Project Code: FC-TRB2

Mapping Underwater Turbulence in Venice

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

Project Team:

Nicholas Angelini
na1303@wpi.edu

Jose Brache
jbrache@wpi.edu

Matt Gdula
mgdula@wpi.edu

Craig Shevlin
cshevlin@wpi.edu

Advisors:

Fred Looft, PhD
fjlooft@wpi.edu

Fabio Carrera
carrera@wpi.edu

Michael Ciaraldi
ciaraldi@wpi.edu

Date: April 26, 2006

gps-05@wpi.edu

100 Institute Road Worcester, MA 01609

Abstract

The goal of this capstone design project was to design a fully automated data collection system that could be installed in diesel or gasoline motorboats to monitor engine RPM. The system is 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 a battery pack 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 occurrence of underwater turbulence in the form of engine RPM. 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.

Table of Contents

List of Figures

List of Tables

Acknowledgments

Fred Looft – For his guidance on the project and patience

Fabio Carrera – For his creative ideas

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

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

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

Larry McMenamy – For providing a yacht for testing.

1. Introduction

Canal walls in the city of Venice, Italy have been subject to severe damage over the past century. These walls serve as the structural foundation for buildings throughout this unique city. Constant repair efforts are necessary in the ongoing effort to limit the structural decay of these buildings; this process comes at a high cost to the government.

There are many theories and known reasons of the causes of canal wall damage. One hypothesis, made by city structural engineers, is that underwater turbulence caused by motorboats that accelerate quickly, to stop and go, as they dock on the very walls of canals put so much pressure on the walls over time that this can lead to collapse. Engineers, however, have not been able to verify this statement, and thus have been unable to conclusively determine the reason for rapid deterioration of canal walls.

This document details the design process undertaken by our project team to design and build a working prototype for a fully automated data collection system that can be used to track these instances of underwater dispersion in the form of motorboat engine RPM.

2. Background

The introduction of the motorboat solved many transportation issues in Venice. Presently motorboats are necessary for the regular day to day operation of the city; they provide transportation for its citizens and millions of tourists yearly, as well as transportation of perishable goods and wastes. The city of Venice has suffered many problems that can be attributed to traffic. Canals are now heavily congested and many blame the increased amount of repairs needed on the structural integrity of many buildings to motorized transportation.

The problem of an increasing number of repairs that have to be made to this unique city has become a major concern for its citizens, public works entities, private businesses, and the government. There are differing opinions on what is causing damage to canal walls and how the problem should be addressed, but currently repair efforts are a serious economic concern to the Venetian government.

2.1. Transportation in Venice

The city of Venice, Italy is dependent on its canals for transportation. Since the city is in the middle of a lagoon, boat transportation is used for transporting goods, public services, and the general public since Venice was founded centuries ago. The waterways that make up the Venetian canals divide the city into more than 100 small islands. These all need to be accessible by boat; the current system does not allow for any other alternatives. This society will continue to thrive on the waterfront as long as these waterways supply the city with the resources needed to support an ever increasing tourism industry.

2.1.1. Canal Traffic

Since the 1950's motorboats have become the primary means of transportation in Venice. The task of rowboats has almost been entirely replaced by the use of motor propelled vessels. This change has lead to an increasing amount of underwater turbulence caused by engine propellers. The actual distribution of traffic within the city is shown in **Figure 1**. As can be seen from the chart, most of traffic within Venice is attributed to cargo boats, taxis, and public transportation.

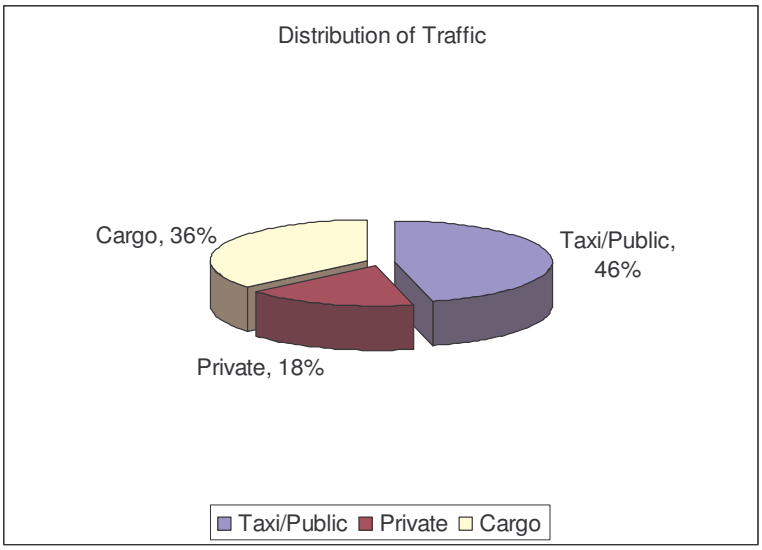

Figure 1: Distribution of Traffic in Venice[1]

2.1.2. Canal Congestion

The congestion of canals in Venice can be attributed to different factors. These factors include the distribution system of cargo boats, sightseeing gondolas, and the physical dimensions of the canals themselves which can limit the passage of multiple boats.

Cargo boats distribute their goods by traveling to and from different canal docks. This can cause bottlenecks at the waterways; an organized distribution system could potentially lower this congestion caused by multiple cargo boats at the same location. An illustration of canal congestion is shown in **Figure 2.**

Figure 2: Canal Congestion[2]

[1] Carrera, Fabio and Caniato, Giovanni. "Venezia la Citta Dei Rii". Pg. 149
[2] Chiu, Jagganath, and Nodine. "The Moto Ondoso Index" IQP. Worcester Polytechnic Institute, July 2002

The famous gondolas that are used primarily for sightseeing and for crossing people across canals also obstruct waterways. Gondolas move slowly, and when many of them are occupying the same area of a canal, traffic can come to a standstill. This is one of the main reasons for congestion in the canals since the gondolas move slowly and stop other boats from passing them in smaller canals.

In addition the physical dimension of a given canal can lead to traffic problems. Larger, heavier boats sit lower on the water; canals that are shallow and narrow impede these boats from navigating across them. As larger and heavier vessels require the use of wider and deeper canals, traffic in these larger canals is also increased. The tides also play a role because they cause some areas of canals to become to shallow or they can rise so as not to let boats pass under bridges.

2.2. Canal Walls

The materials used in the construction of canal walls are primarily Istria stone and brick. Istria stone is non-porous which allows this material to hold up relatively well against salt water. The structural foundation of buildings and lower parts of some canal walls are made up of this stone. The picture shown in **Figure 3** shows a canal lined with the durable Istria stone, known today as *Kirmenjak*. Brick, on the other hand, is porous and quickly deteriorates but since it is inexpensive and readily available, brick still remains a popular building material. The photo shown in **Figure 4** shows a canal wall being repaired with brick and concrete for reinforcement.

Figure 3: Pietra D'Istria (Istria Stone)[3]

[3] http://www.eramarble.com/eng/projects/main.html

Figure 4: Canal Wall Undergoing Repair[4]

2.2.1. Causes of Canal Wall Damage

Two of the main factors attributed to causing canal wall damage include: physical changes to the building material and external events. Boat wakes can start a cycle of events that can cause the mortar holding the bonds between bricks to weaken. The photo shown in **Figure 5** illustrates the wakes produced by boats that can ultimately lead to the erosion of canal walls. Parts of walls can then easily get knocked out of place by other forces, such as a boat docking on a canal wall. Once a small crevice is formed within the walls, underwater boat turbulence can cause more extensive damage.

Figure 5: Wake Produced by a Personal Boat[5]

[4] Chiu, Jagganath, and Nodine. "The Moto Ondoso Index" IQP. Worcester Polytechnic Institute, July 2002
[5] Ibid

Other factors contributing to the damage inflicted to the canal walls are rising sea levels and the sinking of the Venetian land mass. The rising sea level and the extraction of water from aquifers underneath the city, for industrial purposes, have caused the land mass which Venetian buildings rest on to sink. The water levels in the Venetian lagoon have risen approximately 23 centimeters since 1897.

Motorboats, however, cause more significant damage to Venetian structures. In addition to a boats wake produced when a boat propels itself, underwater turbulence caused by engine propellers weaken the structural integrity of canal walls and its foundations. The constant underwater thrusting of the canal walls from propelled water is cause for concern.

Although the sewage system in Venice, Italy depends on the canals for dispersing waste, the sewage lines also can cause costly damages. Sewage enters waterways underwater; the buildup of silt in front of sewage pipes can cause these pipes to clog. When the pipes clog, due to the sewage backing up into the pipes, they can burst and seep to the surrounding mortar. This weakens the structural integrity of canal walls.

In addition to wakes from boats and underwater turbulence, when a boat collides with walls it can cause minor imperfections. The constant exposure to collisions can produce larger crevices where more salt water can seep into and "eat away" at the walls. The picture shown in **Figure 6** shows a damaged canal wall that requires repair.

Figure 6: Canal Wall Damage

2.2.2. Effects of Canal Wall Damage

In December of 1990 a large cavity was found behind one of the walls lining the *Rio Novo* canal. What seemed to be a small hole on the exterior of the wall had actually eroded the supporting structures behind the wall. The *Rio Novo* canal had to be closed down since the supporting structures were on the verge of collapse; boats were rerouted to the *Rio Cerris* canal. Within two years, a building near the *Ponte Rosso* collapsed due to the *moto ondoso*[6], or waves

[6] Moto Ondoso can be roughly translated to mean "wake impact"

caused by motorized boats, from the increased traffic flow. The *Rio Cerris* canal had become a taxi highway and the constant water motion produced by motorboats destroyed the structural foundation of the building beyond repair. Unfortunately, even the twelve owners of the building were fined for failing to make the much needed repairs on the structure. After seven years and 7 billion euros worth of work, *Rio Novo* was reopened in 1997.

In 1994 there was another incidence of a building collapse on the *Rio Dei Greci* canal. The building collapsed due to the erosion caused from excessive traffic; the canal was closed for a period of time until it was repaired.

Buildings on the waterfront of the Grand Canal are constantly being subjected to the effects of *moto ondoso* since it is the major traffic route through Venice. In 1995 the *Ca' Foscari* building was in risk of collapse and was closed for renovation and repairs.

The *Galeazze* Canal, near *Arsenale*, had renovations made and just two years after it was repaired metal sheeting had to be installed alongside the walls as a blockade to stop erosion. Within a year of this installment, in 1999, the entire canal was closed from boat traffic because of the damages that needed to be repaired once again. Some other canals that have been closed for repair include *Rio della Maddalena*, *San Moise*, *San Lorenzo*, and the *Rio de Noal*.

The destructive effects of *moto ondoso* are not only limited to the city of Venice but also extend to other parts of the Venetian lagoon. City Hall on the Lido once had to be closed for fear of collapse because of damage stemming from *moto ondoso*. The cemetery island is in a high traffic area; therefore is also vulnerable to the effects of boat wakes. One billion euros have been spent by the Magistrate of Waters to fix and reinforce the *Codussi* chapel on the cemetery island that was destroyed by erosion caused by the water.

2.2.3. Methods of Preservation of Canal Walls

There are two methods canals are being preserved. Either preventive measures are taken to stop ongoing damage, or damaged walls undergo restoration efforts. The preventive measures currently being used include the required registration of boats and the enforcement of speed limits within the lagoon. Boat registration allows law enforcement officials to know if a boat is permitted to operate within the lagoon; this also allows police to enforce speed limits. The restoration of canal walls involves a long and strenuous process. It includes reforming the bottom of a canal where damage has occurred, sealing crevices behind damaged walls, and rebuilding the wall itself.

2.3. The Age Old Debate

The question of what actually causes erosion in the canals of Venice has long been debated. The group Pax in Aqua attributes *moto ondoso* as the major influence in causing wall damage. Another group called Insula states that there are numerous contributing factors to the deterioration of canal walls. They believe that if canals were reconstructed using modern methods and different building materials *moto ondoso* would be negligent.

2.3.1. Pax in Aqua's Position

Pax in Aqua's position in the debate is that underwater turbulence caused by engine propellers and boat wakes lead to the destruction of canal walls. This group has raised awareness in Venice and has succeeded in appointing a special commissioner that has created regulations regarding these issues for the lagoon.

2.3.2. Insula's Position

Insula attributes canal wall damage to numerous factors. They state that if certain construction practices were used, damage to canal walls can be lessened. They insist that if concrete was used as the building material to form walls that are slanted outward, instead of older vertical walls, underwater turbulence caused by motor boats will not be as significant to the eroding of canal walls. Venetians are skeptical with this concept because concrete and reshaping might affect the aesthetic aspects of these historic canal walls.

2.4. Previous Studies

There have been several observational studies made to determine the effects of motor boats on the Venetian canal walls. One specific study was made by a Worcester Polytechnic Institute (WPI) project team completing their junior year design project while in Venice. A senior year capstone design project was completed in the Electrical Engineering Department of WPI that addressed the same problem, our problem statement is explored in section **3.1**, and also had similar goals; our project goal is stated in section **3.2** of this report.

2.4.1. The Moto Ondoso Index[7]

During the summer of 2002 a team of three Worcester Polytechnic Institute students, completing a junior year design project requirement, they studied the amount of energy dispelled into the canal system using a shore based visual approximation. Data was collected for boats within a certain class type, and canal turbulence was indexed based on the correlation between boat type and annual canal traffic data. Although this was a novel approach addressing the age old debate it was based on a visual approximation. A study of this magnitude could only allow researchers to speculate the amount of underwater turbulence discharged in the studied water ways.

2.4.2. Mapping Turbulence in the Canals of Venice[8]

In January, 2004 a WPI project team of three students completed their senior year capstone design project by creating a tool that could aid researchers in mapping turbulent discharges in the canals of Venice. Their design was an initial effort to map underwater turbulence; the design of the system outlined in this report is a completely reengineered device and approach.

---

[7] Chiu, Jagganath, and Nodine. "The Moto Ondoso Index" IQP. Worcester Polytechnic Institute, July 2002

[8] Chiu, Lacasse, and Menard. "Mapping Turbulence in the Canals of Venice" MQP. Worcester Polytechnic Institute, January 2004

2.5. Summary

The background section of this document discussed several issues pertaining to the transportation in Venice, damage caused to canal walls, debates about the causes of canal wall damage, and previous studies made to find the causes of canal wall damage. This section presented a picture of the current state of Venice and how the city is affected by motorboat transportation in terms of the impact to canal walls and building foundations. Also it discussed how previous studies have attempted to correlate the effect of motorboats to damage cause to canal walls.

3. Problem Statement

This chapter presents the problem statement and defines the overall goal of this capstone design project. In addition to stating the overall goal, this section divides the main project objective into smaller goals; each with their own specific tasks. Each objective was formulated within context of the main project goal.

3.1. Problem Statement and Project Goals

Underwater turbulence caused from motorboat propellers is believed by many to be the cause of erosion to the canal walls in Venice. The problem is that, currently, there are no correlations that have been made to connect underwater turbulence to instances of canal wall damage. One method of studying this correlation is to record instances of motorboat engine RPM, since this is a way of measuring how fast propellers are moving underwater. Instances of engine RPM would have to be mapped across the city, especially where motorboats stop and go, so that underwater turbulence can be correctly correlated to places suffering from canal wall damage.

Our project goal was to: **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 lagoon.**

3.2. Objectives and Tasks

To achieve our project goal, the team had to complete several other objectives, each with their own separate tasks. In order to complete the final objective of creating a working prototype of this automated data collection system, the team outlined several milestones. These objectives are as follows:

1. Familiarize ourselves with the previous senior capstone design project.
2. Develop a method for determining engine RPM from diesel and gasoline engines.
3. Develop a prototype for an embedded system that can collect engine RPM readings, GPS coordinates, and provide the user with relevant information.
4. Write the software to bring all the components of the device together.
5. Test the device on the field and use graphical information software to produce user viewable results from engine RPM data.

As seen above in the objective listing, the first one was to familiarize ourselves with the previous senior design project that attempted to solve the same problem in this design project. Unfortunately the previous project targeted only gasoline engines and its design was confusing. Nonetheless, being familiar with the work that was already done allowed us to iterate new designs more efficiently. We decided to use their choice of using an inductive pickup (placed on spark plugs) for measuring engine RPM from gasoline engines. In addition we researched the microcontroller they used and later determined, for another objective, that a similar one from the same manufacturer was a good choice.

The second objective was to develop a method for measuring engine RPM from diesel and gasoline engines. One way of measuring engine RPM from gasoline engines is by way of an inductive pickup placed on the engines spark plugs. This works for gasoline engines however diesel engines do not have spark plugs. Different methods for obtaining RPM from diesel engines have to be explored, tested, and implemented. Signal processing circuitry for processing the signals representing engine RPM had to be designed and tested for the completion of this objective.

The third objective was to develop a prototype for an embedded system that can collect engine RPM readings, GPS coordinates, and provide the user with relevant information. Requirements, such as the number of inputs, outputs, storage space, etc. had to be determined in choosing a microcontroller. GPS was studied and tested. The entire system was put together in functional blocks. Each subsystem was tested individually. Printed circuit board layout software was then used to build the connections and subsystems of the device. To complete this objective, parts, and the PCB were ordered and then built. Once built, each system was tested again for functionality.

The fourth objective was primarily writing and debugging the software on the microcontroller in order to bring functionality to the entire system. The GPS receiver, analog circuitry, LCD display, and buttons were interfaced with the microcontroller under software control.

The final objective was to test the automated data collection system in the field and produce user viewable results from engine RPM data. Using GIS software, the data from the embedded system has to be processed to produce maps that can be used to interpret the data.

3.3. Summary

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

4. Methods

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

1. Reviewed engine type monitoring requirements (Required to work for diesel or gasoline engines)
2. Reviewed system requirements (What needed to be recorded, for how long, how often e.g. every second, etc.)
3. Implemented the design from system requirements
4. Implemented packaging requirements (Easy to mount, small & compact, water resistant, user interface)
5. Reviewed data processing requirements (How data from unit was going to be used to produce maps)

4.1. Reviewing Engine Type Monitoring Requirements

The first task of our design project was investigating the different methods for obtaining engine RPM from both diesel and gasoline engines. Knowing the types of signals our system was going to be dealing with was the first important step since the next step was establishing our system requirements. One specific method of measuring engine RPM from gasoline engines researched was:

1. Measuring gasoline engine RPM from spark plugs.

Other options had to be reviewed for measuring engine RPM in diesel engines since diesel engines do not have spark plugs. What diesel engines do have, however, are alternators that are used to charge the batteries that start the engines. Signals from an alternator onboard a diesel engine are commonly used today in other systems to calculate engine RPM. To design our system we had to:

1. Measure the tachometer signal from an alternator
2. Design the analog circuitry to condition these signals.

To measure the tachometer signals from an alternator it required that we personally went onboard a diesel boat for a good determination of the signals our system would be dealing with.

4.2. Reviewing System Requirements

The second task of our design project was to outline the system requirements that we were going to use in the design of the automated data collection system. Knowing the signals our system had to interface with allowed us to list more specific system specifications. To complete this task we specified and reviewed:

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

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

4.3. Implementing Design from System Requirements

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

1. Constructed analog circuitry to condition signals from diesel alternator and gasoline engine spark plugs
2. Constructed circuit on a breadboard and tested in lab
3. Designed motherboard on printed circuit board layout software
4. Designed “RPM Module” board on printed circuit board layout software
5. Interconnected system with GPS, LCD, buttons, etc under program control

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

4.4. Implementing Packaging Requirements

Once the multiple modules had been implemented, the next task was building its packaging. From our system requirements we identified the different considerations that had to be

taken into account when packaging the device to one standalone system. These considerations included its size, compactness, water resistance, etc.

4.5. Reviewing Data Processing Requirements

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

4.6. Summary

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

5. System Design

This section presents the overall design approach our team implemented in order to complete our project goal. The system is broken down into logical blocks that represent how the system as a whole is intended to function. These different blocks, or subsystems, include the GPS module, the LCD module, RPM Module, the Microcontroller Module, and the Motherboard Module. The function of each module is explained throughout this section. A block diagram for our entire system is shown below in **Figure 7**.

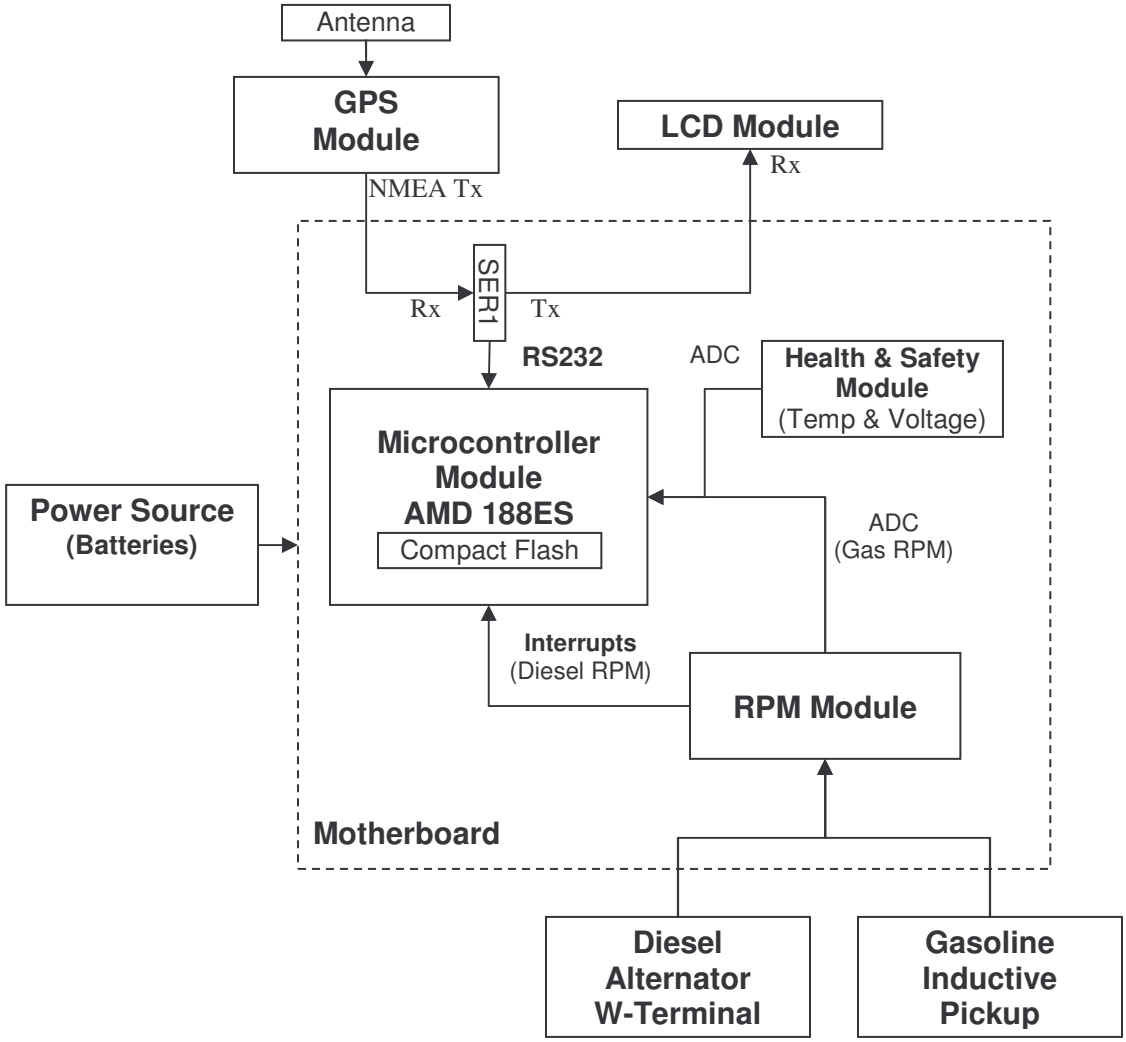

Figure 7: System Block Diagram

5.1. GPS Module

The GPS module of our system, seen in **Figure 8**, consists primarily of a GPS receiver that connects, through serial communication, to the microcontroller and is powered by a +5 V voltage regulator which is located on an external PCB board mounted on the case. The GPS receiver we used in our design was a Garmin[9] GPS receiver, the GPS25-LVS[10], seen in **Figure 9**. This module contains an external antenna that can be mounted onto a glass surface inside the vehicle. The antenna is connected to the GPS receiver through coaxial cable. Once the antenna is placed correctly, the GPS receiver acquires satellites and outputs National Marine Electronics Association (NMEA) strings. These NMEA strings contain useful information such as date, time, speed, location, and other data. The GPS receiver can be programmed to output multiple NMEA strings, each containing different pieces of information. Also, note the connections of the GPS receiver to the microcontroller: the NMEA serial transmit port (Tx) on the GPS receiver is connected to the serial receive port (Rx) of the microcontroller. The signal ground of the GPS receiver is connected to the system ground. The GPS receiver also has a serial receive (Rx) port that can be used to program the types of NMEA strings output. These pin connections can be seen in **Figure 10**.

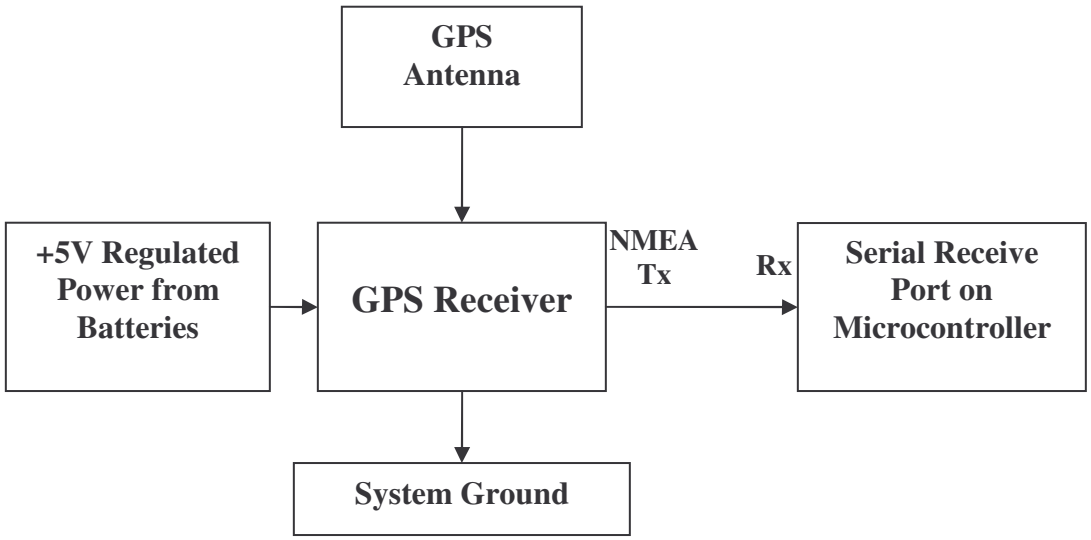

Figure 8: GPS Module Block Diagram

The actual GPS receiver used is shown in **Figure 9**. A PIN-OUT description of the GPS receiver is shown in **Figure 10**. The pin out descriptions shows the connections that were made when connecting power (+5V) to the unit (Pin 10/11), ground (Pin 8), as well as the NMEA Output (Pin 12). The serial NMEA transmit (NMEA Tx) or NMEA Output was connected to the Rx pin on the microcontroller SER1 serial port.

[9] http://www.garmin.com
[10] http://www.garmin.com/products/gps25/

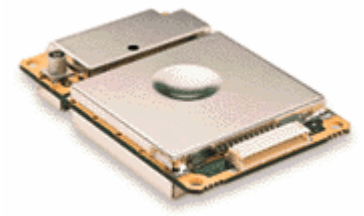

Figure 9: Garmin GPS25-LVS[11]

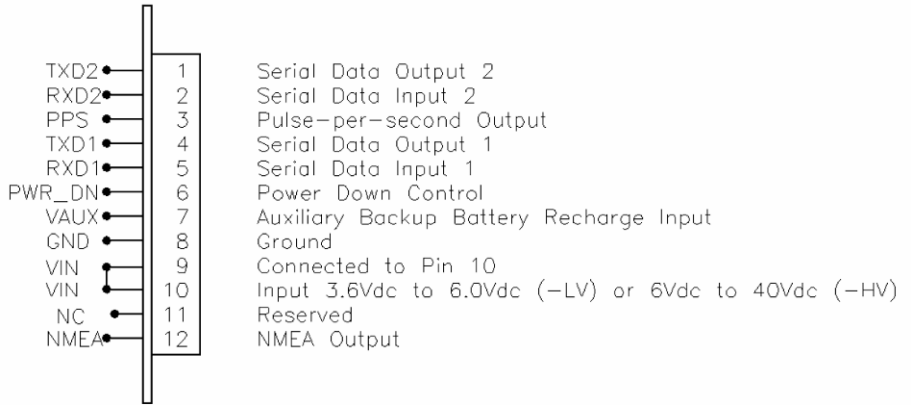

Figure 10: Garmin GPS25-LVS Pin Out Description[12]

5.2. LCD Module

The LCD module of our system, seen in **Figure 11**, consists primarily of a LCD unit that connects, via a RS-232 serial port, to the microcontroller and is powered by a +5 V voltage regulator which is located on an external PCB board mounted on the case. The actual LCD used in our design was a Crystalfontz[13] serial LCD module, model number CFA-632[14], and is displayed in **Figure 12** along with the pin out descriptions in **Figure 13**. This unit is stand alone and displays American Standard Code for Information Interchange (ASCII) characters once ASCII characters are serially transmitted to the LCD display. This device allows our system to display functional information about the device, to the user, while the rest of our system is operating in real time. Also, note the connections of the LCD unit to the microcontroller: the serial receive port (Rx) on the LCD unit is connected to the serial transmit port (Tx) of the microcontroller. The signal ground of the LCD unit is connected to the system ground.

[11] http://www.garmin.com/products/gps25/
[12] http://www.garmin.com/manuals/GPS25LPSeries_TechnicalSpecification.pdf
[13] http://www.crystalfontz.com
[14] http://www.crystalfontz.com/products/632/index.html

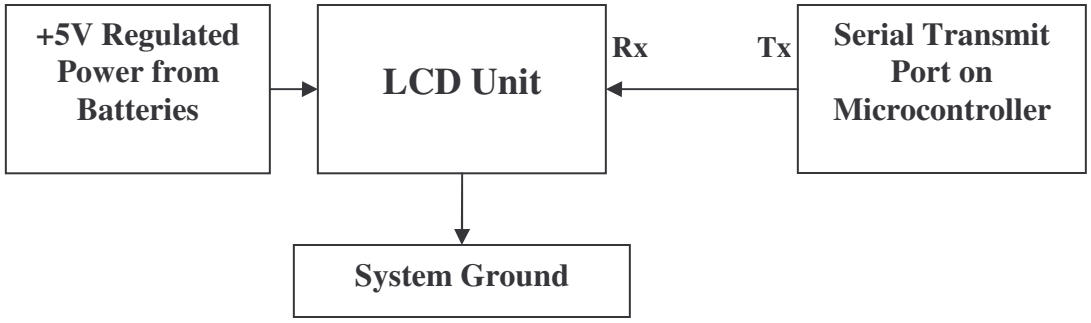

Figure 11: LCD Module Block Diagram

A picture of the LCD unit is shown in **Figure 12**. A PIN-OUT description of the LCD unit is shown in **Figure 13**. The pin out descriptions shows the connections that were made when connecting power to the unit (+5V LCD), ground, as well as the serial receive pin (DATA_IN). The serial receive (Rx) or DATA_IN was connected to the Tx pin on the microcontroller SER1 serial port.

Figure 12: LCD Screen[15]

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

Figure 13: LCD Pin out Description[16]

[15] http://www.crystalfontz.com/products/632/index.html
[16] http://www.crystalfontz.com/products/632/data_sheets/CFA-632_v2.0.pdf

5.3. Health and Safety Module

The health and safety module's function is to provide information about the voltage levels of the power source and ambient temperature. A block diagram of this system is shown in **Figure 14**. We determined that measuring these conditions is important for the overall operation of our device. Whenever these conditions fail, we configured our microcontroller to act accordingly under software control.

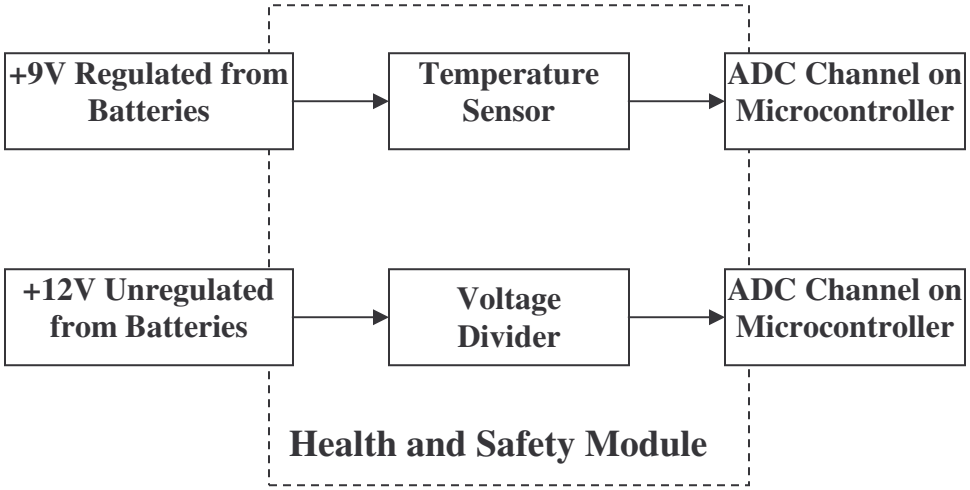

Figure 14: Health and Safety Module Block Diagram

As is seen in **Figure 14**, the temperature sensor is connected to a +9V regulated power supply from the motherboard. The temperature sensor used was an analog devices AD590KF two terminal IC temperature transducer that outputs a current proportional to temperature. The voltage divider (Voltage meter) is connected directly to the power source, to monitor the voltage. Both the temperature sensor and the voltage divider connect to their respective ADC channels on the microcontroller.

5.4. RPM Module

The RPM module's function is to condition the signals from a diesel engine and a gasoline engine. A block diagram of this module is shown in **Figure 15**. For specific details on the design of the RPM module see **APPENDIX F: RPM Module Schematics**. The RPM module has onboard regulators to power the signal conditioning circuitry for both diesel and gasoline engines. The basic operation of this module when it is functioning for diesel engines is that the signal from the alternator w-terminal is connected to our signal conditioning circuit. The conditioned signal is then connected to an external interrupt on the microcontroller. Our program on the microcontroller calculates the RPM in real time depending on the external interrupts. When the device is connected to gasoline engines, the signal from an inductive pickup, connected to the engines spark plugs, is conditioned by the analog circuitry, on the RPM module, to produce

a signal that is read on one of the analog to digital converter (ADC) channels on the microcontroller.

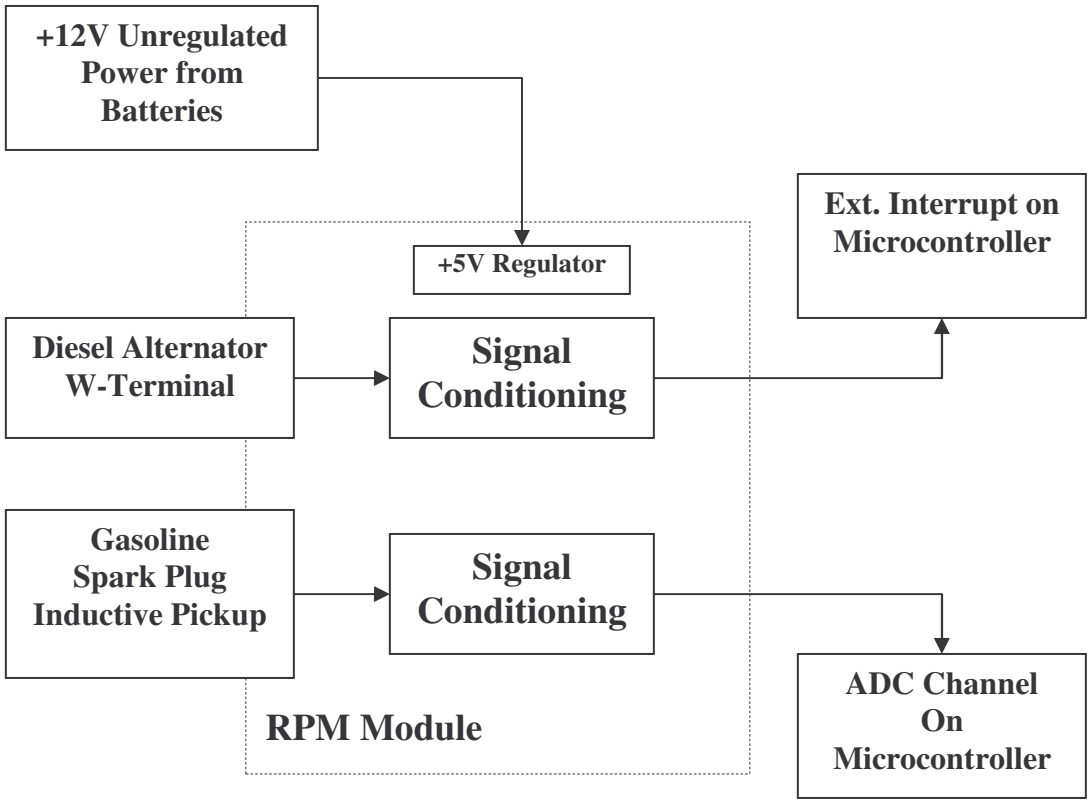

Figure 15: RPM Module Block Diagram

5.5. Microcontroller Module

The microcontroller module is a microcontroller we purchased after performing a value analysis on different microcontrollers that could function in our system. The value analysis we conducted in choosing the TERN FlashCore-B is discussed in **APPENDIX D: Value Analysis**. The functional description of the TERN FlashCore-B is explained in detail in the FlashCore-B(FB)™ Technical Manual found on the Tern Inc. website[17]. A functional block diagram of the FlashCore-B microcontroller can be found in **Figure 17**. A brief description of the FlashCore-B from the technical manual is quoted below[18]:

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

A picture of the FlashCore-B can be seen in **Figure 16**.

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

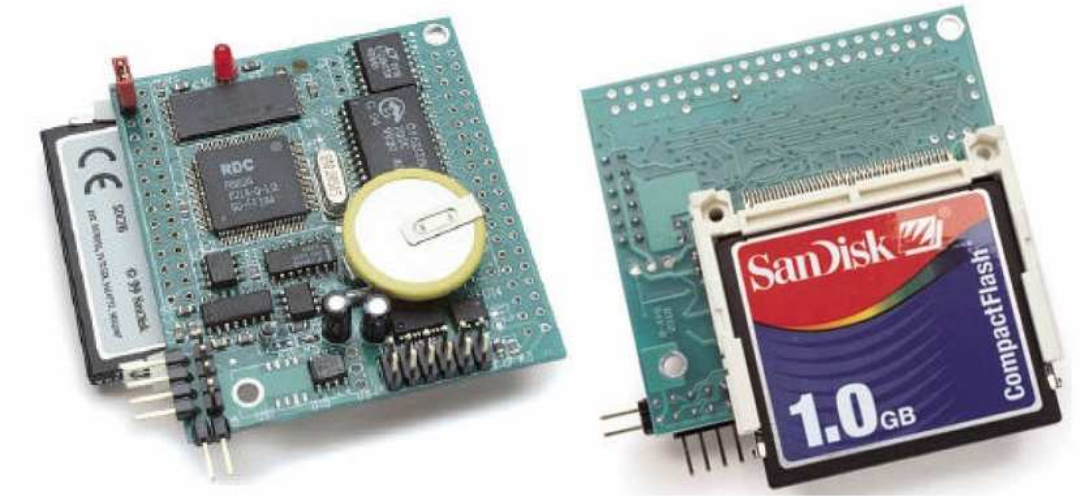

Figure 16: TERN FlashCore-B[19]

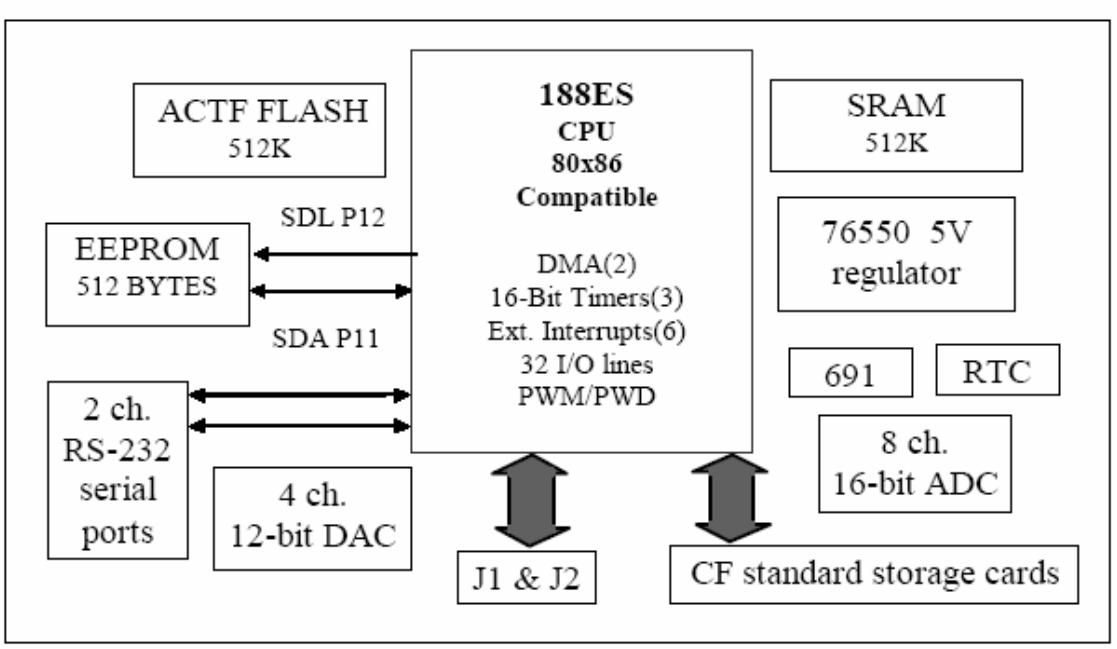

Figure 17: Tern FlashCore-B Functional Block Diagram[20]

We had to interconnect our microcontroller with the other modules explained in this section (RPM, Health and Safety, GPS, LCD Module's). A more detailed block diagram showing the connections between the microcontroller module and the other modules described in this section is shown in **Figure 18**. The block diagram from **Figure 18** shows how the elements of the microcontroller module connect to the other modules. As you can see the GPS and LCD module connect to one of the serial ports on the microcontroller. The health and safety module connects

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

to two of the analog to digital converter (ADC) channels on the microcontroller to provide ambient temperature and voltage readings. The conditioned signal from the RPM module to measure engine RPM in gasoline engines is connected to one of the ADC channels on the microcontroller. The conditioned signal from the RPM module to measure engine RPM in diesel engines is connected to one of the external interrupts of the microcontroller. Power to the microcontroller module is provided by an external +9V regulator that is placed on the motherboard. The motherboard is explained in the next subsection.

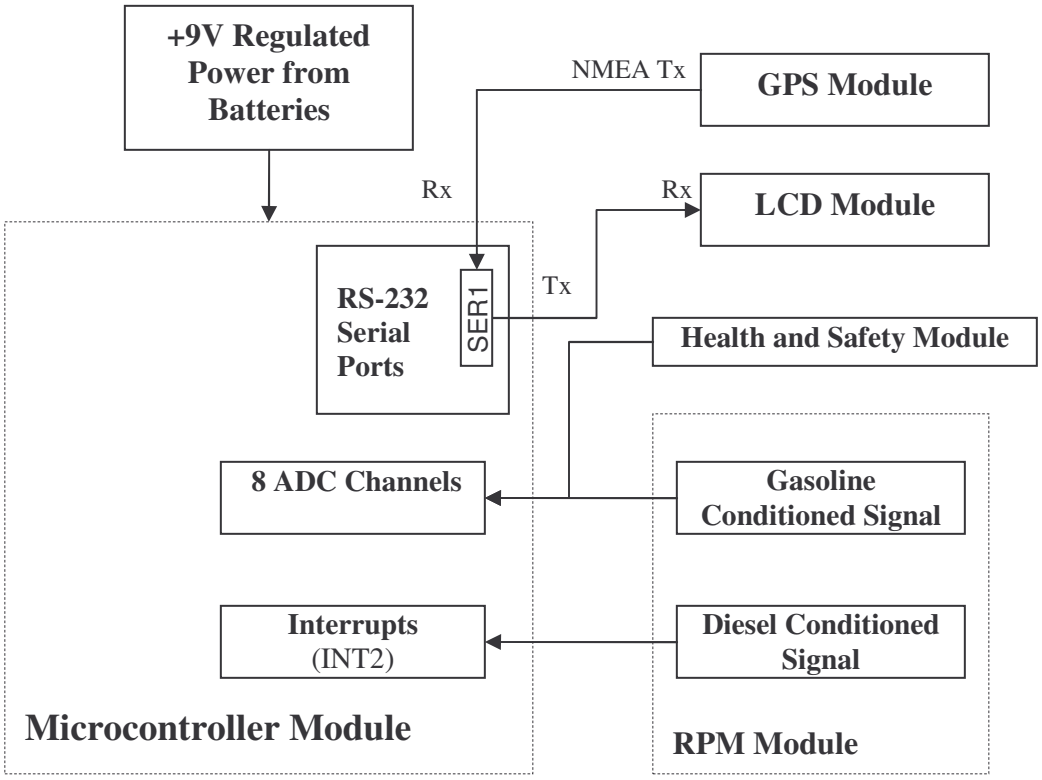

Figure 18: Microcontroller Module Connections

5.6. Motherboard

The motherboard connects all the different modules together. In addition to interconnecting the microcontroller with the GPS, LCD, Health and safety, and RPM Module, the motherboard provides power for all the other modules. Please refer to **Figure 7** to see how the motherboard brings the system together. Everything within the dotted square represents the interconnections made within the motherboard. The microcontroller and the RPM module are the only modules which actually mount, via headers, on the motherboard, shown in **Figure 19**. The GPS module and the LCD module connect to the motherboard using a male header which interfaces the microcontroller SER1 serial port. For specific details on the design of the motherboard see **APPENDIX E: Motherboard Schematics**.

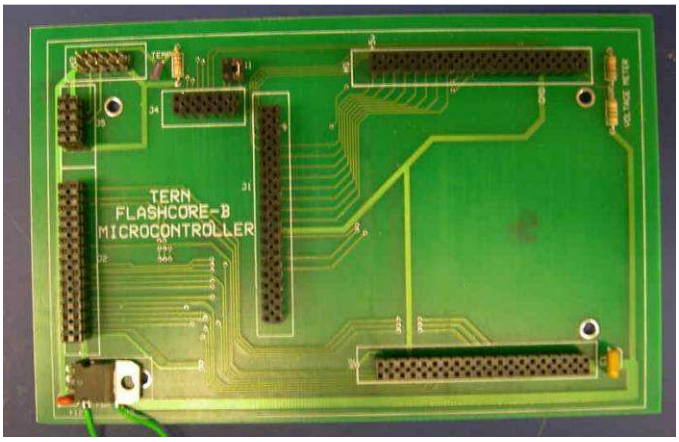

Figure 19: Motherboard Module

5.7. Summary

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

6. Results

This chapter presents the results of our team's work when our system design was implemented. The **Methods** section presented the methods, employed by this project team, which outlined the procedure used to implement our system design. Our team researched methods for obtaining engine RPM from both diesel and gasoline engines and designed the appropriate conditioning circuitry for both. Once we completed specifying our system requirements we implemented the design. The design was then packaged accordingly into one unit. The data recorded by our device was then processed using Geographical Information Systems (GIS) software called MapInfo.

6.1. Diesel Engine RPM Monitoring

This section outlines the process in measuring the signal from a w-terminal and describes the results our team encountered. This section also details the circuitry our project team designed to condition the signals from the w-terminal.

6.1.1. W-Terminal Signal Measurements and Calculations

The design of the diesel engine RPM conditioning circuit was based on the measurements made from an actual alternator on a diesel engine. Using a portable oscilloscope and a laptop our team traveled to Cape Cod, Ma where we measured the alternator w-terminal signal on a diesel boat. The actual engine we obtained measurements was a Volvo Penta Inboard Diesel TAMD63L/P[21]. A photo of the diesel engine we tested is shown in **Figure 20.**

Figure 20: Volvo Penta TAMD63L/P[22]

The alternator on the engine was a Valeo A13N234 alternator that supplied 12 Volts DC, 60 amps. A picture of the alternator is shown in **Figure 21**. One of the terminals on the alternator is labeled as 'W', this is the terminal used to provide a tachometer signal from the engine.

[21] http://www.volvo.com/volvopenta/global/en-gb/marineengines/operators_manual/
[22] http://www.volvo.hu/NR/rdonlyres/A002F3FF-24CC-44E3-BA01-35710D7D14D8/0/tamd63.pdf

Figure 21: Valeo A13N234 Alternator[23]

Several measurements were taken from the alternator using a portable oscilloscope and a laptop. We used the Metex Instruments DG SCOPE-20Mhz digital oscilloscope for obtaining measurements of the signal. **APPENDIX A: DGSCOPE MATLAB Decoder Code** details how we transferred this signal to MATLAB for analysis. We recorded various waveforms from the W-terminal at different engine RPM intervals.

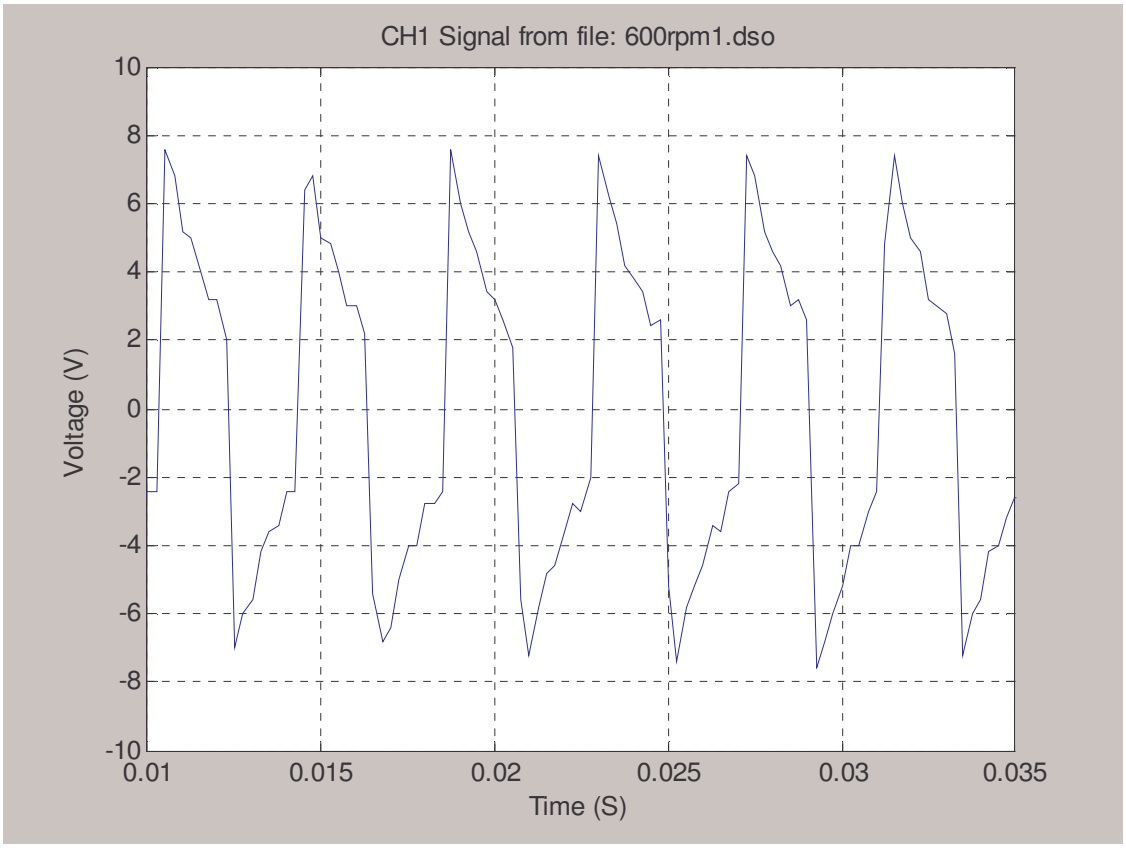

Figure 22: W-Terminal Signal

[23] http://www.go2marine.com/product.do?no=76732F

The plot of the waveform shown in **Figure 22** shows the waveform recorded from the W-Terminal on an alternator. By looking at the Fourier transform of the signal from **Figure 22** we came up with a relationship between alternator w-terminal signal frequency and actual engine RPM after measuring various signals at different RPM intervals. The frequency magnitude spectrum of the signal from the W-terminal, in **Figure 22**, is shown in **Figure 23**. The signal from the W-terminal at approximately 600 RPM was a 243 Hz signal (**Figure 22**). We obtained various recordings, of the signals at the W-terminal, for different engine RPM intervals.

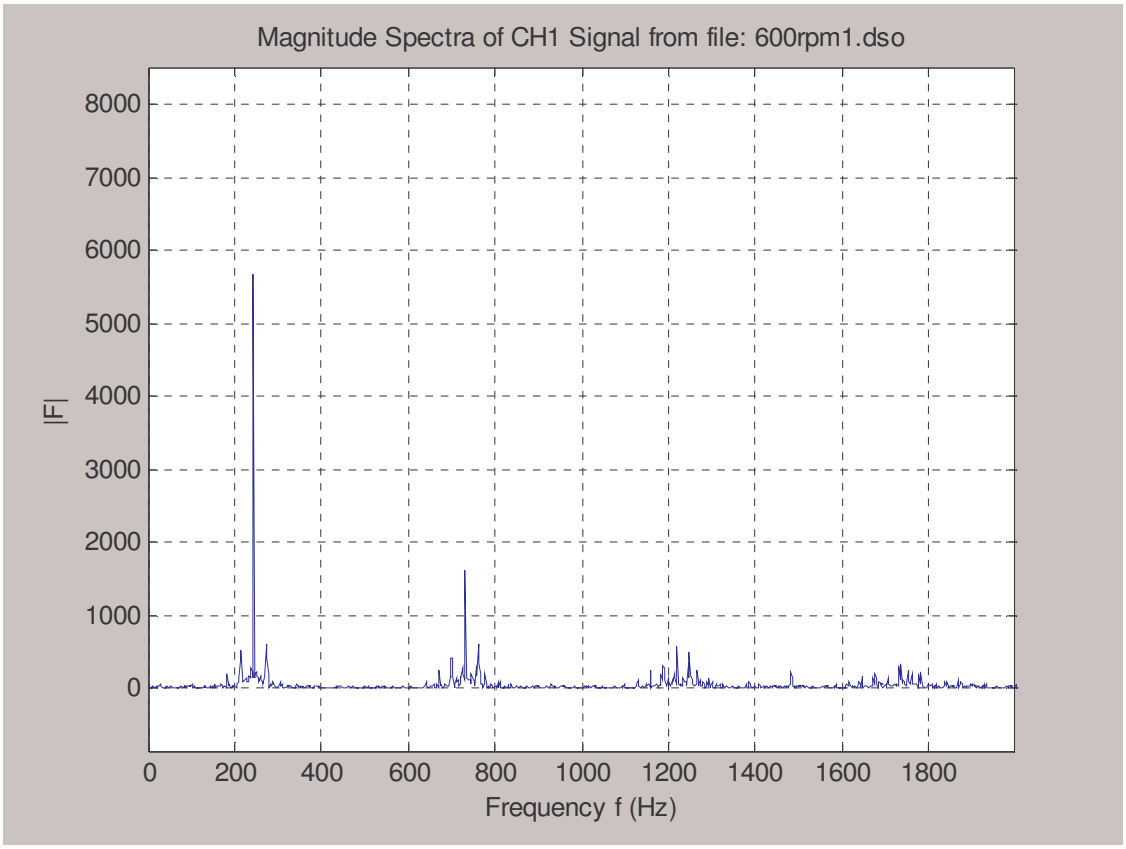

Figure 23: W-Terminal Signal Frequency Spectrum

The different intervals we measured engine RPM from the onboard tachometer and the w-terminal signals respective frequency (Averaged out from several measurements) is shown in **Table 1**.

Measured Engine RPM	W-Terminal Signal Frequency (Hz)
600 RPM	244.3 Hz
1000 RPM	384.57 Hz
1500 RPM	607.67 Hz

Table 1: Measured Engine RPM Vs. W-Terminal Signal Frequency

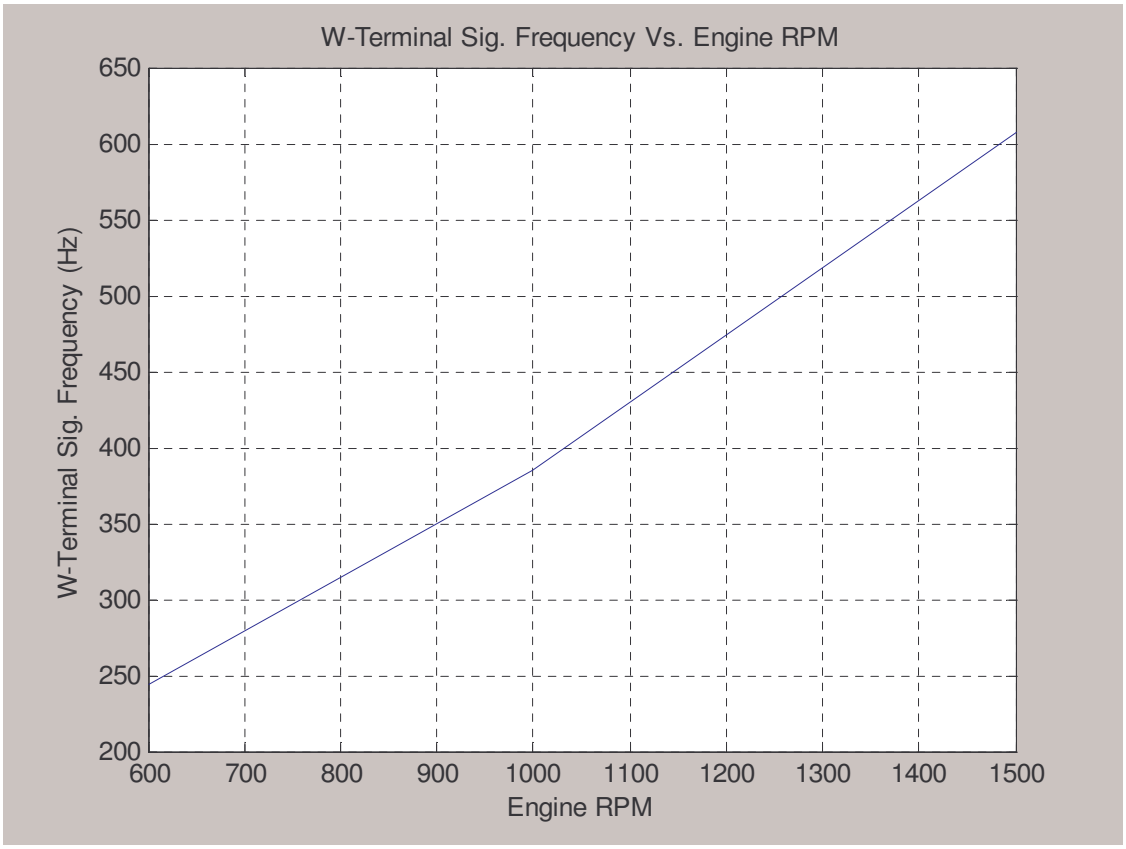

Figure 24: W-Terminal Sig. Frequency vs. Engine RPM Plot

Using the measurements from **Table 1** we plotted the measured relationship between the signal frequency and actual engine RPM; this plot is shown in **Figure 24**. **Figure 24** clearly shows how W-terminal frequency and engine RPM are proportional to each other. In fact, engine RPM can be calculated from certain alternator parameters. These parameters include:

1. W-terminal signal frequency
2. Number of poles on an alternator
3. Pulley ratio

The number of poles on the alternator requires looking at the specific alternators data sheet and the pulley ratio depends on the **Equation 1:**

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

Equation 1: Pulley Ratio Calculation

Equation 1 shows how pulley ratio is an actual ratio determined by the drive pulley diameter and the alternator pulley diameter. This ratio can be calculated by measuring both diameters and often times is specified in engine manuals. Typically in diesel engines there is a 3:1 pulley ratio.

The mathematical equation to calculate actual engine RPM in terms of alternator w-terminal signal frequency, number of poles on the alternator, and the pulley ratio is shown by **Equation 2**:

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

Equation 2: Diesel Engine RPM Calculation

6.1.2. W-Terminal Conditioning Circuitry Results

This section discusses the design of the conditioning circuit that conditions the alternator signal. The output of our conditioning circuitry had to be a TTL compatible waveform since we planned on driving one of the external interrupts on the microcontroller. Using the TTL compatible signal to drive the external interrupts we calculated the signal frequency; this is explained later in the **Software Results** section.

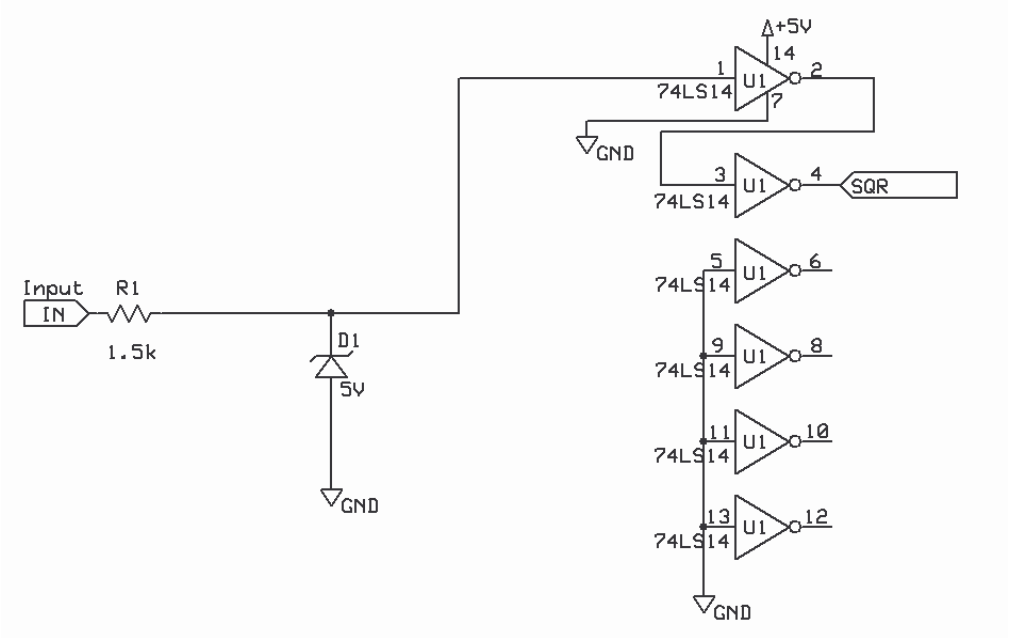

Figure 25: W-Terminal Conditioning Circuit Schematic

The schematic shown in **Figure 25** shows the conditioning circuit we designed to take the w-terminal input and convert that signal to a TTL compatible signal. The w-terminal signal is connected to the port labeled "IN" on the schematic. The resistor R1 limits the current of the w-terminal signal (**Figure 22)** to protect the 5V zener diode D1 and the hex inverting Schmitt trigger: 74LS14. If a waveform like **Figure 22** is connected to the input "IN", the 5V zener diode D1 regulates the voltage of the input waveform (of about (-)8V to +8V) to approximately a 0V to +5V waveform. The resulting waveform at the input 1 of the 74LS14 resembles a 0V to +5V square wave; however, this waveform is not as clean as a perfect square wave. The function of the 74LS14 (Hex inverting Schmitt trigger) is to produce a TTL compatible signal at the output

"SQR", that has the same frequency as the w-terminal signal at "IN". The output signal at "SQR" is supposed to resemble a perfect square wave. For an input signal like the w-terminal signal shown in **Figure 22** the output signal resembles the waveform shown in **Figure 1**.

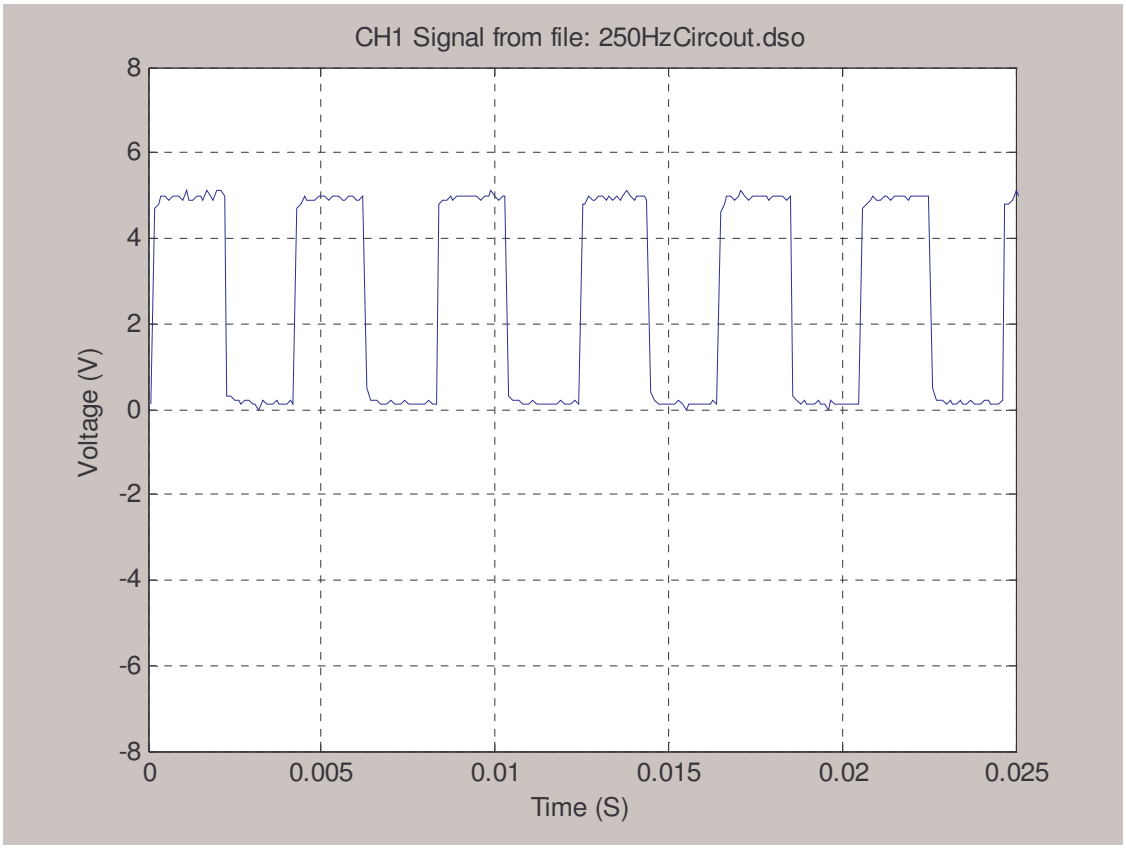

Figure 26: W-terminal conditioning circuit output

The function of this circuit is to produce a TTL compatible signal. The output of the circuit shown in **Figure 25** is shown in **Figure 26**. The output signal shown in **Figure 26** is a TTL compatible signal; it is "cleaner" square wave, with less noise, and is used to drive two of the external interrupts on the microcontroller.

6.1.3. Summary

This section outlined the method used to record the tachometer signal from an alternator. Using the w-terminal signal, engine RPM can be calculated using the w-terminal signal frequency, the number of poles on the alternator, and the pulley ratio. This section also described the conditioning circuitry designed by our team to interface the w-terminal signal to the microcontroller.

6.2. Hardware Results

This section discusses the results from the hardware used to implement our design. To implement all the different modules we separated the design between two printed circuit boards.

The complete schematics and layouts for these two PCB's are included in **APPENDIX E: Motherboard Schematics** and **APPENDIX F: RPM Module Schematics** of this document. Implementing our design between two printed circuit boards allowed the possible use of the motherboard for other functions using the same microcontroller.

6.2.1. Health and Safety Module

The basic function of the health and safety module is to provide temperature and readings and measure the voltage output from the device's power source. Our real-time embedded program on the microcontroller then acts accordingly depending on the voltage and temperature readings it gets (i.e. closes file on compact flash). The program explained in the **Software Results** section.

We used a LM7809 on the motherboard; wired the +9V DC output of the regulator to the temperature sensor. The LM7809 regulator configuration is shown in **Figure 27**.

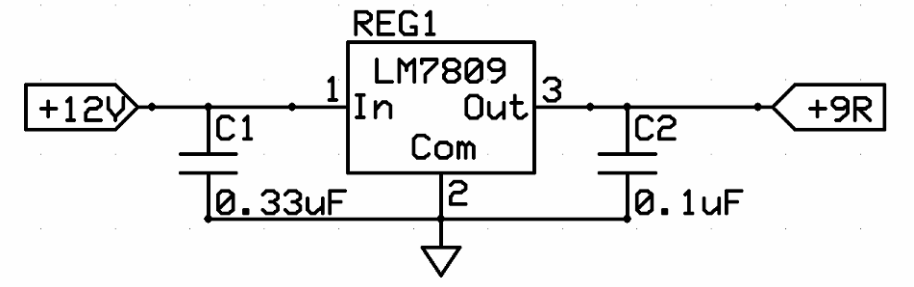

Figure 27: LM7809 Configuration from Motherboard

The output of the LM7809 is used to power the Analog Devices AD590KF temperature sensor shown in **Figure 28**. This temperature sensor acts as a high impedance, constant current regulator passing 1uA/K. The resistor R1 placed at the output of the current sensor creates a voltage potential. At approximately +25°C the sensor outputs 298.2uA, as stated in the datasheet. The resistor network, R1, creates a voltage potential of about 3.0V at +25°C. The voltage potential at the output of the temperature then rises or drops accordingly to the ambient temperature. One of the microcontroller's ADC channels is connected to the output of the temperature sensor, AD0. The real-time program is designed to monitor the voltage at the output of the temperature sensor using the ADC.

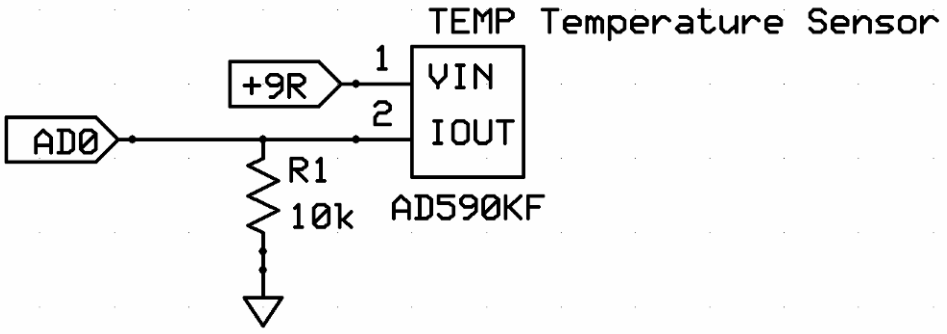

Figure 28: AD590KF Temperature Sensor Configuration

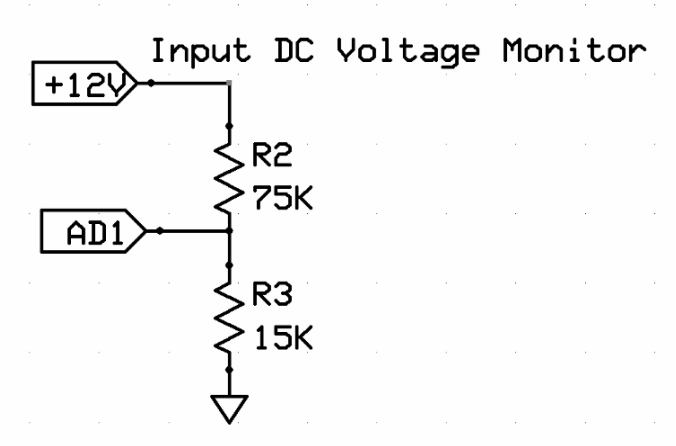

Figure 29: Input Voltage Meter

The "Voltage Meter" shown in **Figure 29** is just a simple voltage divider that takes taps directly into the voltage powering the entire device. One of the microcontroller's ADC channels is connected to the output of the voltage divider, AD1. The real-time program is designed to monitor the voltage at the output of the voltage meter using the ADC.

The health and safety module is placed directly on the motherboard so it can be interfaced directly to the microcontroller without other connections. This is illustrated in **APPENDIX E: Motherboard Schematics**.

6.2.2. RPM Module

The RPM module was designed on one PCB. The details of this printed circuit board are shown in **APPENDIX F: RPM Module Schematics**. The RPM module contains two different circuits that interface through the motherboard to the microcontroller. These two circuits function differently; one is specifically for use on diesel engines and the other for gasoline engines.

The circuit shown in **Figure 30** was previously explained in the **W-Terminal Conditioning Circuitry Results** section and functions specifically for diesel engines. An LM7805 voltage regulator was configured, like the circuit shown in **Figure 32**, to provide power for the signal conditioning circuit shown in **Figure 30**.

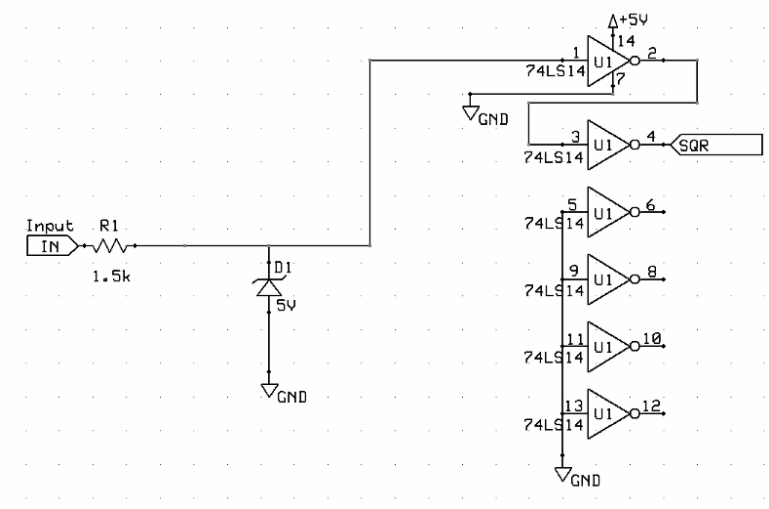

Figure 30: Diesel Engine RPM Conditioning Circuit

The circuit shown in **Figure 31** functions specifically for gasoline engines where an inductive pickup is placed on the spark plugs. The LM2907 IC is a frequency to voltage converter that receives the input from an inductive pickup placed on a spark plug and converts it to a voltage that is read by an ADC channel on the microcontroller, AD4. The configuration of this circuit was taken directly from the datasheet of the LM2907[24]. Software can then be used to determine the respective frequency. Although this circuit was included in the design of our device it has to undergo testing in order for the software design to be implemented. This circuit is also powered by a LM7805 voltage regulator like the one shown in **Figure 32**.

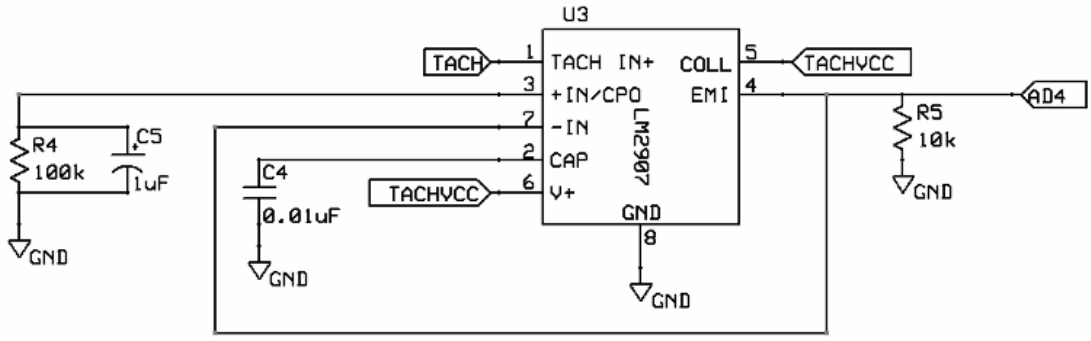

Figure 31: Gasoline Engine RPM Conditioning Circuit

[24] http://www.national.com/ds/LM/LM2907.pdf

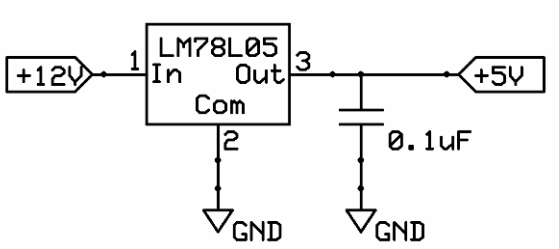

Figure 32: LM7805 Voltage Regulator Configuration

The circuit configurations described in this section were first tested and then designed on PCB layout software. The photo shown in **Figure 33** shows the final PCB board design for the RPM module. As seen in the picture, this module has two long headers that connect to the motherboard. The other headers are used to connect the board to their respective input signal sources.

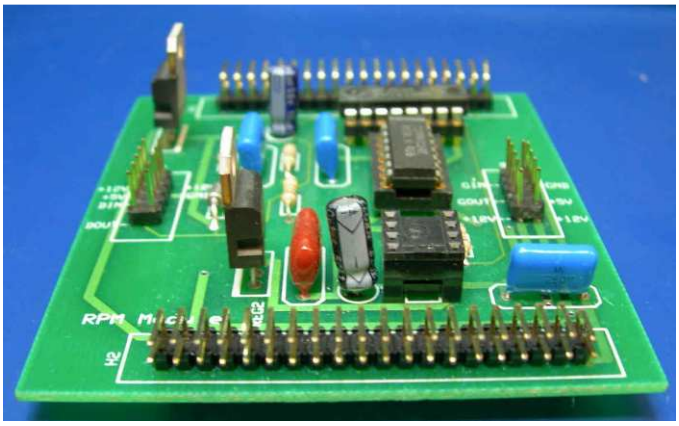

Figure 33: RPM Module PCB Design

6.2.3. Power

The amount of current our device needs to operate was calculated by combining the current requirements from the GPS, Microcontroller, LCD, and the RPM Module. The total current drawn by the entire device is shown in **Table 2**.

Module	Amount of Current
GPS	120mA
LCD	9mA
TERN FashCore-B	160mA
RPM Module	11mA
TOTAL=	**300mA**

Table 2: Current Consumption of System

When considering a power source of two 12V, 7.0 Amp hour batteries and having the device operational for only six hours a day we estimated the length of operation from a complete battery charge. For a completely charged battery pack (Total 14.0 Amp hours), our device could function for about seven and a half days[7.5 Days], approximately six hours per day, if it was drawing about 300mA of current. This length of time is just about what we were expecting to have the device operate for.

6.2.4. Summary

This section described the hardware our team implemented in designing our device. The design and function of the Health and safety module and the RPM module was provided along with schematics detailing the circuitry we implemented. It is important to note that the microcontroller module simply connects to the headers of the motherboard. Also the GPS module, the LCD module, and the RPM module connect to specific headers on the motherboard. Detailed schematics showing the motherboard and the RPM module are shown in **APPENDIX E: Motherboard Schematics** and **APPENDIX F: RPM Module Schematics**, respectively.

6.3. Software Results

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

The software flow chart of our embedded program is shown in **Figure 34**. The basic operation of our code, from looking at this software flow chart, can be followed from the initialization of variables down to the “Main Loop”. Our functions check to see which GPS string type (From the GPS receiver) was received from the SER1 serial port. The program executes different procedures depending on the GPS String.

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

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

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

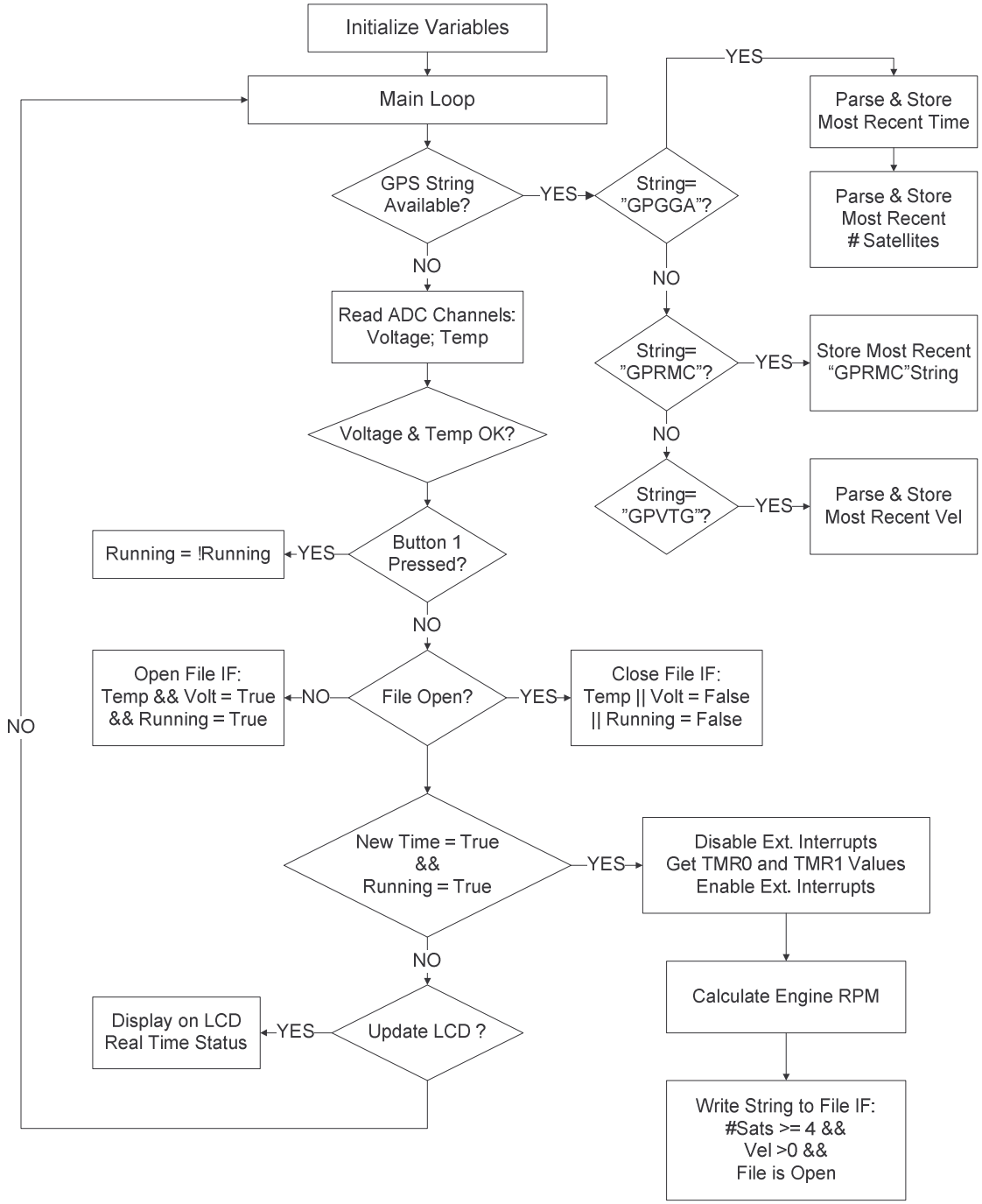

Figure 34: Diesel RPM Software Flow Chart

Once the program processes the current GPS string, it reads the ADC channels to read the voltage (From Voltage Meter), and the temperature (From Temperature Sensor). It sets a Boolean variable to represent if the voltage and temperature readings are within a correct range. If the temperature and voltage readings are outside the given range, either the temperature or voltage "statistic" is set to false signifying that one of them is invalid for device operation.

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

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

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

The procedure that writes to compact flash first disables the external interrupts so it can read from the timer 1 and timer 2 on the processor.

The AMD AM188ES processor has a pulse width demodulation (PWD) option that enables us to measure an input signal's frequency. The illustration in **Figure 35** shows how the timer 1 and timer 2 are used to measure the time elapsed for high and low edges of the waveform. The program disables the external interrupts for a moment to capture the values read from the timers and then re-enables the interrupts. If interrupts weren't disabled there is a possibility an interrupt is interleaved with the execution of the code that reads from the timers. This might lead to erroneous measurements.

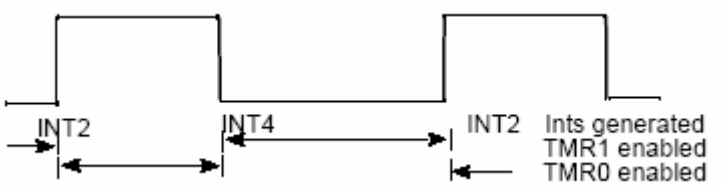

Figure 35: Pulse Width Demodulation[25]

Combining the values from timer 1 and timer 2 yields the period, the program then calculates the signal frequency based on the period of the waveform and then uses **Equation 2**Error! Reference source not found. to calculate the diesel engine RPM.

The program then writes to compact flash depending on conditions like the number of satellites the receiver is tracking and if the file is open. Next the program updates the LCD screen depending real time data like: voltage, temperature statistics, the calculation of RPM, and the number of satellites "in view".

The actual data recorded onto the Compact Flash card resembles the lines shown in **Table 3**:

[25] Am186 TMES and Am188 TMES Users Manual. Advanced Micro Devices, Inc. Http://www.amd.com

$GPRMC,063403,V,4216.5799,N,07148.4762,W,000.0,000.0,300306 RPM=0980 Hz=0392 #SATS=00
$GPRMC,063404,V,4216.5799,N,07148.4762,W,000.0,000.0,300306 RPM=0980 Hz=0392 #SATS=00
$GPRMC,063405,V,4216.5799,N,07148.4762,W,000.0,000.0,300306 RPM=0980 Hz=0392 #SATS=00
$GPRMC,063406,V,4216.5799,N,07148.4762,W,000.0,000.0,300306 RPM=0980 Hz=0392 #SATS=00

Table 3: Compact Flash Stored Data

The data is stored as a text file on compact flash and then GIS software can be used to interpret the data graphically on a Venice, Italy city map. A description of the GIS software that can be used to process this data and source code is included in **APPENDIX C: MapInfo/MapBasic Software** of this document.

6.4. Test Results

To test the device, we had to model the W-terminal signal (**Figure 22**) using an Instek Function Generator (CFG-8219A). We used an input square wave like the one shown in **Figure 36** to our W-terminal conditioning circuit. The output of the W-terminal conditioning circuit resembled the waveform shown in **Figure 26**.

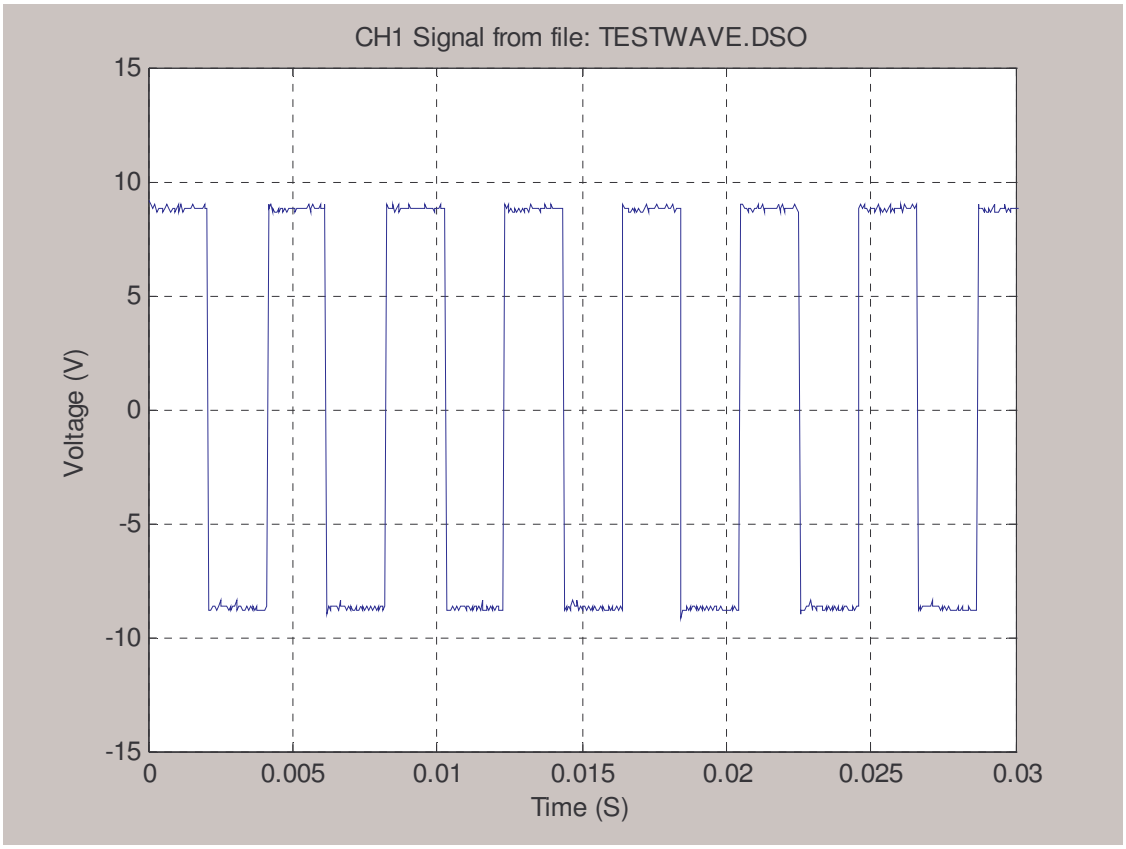

Figure 36: Square Wave used to model the W-Terminal

The frequency of the input square wave (**Figure 36**) we connected to our W-terminal conditioning circuit is shown by the image capture of the function generator in **Figure 37**. The input square wave had a frequency of about 283 Hz.

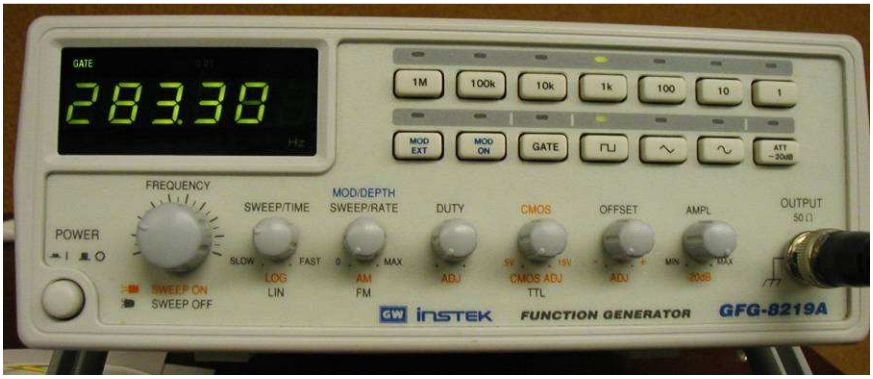

Figure 37: Input Square Wave Frequency (283 Hz)

Our device measured an RPM of 708. This can be seen from the figure shown in **Figure 38**. When comparing the measured 708 RPM to an input signal having a frequency of 283 Hz we used **Figure 39** for comparison. As seen in **Figure 39**, these results are very close to the theoretical values.

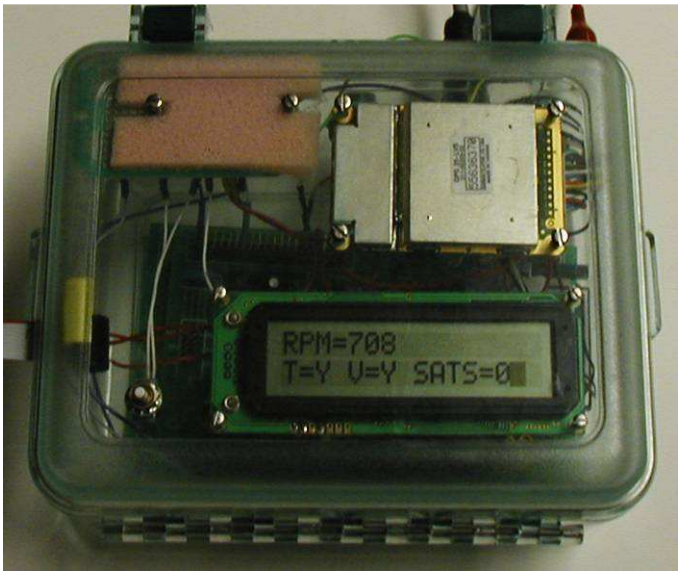

Figure 38: Device Displaying RPM in Real Time

The image in **Figure 38** also shows how other modules in the device are functioning. The device is displaying in the bottom line "T=Y V=Y SATS=0". This shows that the temperature "T" is equal to "Y "for yes meaning good. Inversely if the temperature were bad "T" would be equal to "N" for no meaning bad. The voltage "V" is equal to "Y "for yes meaning good. Inversely if the input voltage levels were bad "V" would be equal to "N" for no meaning bad. The device also shows the number of satellites in view, currently "SATS=0" because we tested the

device indoors. The device only starts to record data once “SATS=04” or the number of satellites in view is greater than 3 (More accurate coordinates).

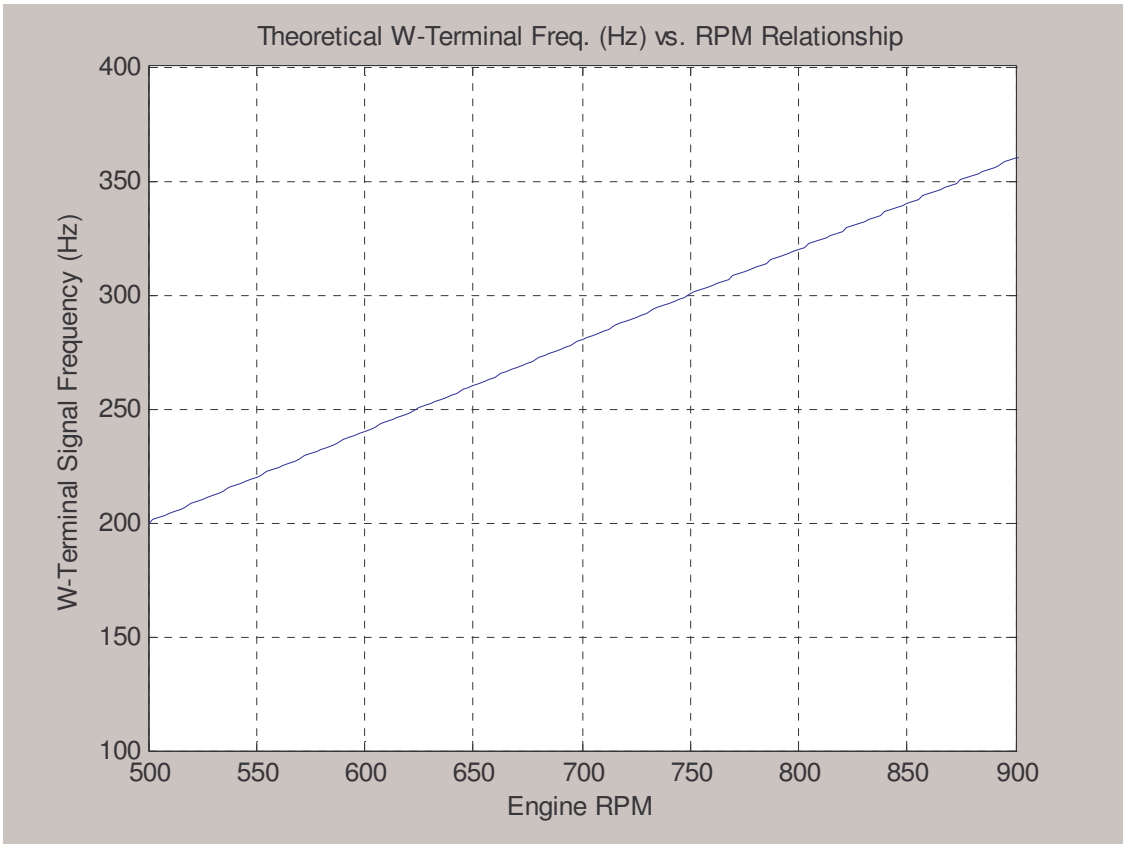

Figure 39: Theoretical W-Terminal Freq. vs. RPM Relationship

6.5. Summary

This chapter presented the results of our system design. The design of the RPM module was discussed in detail; detailing the input signals expected to drive the signal conditioning circuitry as well as the expected outputs. The design of hardware such as the health and safety module was also illustrated. The power requirements of the system were detailed as well as the expected length of time the device can operate, for a fully charged battery pack. In addition to the hardware results, our team presented the software implemented in bringing functionality to the entire system; software flow charts and source code were included as reference. The tests we performed on our system were all done using signals that model the W-terminal signal from a diesel engine. We verified that the tests matched the theoretical results for measuring diesel engine RPM.

7. Summary and Conclusions

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

7.1. Summary of Project Design, System Design and Results

To complete our project design we followed a methodology that consisted of researching how to measure engine RPM for both diesel and gasoline engines, establishing our system requirements, building our system from those system requirements, and finally processing the data produced by our device to provide analysis.

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

1. GPS Module
2. LCD Module
3. Health and Safety Module
4. RPM Module
5. Microcontroller Module
6. Motherboard

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

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

The health and safety module allowed the device to monitor ambient temperature's as well as power source voltage levels. This module provides a means of letting a user know of a fault that is causing the system function incorrectly. It also allows the microcontroller to stop writing to compact flash so that data does not become corrupted.

The RPM module's function is to condition signals from the W-terminal on a diesel engine, and from an inductive pickup placed on gasoline engines spark plugs. These signals are conditioned so that they can be interfaced to the microcontroller for processing and engine RPM calculation.

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

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

Our project team was able to successfully implement all these modules in hardware. In addition we also wrote the software that allowed the system to properly function as an automated data collection system for monitoring a diesel engines RPM.

The resulting data acquisition unit was packaged for use, the system, however, was not tested on the field. Current results are based off inputs that resemble the types of signals from the W-terminal on an alternator field; further testing is required.

7.2. Overall Assessment, Future Work

While our team successfully modeled, developed, and implemented our system design there still remains many challenges in the testing stages and post processing stage. The project completed many of its primary objectives. Overall our project team:

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 external memory, and provide the user with relevant information.
3. Wrote the software to bring all the components of the device together.

Some future work includes the part of our design describing the use of an inductive pickup placed on gasoline engines spark plugs. This also needs to be tested for functionality and have software implemented. Since we designed a modular system, this function could be easily implemented if a different direction (Instead of the inductive pickup option) is desired.

In addition the current system has to undergo various field tests. At the completion of this project our team was unable to test the device on a diesel motorboat. The results that we achieved were based on signals resembling the expected W-terminal signals from a diesel engines alternator.

Once field tests are completed, future work would involve using the geographical information systems (GIS) software we provided to prepare the data files produced by our system for mapping on a Venice, Italy city map. The purpose of the post processing needed for this data is that a correlation has to be made between engine RPM and damage caused to Venetian canal walls.

Engine RPM is related to underwater turbulence because it shows how fast an engine's propellers are moving; future work may be in the finding ways to correlate instances of RPM to the underwater turbulence that causes underwater damage. Furthermore, instances of engine RPM has to be correlated with damaged canal walls.

The current packaging of the device is also a little big. The whole device could be designed to fit a much smaller enclosure. Having the device plug into a boats electrical system for

power could also be an option for making the device more portable. Power requirements of the system should be considered when attempting to make the entire system consume less power; we feel the device could draw much less current with modern GPS units, and other components.

Once these issues are assessed, the operation of a device like the one our project team, that is correctly implemented could potentially aid the city of Venice, Italy in the efficient release of canal repair resources. In addition, this tool could aid city planners in the monitoring of underwater turbulence caused by engine propellers, and find correlations between them and canal wall damage.

7.3. Conclusions

Our team successfully implemented an automated data collection system that can be installed in motorboats propelled by diesel engines to monitor and store engine RPM as well as GPS positional data, for boats that navigate the Venetian lagoon. This document outlines the design process our project team followed in building a working prototype for this automated data collection system. Options in having this device placed in gasoline motorboats have to be tested and implemented. In addition, the post processing of data stored on compact flash by our system has to be considered after multiple field tests are conducted. In addition to the design of our system we also presented different considerations that should be taken to improve the functionality of the device and system. Future designs and methods could serve as a tool for aiding the city of Venice, Italy in the repair of damaged canal walls.

8. References

Introduction

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

1. Carrera, Fabio and Caniato, Giovanni. “Venezia la Citta Dei Rii”. Pg. 149
2. Chiu, Jagganath, and Nodine. ”The Moto Ondoso Index” IQP. Worcester Polytechnic Institute, July 2002
3. http://www.eramarble.com/eng/projects/main.html
4. Chiu, Jagganath, and Nodine. ”The Moto Ondoso Index” IQP. Worcester Polytechnic Institute, July 2002
5. Chiu, Jagganath, and Nodine. ”The Moto Ondoso Index” IQP. Worcester Polytechnic Institute, July 2002
6. Moto Ondoso can be roughly translated to mean “wake impact”
7. Chiu, Jagganath, and Nodine. ”The Moto Ondoso Index” IQP. Worcester Polytechnic Institute, July 2002
8. Chiu, Lacasse, and Menard. “Mapping Turbulence in the Canals of Venice” MQP. Worcester Polytechnic Institute, January 2004
9. http://www.garmin.com
10. http://www.garmin.com/products/gps25/
11. http://www.garmin.com/products/gps25/
12. http://www.garmin.com/manuals/GPS25LPSeries_TechnicalSpecification.pdf
13. http://www.crystalfontz.com
14. http://www.crystalfontz.com/products/632/index.html
15. http://www.crystalfontz.com/products/632/index.html
16. http://www.crystalfontz.com/products/632/data_sheets/CFA-632_v2.0.pdf
17. http://www.tern.com
18. FlashCore-B(FB)^{TM} Technical Manual. Tern Inc. http://www.tern.com
19. FlashCore-B(FB)^{TM} Technical Manual. Tern Inc. http://www.tern.com
20. FlashCore-B(FB)^{TM} Technical Manual. Tern Inc. http://www.tern.com

21. http://www.volvo.com/volvopenta/global/en-gb/marineengines/operators_manual/
22. http://www.volvo.hu/NR/rdonlyres/A002F3FF-24CC-44E3-BA01-35710D7D14D8/0/tamd63.pdf
23. http://www.go2marine.com/product.do?no=76732F
24. http://www.national.com/ds/LM/LM2907.pdf
25. Am186 ™ES and Am188 ™ES Users Manual. Advanced Micro Devices, Inc. Http://www.amd.com
26. FlashCore-B(FB)™ Technical Manual. Tern Inc. http://www.tern.com

APPENDIX A: DGSCOPE MATLAB Decoder Code

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

decode('FILENAME.DSO');

Once the code executes, both the signal waveform and the frequency spectrum of the waveform is shown. A Screen capture showing the command window is shown in **Figure 40**. Changing the views on the output waveforms can be modified using the plot parameters in the end of the MATLAB code.

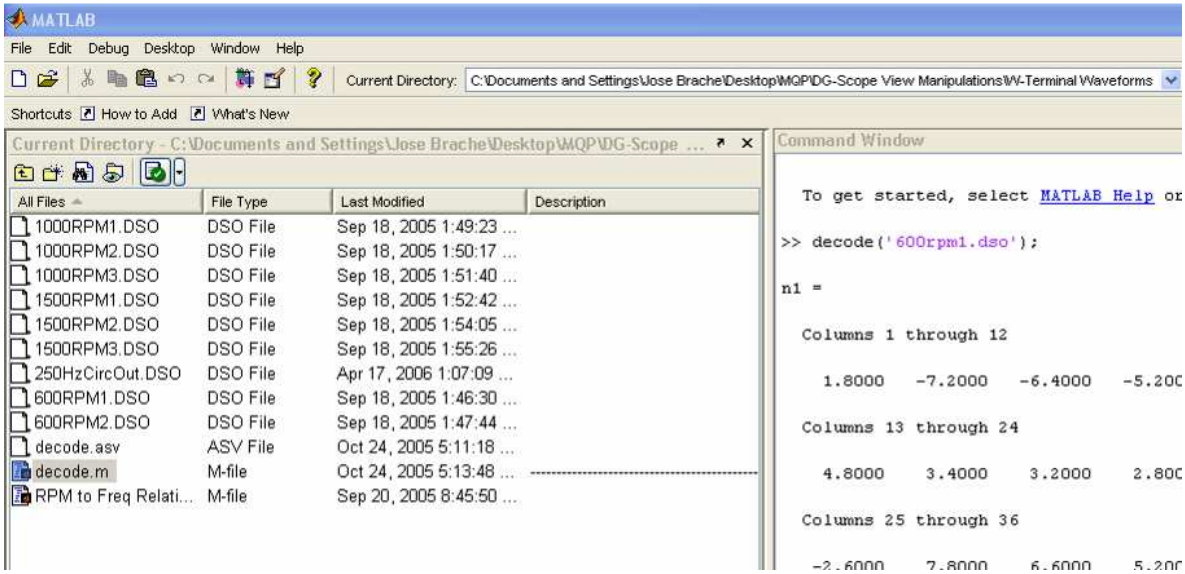

Figure 40: MATLAB Command Window

1. MATLAB Code

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

%--------------------------------------------------------------------
% IDENTIFY AND SET THE SCALE PER DIVISION BEING USED
```

```
        % After playing around with all the possible voltage division
         % combinations, the HEX value for all the possible scales was
        % identified. The HEX value is stored in c(6) for CH1 and c(12) for CH2.
        % These two variables: ampopt and ampscale are used to set up the correct
        % scale in the loops where the array's n1 (CH1) and n2 (CH2) are created.
ampopt = [00 01 02 03 04 05 06 07 08];
ampscale = [1/4000 1/2000 1/1000 1/400 1/200 1/100 1/50 1/20 1/10];

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

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

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

```
        i = i+2;
        x = x+1;
      end
   end

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

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

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

% This while loop identifies the scale used for the channel and applies
```

```
% it to the respective array
   i = 1;
   while i <= 9;
      if (c(12) == ampopt(i));
         n2 = n2 .* ampscale(i);
         i = 9;
      end
      i = i+1;
   end
else
   n2 = 0;
end

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

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

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

```
    96/1920*1*10^(-3)          %13    1ms/Div timediv = 1*10^(-3);
    96/1920*2*10^(-3)          %14    2ms/Div timediv = 2*10^(-3);
    96/1920*5*10^(-3)          %15    5ms/Div timediv = 5*10^(-3);
    96/1920*10*10^(-3)         %16    10ms/Div timediv = 10*10^(-3);
    96/1920*20*10^(-3)         %17    20ms/Div timediv = 20*10^(-3);
    96/1920*50*10^(-3)         %18    50ms/Div timediv = 50*10^(-3);
    96/1920*0.1                %19    0.1s/Div timediv = 0.1;
    12/240*0.2                 %20  *   0.2s/Div timediv = 0.2;
    12/240*0.5                 %21  *   0.5s/Div timediv = 0.5;
    12/240*1                   %22  *   1s/Div timediv = 1;
    12/240*2                   %23  *   2s/Div timediv = 2;
    ];

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

format short;

fs = 1/timescalar;

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

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

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

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

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

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

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

    figure()
    plot(f1, fF1)
    grid on

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

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

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

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

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

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

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

```
    figure()
    plot(f2, fF2)
    grid on

    famp = max(fF2) * 3/2;
    axis([0 max(f2)/2 -(famp)/10 famp])
    title(['Magnitude Spectra of Signal in CH2 from file: ', file]);
    ylabel('|F|');
    xlabel('Frequency f (Hz)');

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

    title(['Aquired signal in X-Y Mode from file: ', file]);
    xlabel('CH1 (Volts)')
```

```
    ylabel('CH2 (Volts)')
    axis([-amp1 amp1 -amp2 amp2])
end
return
```

APPENDIX B: C++ Embedded Program Source Code

The source code provided in the **Program Source Code** section of this appendix comprises the entire operation of the device from a software standpoint. A software flow chart is included in **Figure 41** to provide a functional top level view of our program. Some other functions defined in other header files are not included in this document. Please explore the files included in the [Paradigm C++ Light – Tern Edition IDE] CITYLAB project file. In addition to these files please see the TERN FlashCore-FB technical manual[26] for the description of the functions controlling the FlashCore-B provided by TERN. This code was specifically written for use when the device is connected to the W-terminal of a diesel engine.

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

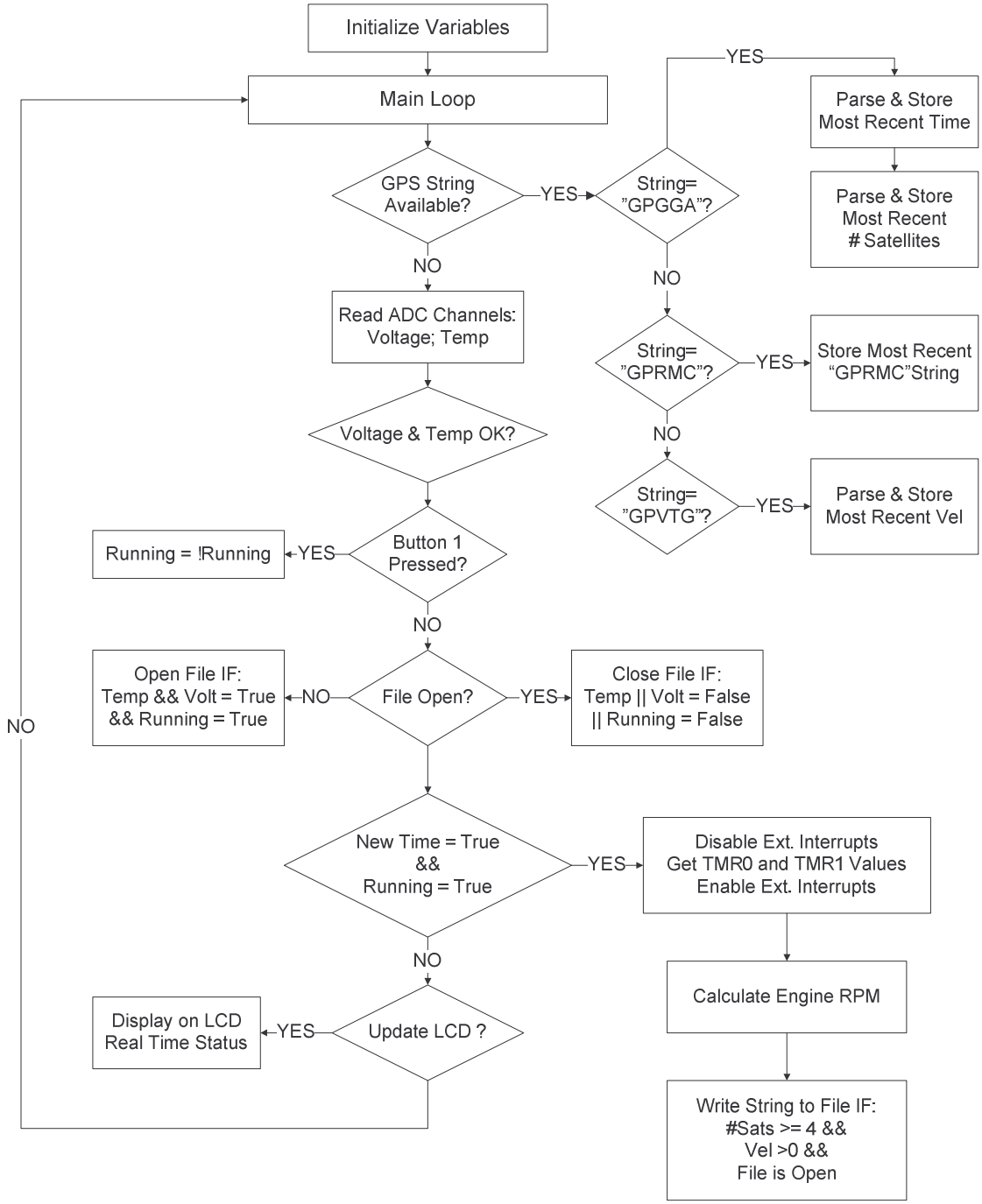

Figure 41: Diesel RPM Software Flow Chart

1. Program Source Code

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

  extern "C" {
```

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

#define BUFFSIZE 1024

#ifndef NULL
#define NULL 0
#endif

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

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

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

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

```
*
*                   where:  int adcVal is value read from ADC (Decimal)
*****************************************************/
double ConvertVolts(unsigned int adcVal)         // Change value read in from
{
        double retVal=adcVal;
        retVal /= 0xFFFF;
        retVal *= 5;
        return retVal;
}
/*****************************************************
*       Function: int FindNumSats(const char* gpsStr)
*                   Output: type int the number of satellites according to
*                                       GPS string
*
*                   where:  gpsStr is the GPS string type '$GPGGA'
*****************************************************/
int FindNumSats(const char* gpsStr)
{
        const char* curTok = gpsStr+41;
        int retVal = ((curTok[0]-0x30)*10)+(curTok[1]-0x30);
        return retVal;
}
/*****************************************************
*       Function: int FindVelocity(const char* gpsStr)
*                   Output: type int the velocity in knots (without decimal point) as parsed
*                           from the GPS string
*
*                   where:  gpsStr is the GPS string type '$GPVTG'
*****************************************************/
int FindVelocity(const char* gpsStr)
{
        const char* curTok = gpsStr+19;
        int retVal = ((curTok[0]-0x30)*1000)+((curTok[1]-0x30)*100)+((curTok[2]-
                0x30)*10)+((curTok[4]-0x30));
        return retVal;
}

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

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

        if(tStat)                       // If temp status is true, temp is 'Y', else 'N'
                tempStr = "Y";
        else
                tempStr = "N";
        if(vStat)                       // If voltage status is true, voltage is 'Y', else 'N'
                voltStr = "Y";
        else
                voltStr = "N";
        sprintf(bottomLine, "T=%s V=%s SATS=%i", tempStr, voltStr, numSats);

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

#define MAX_TEMP 0xCCCC
#define MIN_TEMP 0x3333

#define MAX_VOLT 0x80F0
#define MIN_VOLT 0x5300

#define MIN_SATS 0
#define MIN_VEL -1

#define MIN_HZ 0

void interrupt far int2_isr(void);
void interrupt far int4_isr(void);
void interrupt far t2_isr(void);

unsigned int ta,tb,tm,t0_cnt,t1_cnt,t2_cnt;

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

        for(int i=0;i<=5;i++)           // Delay for about 5sec to wait for GPS
                delay_ms(999);

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

```
pio_init(26, 3);                    // Button Init as an input

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

// Initialize flash
fs_descrip* file = NULL;

if(fs_initPCFlash()!=0)
{
        putser1('\f', com1);
        putsers1("Restart with", com1);
        putser1('\n', com1);
        putser1('\r', com1);
        putsers1("Compact Flash", com1);
        while(1)
                delay_ms(999);
}

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

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

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

bool running = true;                // All ENV ok or Button pressed?
bool butDown = false;               // Button pressed?
bool record = false;                // Has one second elapsed for next recording?

GPSStringReader sr(com1);           // Object to read GPS strings
char* gpsStr = NULL;                // The string being read in from GPS
char curGPS[60] = "";               // The last gps string we care about
int numSats = -1;                   // The current number of satellites in view

led(0);
char strType[6] = {0,};             // Array that holds GPS String type ex. '$GPGGA'
char strnewTime[7] = "";            // String holding new time
char strTime[7] = "";               // String holding current time
int Vel = 0;                        // Velocity in knots

double t1, t1new, t0, t0new, duty, T, Hz, RPM=0;
int iRPM,iHz,telapse=0;

// Pulley factor is alternator pulley diameter divided by crank pulley diameter
double pulleyRatio = 0.333333;
// Obtain from manufacturers data sheet (6, 8, 10, etc.)
double alternatorPoles = 8;

/*****************************************/
```

```
t0_cnt=0;
t1_cnt=0;

int2_init(1,int2_isr);
int4_init(1,int4_isr);

ta=(unsigned int)0xffff; // MAXCOUNT
tb=(unsigned int)0xffff; //

tm = 0xc009;              // start timer0, int. enabled, external counts
t0_init(tm,ta,tb,int4_isr);

tm = 0xc009;              // start timer1, int. enabled, external counts
t1_init(tm,ta,tb,int2_isr);

// 40 MHz, 25x4=100 ns per timer clk
ta=(unsigned int)10;      // pre-scale for timer0 & timer1 .1us*10 = 1us
tm = 0xc001;              // start 1/4 clk countdown, int. disabled
t2_init(tm,ta,t2_isr);

outport(0xfff0, inport(0xfff0)|0x1000);   //Enable PWD
pio_init(31,0);                           //Init INT2
/*****************************************/

while(true){              //main loop

/******GPS STRING HANDLING**********************/
if(gpsStr = sr.GetString())
{
        for(int i=0;i<5;i++)
                strType[i] = gpsStr[i+1];          // Parses for string type

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

                int newNumSats = FindNumSats(gpsStr);
                if(newNumSats!=numSats)        // Need to change display
                {
                        numSats = newNumSats;
                        needUpdate = true;
                }
                if(strcmp(strnewTime, strTime)!=0)       // Need to record if sec elapsed
                {
                        strcpy(strTime, strnewTime);
                        record = true;
                }
        }

        if(!strcmp(strType, "GPRMC"))                    // GPS string is 'GPRMC'
```

```
                strncpy (curGPS, gpsStr, 59);

        if(!strcmp(strType, "GPVTG")&&(numSats>=MIN_SATS))
        // GPS string is 'GPVTG'
        {
                int newVel = FindVelocity(gpsStr);
                if(newVel!=Vel)          // need to change display
                {
                        Vel = newVel;
                        // needUpdate = true;
                }
        }

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

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


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

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

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

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

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

        /******Recording Every Second*************************/
        if(record&&running)
        {
                /********Calculate the Signal Frequency********/
                disable();
                t1new = t1_cnt;
                t0new = t0_cnt;
                enable();
```

```
if((t0new!=0) && (t1new!=0))
{
	if(t0new==t0 && t1new==t1)
		telapse++;
	else
		telapse=0;
	if(telapse<11)
	{
		//duty = t0/t1;
		t0=t0new;
		t1=t1new;
		T = t0+t1;
		Hz = 1/(T*0.000001);
	}
	else
	{
		telapse=11;
		Hz=0;
	}
}
else
	Hz=0;

iHz = Hz;

/********Convert Frequency to RPM********/

RPM = Hz;
RPM *= 60;
RPM /= alternatorPoles;
RPM *= pulleyRatio;
iRPM = RPM;
needUpdate = true;
record = false;

/********RECORD********/

if((numSats>=MIN_SATS)&&(iHz>MIN_HZ)&&file)
{
	char strSats[] =  {((numSats/10)+0x30),((numSats%10)+0x30), '\0'};
	char strHz[] =    {((iHz/1000)%10)+0x30,
			   ((iHz/100)%10)+0x30,
			   ((iHz/10)%10)+0x30,
			   (iHz%10)+0x30,
			   '\0'};
	char strRPM[] =		{((iRPM/1000)%10)+0x30,
				((iRPM/100)%10)+0x30,
				((iRPM/10)%10)+0x30,
				(iRPM%10)+0x30,
				'\0'};
```

```
                        fs_fprintf(file, "%s Hz=%s RPM=%s #SATS=%s\r\n", curGPS, strHz,
                                strRPM, strSats);
                        //fs_fflush(file);
                }
        }
/*********************************************/

        // Update the display if it needs to
        if(needUpdate)
        {
                if(!vStat||!tStat)
                        sprintf(topLine, "Env Unsafe!");
                else if(!running)
                        sprintf(topLine, "Paused");
                else
                {
                        if(iRPM==0)
                                sprintf(topLine, "No Signal");
                        else
                                sprintf(topLine, "RPM=%i", iRPM);
                }

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

// Function:  int4_isr
// INTP4 interrupt handler.  Increment int4_cnt
void    interrupt far int4_isr(void)
{
        t0_cnt = inport(0xff50);            // Read timer 0
        outport(0xff50,0);                  // Reset timer
        /* Issue the Non-Specific EOI for the interrupt */
        outport(0xff22,0x8000);
}

// Function:  int2_isr
// INTP2 interrupt handler.  Increment int2_cnt
void    interrupt far int2_isr(void)
{
        t1_cnt = inport(0xff58);            // Read timer 1
        outport(0xff58,0);                  // Reset timer
        /* Issue the Non-Specific EOI for the interrupt */
        outport(0xff22,0x8000);
}

// Function:  t2_isr
// t2 interrupt handler.  Increment t2_cnt
void    interrupt far t2_isr (void)
```

```
{
        t2_cnt++;
        /* Issue EOI for the interrupt */
        outport(0xff22,0x8000);
}
```

APPENDIX C: MapInfo/MapBasic Software

1. MapBasic Code Description

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

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

When called to in the main function, the "inputturbulence" function will prompt the user the for the rpm text file (which can be located on the flash card). Once the text file is inputted, MapInfo will then create a table of GPS coordinates including the rpm value, frequency value, number of satellites, and Coordinated Universal Time (UTC date and time). The first step in this function is to create the variables that are going to be needed in this function. The variables in this function are tablename, yNorS, xEorW, readline, temp1, temp2, temp3, temp4, temp5, temp6, P, x, y, v, f, s, day, month, year, date, hour, min, sec, and time. Declaring these variables is done by using the "dim" statement. Once the variables are declared the program prompts the user to open the text file which contains the pothole information. This is done by the "open file FileOpenDlg" statement. Next, the table that contains the turbulence information is created which is executed by "create table." Once created, a while loop begins to read all the needed information off of the text file. Each piece of needed information is located on the same spot of each line of text which makes the "mid$" statement very valuable in this process. Each line of text is stored into the "readline" variable and the "mid$" statement can take information from a given location on the stored line of text. Once this text is located MapBasic uses a statement called "val" which takes the string and converts it into a numeric value. The first numeric value is stored in temp1 and the following in temp2. Temp2 is then divided by 60 and then stored into temp3. This process is done because the GPS information needs to be converted from decimal minutes to decimal degrees. Once the correct GPS value is calculated the program checks if it is in the northern or southern hemisphere. This is done by using an "if…then" statement. After the latitude is finished being calculated the longitude goes under the same process. Next, the rpm and frequency values are read in along with the number of satellites and this information is stored into variables "r", "f", and "s" respectively. After the location, voltage, and satellites are stored, MapBasic reads in the date and time of each point. This text is then formatted for easy use. The next step sets the style of points. As of right now each point is the same but once a correlation is established between RPMs

and turbulence, points can be created to distinguish differences. This step still requires a lot of testing. Once the value has been calculated, MapBasic creates a point using the "create point into variable" statement. Each point is put into the table and is ready to be plotted onto a map. When the text file is ended, the program will close the text file and return to the main function.

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

2. MapInfo Software

This part of the supplement describes how to use the created executable MapBasic file. First, double click on the "mqp.mbx" file to open it. On the menu bar there will be an "Add Turbulence" menu as seen in **Figure 42**.

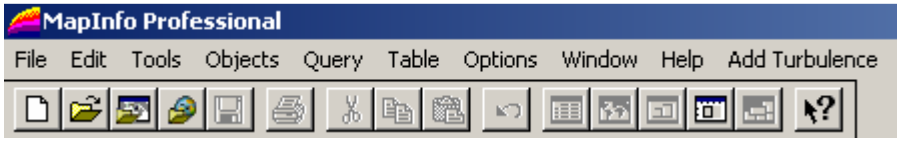

Figure 42: Add Turbulence Menu

Click on this menu to open it and two options will be given, "Read Turbulence" and "Create Map". Click on the "Read Turbulence" option first which can be seen in **Figure 43**.

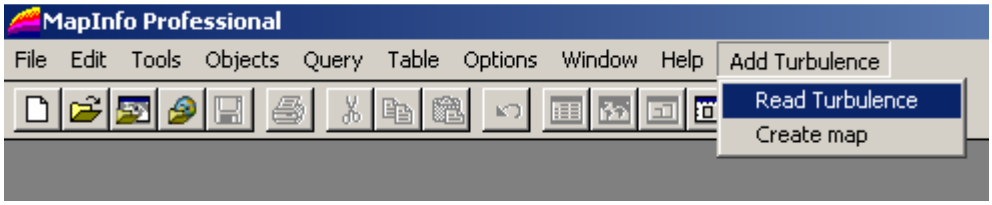

Figure 43: Read Turbulence

This will open a dialog box asking to open the RPM.txt file which can be seen in Figure 44. Move to the directory where the RPM file is located and open it.

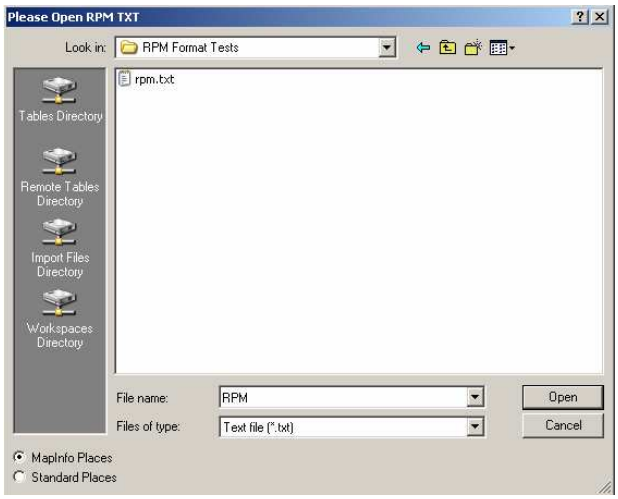

Figure 44: RPM Dialog Box

Next, follow Figure 45 and click on the Create Map option in the menu.

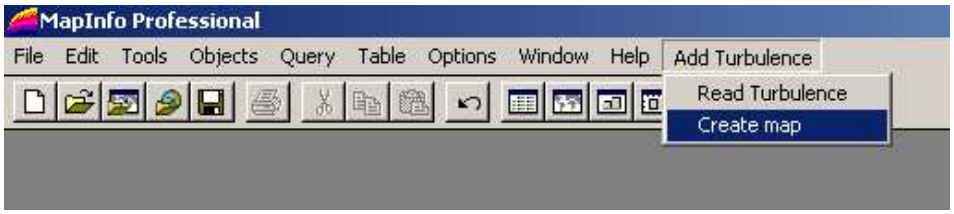

Figure 45: Create Map

This will open a dialog box that asks the user the open the Worcester workspace. This can be seen in **Figure 46**.

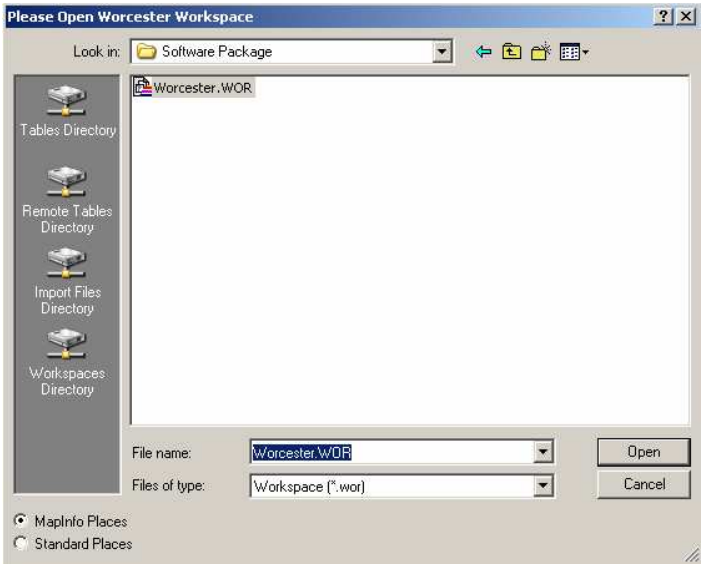

Figure 46: Worcester Workspace

After the workspace has been open, another dialog box appears and prompts the user to open the potholes table that was created when “Add Turbulence” was executed. This dialog box is shown in **Figure 47**. After being opened a map with the points will be displayed. This is the file product of this program. A sample map can be seen in **Figure 48** and a sample browser window can be seen in **Figure 49**. When exiting MapInfo you will be asked to save the file, the option is up to the user to save the results.

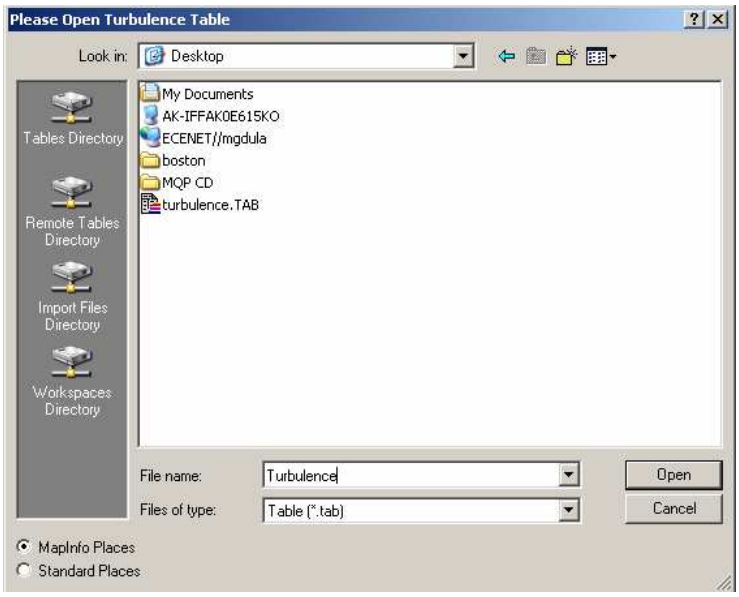

Figure 47: Turbulence Table

Figure 48: Sample Map

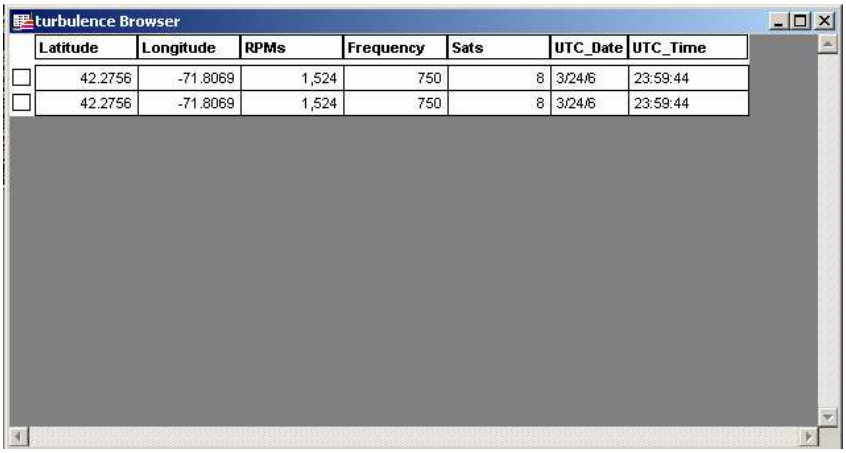

Figure 49: Browser Window

3. Map Basic Code

```
' City Lab MQP
' Adding Turbulence points

' Declare functions
declare sub main
declare sub inputturbulence
declare sub map

sub main

' Create a menu for MapInfo to execute program
create menu "Add Turbulence" as "Read Turbulence" calling inputturbulence, "Create map"
calling map

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

end sub

' plot the worcester table and turbulence table
sub map

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

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

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

' remove zoom layering constriction to all grids

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

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

Browse * from turbulence
end sub

' input the rpm text file and convert to latitude and longitutde, making a table
sub inputturbulence

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


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

' give the table's name "turbulence"
tablename = "turbulence"

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

' create a map for turbulence
create map for tablename

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

' read entire line
line input #1, readline

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

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

'Read in RPMs, sats, and frequency
r = val (mid$(readline,65,4))
s = val (mid$(readline,84,2))
f = val (mid$(readline,73,4))

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

time = hour + ":" + min + ":" + sec

day = val (mid$(readline,54,2))
month = val (mid$(readline,56,2))
year = val (mid$(readline,58,2))
date = month + "/" + day + "/" + year


' set symbol for potholes
Include "mapbasic.def"

Set style Symbol MakeSymbol (33,GREEN, 15)


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

wend

' close rpm.txt file
close file 1

end sub
```

APPENDIX D: Value Analysis

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

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

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

Quality

Size: How large is the microcontroller

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

Temperature Range: How severe of conditions can the microcontroller withstand

Extremely Well – 5
Good – 4
Fair – 3
Bad -2
Terrible – 1

Specifications

Ports: Amount of Serial Ports

- Five or more – 5
- Four to Five – 4
- Two to Three – 3
- One – 2
- Zero – 1

Expandability: The ease to expand the microcontroller

- Very Easy – 5
- Easy – 4
- Moderate – 3
- Poor – 2
- Incapable – 1

Development: The software and peripherals included

- Everything needed – 5
- Complete Software and Microcontroller – 4
- Partial Software and Microcontroller – 3
- Just Software – 2
- Just Microcontroller – 1

Language: The ease of programming it

- Very Easy – 5
- Easy – 4
- Moderate – 3
- Hard – 2
- Very Hard – 1

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

- Ten or more ADC and Twenty Digital I/O – 5
- Seven to Nine ADC and Fifteen to Nineteen Digital I/O – 4
- Four to Six ADC and Ten to Fifteen Digital I/O – 3
- Two or Three ADC and Five to Nine Digital I/O – 2
- One ADC and One to Four Digital I/O – 1

Power: The amount of power it takes to run

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

Speed: How fast it runs

Extremely Fast – 5
Fast – 4
Moderate – 3
Slow – 2
Very Slow – 1

Price: The Cost of the Development Kit

Expensive – 5
Costly – 4
Reasonable – 3
Economical – 2
Inexpensive – 1

Once going through each individual criterion and giving the microcontroller their values, we came up with **Table 4**.

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

Table 4: Microcontroller Value Analysis

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

APPENDIX E: Motherboard Schematics

This appendix illustrates the interconnections made on the motherboard to connect the TERN FlashCore-B microcontroller to a temperature sensor, a voltage meter, power, as well as different headers. The different headers on the motherboard connect to the FlashCore-B pins for expansion. **Figure 50** details all the wirings on the motherboard. The PCB layout shown in **Figure 51** displays the layout of the motherboard, and **Figure 52** shows the PCB wiring schematic. The blue wire layer on the wiring schematic shown in **Figure 52** is the top copper layer and the green is the bottom copper layer. Please see source Schematic and PCB files on our project CD for more details.

IMPORTANT NOTE:

When programming the TERN FlashCore-B microcontroller, a jumper must be placed or removed between J2.1 and J2.3 header (On the actual FlashCore-B board). If the microcontroller is mounted on the motherboard, these two pins can be accessed from H2 on the motherboard. Placing a jumper between H2.7 and H2.8 allows setting up the microcontroller for debug mode or for standalone mode. Likewise if a module is placed on H1 and H2 (On the motherboard), these two pins (H2.7 and H2.8) still have to be accessible for the placement or removal of a jumper between them. Please see the TERN FlashCore-B(FB)^{TM} Technical Manual for more information on the FlashCore-B Programing Overview. The red square, shown in **Figure 51**, shows where a jumper should be placed on the Motherboard PCB schematic.

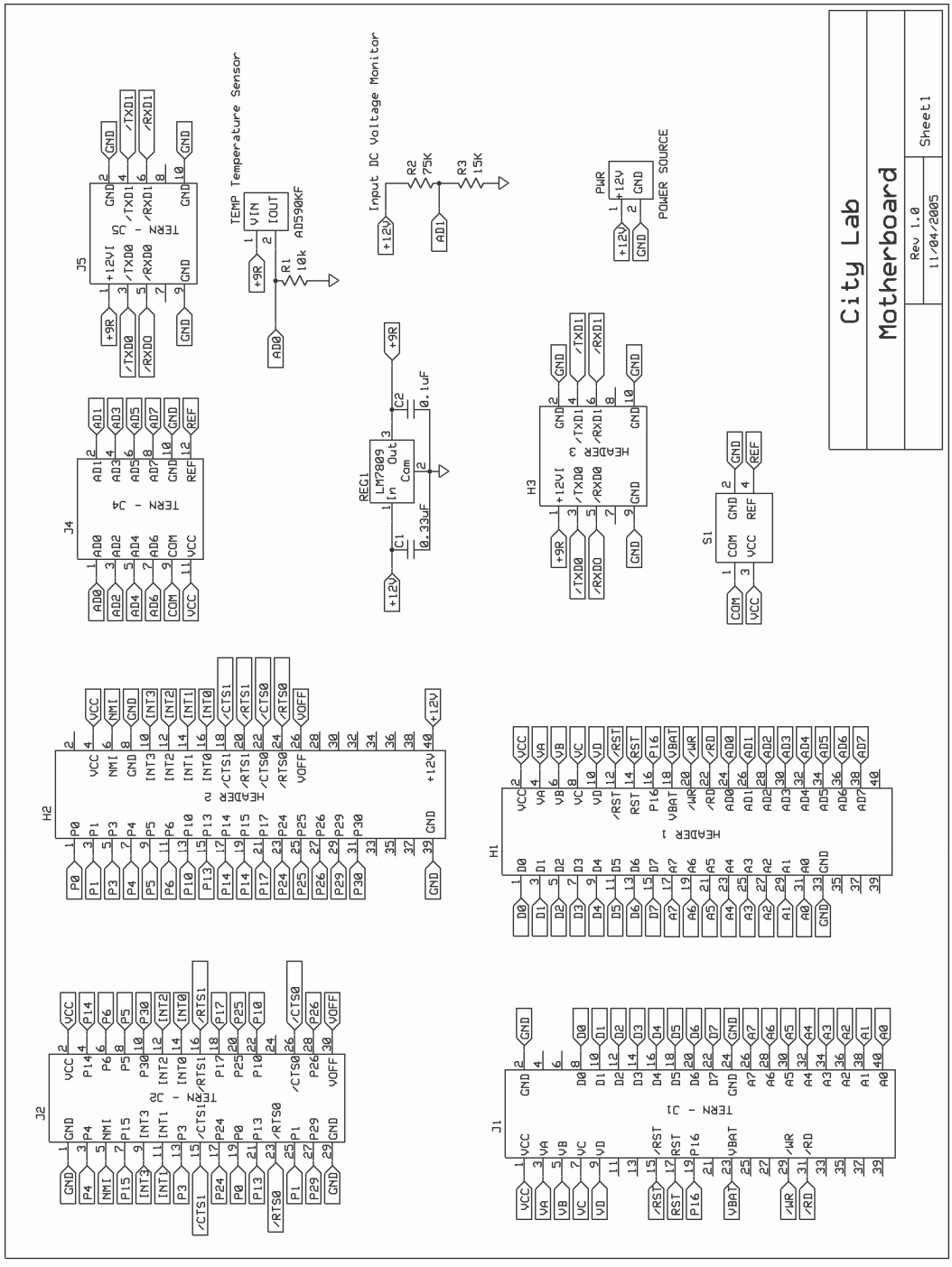

Figure 50: Motherboard Inter Connections

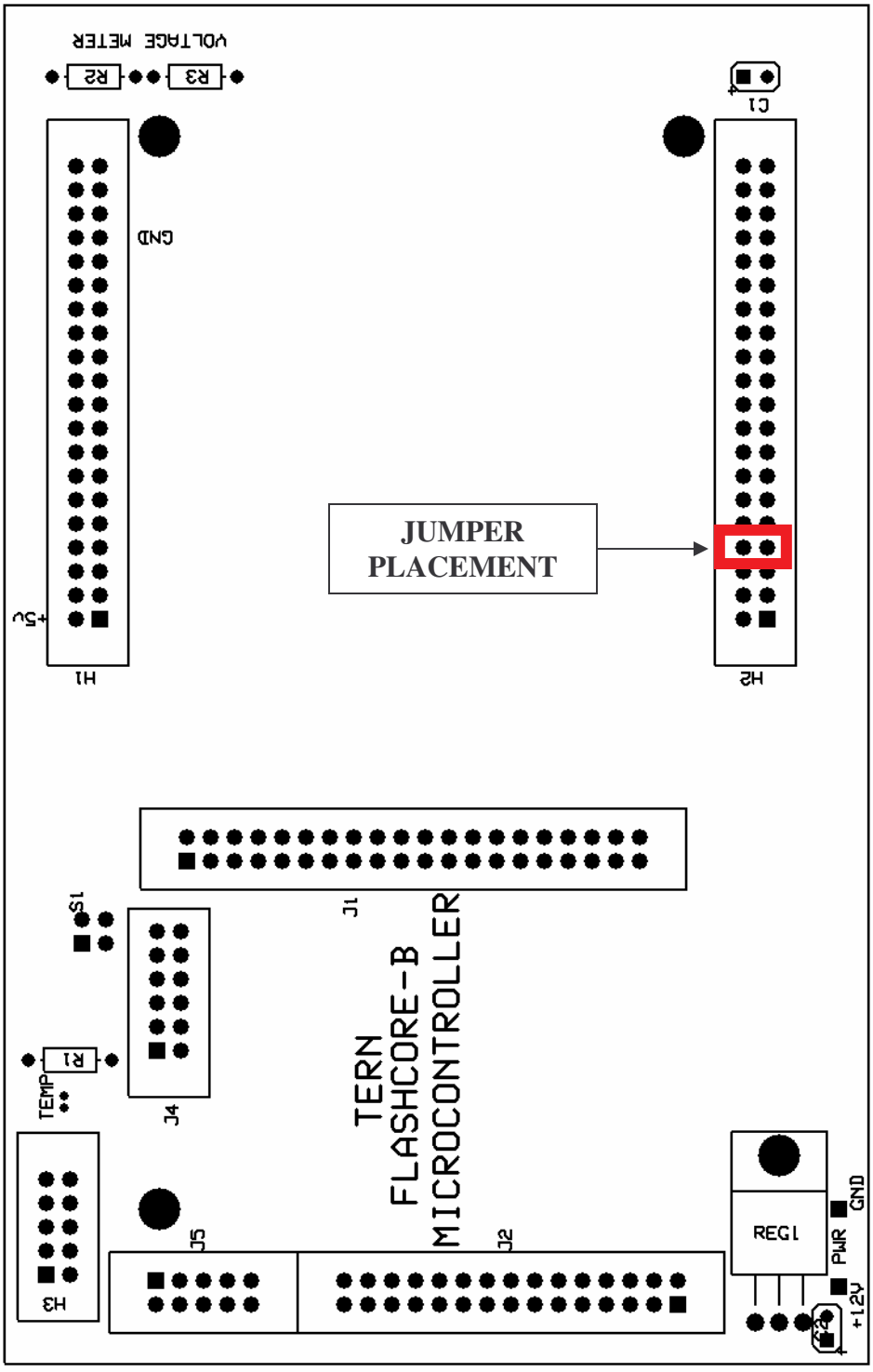

Figure 51: Motherboard PCB Silkscreen Schematic

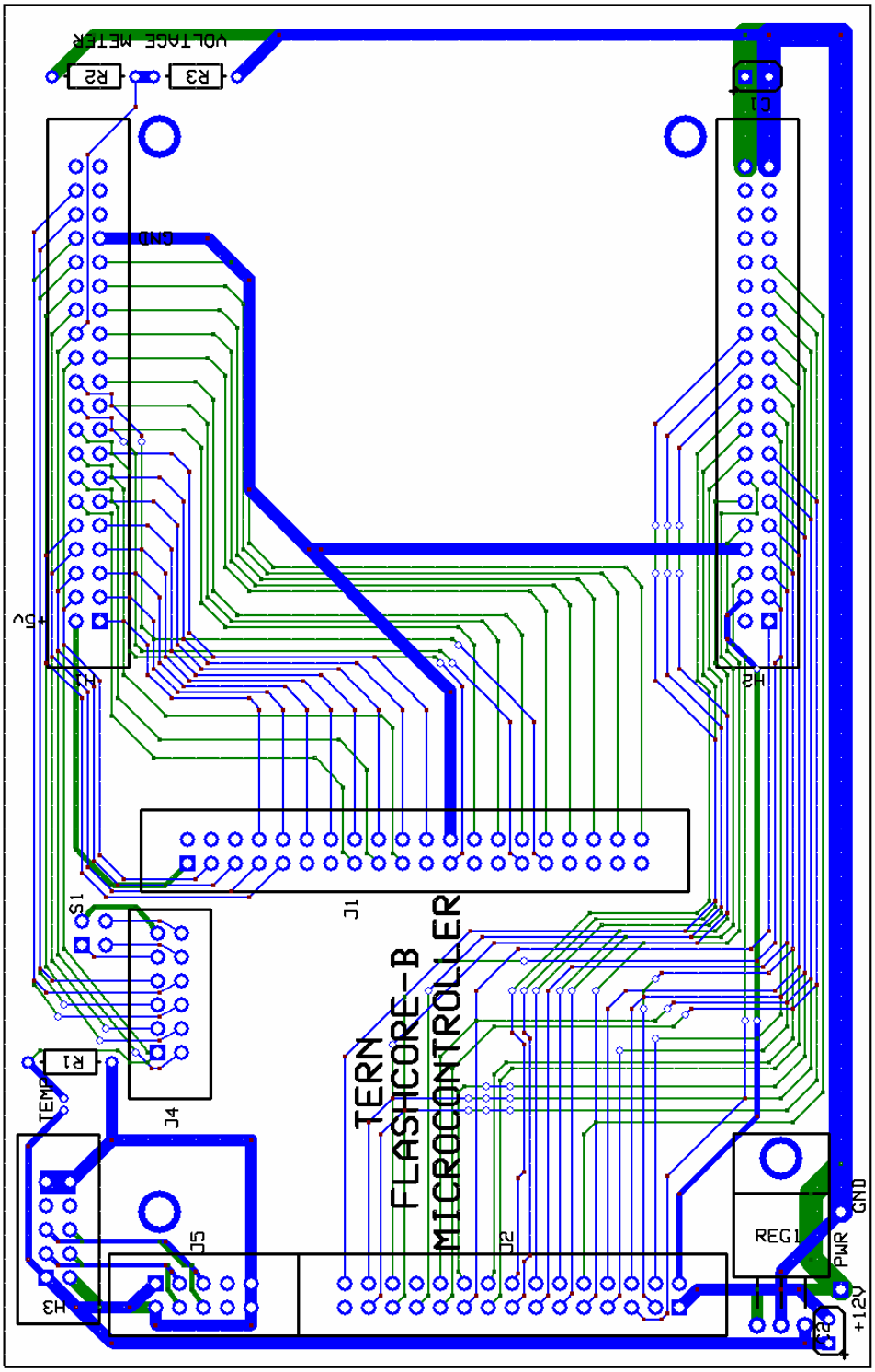

Figure 52: Motherboard PCB Wiring Schematic

APPENDIX F: RPM Module Schematics

This appendix illustrates the schematics of the RPM Module circuits designed by our project team. The H1 and H2 headers shown in **Figure 55** connect the outputs of the signal conditioning circuits to the same headers (H1 and H2) on motherboard; the motherboard allows the RPM module to interface with the FlashCore-B. The H3 and H4 headers, also shown in **Figure 56**, provide the pins to connect the input signals to their respective signal conditioning circuitry. The PCB layout shown in **Figure 56** displays the layout of the RPM module, and **Figure 57** shows the PCB wiring schematic. The blue wire layer on the wiring schematic shown in **Figure 57** is the top copper layer and the green is the bottom copper layer. Please see source Schematic and PCB files on our project CD for more details.

IMPORTANT NOTE:

When programming the TERN FlashCore-B microcontroller, a jumper must be placed or removed between J2.1 and J2.3 header (On the actual FlashCore-B board). If the microcontroller is mounted on the motherboard, these two pins can be accessed from H2 on the motherboard. Placing a jumper between H2.7 and H2.8 allows setting up the microcontroller for debug mode or for standalone mode. Likewise if a module is placed on H1 and H2 (On the motherboard), these two pins (H2.7 and H2.8) still have to be accessible for the placement or removal of a jumper between them. Please see the TERN FlashCore-B(FB)^{TM} Technical Manual for more information on the FlashCore-B Programing Overview. The red square, shown in **Figure 56**, shows where a jumper should be placed on the RPM Module PCB.

IMPORTANT NOTE:

A revision to our design was made after PCB's of the RPM module were printed. Please see the source Schematic and PCB files for more details on the connections. The double sided arrow shown in **Figure 56** shows the two pins that have to be connected, via a long jumper, **for the proper operation of the device using the code for the diesel RPM monitoring application. No IC is supposed to be placed in U2. U2.2 should be connected to H2.12**

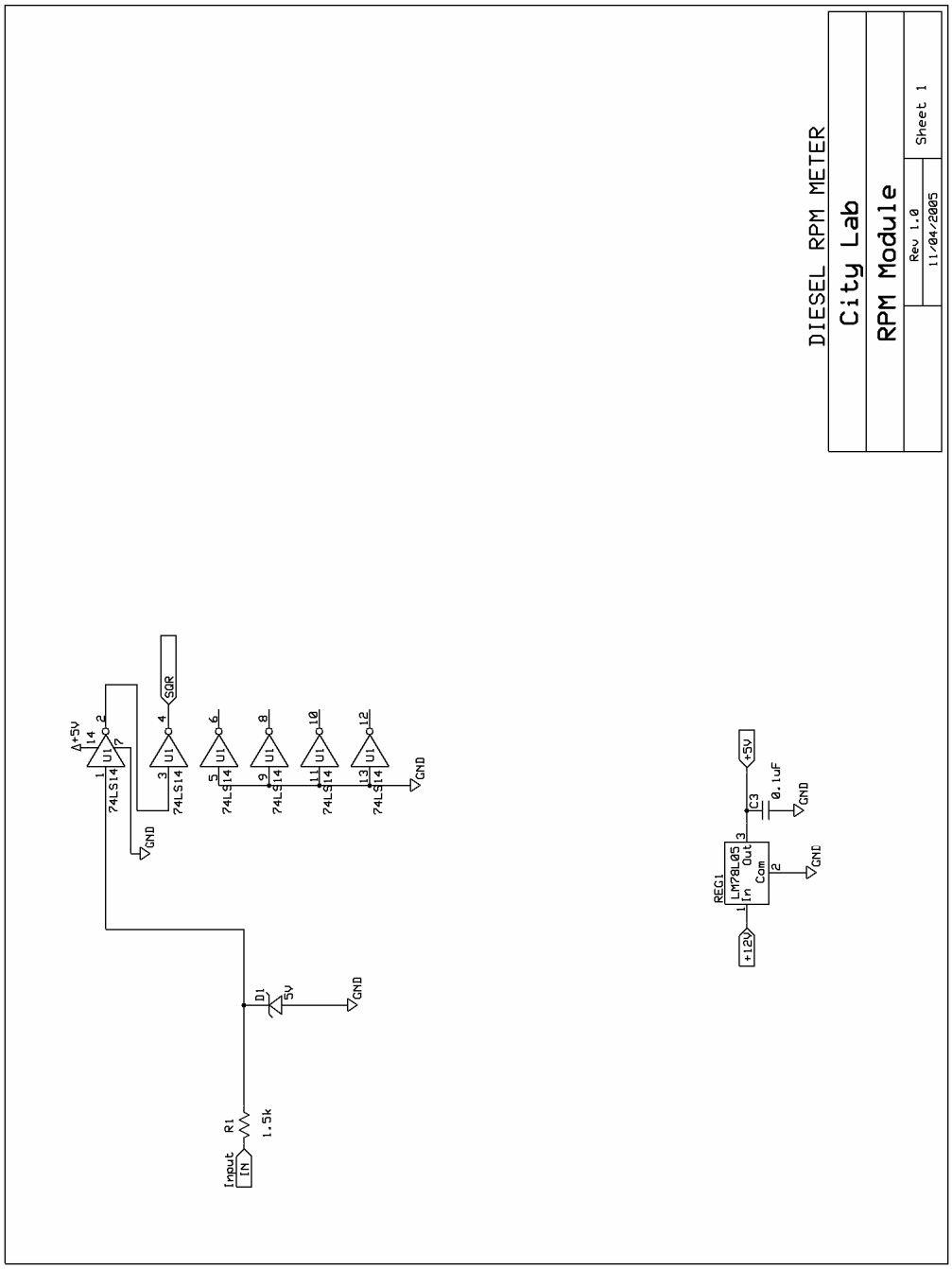

Figure 53: Diesel RPM Conditioning Circuit

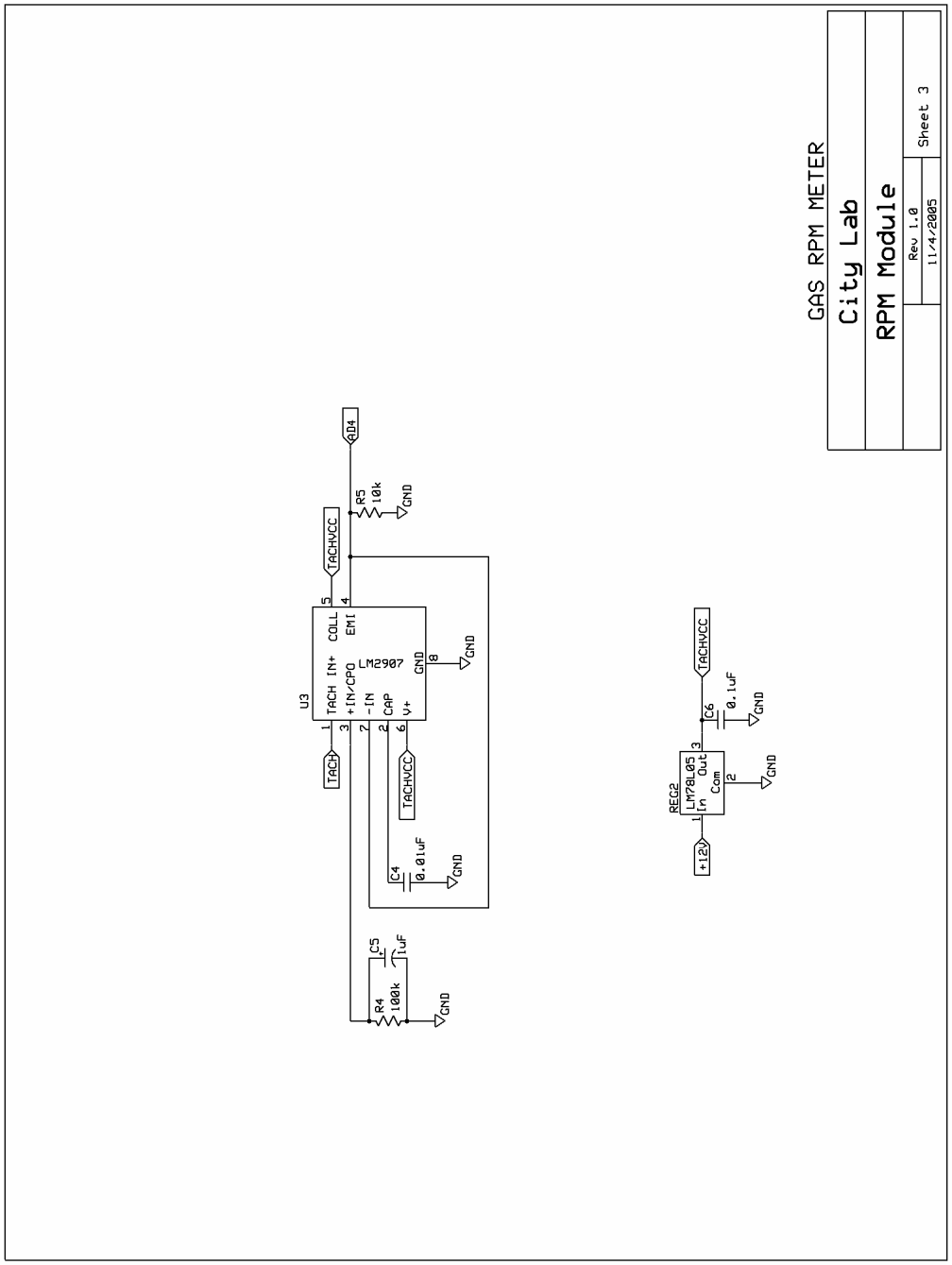

Figure 54: Gasoline Engine Conditioning Circuit

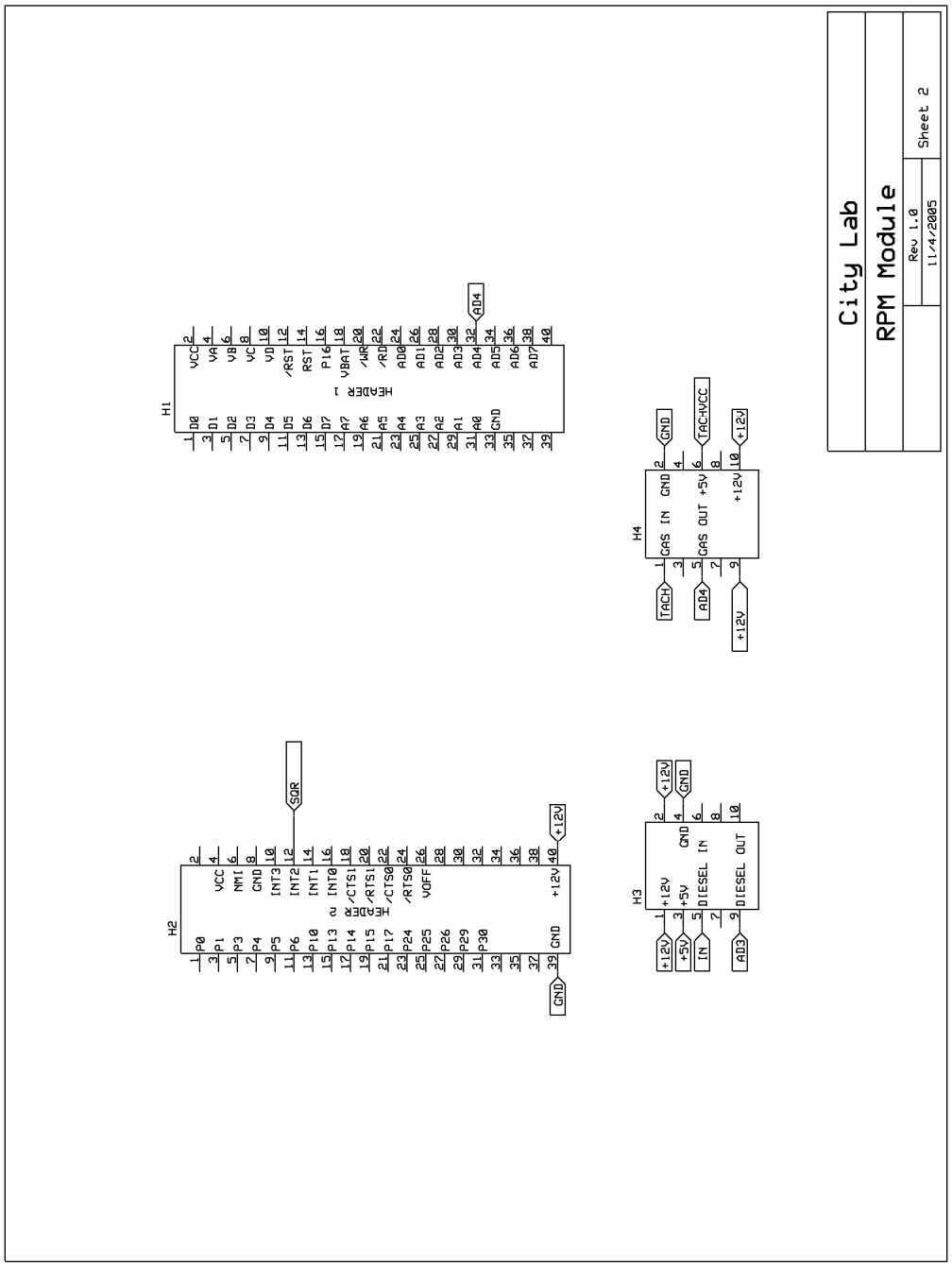

Figure 55: RPM Module Header Connections

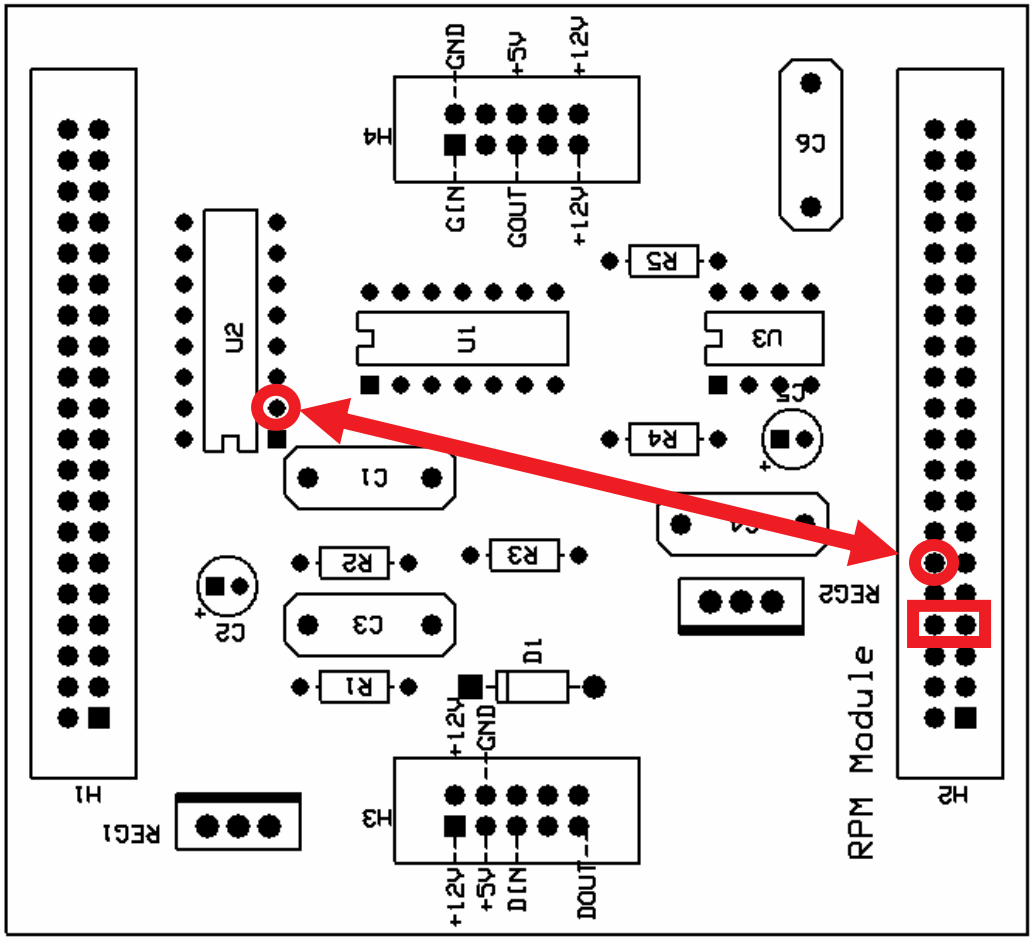

Figure 56: RPM Module PCB Silkscreen Schematic

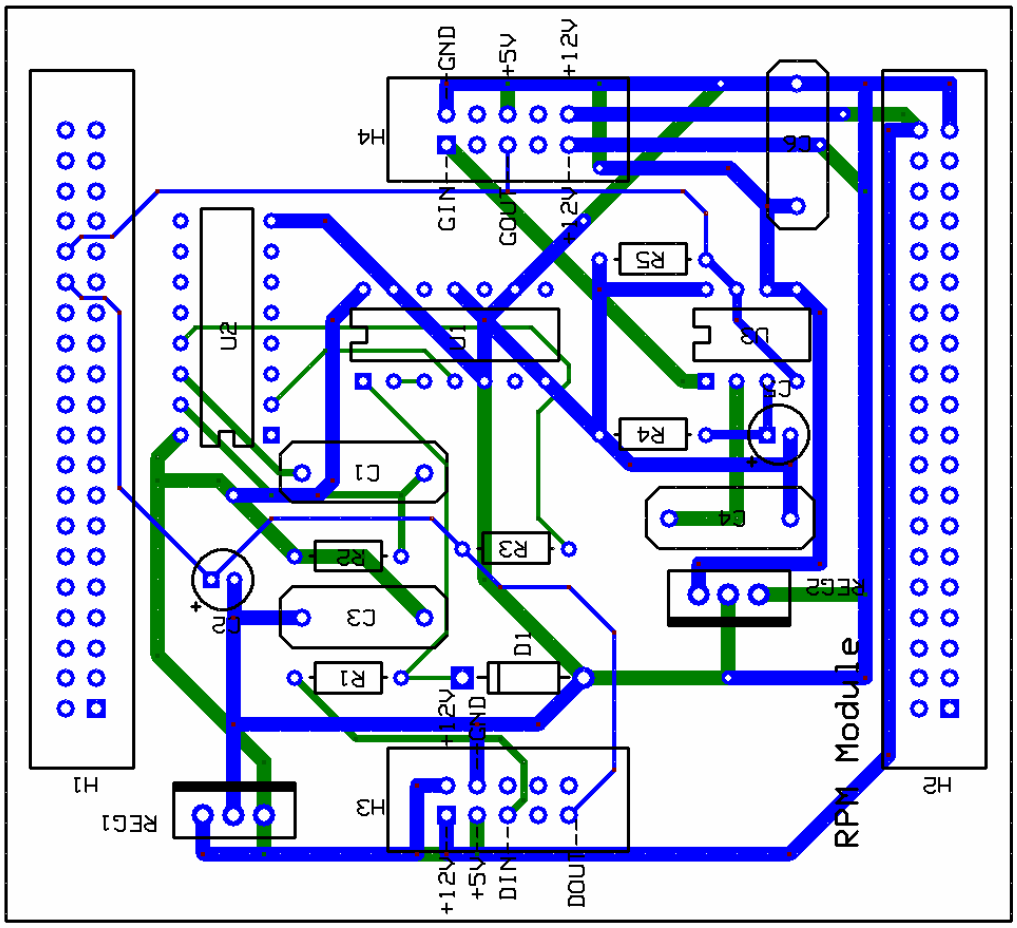

Figure 57: RPM Module PCB Wiring Schematic

APPENDIX G: Executive Summary

Introduction

This appendix presents the executive summary of our report to serve as a brief overview and summary of the full report.

Project Code: FC-TRB2

Mapping Underwater Turbulence in Venice

Executive Summary

Project Team:

Nicholas Angelini
na1303@wpi.edu

Jose Brache
jbrache@wpi.edu

Matthew Gdula
mgdula@wpi.edu

Craig Shevlin
cshvelin@wpi.edu

Advisors:

Fabio Carrera
carrera@wpi.edu

Michael Ciaraldi
ciaraldi@wpi.edu

Fred Looft
fjlooft@wpi.edu

Date: April 26, 2006

gps-05@wpi.edu

100 Institute Road Worcester, MA 01609

1. Introduction

Canal walls in the city of Venice, Italy have been subject to severe damage over the past century. These walls serve as the structural foundation for buildings throughout this unique city. Constant repair efforts are necessary in the ongoing effort to limit the structural decay of these buildings; this process comes at a high cost to the government.

There are many theories and known reasons of the causes of canal wall damage. One hypothesis, made by city structural engineers, is that underwater turbulence caused by motorboats that accelerate quickly, to stop and go, as they dock on the very walls of canals put so much pressure on the walls over time that this can lead to collapse. Engineers, however, have not been able to verify this statement, and thus have been unable to conclusively determine the reason for rapid deterioration of canal walls.

This document details the design process undertaken by our project team to design and build a working prototype for a fully automated data collection system that can be used to track these instances of underwater dispersion in the form of motorboat engine RPM.

2. Background

The introduction of the motorboat solved many transportation issues in Venice. Presently motorboats are necessary for the regular day to day operation of the city; they provide transportation for its citizens and millions of tourists yearly, as well as transportation of perishable goods and wastes. The city of Venice has suffered many problems that can be attributed to traffic. Canals are now heavily congested and many blame the increased amount of repairs needed on the structural integrity of many buildings to motorized transportation.

The problem of an increasing number of repairs that have to be made to this unique city has become a major concern for its citizens, public works entities, private businesses, and the government. There are differing opinions on what is causing damage to canal walls and how the problem should be addressed, but currently repair efforts are a serious economic concern to the Venetian government.

Two of the main factors attributed to causing canal wall damage include: physical changes to the building material and external events. Boat wakes can start a cycle of events that can cause the mortar holding the bonds between bricks to weaken. Parts of walls can then easily get knocked out of place by other forces, such as a boat docking on a canal wall. Once a small crevice is formed within the walls, underwater boat turbulence can cause more extensive damage.

Other factors contributing to the damage inflicted to the canal walls are rising sea levels and the sinking of the Venetian land mass. The rising sea level and the extraction of water from aquifers underneath the city, for industrial purposes, have caused the land mass which Venetian buildings rest on to sink. The water levels in the Venetian lagoon have risen approximately 23 centimeters since 1897.

Motorboats, however, cause more significant damage to Venetian structures. In addition to a boats wake produced when a boat propels itself, underwater turbulence caused by engine propellers weaken the structural integrity of canal walls and its foundations. The constant underwater thrusting of the canal walls from propelled water is cause for concern.

3. Project Goal

Underwater turbulence caused from motorboat propellers is believed by many to be the cause of erosion to the canal walls in Venice. The problem is that, currently, there are no correlations that have been made to connect underwater turbulence to instances of canal wall damage. One method of studying this correlation is to record instances of motorboat engine RPM, since this is a way of measuring how fast propellers are moving underwater. Instances of engine RPM would have to be mapped across the city, especially where motorboats stop and go, so that underwater turbulence can be correctly correlated to places suffering from canal wall damage.

Our project goal was to: **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 lagoon.**

4. Methods

This section outlines the methods used by our project team to accomplish our project goals. Briefly, we had to determine our system requirements when choosing a microcontroller to fit our needs. The team also had to research different means of measuring engine RPM for both gasoline engines and diesel engines. As a result, the approach used to develop our system was as follows:

1. Reviewed engine type monitoring requirements (Required to work for diesel or gasoline engines)
2. Reviewed system requirements (What needed to be recorded, for how long, how often e.g. every second, etc.)
3. Implemented the design from system requirements
4. Implemented packaging requirements (Easy to mount, small & compact, water resistant, user interface)
5. Reviewed data processing requirements (How data from unit was going to be used to produce maps)

Once completing this process, we feel that our system will be fully functional.

5. Results

This section outlines the process in measuring the signal from a w-terminal and describes the results our team encountered. The design of the diesel engine RPM conditioning circuit was based on the measurements made from an actual alternator on a diesel engine. Using a portable oscilloscope and a laptop our team traveled to Cape Cod, Ma where we measured the alternator w-terminal signal on a diesel boat. The signal obtained from the W-terminal can be seen in **Figure 1**, along with the frequency magnitude spectrum of the signal from the W-terminal shown in **Figure 2**. The signal from the W-terminal in **Figure 1** is at approximately 600 RPM was a 243 Hz signal.

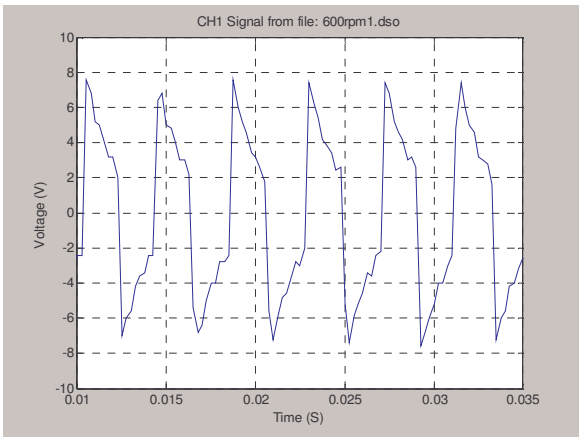

Figure 1: W-terminal signal

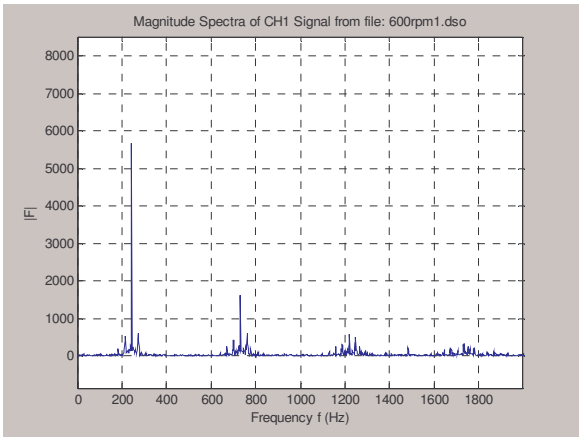

Figure 2: W-Terminal Signal Frequency Spectrum

Obtaining multiple signals with this process, we were able to plot plotted the measured relationship between the signal frequency and actual engine RPM which is shown in **Figure 3**.

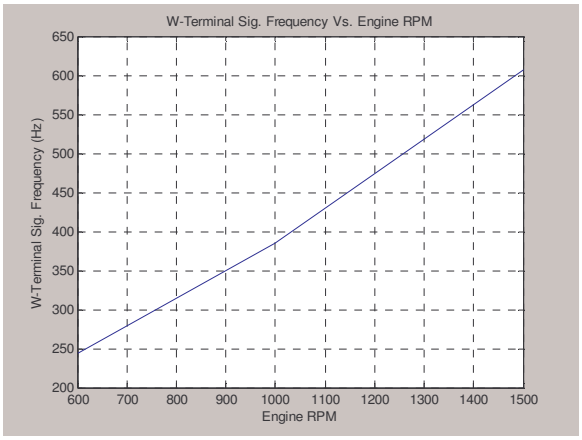

Figure 3: W-Terminal Sig. Frequency vs. Engine RPM Plot

Engine RPM can be calculated from certain alternator parameters. These parameters include:

1. W-terminal signal frequency
2. Number of poles on an alternator
3. Pulley ratio

The number of poles on the alternator requires looking at the specific alternators data sheet and the pulley ratio depends on the **Equation 1**.

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

Equation 1: Pulley Ratio Calculation

Equation 1 shows how pulley ratio is an actual ratio determined by the drive pulley diameter and the alternator pulley diameter. This ratio can be calculated by measuring both diameters and often times is specified in engine manuals. Typically in diesel engines there is a 3:1 pulley ratio.

The mathematical equation to calculate actual engine RPM in terms of alternator w-terminal signal frequency, number of poles on the alternator, and the pulley ratio is shown by **Equation 2**:

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

Equation 2: Diesel Engine RPM Calculation

Once this relationship was found, we had to condition the signal to be a TTL compatible waveform since we planned on driving one of the external interrupts on the microcontroller. Using the TTL compatible signal to drive the external interrupts we calculated the signal frequency.

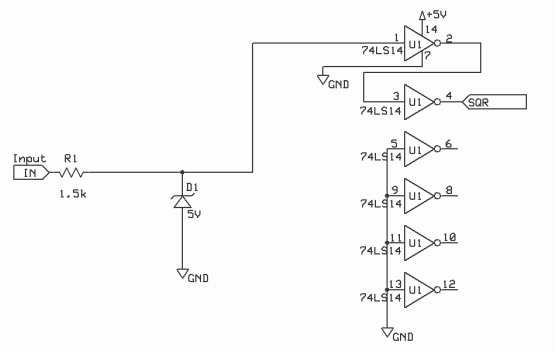

Figure 4: W-Terminal Conditioning Circuit Schematic

The schematic shown in **Figure 4** shows the conditioning circuit we designed to take the w-terminal input and convert that signal to a TTL compatible signal. The w-terminal signal is connected to the port labeled "IN" on the schematic. The resistor R1 limits the current of the w-terminal signal (**Figure 1)** to protect the 5V zener diode D1 and the hex inverting Schmitt trigger: 74LS14. If a waveform like **Figure 1** is connected to the input "IN", the 5V zener diode D1 regulates the voltage of the input waveform (of about (-)8V to +8V) to approximately a 0V to +5V waveform. The resulting waveform at the input 1 of the 74LS14 resembles a 0V to +5V square wave; however, this waveform is not as clean as a perfect square wave. The function of the 74LS14 (Hex inverting Schmitt trigger) is to produce a TTL compatible signal at the output "SQR", that has the same frequency as the w-terminal signal at "IN". The output signal at "SQR" is supposed to resemble a perfect square wave. For an input signal like the w-terminal signal shown in **Figure 1** the output signal resembles the waveform shown in **Figure 5**.

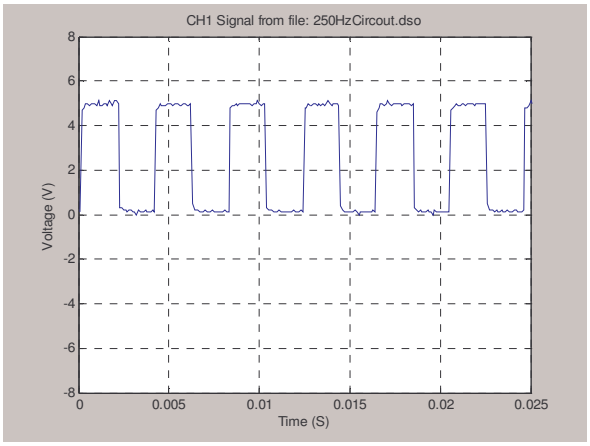

Figure 5: W-terminal conditioning circuit output

Once the design of the conditioning circuit was complete, the module devices were created. The modules consist of the RPM module and Motherboard Module. The RPM module consisted of the RPM conditioning circuit shown above. The RPM module, along with the microcontroller, were plugged into the Motherboard Module. Also on the Motherboard Module was the Health & Safety module. The basic function of the health and safety module is to provide temperature and readings and measure the voltage output from the device's power source. Our real-time embedded program on the microcontroller then acts accordingly depending on the voltage and temperature readings it gets (i.e. closes file on compact flash). After the hardware was designed and working together, the next step was to create the software to bring functionality to the entire system. The software created was based on the flowchart seen in **Figure 6.** Once going through a successful loop of the flowchart, the program then writes to compact flash depending on conditions like the number of satellites the receiver is tracking and if the file is open. The actual data recorded onto the Compact Flash card resembles the lines shown in **Table 1**. The data is stored as a text file on compact flash and then GIS software can be used to interpret the data graphically on a Venice, Italy city map.

$GPRMC,063403,V,4216.5799,N,07148.4762,W,000.0,000.0,300306 RPM=0980 Hz=0392 #SATS=00
$GPRMC,063404,V,4216.5799,N,07148.4762,W,000.0,000.0,300306 RPM=0980 Hz=0392 #SATS=00
$GPRMC,063405,V,4216.5799,N,07148.4762,W,000.0,000.0,300306 RPM=0980 Hz=0392 #SATS=00
$GPRMC,063406,V,4216.5799,N,07148.4762,W,000.0,000.0,300306 RPM=0980 Hz=0392 #SATS=00

Table 1: Compact Flash Stored Data

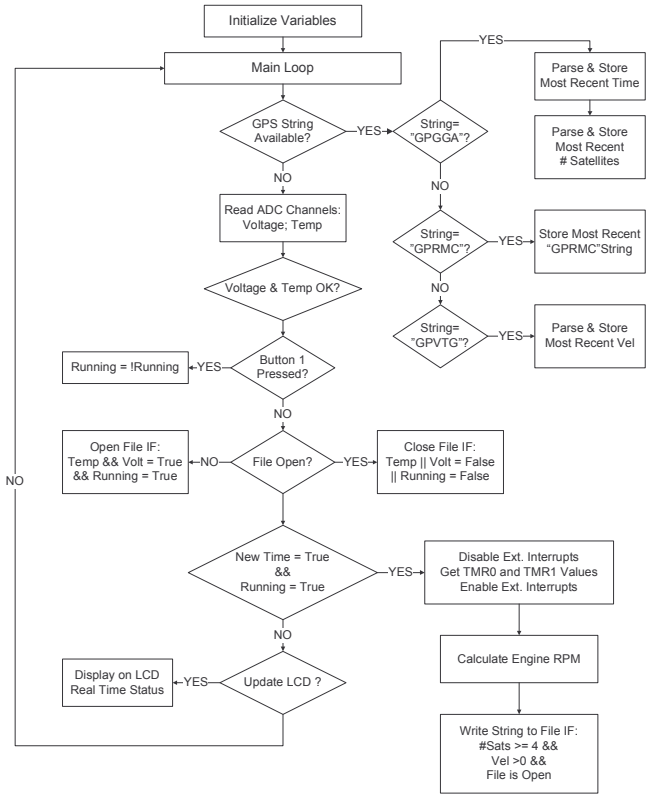

Figure 6: Software Flowchart

Upon completion of the software, in lab tests were done. To test the device, we had to model the W-terminal signal using an Instek Function Generator (CFG-8219A). We used an input square wave from the function generator as our signal. The frequency of the input square wave we connected to our W-terminal conditioning had a frequency of about 283 Hz, and the device measured an RPM of 708. This can be seen from the figure shown in **Figure 7**. When comparing the measured 708 RPM to an input signal having a frequency of 283 Hz we used **Figure 8** for comparison. As seen in **Figure 8**, these results are very close to the theoretical values.

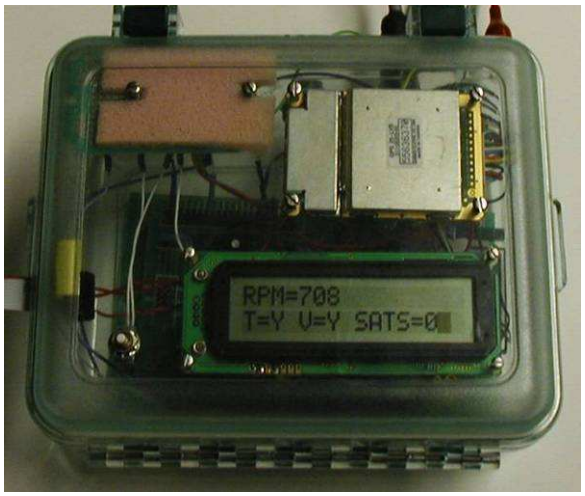

Figure 7: Device Displaying RPM in Real Time

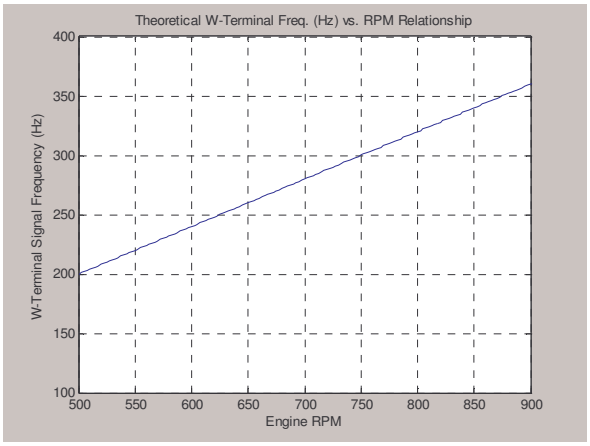

Figure 8: Theoretical W-Terminal Freq. vs. RPM Relationship

6. Summary and Conclusions

Our team successfully implemented an automated data collection system that can be installed in motorboats propelled by diesel engines to monitor and store engine RPM as well as GPS positional data, for boats that navigate the Venetian lagoon. This document briefly outlines the design process our project team followed in building a working prototype for this automated data collection system. Options in having this device placed in gasoline motorboats have to be tested and implemented. In addition, the post processing of data stored on compact flash by our system has to be considered after multiple field tests are conducted. Different considerations, such as reducing power consumption, the size of the current package, and further testing should be taken to improve the functionality of the device and system. Future designs and methods could serve as a tool for aiding the city of Venice, Italy in the repair of damaged canal walls.