIW); /* set I/O option for output */
P2OUT |= (SPIS);            /* falling edge of select after power-up, turn on */
swDelay( 2, DHUSEC );       /* wait 200 us */
P2OUT &= ~(SPIHOLD|SPIS|SPIW); /* clear hold, select and disable write on
slave (all active low)*/

dusart0.tx.buffer = char32;
dusart0.tx.length = 0;
dusart0.tx.index = 0;
dusart0.tx.size = 32;
dusart0.tx.status = FREE;

dusart0.rx.buffer = char125;
dusart0.rx.length = 0;
dusart0.rx.size = 125;
dusart0.rx.status = FREE;

dusart0.port = SPI0;
dusart0.recv = sfmRecv;
dusart0.send = sfmSend;

hsfm = &dusart0;

sfmbuffer.buffer = char256;
sfmbuffer.index = 0;
sfmbuffer.length = 0;
sfmbuffer.size = 256;
sfmbuffer.status = FREE;

sfmpos = 0;
}

/***** init_spi1() *****/
* function to initialize USART 1 for SPI
* post: USART 1 set up for 3-wire SPI connection, 8-bit master

```

```

*/
void sfmInit1(void){
    P5SEL |= (BIT1|BIT2|BIT3); /* setup P5 for SPI mode */
    P5OUT = 0x20;
    P5DIR |= 0x30;

    U1ME |= UTXE1 + URXE1; /* enable USART1 transmit and receive modules */
    UCTL1 = CHAR + SYNC + MM; /* 8-bit, SPI, Master */
    UTCTL1 = CKPL + SSEL1 - STC; /* Polarity, SMCLK, 3-wire */
    U1BR0 = 0x02; /* SPICLK = SMCLK/2 */
    U1BR1 = 0x00;
    UMCTL1 = 0x00;

    IE2 |= URXIE1 + UTXIE1; /* RX and TX interrupt enable */
    UCTL1 &= ~SWRST; /* SPI enable */
    IFG2 &= ~UTXIFG1; /* clear initial flag on POR */

    P2DIR |= (SPIHOLD|SPIS|SPIW); /* setup for P2.4-6 for HOLD, S and W/VPP */
    P2SEL &= ~(SPIHOLD|SPIS|SPIW); /* set I/O option for output */
    P2OUT |= (SPIS); /* falling edge of select after power-up, turn on */
    swDelay( 2, DHUSEC); /* wait 200 us */
    P2OUT &= ~(SPIHOLD|SPIS|SPIW); /* clear hold, select and disable write on
slave (all active low)*/

    dusart1.tx.buffer = char32;
    dusart1.tx.index = 0;
    dusart1.tx.length = 0;
    dusart1.tx.size = 32;
    dusart1.tx.status = FREE;

    dusart1.rx.buffer = char125;
    dusart1.rx.length = 0;
    dusart1.rx.size = 125;
    dusart1.rx.status = FREE;

    dusart1.port = SPI1;
    dusart1.recv = sfmRecv;
    dusart1.send = sfmSend;

    hsfm = &dusart1;

    sfmbuffer.buffer = char256;
    sfmbuffer.index = 0;
    sfmbuffer.length = 0;
    sfmbuffer.size = 256;
    sfmbuffer.status = FREE;

```

```

    sfmpos = 0;
}

/***** sfmEnable() *****/
* function to enable interrupts for SFM
*/
void sfmEnable(void)
{
    if( (*hsfm).port == SPI0 ){
        ME1 |= USPIE0;          /* enable the SPI module */
    } else if( (*hsfm).port == SPI1 ){
        ME2 |= USPIE1;          /* enable the SPI module */
    }
}

/***** sfmDisable() *****/
* function to disable interrupts for SFM
*/
void sfmDisable(void)
{
    if( (*hsfm).port == SPI0 ){
        ME1 &= ~(USPIE0);      /* enable the SPI module */
    } else if( (*hsfm).port == SPI1 ){
        ME2 &= ~(USPIE1);      /* enable the SPI module */
    }
}

/***** sfmRecv() *****/
* function to
*/
void sfmRecv(void){
    (*hsfm).rx.status=FREE;

    if( (*hsfm).rx.length<(*hsfm).rx.size ){
        if( (*hsfm).port == SPI0 ){
            (*hsfm).rx.buffer[ (*hsfm).rx.length ] = RXBUF0;
        } else if( (*hsfm).port == SPI1 ){
            (*hsfm).rx.buffer[ (*hsfm).rx.length ] = RXBUF1;
        }
    }

    (*hsfm).rx.length++;
}

/***** sfmSend() *****/

```

```

* function to
*/
void sfmSend(void){
    (*hsfm).tx.status=WORK;

    (*hsfm).tx.index++;
    if( (*hsfm).tx.index < (*hsfm).tx.length ){
        if( (*hsfm).port == SPI0 ){
            TXBUF0 = (*hsfm).tx.buffer[(*hsfm).tx.index];
        } else if( (*hsfm).port == SPI1 ){
            TXBUF1 = (*hsfm).tx.buffer[(*hsfm).tx.index];
        }
    }
    else{
        (*hsfm).tx.status = FREE;
        (*hsfm).tx.index = 0;
        (*hsfm).tx.length = 0;
    }
}

/***** sfmPost() *****/
* function to do a Power-on Self Test for the SFM
*/
void sfmPost(void){
    sfmEnable();
    _EINT();

    /* FIX THIS: DON'T LOOK FOREVER! */
    while( rRetVal != Flash_Success ){
        /* read manufacturer identification */
        rRetVal=Flash(ReadManufacturerIdentification, &fp );
        foo = fp.ReadManufacturerIdentification.ucManufacturerIdentification;

#ifdef DEBUG
        lcdClear();
        lcdPrints( FlashErrorStr( rRetVal ) );
        lcdPrints( "\n" );
        lcdPrints( FlashErrorStr( rRetVal ) + 8 );
        swDelay( 1, DHSEC );
#endif
    }

    /* read device identification */
    rRetVal=Flash(ReadDeviceIdentification, &fp );
    foo = fp.ReadDeviceIdentification.ucDeviceIdentification;

```

```

    sfmDisable();
    _DINT();
}

void sfmErase(void)
{
    lcdClear();
    lcdPrint( '5' );
    swDelay( 3, DHSEC );
    lcdPrint( '4' );
    swDelay( 3, DHSEC );
    lcdPrint( '3' );
    swDelay( 3, DHSEC );
    lcdPrint( '2' );
    swDelay( 3, DHSEC );
    lcdPrint( '1' );
    swDelay( 3, DHSEC );
    lcdPrint( '0' );
    swDelay( 3, DHSEC );
    lcdPrints( "\n" );
    lcdPrints( "Deleting" );

    sfmEnable();
    _EINT();

    /* erase all memory */
    rRetVal = Flash( BulkErase, &fp);

    sfmDisable();
    _DINT();

    lcdClear();
    lcdPrints( FlashErrorStr( rRetVal ) );
    lcdPrints( "\n" );
    lcdPrints( FlashErrorStr( rRetVal ) + 8 );
    swDelay( 3, DHSEC );
}

/***** sfmBuffer() *****/
/* function to buffer more data to send for SFM or return saying it's full
*/
int sfmBuffer( char* buffer, int length )
{
    /* is it full or has too much been stored to buffer this? */
    if( sfmbuffer.length + length < sfmbuffer.size ){
        strncpy( sfmbuffer.buffer + sfmbuffer.length, buffer, length );
    }
}

```

```

    sfmbuffer.length += length;
} else{ /* yep, say it's full */
    return SFMFULL;
}

return SFMNFULL; /* otherwise, say it's not full */
}

/***** sfmFlush() *****/
* function to flush data from the SFM buffer to the SFM
* pre: assumes interrupts enabled
*/
int sfmFlush(void)
{
    sfmEnable();
    _EINT();

    /* write some data */
    fp.Program.udAddr = sfmpos;
    fp.Program.udNrOfElementsInArray = sfmbuffer.length;
    fp.Program.pArray = (void*)sfmbuffer.buffer;
    rRetVal = Flash( Program, &fp );

    lcdClear();
    lcdPrints( FlashErrorStr( rRetVal ) );
    lcdPrints( "\n" );
    lcdPrints( FlashErrorStr( rRetVal ) + 8 );
    swDelay( 3, DHSEC );

    sfmpos += sfmbuffer.length; /* update position */

    /* say it's empty */
    sfmbuffer.length = 0;

    sfmDisable();
    _DINT();

    return SFMFLUSH;
}

#endif

/* mqptimer.h */
/*****header file for timer functions and structures *****/

```

```
#ifndef MQPTIMER_H
#define MQPTIMER_H
```

```
enum TimerStatus {
    DONE, RUNNING
};
```

```
typedef struct _Timer{
    void (*start)(void); /* function pointer to start() function */
    void (*stop)(void); /* function pointer to stop() function */
    int count;          /* number of iterations to do */
    int counter;        /* current number of iterations run */
    int offset;         /* offset to start TAR at */
    enum TimerStatus status; /* status of timer */
} Timer;
```

```
void swDelay(unsigned int, unsigned int); // simple SW delay loop
void hwDelay(Timer*, unsigned int, unsigned int);
```

```
void taInit(void); /* initialize Timer A */
void taStart(void); /* start Timer A */
void taStop(void); /* stop Timer A */
```

```
Timer timerA;
```

```
void taInit(void)
{
    timerA.count = 1;
    timerA.offset = 0;
    timerA.status = DONE;
    timerA.start = taStart;
    timerA.stop = taStop;
}
```

```
void taStart(void)
{
    timerA.counter = 0;
    timerA.status = RUNNING;
    TAR = timerA.offset; /* set TAR at offset value */
    TACTL = TASSEL_1 + TAIE + MC_2; /* use ACLK, interrupt-driven, cont counter */
}
```

```
void taStop(void)
{
    TACTL = MC_0; /* set to stop counting */
}
```

```

    timerA.status = DONE;
}

#pragma vector=TIMERA1_VECTOR
__interrupt void timerA_interrupt(void)
{
    timerA.counter++; /* increment counter */

    /* check if done counting */
    if( timerA.counter >= timerA.count ){
        taStop(); /* stop the clock */
    } else{
        TACTL &= ~(TAIFG); /* otherwise, clear TAIFG flag */
    }
}

/***** hwDelay() *****/
void hwDelay(Timer *ptimer, unsigned int count, unsigned int offset )
{
    (*ptimer).count = count;
    (*ptimer).offset = offset;
    ((*ptimer).start)();
}

#define DHSEC 65535 /* half-second delay multiplier */
#define DMSEC 33 /* millisecond delay multiplier */
#define DHUSEC 3 /* 100 us delay multiplier */

/***** swDelay() *****/
void swDelay(unsigned int max_cnt, unsigned int multiplier)
{
    unsigned int cnt1=0, cnt2;

    while (cnt1 < max_cnt)
    {
        cnt2 = 0;
        while (cnt2 < multiplier)
            cnt2++;
        cnt1++;
    }
}
#endif

/* mqpuart.h */
/***** header file for UART interface *****/

```

```

#ifndef MQPUART_H
#define MQPUART_H

#include "mqp.h"

void init_uart(void);

/***** init_uart() *****/
* function to initialize USART 0
*/
void init_uart(void){
    P3SEL = 0x30;           // P3.3,4 = USART0 TXD/RXD
    ME1 |= UTXE0 + URXE0;  // Enabled USART0 TXD/RXD
    UCTL0 |= CHAR;         // 8-bit character, SWRST=1
    UTCTL0 |= SSEL0;       // UCLK = ACLK

    /* 1 baud */
    UBR00 = 0x00;
    UBR10 = 0x80;
    UMCTL0 = 0x00;

    UCTL0 &= ~SWRST;       // Initialize USART state machine
    IE1 |= URXIE0 + UTXIE0; // Enable USART0 RX/TX interrupt
    IFG1 &= ~UTXIFG0;     // Clear initial flag on POR
    _BIS_SR(GIE);         // Enable interrupts
}

/***** usart0_tx() *****/
* interrupt service routine for USART 0 transmission
* pre: to be entered, GIE must be set and a new value has been
*   written to TXBUF0
*/
#pragma vector=UART0TX_VECTOR
__interrupt void usart0_tx (void)
{
}

/***** usart0_rx() *****/
* interrupt service routine for USART 0 reception
* pre: to be entered, GIE must be set and a new value has been
*   recieved in RXBUF0
*/
#pragma vector=UART0RX_VECTOR
__interrupt void usart0_rx (void)
{
}

```

```

    TXBUF0=RXBUF0;
}

#endif

/* mquart.h */
/***** header file for UART interface *****/

#ifndef MQPUSART_H
#define MQPUSART_H

#include "mqpio.h"
IOdevice dusart0, dusart1; /* IOdevices on USART ports */
IOhandle husb, hsfm, hgps; /* IO handles for USB, SFM and GPS */

/***** usart0_tx() *****/
* interrupt service routine for USART 0 transmission
* pre: to be entered, GIE must be set and a new value has been
* written to TXBUF0
*/
#pragma vector=USART0TX_VECTOR
__interrupt void usart0_tx (void)
{
    dusart0.tx.status=WORK;
    (*dusart0.send)();
}

/***** usart0_rx() *****/
* interrupt service routine for USART 0 reception
* pre: to be entered, GIE must be set and a new value has been
* recieved in RXBUF0
* post: latest received character added to buffer, unless buffer overflow
*/
char foo;
#pragma vector=USART0RX_VECTOR
__interrupt void usart0_rx (void)
{
    dusart0.rx.status=WORK;
    (*dusart0.recv)();
    foo=RXBUF0; /* pull out of buffer */
}

/***** usart1_tx() *****/
* interrupt service routine for USART 1 transmission
* pre: to be entered, GIE must be set and a new value has been

```

```

*   written to TXBUF1
*/
#pragma vector=UART1TX_VECTOR
__interrupt void usart1_tx (void)
{
    dusart1.tx.status=WORK;
    (*dusart1.send)();
}

/***** usart1_rx() *****/
* interrupt service routine for USART 1 reception
* pre: to be entered, GIE must be set and a new value has been
*   recieved in RXBUF1
* post: latest received character added to buffer, unless buffer overflow
*/
#pragma vector=UART1RX_VECTOR
__interrupt void usart1_rx (void)
{
    dusart1.rx.status=WORK;
    (*dusart1.recv)();
    foo=RXBUF1;      /* pull out of buffer */
}

#endif

/* mqpusb.h */
/***** header file for USB *****/

#ifndef MQPUSB_H
#define MQPUSB_H

void usbInit(void);
void usbEnable(void);
void usbDisable(void);
void usbRecv(void);
void usbSend(void);
void usbPost(void);
void usbPrint(char*);
void usbPrintMenu(void);
void usbDump(void);

#include "mqpusart.h"
#include <string.h>
#include <stdlib.h>

```

```

/***** usbInit() *****/
* function to initialize USART 1
* post: USART 1 set up for 921600 baud 8-none-1 connection
* using a 8 MHz crystal placed in XT2
* with no flow control and interrupts are enabled
*/
void usbInit(void){
    P3SEL |= (BIT6|BIT7);           // P3.6,7 = USART1 TXD/RXD

    BCSCCTL1 &= ~XT2OFF;           // XT2on
    BCSCCTL2 |= SELM_2 + SELS;     // MCLK= SMCLK= XT2 (safe)

    UCTL1 |= CHAR;                 // 8-bit character
    UTCTL1 |= SSEL1;               // UCLK = SMCLK

    /* 921600 baud */
    UBR01 = 0x08;
    UBR11 = 0x00;
    UMCTL1 = 0x5B;                 // modulation

    UCTL1 &= ~SWRST;               /* initialize USART1 state machine */
    IE2 |= URXIE1 + UTXIE1;        /* enable USART1 RX/TX interrupts */
    IFG2 &= ~UTXIFG1;              /* clear initial flag on POR */

    dusart1.tx.buffer = char180;
    dusart1.tx.index = 0;
    dusart1.tx.length = 0;
    dusart1.tx.size = 180;
    dusart1.tx.status = FREE;

    dusart1.rx.buffer = char1;
    dusart1.rx.length = 0;
    dusart1.rx.size = 1;
    dusart1.rx.status = FREE;
    dusart1.rx.stopbyte='\n'; /* end of input */

    dusart1.port = UART1;
    dusart1.recv = usbRecv;
    dusart1.send = usbSend;

    husb = &dusart1;
}

/***** usbEnable() *****/
* function to enable interrupts for USB
*/

```

```

void usbEnable(void)
{
    ME2 |= UTXE1 + URXE1;          /* enable USART1 TXD/RXD */
}

/***** usbDisable() *****/
* function to disable interrupts for USB
*/
void usbDisable(void)
{
    ME2 &= ~(UTXE1);
    ME2 &= ~(URXE1);
}

/***** usbRecv() *****/
* function to receive data over USB
*/
void usbRecv(void)
{
    (dusart1).rx.status=FREE;

    if( (dusart1).rx.length<(dusart1).rx.size ){
        (dusart1).rx.buffer[ (dusart1).rx.length ] = RXBUF1;
        (dusart1).rx.length++;
    } else {
        (dusart1).rx.buffer[0] = RXBUF1;
        (dusart1).rx.length = 0;
    }
}

/***** usbSend() *****/
* function to send data over USB
*/
void usbSend(void)
{
    (dusart1).tx.status=WORK;

    (dusart1).tx.index++;
    if( (dusart1).tx.index < (dusart1).tx.length ){
        TXBUF1 = (dusart1).tx.buffer[(dusart1).tx.index];
    }
    else {
        (dusart1).tx.length = 0;
        (dusart1).tx.index = 0;
        (dusart1).tx.status = FREE;
    }
}

```

```

}

/***** usbPost() *****/
* function to do Power On Self-Test of USB
*/
void usbPost(void)
{
  usbEnable();
  _EINT();

  usbPrint( "GPS Road Roughness\n\rv0.9\n\r" );

  usbDisable();
  _DINT();
}

/***** usbPrint() *****/
* function to print string to USB
* pre: assumes USB interrupts enabled
*/
void usbPrint( char* buffer )
{
  // swDelay( 5, DMSEC ); /* wait just a tiny bit */

  /* copy passed-in string into transmit buffer */
  strncpy( (dusart1).tx.buffer, buffer, strlen(buffer) );
  (dusart1).tx.index=0;
  (dusart1).tx.length=strlen( buffer );

  /* send the first char to initialize transfer */
  TXBUF1 = (dusart1).tx.buffer[(dusart1).tx.index];
}

/***** usbPrintMenu() *****/
* function to print menu of commands to USB
* pre: assumes USB interrupts enabled
*/
void usbPrintMenu(void)
{
  usbPrint( "\n\r USB Menu\n\r-----\n\r  c clear memory\n\r\
  d download from memory\n\r  h print this help menu again\n\r" );
}

/***** usbPrint() *****/
* function to print entire memory out to USB
* pre: assumes interrupts enabled

```

```

*/
void usbDump(void)
{
    sfmEnable();

    sfmreadpos = 0;

    while( sfmreadpos < 0x2000000 ){
        fp.Read.udAddr = sfmreadpos;
        fp.Read.udNrOfElementsToRead = (dusart1).tx.size;
        fp.Read.pArray = (void*)(dusart1).tx.buffer;
        rRetVal=Flash(Read, &fp );

        sfmreadpos += (dusart1).tx.size;

        /* send along first char to initiate transfer */
        (dusart1).tx.index=0;
        (dusart1).tx.length=(dusart1).tx.size;
        TXBUF1 = (dusart1).tx.buffer[(dusart1).tx.index];
    }

    sfmDisable();
}

#endif

/* SPI MEMORY TEST PROGRAM */

#include "msp430x16x.h"    // Definitions, constants, etc for msp430F169
#include "mqp.h"
#include "mqplcd.h"
#include "mqpusart.h"

/* SPI memory library */
#include "c2082.h"
#include "c2082.c"
#include "Serialize.h"
#include "Serialize.c"

// ***** FUNCTION DECLARATIONS *****
void init_sys(void);    // MSP430 Initialization routine

void usbPrintMenu(void);

```

```

/* modes */
void mode_datalog(void);
void mode_usb(void);
void mode_selftest(void);
void mode_default(void);

void ledOn(void); // turn on LED
void ledOff(void); // turn off LED

/* GLOBALS */
signed long adc;
float fadc;
char dipsw, olddipsw;

/*****
*/
/*                                     */
/* main() variable declarations                                     */
/*                                     */
/*****
*/

/***** MAIN FUNCTION *****/
void main(void)
{
    WDTCTL = WDTPW + WDTHOLD; // Stop watchdog timer
    init_sys(); // Initialize the MSP430

    /* print welcome message */
    lcdPrints( "CIT2 MQP\nv.0.8" );
    swDelay( 5, DHSEC ); /* wait 2.5 seconds */
    lcdClear(); /* clear the LCD */

    lcdPrints( "USB Mode" );
    swDelay( 5, DHSEC ); /* wait 2.5 seconds */

    init_spi0(); /* set up SPI memory on USART 0 */
    init_uart1(); /* enable the USB-UART on USART 1 */
    _EINT(); /* enable interrupts */

    usbPrint( "\n\n\rBegin USB Communication...\n\r" );
    while( !(tx1.status) );
    usbPrintMenu();

    rx1.stop='\r'; /* wait for carriage return */

```

```

/* loop forever */
while(1){
  /* echo */
  // if( rx1.status ){
  //   switch( rx1.buffer[rx1.length-1] ){
  //     case 'i':
  //       usbPrint( "\n\r-----");
  //       Flash(ReadManufacturerIdentification, &fp );
  //       foo = fp.ReadManufacturerIdentification.ucManufacturerIdentification;
  //       usbPrint( "\n\rManufacturer Identification: " );
  //       TXBUF1 = '0'+(char)foo;

  //       usbPrint( "\n\r-----");
  //       Flash(ReadDeviceIdentification, &fp );
  //       usbPrint( "\n\rDevice Identification: " );
  //       foo = fp.ReadDeviceIdentification.ucDeviceIdentification;
  //       TXBUF1 = '0'+(char)(foo >> 8);
  //       TXBUF1 = '0'+(char)foo;

  //       swDelay( 5, DHSEC );
  //       break;
  //     case 'h':
  //       usbPrintMenu();
  //       break;
  //     case 'c':
  //       break;
  //     case 'd':
  //       break;
  //     case 'l':
  //       usbPrint( "\n\rEnter string: " );
  //       rx1.status=0; /* clear "received" */
  //       rx1.length=0; /* clear received buffer */
  //       while( !(rx1.status) ); /* wait until received */
  //       lcdClear();
  //       rx1.buffer[rx1.length]='\0'; /* add null terminator */
  //       lcdPrints( rx1.buffer );
  //       usbPrint( "\n\r" );
  //       break;
  //     case 's':
  //       usbPrint( "\n\rRunning spinner on LCD for 10 seconds..." );
  //       for( i=0; i<10; i++ ){
  //         lcdClear();
  //         lcdPrint( '+' );
  //         swDelay( 1, DHSEC );
  //         lcdClear();
  //         lcdPrint( 'x' );

```

```
//      swDelay( 1, DHSEC );
//      }
//      usbPrint( "done.\n\r" );
//      break;
//      }
//      rx1.status=0; /* clear UART 1 received semaphore */
//      }
//      }
}
```

## Appendix B: Google Earth Code

```
% Input file name to which .kml file will be saved, test name, and test
% description
file_input_name = input('Enter Textfile Name to Examine (no extention):
', 's');
file_name = input('Enter .kml File Name to Save (no extention): ',
's');
test_name = input('Enter Test Name: ', 's');
desc_name = input('Enter Test Description: ', 's');

% Import data into MATLAB and parse for ','
fid = fopen([file_input_name '.txt'], 'r');
A = fread(fid, 'uint8=>char');
fclose(fid);

% Sample GPS data
% A = [Time, Latitude, Hemisphere, Longitude, Hemisphere,
%      xmin, xmax, ymin, ymax, zmin, zmax, rpm; ...];

% Variables for parsing GPS strings:
% time = time of GPS reading
% lat = latitude of GPS reading
% n_s = hemisphere of latitude reading (North or South)
% long = longitude of GPS reading
% e_w = hemisphere of longitude reading (East or West)
% xmin = minimum x axis output from accelerometer
% xmax = maximum x axis output from accelerometer
% ymin = minimum y axis output from accelerometer
% ymax = maximum y axis output from accelerometer
% zmin = minimum z axis output from accelerometer
% zmax = maximum z axis output from accelerometer
% rpm = rpm reading from circuit
time = char(zeros(1, length(A)));
lat = char(zeros(1, length(A)));
n_s = char(zeros(1, length(A)));
long = char(zeros(1, length(A)));
e_w = char(zeros(1, length(A)));
xmax = char(zeros(1, length(A)));
xmin = char(zeros(1, length(A)));
ymax = char(zeros(1, length(A)));
ymin = char(zeros(1, length(A)));
zmax = char(zeros(1, length(A)));
zmin = char(zeros(1, length(A)));
rpm = char(zeros(1, length(A)));
f1 = zeros(1, length(A));
f2 = zeros(1, length(A));
f3 = zeros(1, length(A));
f4 = zeros(1, length(A));
f5 = zeros(1, length(A));
f6 = zeros(1, length(A));
f7 = zeros(1, length(A));
f8 = zeros(1, length(A));
f9 = zeros(1, length(A));
f10 = zeros(1, length(A));
f11 = zeros(1, length(A));
```

```

f12 = zeros(1,length(A));
b = 1;
c = 1;
d = 1;
e = 1;
f = 1;
g = 1;
h = 1;
i = 1;
j = 1;
k = 1;
l = 1;
m = 1;
n = 0;

% Parsing the GPS string
% ----- BEGIN PARSING-----
remain = A;
while true
%   if m == 0
    [str, remain] = strtok(remain, ',');
    if isempty(str), break; end
    f1 = sprintf('%s', str);
    time(b:b + length(f1) - 1) = f1;
    time(b + length(f1)) = ' ';
    b = b + length(f1) + 1;
    n = 1;
%   else
%       [str, remain] = strtok(remain, ',');
%       if isempty(str), break; end
%       f1 = sprintf('%s', str);
%       time(b:b + length(f1) - 4) = f1(4:length(f1));
%       time(b + length(f1) - 2) = ' ';
%       b = b + length(f1) - 1;
%   end

    [str, remain] = strtok(remain, ',');
    if isempty(str), break; end
    f2 = sprintf('%s', str);
    lat(c:c + length(f2) - 1) = f2;
    lat(c + length(f2)) = ' ';
    c = c + length(f2) + 1;

    [str, remain] = strtok(remain, ',');
    if isempty(str), break; end
    f3 = sprintf('%s', str);
    n_s(d:d + length(f3) - 1) = f3;
    n_s(d + length(f3)) = ' ';
    d = d + length(f3) + 1;

    [str, remain] = strtok(remain, ',');
    if isempty(str), break; end
    f4 = sprintf('%s', str);
    long(e:e + length(f4) - 1) = f4;
    long(e + length(f4)) = ' ';
    e = e + length(f4) + 1;

```

```

[str, remain] = strtok(remain, ',');
if isempty(str), break; end
f5 = sprintf('%s', str);
e_w(f:f + length(f5) - 1) = f5;
e_w(f + length(f5)) = ' ';
f = f + length(f5) + 1;

[str, remain] = strtok(remain, ',');
if isempty(str), break; end
f6 = sprintf('%s', str);
xmax(g:g + length(f6) - 1) = f6;
xmax(g + length(f6)) = ' ';
g = g + length(f6) + 1;

[str, remain] = strtok(remain, ',');
if isempty(str), break; end
f7 = sprintf('%s', str);
xmin(h:h + length(f7) - 1) = f7;
xmin(h + length(f7)) = ' ';
h = h + length(f7) + 1;

[str, remain] = strtok(remain, ',');
if isempty(str), break; end
f8 = sprintf('%s', str);
ymax(k:k + length(f8) - 1) = f8;
ymax(k + length(f8)) = ' ';
k = k + length(f8) + 1;

[str, remain] = strtok(remain, ',');
if isempty(str), break; end
f9 = sprintf('%s', str);
ymin(i:i + length(f9) - 1) = f9;
ymin(i + length(f9)) = ' ';
i = i + length(f9) + 1;

[str, remain] = strtok(remain, ',');
if isempty(str), break; end
f10 = sprintf('%s', str);
zmax(j:j + length(f10) - 1) = f10;
zmax(j + length(f10)) = ' ';
j = j + length(f10) + 1;

[str, remain] = strtok(remain, ',');
if isempty(str), break; end
f11 = sprintf('%s', str);
zmin(l:l + length(f11) - 1) = f11;
zmin(l + length(f11) - 1) = ' ';
l = l + length(f11) + 1;

[str, remain] = strtok(remain, ';');
if isempty(str), break; end
f12 = sprintf('%s', str);
rpm(m:m + length(f12) - 2) = f12(2:length(f12));
rpm(m + length(f12) - 1) = ' ';

```

```

    m = m + length(f12);
end
% ----- END PARSING-----

% Calculating input variables:
% mag_test = magnitude voltage from accelerometer [V]
% long_test = longitude values from GPS
% lat_test = latitude values from GPS
% time = time from GPS
% mag_RPM = RPM
% -----BEGIN CALCULATIONS-----
% mag_test
% Find maximum magnitude from maximum and minimum outputs (minimum
output
% rearranged to compare with maximum)
xlmin = zeros(1,length(xmin));
ylmin = zeros(1,length(ymin));
zlmin = zeros(1,length(zmin));

xlmax = zeros(1,length(xmax));
ylmax = zeros(1,length(ymax));
zlmax = zeros(1,length(zmax));

x1 = zeros(1,length(xmax));
y1 = zeros(1,length(ymax));
z1 = zeros(1,length(zmax));

xmin1 = str2num(xmin) .* (3.3 / 4095);
ymin1 = str2num(ymin) .* (3.3 / 4095);
zmin1 = str2num(zmin) .* (3.3 / 4095);

xlmin = (3.3/2) + ((3.3/2) - xmin1);
ylmin = (3.3/2) + ((3.3/2) - ymin1);
zlmin = (3.3/2) + ((3.3/2) - zmin1);

xlmax = str2num(xmax) .* (3.3 / 4095);
ylmax = str2num(ymax) .* (3.3 / 4095);
zlmax = str2num(zmax) .* (3.3 / 4095);

x1 = max(xlmin, xlmax) - ((3.3)/2);
y1 = max(ylmin, ylmax) - ((3.3)/2);
z1 = max(zlmin, zlmax) - ((3.3)/2);
mag_test = sqrt((x1.^2)+(y1.^2)+(z1.^2));

mag_RPM = zeros(1,length(str2num(rpm)));
mag_RPM = str2num(rpm);

% Calculate number of g's from magnitude
g_s = zeros(1,length(mag_test));
for i = 1:length(mag_test)
    if mag_test(i) < 0.33
        g_s(i) = (0.33 + (0.33 - mag_test(i))) / 0.33;
    else
        g_s(i) = mag_test(i) / 0.33;
    end
end

```

```

end

% long_test and lat_test
format long g
long_new = zeros(1,length(str2num(long)));
lat_new = zeros(1,length(str2num(lat)));
long_new = str2num(long);
lat_new = str2num(lat);

% Convert MNEA longitude and latitude style (degrees [D] and minutes
[M]) into
% degress
% longitude (NMEA): DDDMM.MMMM
% latitude (NMEA): DDMM.MMMM
% NOTE: DDD (or DD) = whole degrees
%       MM = whole minutes
%       .MMMM = partial minutes
long1 = long_new.*(1e-2);
long2 = long1 - floor(long1);
long3 = long2.*(1e2);
long4 = long3./60;
long5 = floor(long1)+long4;

% Take into account north or south hemisphere for Google Earth
if e_w(1) == 'W'
    long_test = long5.*(-1);
elseif e_w(1) == 'E'
    long_test = long5;
end

lat1 = lat_new.*(1e-2);
lat2 = lat1 - floor(lat1);
lat3 = lat2.*(1e2);
lat4 = lat3./60;
lat5 = floor(lat1)+lat4;

% Take into account east or west hemisphere for Google Earth
if n_s(1) == 'S'
    lat_test = lat5.*(-1);
elseif n_s(1) == 'N'
    lat_test = lat5;
end

% Convert MNEA time style into standard style
% date (NMEA): DDMMYY
% time (NMEA): HHMMSS.SS (UTC)
% NOTE: DD = day
%       MM = month
%       YY = year
%       HH = hours
%       MM = minutes
%       SS = whole seconds
%       .SS = partial seconds

% time

```

```

t1 = str2num(time).*(1e-4);
t2 = floor(t1);
t3 = str2num(time).*(1e-2);
t4 = t3 - floor(t3);
t5 = round(t4.*(1e2));
t6 = str2num(time) - t5;
t7 = str2num(time).*(1e-4);
t8 = t7 - floor(t7);
t9 = round(t8.*(1e2));
hour = t2 - 5;
minute = t9;
second = t5;
% -----END CALCULATIONS-----

% Start saving .kml file
diary ([file_name '.kml'])
diary on

% Start creating .kml file
% Adds the appropriate heading, test name, test description, and style
of
% pin for roughness classification
disp('<?xml version="1.0" encoding="UTF-8"?>')
disp('<kml xmlns="http://earth.google.com/kml/2.1">')
disp('<Document>')
disp('  <Style id="style1">')
disp('    <Icon>')
disp('      <href>http://maps.google.com/mapfiles/kml/pushpin/grn-pushpin.png</href>')
disp('    </Icon>')
disp('  </Style>')
disp('  <Style id="style2">')
disp('    <Icon>')
disp('      <href>http://maps.google.com/mapfiles/kml/pushpin/ltblu-pushpin.png</href>')
disp('    </Icon>')
disp('  </Style>')
disp('  <Style id="style3">')
disp('    <Icon>')
disp('      <href>http://maps.google.com/mapfiles/kml/pushpin/ylw-pushpin.png</href>')
disp('    </Icon>')
disp('  </Style>')
disp('  <Style id="style4">')
disp('    <Icon>')
disp('      <href>http://maps.google.com/mapfiles/kml/pushpin/pink-pushpin.png</href>')
disp('    </Icon>')
disp('  </Style>')
disp('  <Style id="style5">')
disp('    <Icon>')
disp('      <href>http://maps.google.com/mapfiles/kml/pushpin/red-pushpin.png</href>')
disp('    </Icon>')
disp('  </Style>')
disp('</Folder>')

```

```

disp([' <name>' test_name '</name>'])
disp(' <open>1</open>')
disp([' <description>' desc_name '</description>'])

% Adds placement pins to GPS locations with RPM data
% -----BEGIN LOOP-----
% Loop to determine level of road roughness, point number, and
description
% Level 1 (Forwards) = Magnitudes less than 0.34 (green) <= CHANGE
% Level 2 (Backwards) = Magnitudes between 0.34 and 0.375 (light blue)

for i=1:length(mag_RPM)
disp(' <Placemark>')
if (xmin(i) < 2047)
if minute(i) < 10
    if second(i) < 10
        disp([' <description><b>Point ' num2str(i) '</b><br/>Time: '
' num2str(hour(i)) ':0' num2str(minute(i)) ':0' num2str(second(i))
'.<hr /><b>Forward</b><br/>The boat is moving forward.<br/> Magnitude
is ' num2str(mag_RPM(i)) ' .</description>'])
    else
        disp([' <description><b>Point ' num2str(i) '</b><br/>Time: '
' num2str(hour(i)) ':0' num2str(minute(i)) ':' num2str(second(i)) '<hr
/><b>Forward</b><br/>The boat is moving forward.<br/> Magnitude is '
num2str(mag_RPM(i)) ' .</description>'])
    end
elseif second(i) < 10
    disp([' <description><b>Point ' num2str(i) '</b><br/>Time: '
num2str(hour(i)) ':' num2str(minute(i)) ':0' num2str(second(i)) '<hr
/><b>Forward</b><br/>The boat is moving forward.<br/> Magnitude is '
num2str(mag_RPM(i)) ' .</description>'])
else
    disp([' <description><b>Point ' num2str(i) '</b><br/>Time: '
num2str(hour(i)) ':' num2str(minute(i)) ':' num2str(second(i)) '<hr
/><b>Forward</b><br/>The boat is moving forward.<br/> Magnitude is '
num2str(mag_RPM(i)) ' .</description>'])
end
disp(' <styleUrl>#style1</styleUrl>')
end

if (xmin(i) >= 2047)
if minute(i) < 10
    if second(i) < 10
        disp([' <description><b>Point ' num2str(i) '</b><br/>Time: '
' num2str(hour(i)) ':0' num2str(minute(i)) ':0' num2str(second(i))
'.<hr /><b>Backward</b><br/>The boat is moving backward.<br/> Magnitude
is ' num2str(mag_RPM(i)) ' .</description>'])
    else
        disp([' <description><b>Point ' num2str(i) '</b><br/>Time: '
' num2str(hour(i)) ':0' num2str(minute(i)) ':' num2str(second(i)) '<hr
/><b>Backward</b><br/>The boat is moving backward.<br/> Magnitude is '
num2str(mag_RPM(i)) ' .</description>'])
    end
end

```

```

elseif second(i) < 10
    disp(['          <description><b>Point ' num2str(i) '</b><br/>Time: '
num2str(hour(i)) ':' num2str(minute(i)) ':0' num2str(second(i)) '.<hr
/><b>Backward</b><br/>The boat is moving backward.<br/> Magnitude is '
num2str(mag_RPM(i)) ' .</description>'])
else
    disp(['          <description><b>Point ' num2str(i) '</b><br/>Time: '
num2str(hour(i)) ':' num2str(minute(i)) ':' num2str(second(i)) '.<hr
/><b>Backward</b><br/>The boat is moving backward.<br/> Magnitude is '
num2str(mag_RPM(i)) ' .</description>'])
end
disp('          <styleUrl>#style5</styleUrl>')
end

% Adds GPS location to point
disp('          <Point>')
disp(['          <coordinates>' num2str(long_test(i)) num2str(',')
num2str(lat_test(i)) '</coordinates>'])
disp('          </Point>')
disp('    </Placemark>')
end
% -----END LOOP-----

% Finishes .kml file
disp('</Folder>')
disp('</Document>')
disp('</kml>')
diary off
disp('The file is complete. It is located in the current directory.')

```

