OnBoard Travel
Technologies

School of Engineering Science
Simon Fraser University
Burnaby BC V5A 1S6
onboardtravel@gmail.com

March 9, 2006

Dr. Andrew Rawicz & Mr. Steve Whitmore
School of Engineering Science
Simon Fraser University
Burnaby BC V5A 1S6

Dear Dr. Rawicz and Mr. Whitmore

Please find the attached document, *Design Specification for a Mileage Recorder for Small Business Owner/Operators*. The document is comprehensive and outlines in detail the design of our ENSC440/305 project.

The Design Specification document will detail the design of the hardware, software, and firmware as needed to build a functional prototype of our mileage recorder device. The document will also outline our test plan to ensure that our device meets our functional specifications.

The Design Specification document will aid all members of our group as we strive to finish our prototype for the beginning of April 2006.

OnBoard Travel Technologies is comprised of four highly motivated and skilled engineers: Bergen Fletcher, Ali Abdul-Hussein, Lena Lee, and Patrick Perrella. Please forward any questions or concerns to onboardtravel@gmail.com.

Best Regards,

Bergen Fletcher
Chief Executive Officer
OnBoard Travel Technologies

**Enclosure:** *Design Specification for a Mileage Recorder for Small Business Owner/Operators*

OnBoard Travel Technologies

School of Engineering Science
Simon Fraser University
Burnaby BC V5A 1S6
onboardtravel@gmail.com

## Design Specification for a

# Mileage Recorder for Small Business Owner/Operators

Project Team: Bergen Fletcher
Ali Abdul-Hussein
Lena Lee
Patrick Perrella
Contact: Patrick Perrella
onboardtravel@gmail.com

Submitted to: Dr. Andrew Rawicz, ENSC 440
Steve Whitmore, ENSC 305
School of Engineering Science
Simon Fraser University
Issued Date: March 9, 2006
Revision 1.0

# Executive Summary

Many business owners track their vehicle business usage versus total use for income tax deduction purposes using methods such as estimation and logbooks, but often they find those methods inconvenient and inaccurate. OnBoard Travel Technologies (OBTT) aims to provide a novel solution that is convenient, accurate, and cost-effective. OBTT plans to unveil an operational prototype of our solution early in April 2006.

OBTT's Plug n'Go system consists of two main subsystems; the Mileage Recorder Device (MRD) which is to be connected to the vehicle during all trips, and a PC application, which is used to display trip records and configure the MRD.

The development of Plug n'Go will take place in two main phases; the first phase will primary deal with the functional specifications of both subsystems as well as the implementation of the MRD hardware. The second phase will include full integration of the MRD firmware, PC software, and system testing and optimization. Upon the completion of the development phases, the system should support the following functions:

- The user may easily display recorded trip data using a PC with the Plug n'Go Windows application.
- The user may access and modify MRD configuration parameters.
- The MRD can be quickly connected to a vehicle and removed anytime the vehicle is stopped so that the MRD can be taken to a PC to which it can be connected facilitating communication with the Plug n'Go Windows application.
- The ability to accommodate multiple vehicles and multiple drivers.
- The user may easily read, interpret, and respond to the MRD user interface when the MRD is connected to the vehicle.
- The user may manually enter fuel purchases to be store in the MRD's memory.
- The MRD is capable of interpreting the vehicle speed sensor (VSS) signal and convert it to mileage data. The MRD can store this mileage information along with other relevant trip information in non-volatile memory.

This document contains full details of the proposed design specifications for the MRD hardware, device firmware, and Windows software as well a plan for testing these components.

OBTT is scheduled to produce operational prototypes of the MRD, the PC software and required interfaces to be ready for demonstration by the first week of April 2006.

# Table of Contents

# List of Figures

# List of Tables

# Acronyms

CRA – Canada Revenue Agency
CSV – Comma Separated Value
$I^2C$ – Inter-Integrated Circuit Bus
IC – Integrated Circuit
MRD – Mileage Recorder Device
MRDUI – Mileage Recorder Device User Interface
OBTT – OnBoard Travel Technologies
PC – Personal Computer
TWI – Two Wire (Serial) Interface
UART – Universal Asynchronous Receiver Transmitter
UML – Universal Modeling Language
USB – Universal Serial Bus
VAC – Volts, Alternating Current
VCD – Vehicle Cradle Device
VSS – Vehicle Speed Sensor

# 1 Introduction

OnBoard Travel Technologies' Plug n'Go is a system which consists of a compact device which can be connected to a vehicle's odometer to record the vehicle's mileage. The system also includes Windows® software which is capable of configuring the recorder device and downloading trip records from the device via a standard PC to peripheral interface, such as USB. The Plug n'Go system is intended to aid small business owner/operators in keeping adequate vehicle usage records such that they can claim income tax deductions correctly; ensuring proper records are kept in the event of an audit.

## 1.1 Intended Audience

The intended purpose of this document is to guide design engineers in developing the prototype of the system. The design specification document will also serve as a checklist for the company executives and project managers as they ensure that the development of the product is running inline with the intended design of the product. Certain design aspects of the device can also serve as marketing points and portions could be featured in future promotional material for the product.

## 1.2 General System Overview

The system will consist of a Mileage Recorder Device (MRD) which can be connected to a vehicle to acquire trip information and store such information in memory. The MRD can be easily removed from the vehicle and connected to a PC via a standard interface allowing the MRD to exchange data with the PC. Most notably, this data transaction will consist of downloading trip records to the PC. The system will include PC software which facilitates this data transaction and organizes the downloaded trip information into a user desirable format.

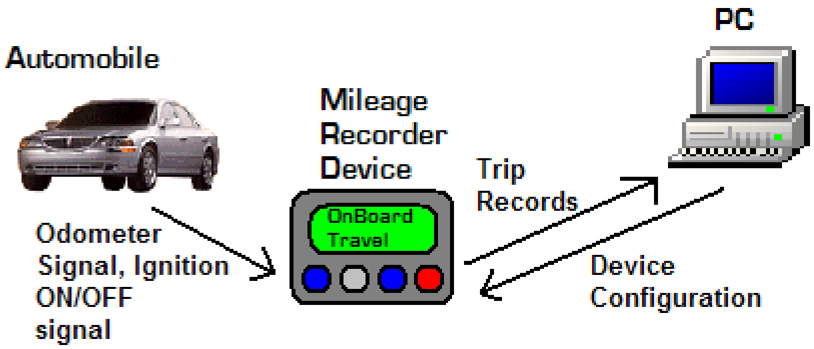


**Figure 1: System Overview**

# 2 Hardware Design

The Plug n'Go hardware is comprised of two components: the Mileage Recorder Device (MRD), and the Vehicle Cradle Device (VCD). Figure 2 outlines the major components of the MRD and Figure 3 outlines the major components of the VCD. The number of connections between blocks shown in these diagrams do not include power and ground, only data connections are shown.

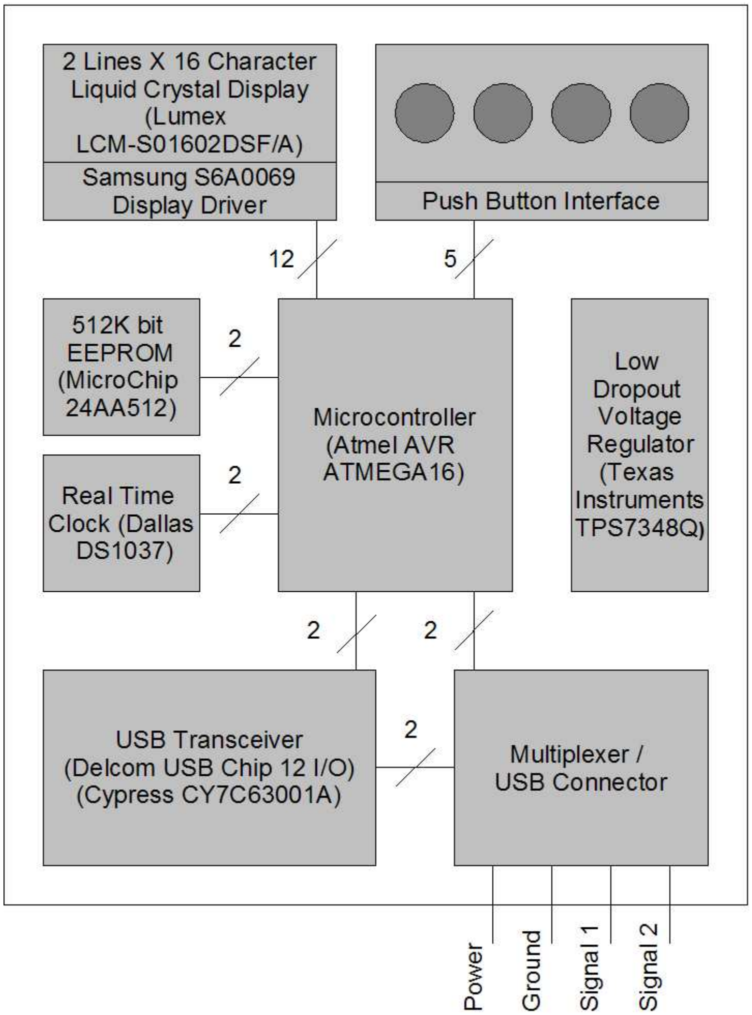

2 Lines X 16 Character Liquid Crystal Display (Lumex LCM-S01602DSF/A)
Samsung S6A0069 Display Driver
Push Button Interface
12
5
512K bit EEPROM (MicroChip 24AA512)
2
Real Time Clock (Dallas DS1037)
2
Microcontroller (Atmel AVR ATMEGA16)
Low Dropout Voltage Regulator (Texas Instruments TPS7348Q)
2
2
USB Transceiver (Delcom USB Chip 12 I/O) (Cypress CY7C63001A)
2
Multiplexer / USB Connector
Power
Ground
Signal 1
Signal 2

**Figure 2: Mileage Recorder Device - Hardware Block Diagram.**

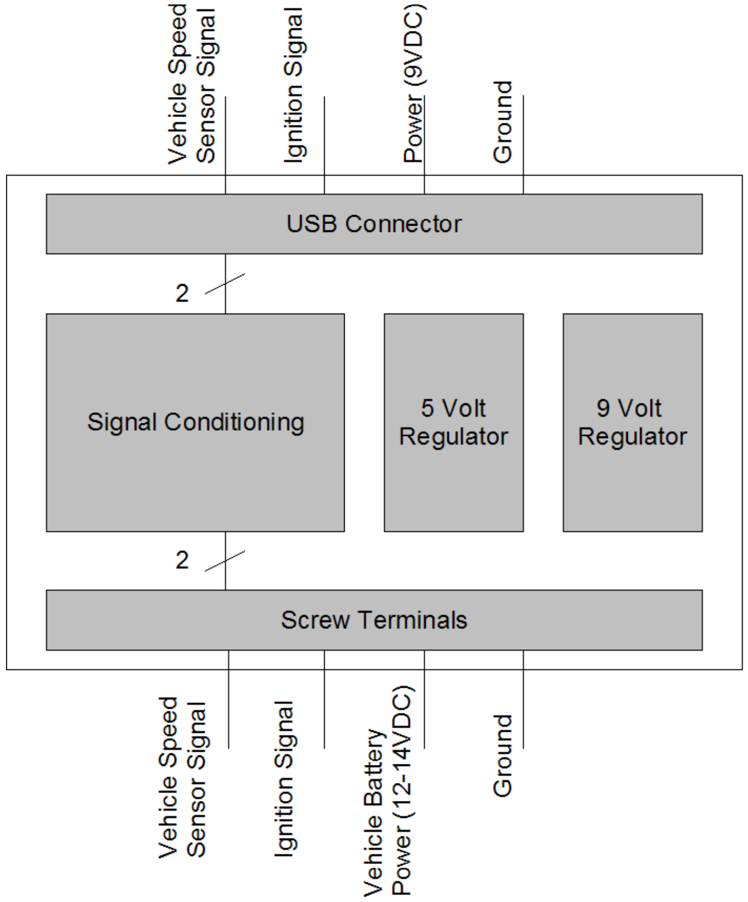


**Figure 3: Cradle device for signal and power supply conditioning in vehicle.**

# 2.1 Vehicle Cradle Device (VCD)

The VCD's primary purpose is to provide the MRD with a connection via a USB Type B connector to the vehicle's power and ignition and vehicle speed sensor (VSS) signals. The VCD pre-regulates the power from the vehicle from a level of 12-14 VDC down to 9 VDC and conditions the ignition and vehicle speed sensor signals. The VCD provides four screw terminals to allow wires to be connected into the vehicle's electrical system. In addition, a 1 amp quick-blow fuse should be connected between the vehicle's battery and the power input of the VCD. A complete schematic of the VCD circuit can be found in Appendix B.2.

## 2.1.1 Signal Conditioning

The signal conditioning in the VCD is accomplished by two LF347N op-amps that are configured to output a maximum voltage of 5VDC to ensure that the hardware within the MRD is protected from damage caused by over-voltage.

# 2.2 Mileage Recorder Device (MRD)

The MRD is a much more sophisticated system that the VCD – it contains all the necessary hardware to gather information from the user, interpret signals from the vehicle, store data in non-volatile memory, and transfer data to a PC. The MRD is a transportable device that can be either connected to the vehicle via the VCD or to a PC with an available USB port. The USB connection requires a cable with a type B male connector to the MRD and a type A male connector to the PC. The MRD multiplexes the USB connector so that it can also be used to acquire VSS and Ignition signals from the vehicle. The major components of the MRD will be discussed in this section while a complete schematic diagram can be found in Appendix B.1.

## 2.2.1 Atmel AVR ATMEGA16 Microcontroller

The Atmel AVR line of microcontrollers is a popular industry choice for many embedded systems applications because of its advanced RISC architecture, C programming language support, low cost, and a wide range of options to choose from in terms of flash program memory size, peripheral support, etc. We have selected the ATMEGA16 processor for numerous reasons: it is available in a 40 Dual Inline Pin (DIP) package easing the prototype assembly process, it has 16 KiloBytes of Flash Memory which should be sufficient to accommodate our application; it has a built-in counter that will allow us to count pulses from the VSS; it can be easily programmed using a relatively inexpensive development board and software; and it supports Inter-Integrated Circuit Bus ($I^2C$) peripherals. The ATMEGA16 will serve as the “brains” of the MRD and thus is the most important component in the circuit.

## 2.2.2 Delcom Engineering USB Chip

The Delcom Engineering USB chip is possibly the second most important component in the circuit as it enables communication between a PC and the MRD. The Delcom USB chip is basically a pre-programmed Cypress CY7C63001A USB-enabled microcontroller. Delcom has programmed the Cypress microcontroller to respond to commands issued by a Windows USB device driver that Delcom also provides. Delcom provides C headers and application examples that make development of Windows applications capable of communicating to the Delcom USB chip easy. The Delcom USB chip also supports a Universal Asynchronous Receiver Transmitter (UART) interface that is compatible with the ATMEGA16, allowing the ATMEGA16 to transfer data to and from a PC.

## 2.2.3 Liquid Crystal Display

The Liquid Crystal Display (LCD) is an important element of the MRD user interface (MRDUI) since it is used to provide information to the user. The LCD module we have chosen to use is a 2 line by 16 character display manufactured by Lumex. Each character is 5X8 dots and capable of displaying all alpha-numeric symbols used in the English Language. The LCD module includes a driver IC based on the industry standard Samsung S6A0069. The driver IC provides the logic which controls the LCD based on a set of commands received over an 8-bit parallel interface. The parallel interface also requires 3 additional signals: read/write, address/data register select, and read/write enable. This 11-wire interface is connected to general purpose input/output pins (GPIOs) on the ATMEGA16, allowing the ATMEGA16 to control the content displayed on the LCD. In addition the LCD module includes an LED backlight system and provides direct access to the anode and cathode of the LEDs. With the addition of a transistor to provide adequate current to the LED backlight, the ATMEGA16 can control the on/off state of the LED backlight.

### 2.2.4 Push Button Interface

The push button interface is another important component of the MRDUI since it allows the user to provide input and respond to the device. The push button interface consists of 4 momentary-on switches which pull ATMEGA GPIOs high when activated. The ATMEGA16 is not capable of generating interrupts for every GPIO and only provides 2 external interrupts which can be triggered by changes in logic levels. In order to have the push button interface generate an interrupt, two three-input or gates have been cascaded to allow any one of the 4 pushbuttons to generate an interrupt on the same interrupt pin. When the ATMEGA16 receives an interrupt on this pin it can read the state of the GPIOs for each of the 4 buttons to determine which has been pressed. Having the push buttons generate an interrupt is extremely important for program flow on the ATMEGA16 as it eliminates the need for constant polling of the GPIOs to which the buttons are connected.

### 2.2.5 Real Time Clock (RTC)

The RTC keeps track of the date and time so that trip and gas records can be associated with a certain date and time. Since the MRD needs to be disconnected from a power source when it is transferred from vehicle to PC, the RTC must be battery backed such that it can continue to keep time regardless of whether or not the MRD is connected to a power source. We have chosen the Dallas DS1307 RTC because it power itself from a small lithium 3V battery when disconnected from its regular power source. In addition the DS1307 can be configured and transmit time and date information via an Inter-Integrated Circuit ($I^2C$) interface. The $I^2C$ interface is supported by the ATMEGA16 and requires only 2 wires for interconnection between the two devices. Atmel refers to the $I^2C$ interface as the Two Wire Interface (TWI) for legal reasons, but for all intents and purposes $I^2C$ and TWI are the same thing. The use of 2 wires as opposed to a much larger parallel interface elegantly simplifies the layout and design of the circuit.

### 2.2.6 EEPROM

The EEPROM is used to store configuration data and trip and gas records in the MRD. To satisfy the functional requirement[2] of the Plug n'Go system the EEPROM must have enough non-volatile storage space to store at the very least 3 months of user trip and gas records. To more than satisfy this requirement we have selected a 512 KiloBit EEPROM, Microchip's 24AA512. The 512 KiloBit of memory will enable the storage of roughly 21 months worth of user trip and gas records. Just like the DS1307 RTC, the 24AA512 can be written/read via an $I^2C$ interface. The 24AA512 is also rated for 1 million erase/write cycles which means that if every 3 months the user were to download to a PC then erase data from the MRDUI, the EEPROM will last for over 25000 years.

### 2.2.7 USB Connector Multiplexer

To reduce the number of external connections on the MRD, the 4 pin USB Type-B female connector is multiplexed to allow it to be used not only for USB communication but also for acquiring the ignition and VSS signals from a vehicle. The multiplexing is implement on both the Data+ and Data- pins of the USB connector, and is achieved though the use of 4 digitally controlled analog switches built into one package in the standard CMOS CD4016. When connected to a vehicle the voltage level on the power supply pin of the USB connector is a nominal 9 volts. When connected to a PC the voltage level on the power supply pin of the USB connector is a nominal 5 volts. With the use of voltage divider circuitry and a couple of logic inverters, the CD4016 switches can be toggled by the voltage level of the power supply. At 5 volts – indicating a PC connection - two of the switches turn on connecting the Data+ and Data- pins of the USB connector to the appropriate pins on the Delcom USB chip. At 9 volts – indicating a vehicle connection – the other two switches turn on connecting the Data+ and Data- pins of the USB connector to the appropriate pins of the ATMEGA16 to acquire the VSS and ignition signals. It is important to note that each of the two sets of switches is never connected at the same time. Figure 4 illustrates the power supply controlled multiplexed connector.

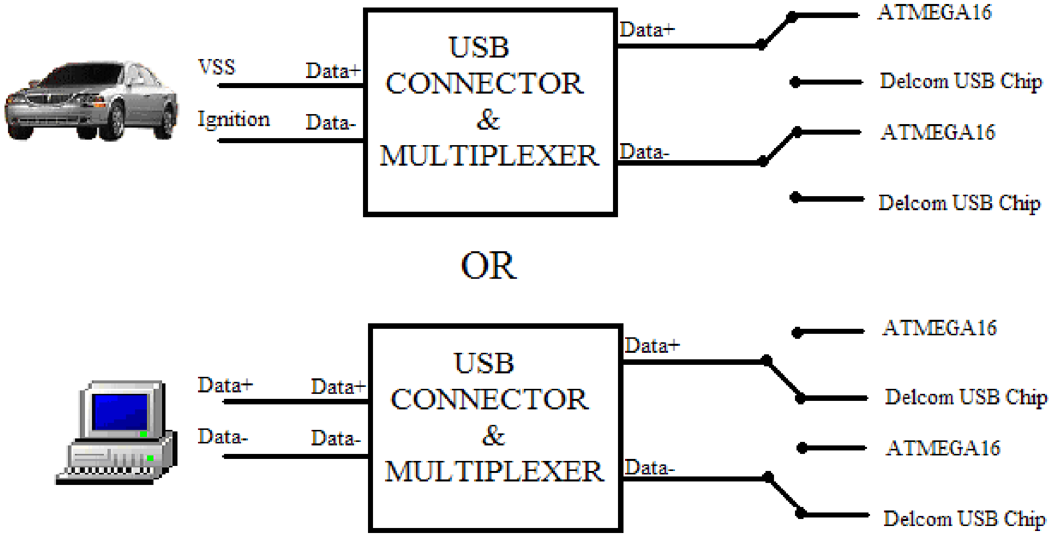


**Figure 4: USB connector multiplexer**

### 2.2.8 Low-Dropout Voltage Regulator (LDO)

The LDO is used to ensure that the power supplied to the MRD through the USB connector is at a level which is acceptable to all the IC components of the device. Since the input voltage level can be either 5 or 9 volts depending on whether the MRD is connected to a PC or a vehicle, the LDO must handle an input voltage up to a least 10 V. In addition, some of the IC's in the MRD require a voltage of at least 4.5V and as such the LDO must have an output voltage of at least 4.5V regardless of the input voltage. To satisfy this requirement we have selected the Texas Instruments TPS7348Q LDO which can output a voltage of 4.85 volts at input voltages as low as 5 volts and as high as 10V. For those unfamiliar with the term low-dropout, it refers to the ability of a voltage regulator to output a voltage at a small negative differential of the input voltage to the regulator. For instance in this case the TPS7348Q can output 4.85 volts with 5 volts input, a differential of only .15 volts. Many voltage regulators are only capable of differentials of 1 volt or more as the minimum. While the ATMEGA16 microcontroller may be the "brains" of the MRD, the LDO is most certainly the "heart" which keeps every other IC running.

# 3 Device Firmware Design

## 3.1 Overview

Figure 5 shows the block diagram of the different modules employed by the microcontroller firmware for the MRD. The inputs are the car signals and serial data from the PC.

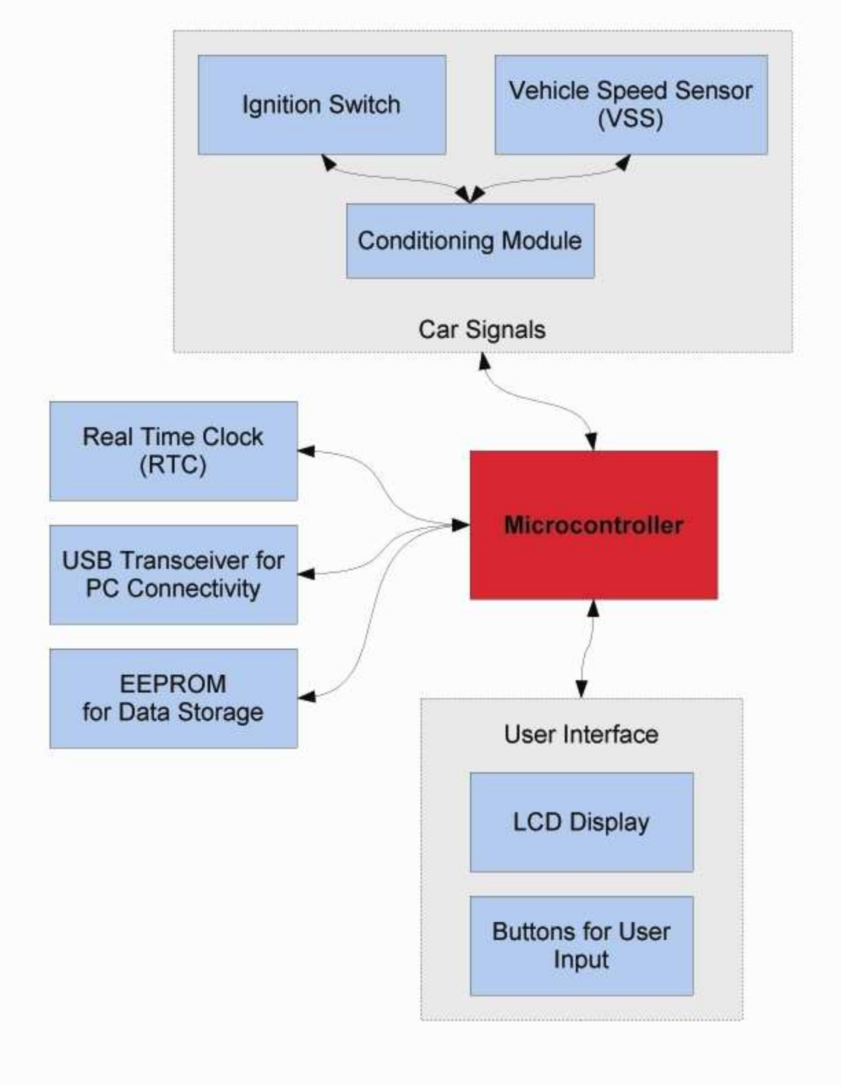


**Figure 5: Controller System Block Diagram**

The car signal inputs include the vehicle speed sensor (VSS) signal as well as the ignition signal. Both of these signals are conditioned in hardware before entering the microcontroller. The VSS signal provides pulses indicating the distance the car travels. This number of pulses is then converted into a meaningful number by dividing by a pulse/km or pulse/mile rate, which is car-specific. The ignition signal on the other hand, indicates when the car is turned on or off; this signal can be used as an external interrupt to indicate the start and end of trips.

Serial data transfer to and from the PC is achieved through a USB connection to the MRD. Data transfers include new trip information transferred from the MRD data storage, the EEPROM, under the supervision of the microcontroller. Configuration data from the PC software also can be transferred to the EEPROM in the same manner.

In addition, the MRD contains a user interface (MRDUI) which employs four push buttons and an LCD display. The user can input user, trip, destination, and car information with the buttons and the MRD can provide system status, trip options and trip indicators with the LCD display.

As for data storage, the MRD contains a 512Kbit EEPROM to be used to store both the configuration and trip words. This data can then be accessed and modified at anytime.

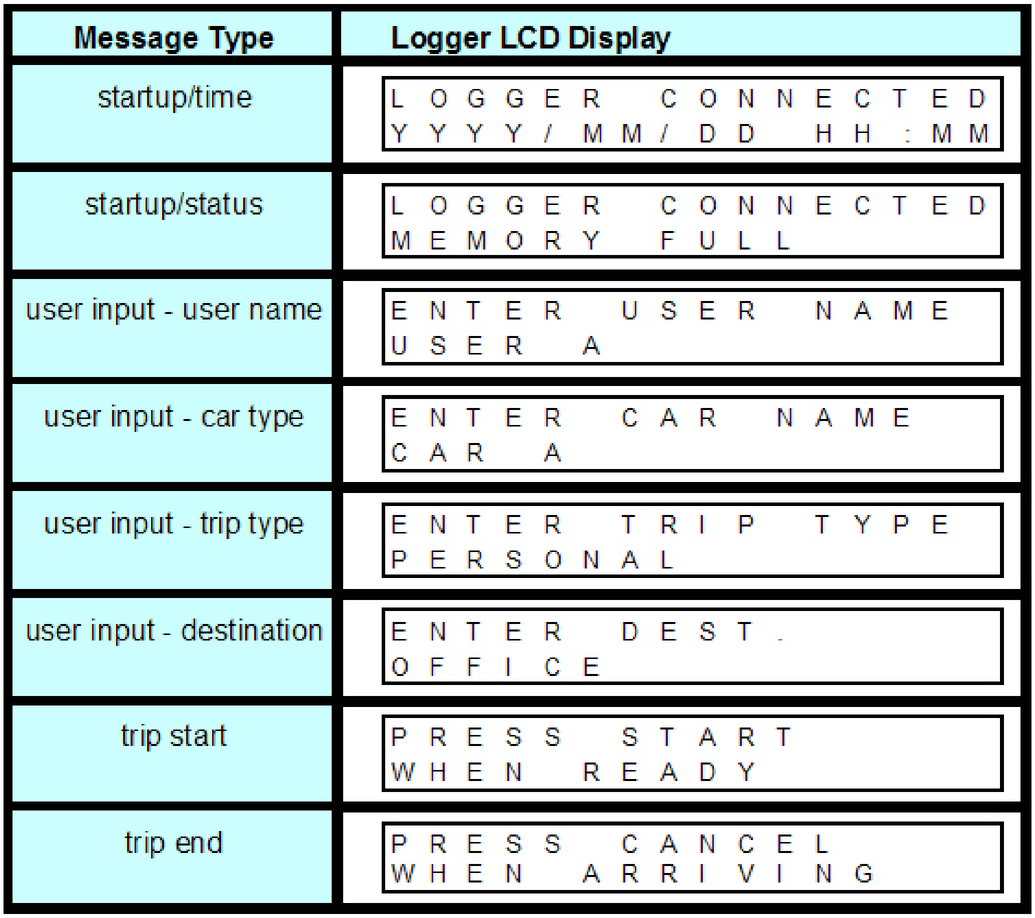
Finally, to provide current time/date to be used with the trip data, the MRD uses a Real Time Clock to provide accurate time to the minute and date in a specific format. The following sections discussed each module in details.

## 3.2 Logger User Interface (MRDUI)

### 3.2.1 LCD Display

The MRDUI uses the MRD’s 2 line by 16 character LCD display to display status and trip messages as in Figure 6.

For trip parameter setting, the four push buttons can be used to browse, select, and cancel the available options.



| Message Type | Logger LCD Display |
|---|---|
| startup/time | LOGGER CONNECTED<br>YYYY/MM/DD HH:MM |
| startup/status | LOGGER CONNECTED<br>MEMORY FULL |
| user input - user name | ENTER USER NAME<br>USER A |
| user input - car type | ENTER CAR NAME<br>CAR A |
| user input - trip type | ENTER TRIP TYPE<br>PERSONAL |
| user input - destination | ENTER DEST.<br>OFFICE |
| trip start | PRESS START<br>WHEN READY |
| trip end | PRESS CANCEL<br>WHEN ARRIVING |

**Figure 6: Logger LCD Messages**

### 3.2.2 Push Buttons

The push buttons are used in the MRDUI to get user input, set trip parameters, and indicate the start and end of each trip. Four push buttons are employed to provide a flexible interface for menu and item selection. The (<<) and (>>) buttons are used to browse different items in menus. The (S) or select button is used to select items and start a trip. The (C) or cancel button is used to skip menu and end trip. The push buttons and LCD interface is shown in Figure 7.

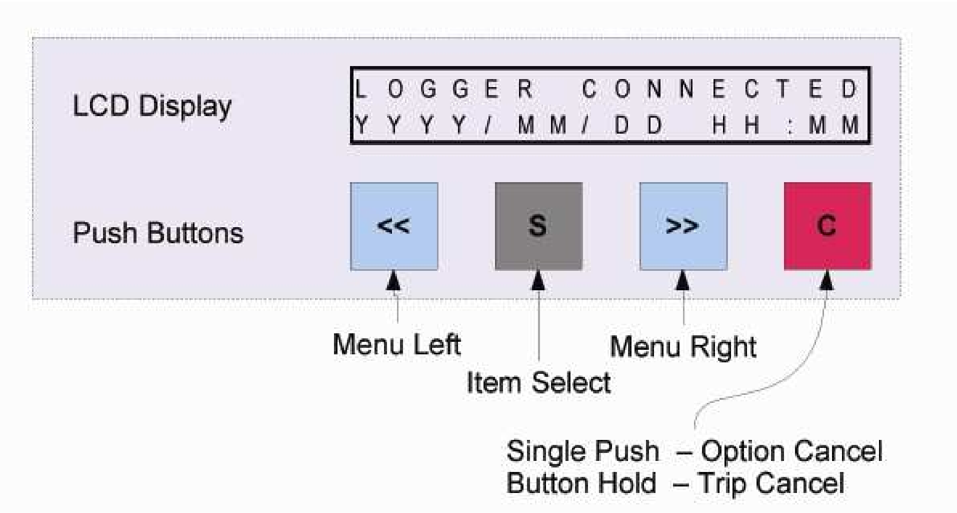

LCD Display
L O G G E R C O N N E C T E D
Y Y Y Y / M M / D D H H : M M
Push Buttons
<<
S
>>
C
Menu Left
Item Select
Menu Right
Single Push – Option Cancel
Button Hold – Trip Cancel

**Figure 7: Push Buttons**

## 3.3 Data Flow and Storage

With the aid of the MRDUI, the user can select trip parameters before the start of the trip. Those parameters, along with the mileage, start and end trip time/date are encapsulated into a *trip word* and stored in memory. The trip word contains 7 fields as listed in Table 1. The total size of the trip word is 10 bytes. A trip word is used for each new trip.

Table 1 also shows the possible values for each trip parameters, format and bit size. For example, trip date is in YY:MM:DD format which required 7+4+5=16 bits for trip start date and another 16 bits for trip end date. To achieve maximum storage utilization, we determined the format and resolution for each field. For instance, the addition of a second (sec) field to the trip start/end time would require an additional 12 bits for each trip word[1].

[1] More details on data storage utilization are included in APPENDIX A.1. The appendix lists four different possible formats for a trip word and their expected utilization. Type II word was selected since it provided maximum utilization with acceptable accuracy margin.

| Field | Format | | Bits |
|---|---|---|---|
| **Car Type** | 4 car names | 0->3 | 2 |
| **User Name** | 4 user names | 0->3 | 2 |
| **Trip Type** | Business/Personal | 0/1 | 1 |
| **trip start/end time** | | 2*11 bits = | 22 |
| | 5 bits for hr | 0->23 | |
| | 6 bits for min | 0->59 | |
| **trip start/end date** | | 2*16 bits = | 32 |
| | 7 bits for year | 2000->2099 | |
| | 4 bits for month | 1->12 | |
| | 5 bits for day | 1->31 | |
| **Destination** | | 0->31 | 5 |
| **Mileage** | | 0->2047 Km | 11 |
| **Total bits** | | | 75 |
| **Total bytes** | | | 10 |

**Table 1: Trip Word Format**

Another word of information that needs to be stored is the configuration word. As mentioned above, and listed in Table 1, there will be several pre-defined values for the display menu field such as: car type, user name, and destinations; for more flexible and user-friendly interface, the MRDUI will not simply display User A…D, or Car Type A…B, or Destination 1..31 but rather, it will display distinctive and meaningful names for each one of those parameters as in Table 2 for example.

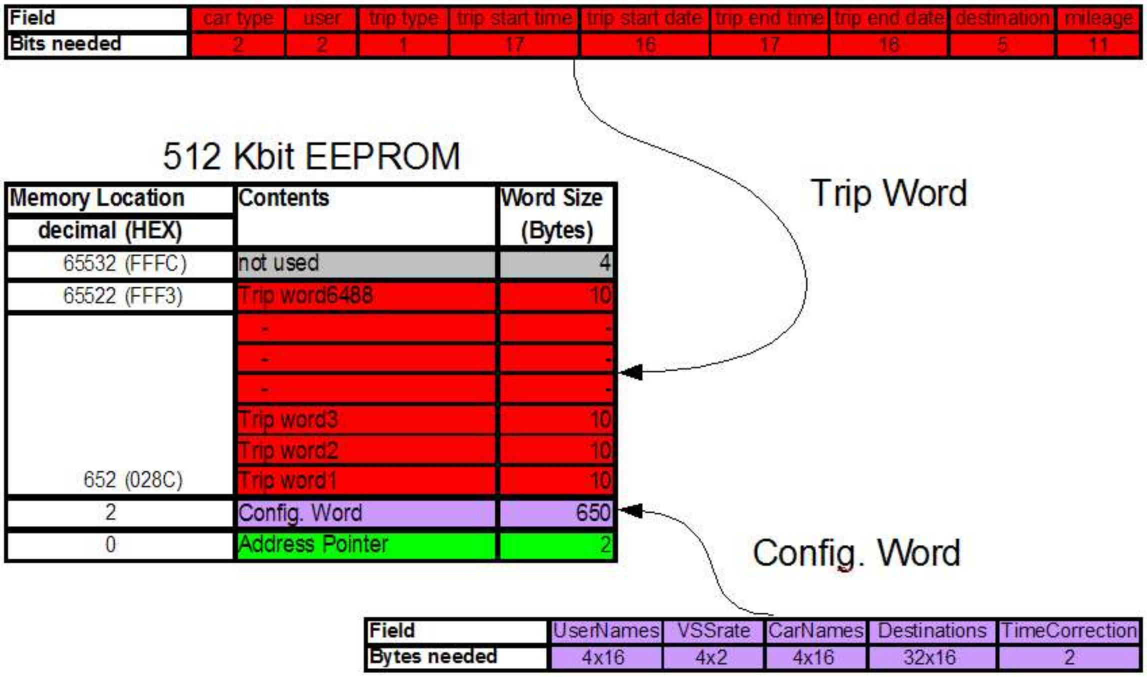
| User Names | Car Type | Destination |
|---|---|---|
| Joe L. M.<br>Mike S.C<br>Selena L.<br>Unknown | Nissan<br>BMW<br>Acura<br>Unknown | Office 1<br>Office 3<br>High School<br>Shopping<br>Business 5<br>Agency…etc. |

**Table 2: Example String Initializations**

Those possible string values will not be stored in the microcontroller's flash memory due to size limitations; they will rather be stored in the EEPROM data storage and will be read into the MRD controller program only once at the beginning of the program initialization. Those strings will be inserted by the user through the PC GUI software and transferred to the EEPROM whenever modifications required. Those initialization strings and possible values is what constitute the configuration word.

Another field in the configuration word is the corresponding VSS rate for each car; this number indicates the number of pulses per (Km) traveled specific to each car type. A detailed diagram of the configuration word can be found in Appendix A.1.

Figure 8 shows the packing of the two words into the EEPROM data storage; only one configuration word is required since it is only needed to initialize value required within the main program.  This word is allocated 650 byte in the memory location indicated below.



| Field | car type | user | trip type | trip start time | trip start date | trip end time | trip end date | destination | mileage |
|---|---|---|---|---|---|---|---|---|---|
| Bits needed | 2 | 2 | 1 | 17 | 16 | 17 | 18 | 5 | 11 |

| Memory Location decimal (HEX) | Contents | Word Size (Bytes) |
|---|---|---|
| 65532 (FFFC) | not used | 4 |
| 65522 (FFF3) | Trip word6488 | 10 |
| | - | - |
| | - | - |
| | - | - |
| | Trip word3 | 10 |
| | Trip word2 | 10 |
| 652 (028C) | Trip word1 | 10 |
| 2 | Config. Word | 650 |
| 0 | Address Pointer | 2 |

| Field | UserNames | VSSrate | CarNames | Destinations | TimeCorrection |
|---|---|---|---|---|---|
| Bytes needed | 4x16 | 4x2 | 4x16 | 32x16 | 2 |

**Figure 8: Memory Allocation**

The remaining memory block is used to store trip words. This memory block leaves enough space to store 6488 trip words of 10 bytes each. If the user consumes 10 trips per day, this storage can hold up to 21 months worth of trips before downloading to the PC.

The address of the next free memory location can be retrieved from a 2-byte address field stored in memory location 0 (green in Figure 8).  This next free address can be used to write new trip words to.

## 3.4 In-System Real Time Clock (RTC)

The In-System Real Time Clock (RTC) an IC that is controlled by microcontroller via $I^2C$.  The real time clock has a small amount of memory (organized using a standard memory address space) where it stores the current date and time, assuming it is powered by the battery.

The current date and time stored by the clock can be reset (to any other date and time) by the microcontroller simply by writing the new date and time values to the correct memory location (defined by the RTC manufacturer). Whenever required, the date and time can be read from the RTC by reading those same memory locations

The $I^2C$ is a serial bus interface that many different devices can use (to a maximum of 127). Each device that uses this interface has a unique device ID (this a 7-bit number). Before data is read or transmitted a device must secure the transmission line by asserting a “start” command, followed by a device ID. If this is successfully the identified device will reply, where data transmission can begin. The data transmission is specified by the $I^2C$ “stop” command. For the RTC, the first part of data that is transmitted is always the address of the memory location that is of concern.

## 3.5 Firmware Control Algorithm

Upon detection of any connection via the USB connector to the device, the MRD controller will determine the type of connection: PC software (GUI) or Vehicle Cradle Device (VCD) connection. It can then run the appropriate routine accordingly to perform the required tasks.

### 3.5.1 MRD/VCD Routine

Once it establishes a connection to the VCD, the MRD will have to perform certain initialization and user trip selection steps before it can record any VSS signals. The steps are shown in the flow chart in Figure 9.

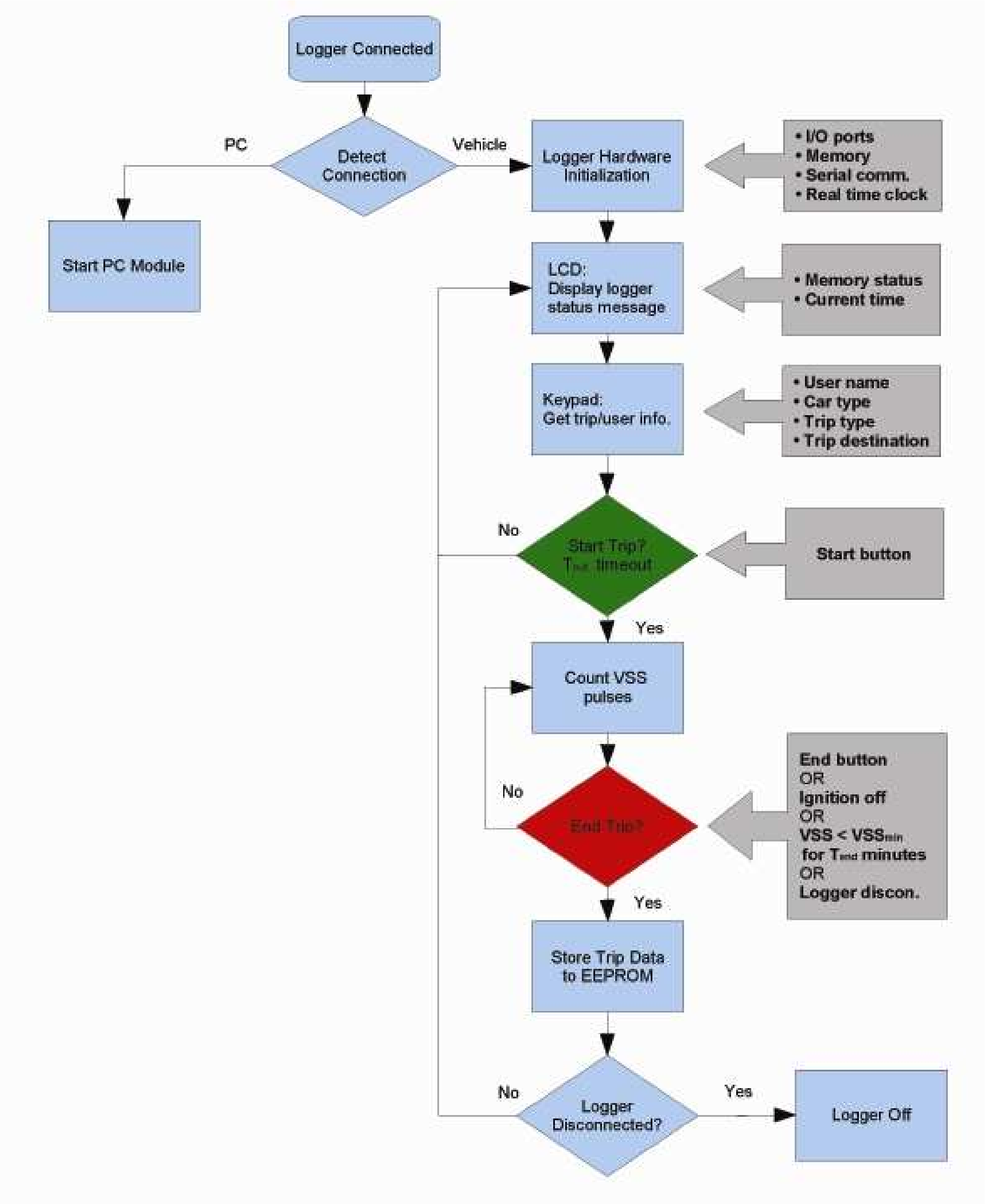


**Figure 9: Firmware Control Flowchart**

### 3.5.1.1 Controller Initialization

The MRD initializes the following modules upon startup and detection of a VCD connection:

1. I/O ports directionality, i.e all of PORTA pins set to output to be used to write commands to the LCD.
2. Initialize the LCD display mode and settings.
3. Initialize interrupt registers to enable external interrupts to be used to detect push buttons, car ignition, VSS pulses, and timer interrupts.
4. Initialize the real time clock to provide accurate time and date in the previously mentioned formats.
5. Initialize data buffers and character strings to be used in LCD menus.

### 3.5.1.2 System Status and Trip Parameter Selection

After port and modular initialization, the controller will then display a system status message which indicates the capacity of the memory as well as the current time and date. Figure 6 contains the pre-defined LCD messages to be used in the MRDUI.

The MRDUI then allows the user to select trip options and get ready to start the trip by simply pressing the Start button (S).

### 3.5.1.3 VSS/Mileage Counting and Trip End

After pushing the Start button, the controller will poll the VSS signal and count the number of pulses, which is in the form of an external interrupt. This variable is then converted into kilometers by dividing it by the VSSrate specific to the car being used. The conversion takes place at the end of the trip.

Trip end is indicated by two different flags:

1. When the Cancel button (C) is held for 2 seconds
2. When the car ignition is turned off, this can be indicated by the ignition interrupt signal. Note that when the ignition is turned off for gas fill up or resting breaks, the ongoing trip will be automatically sent to data storage and a new trip will be started upon ignition on.

Upon the trip end, trip data will be encapsulated and sent to the EEPROM to be stored for record. Trip data will be permanently erased when the logger is physically disconnected from the car during an ongoing trip.

### 3.5.2 MRD/PC Windows GUI Routine

Once the MRD detects a connection with the PC, it will automatically wait for the PC for the next instruction to execute. Data and commands from the PC will be sent to the MRD via the USB interface.

The PC GUI user can execute one or two of the following MRD tasks:

1. Download all trip data stored in EEPROM. The MRD controller will be the channel through which this data is transferred.
2. Set and send new configuration word to be sent to EEPROM which can later be used by the MRD to initialize trip options.

# 4 Windows Software Design

The Plug n'Go software application's main purpose is to act as an interface between the device, the PC, and the user. The program will have a graphical user interface (GUI) for viewing and manipulating data, and will allow the user to read and save data from the device and the computer's hard drive.

These design constraints can mandate roughly four main functional areas of the program:

- Data Handling: Data structures for storing, moving, and parsing data.
- Device Interfacing: Software module concerned with communicating with the device.
- File System Interfacing: Software module that handles reading and writing to the computer's file system
- Core Application: Software that coordinates the actions and interprets the results of the above three areas in conjunction with input from the user.

These functional areas are shown in the diagram below, along with arrows indicating how they communicate with each other.

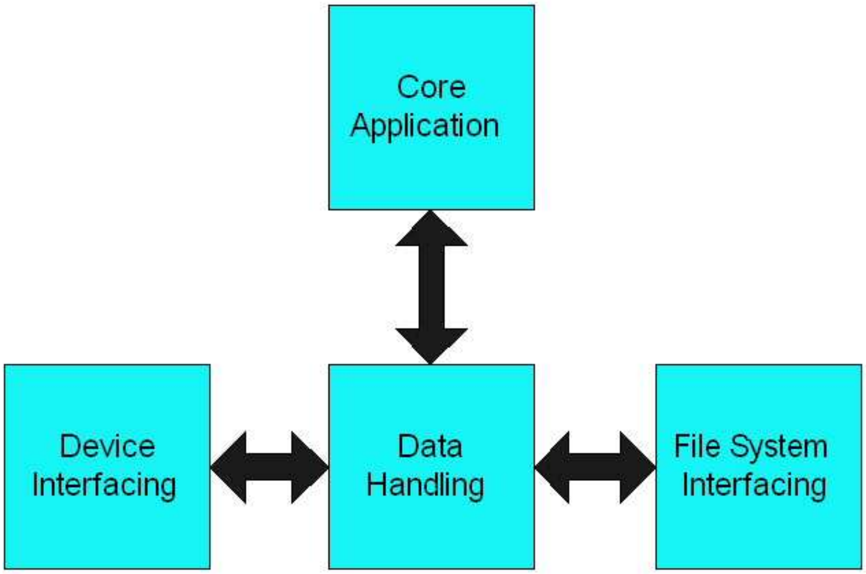


**Figure 10: Functional Areas of Software Application**

In the sections below we will elaborate on the system's use cases, its class design, and user interface design.

## 4.1 Use Case Overview

The use of the program can be viewed by looking at use cases, which are basically user scenarios. They illustrate what kinds of operations the users of the system will need to complete. In the "Use Case Diagram", below, these scenarios are illustrated in a diagrammed form. The stick men represent "actors" using the system. Each represents a "role", not necessarily a single user (i.e. one user could have serve in multiple roles). Likewise, an actor need not be a person, but can be an object, such as the MRD (Mileage Recorder Device). The ovals represent the different use case scenarios, which are each described below the diagram. This diagram is very high-level and is not a precise model; the purpose of the diagram is to give insight into the different functions of the software and the different user roles the system must accommodate.

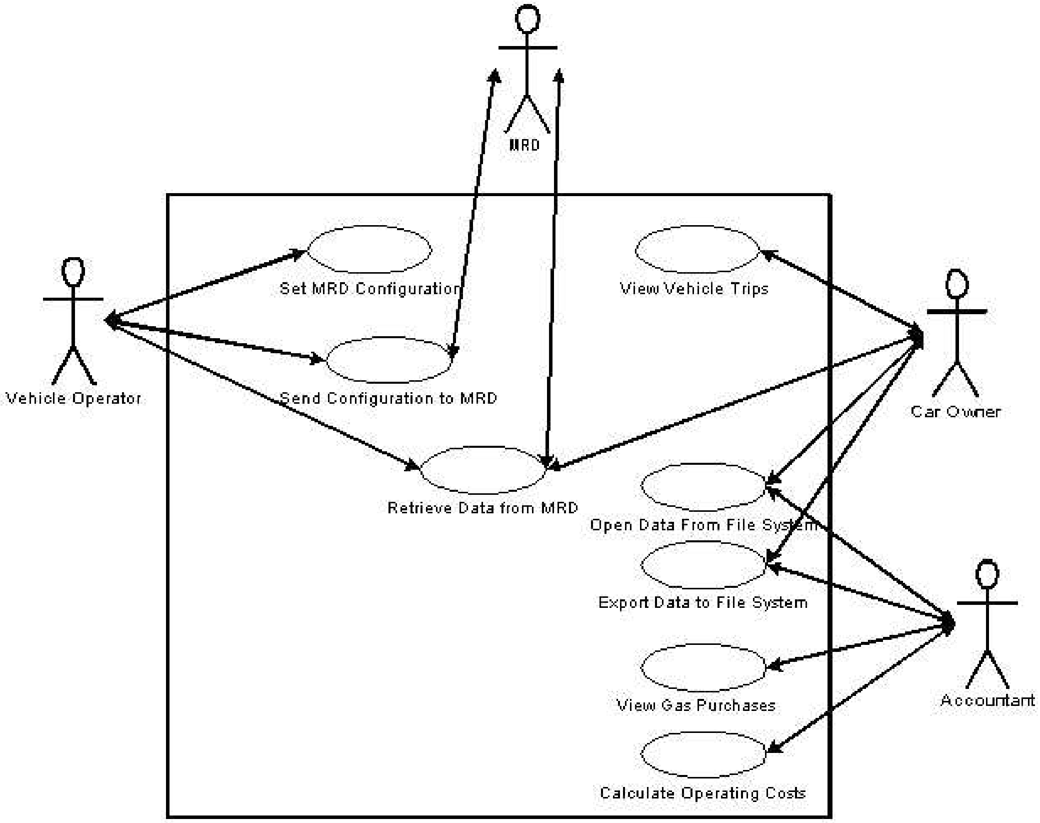


**Figure 11: Use Case Diagram for Software Application**

There are four actors in the use case diagram. The one at the top of the diagram, the MRD, represents the mileage recorder device itself. It is concerned only with the transmission and receiving of data between itself and the software application.

The other three actors, the Vehicle Operator, Car Owner, and Accountant, will often be the same person, although each actor represents a different role. The vehicle operator role need only be concerned that the MRD is configured correctly. This means the vehicle operator must make sure the time is correct, the MRD is set to the correct car model, and that the destination list reflects the destinations that the operator travels to. The vehicle operator may use the application to retrieve data from the device, though this role will most likely be the concern of the person responsible for the car, usually the owner (once again, the vehicle operator and owner could be the same person).

The car owner role's main concern is getting information about car usage. This includes retrieving the data form the device, viewing it, as well as saving and opening the data to and from the file system.

The accountant role is mainly concerned with the financial aspects of the car's operation. Mainly, this is the operating cost information (functionality in the program that allows the user to view a summary of all car costs), as well as information on gas purchases. To reiterate, the person who fills this role could fill any of the other roles, but mainly these are “accounting concerns”.

## 4.2 Class Descriptions

This section will go through the breakdown of the program into “classes”, which are different pieces of code that work together to provide the complete program. The figure below illustrates this with a static class diagram in Unified Modeling Language (UML) notation. This diagram shows a high-level view of the different classes in the program in order to demonstrate what classes the software implementation of the program consists of, as well as the relationship of the classes with each other.

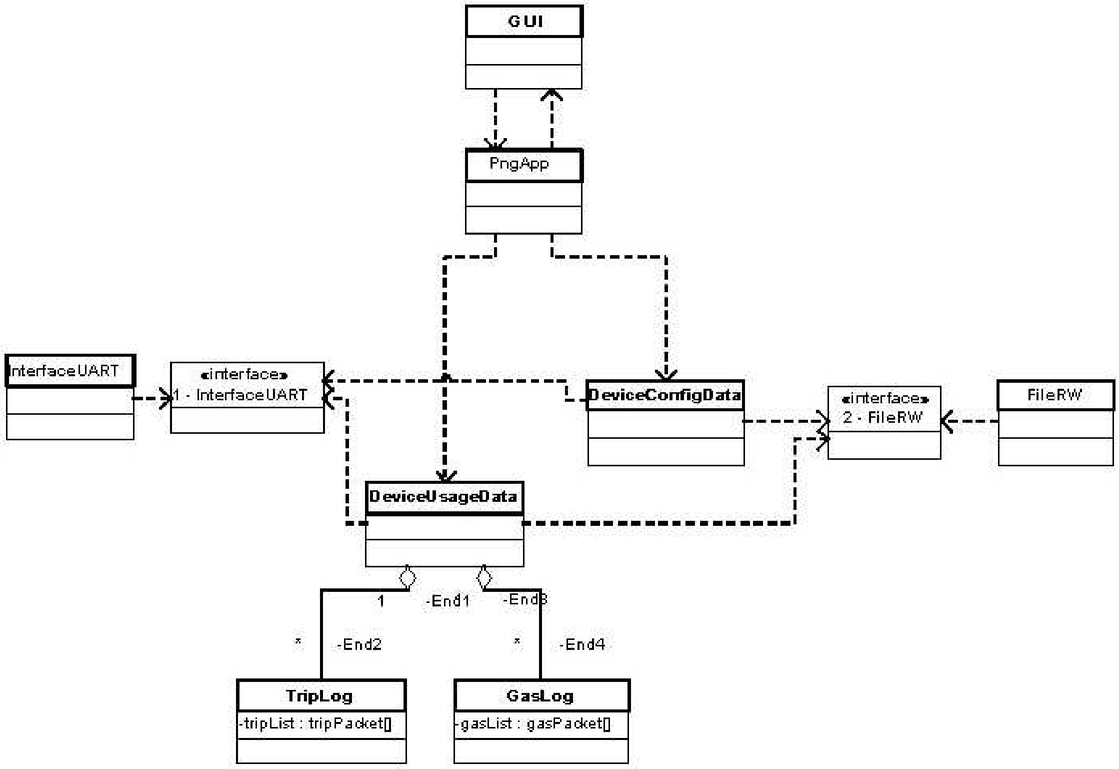


**Figure 12: UML Static Class Diagram**

### 4.2.1 DeviceUsageData

This class defines the data structure that holds vehicle usage data, collected by the device, and provides functions for working with this data (such as parsing, sending, and retrieving data through its dependency relationships).

**Aggregate Relationships** (classes that DeviceUsageData is partly composed of):
The data structure contains two other classes TripLog and GasLog. In other words, DeviceUsageData is partly composed of these classes. Each DeviceUsageData instance (or "working copy") will contain a TripLog and GasLog instance, which will hold the trip data and the gas data, respectively. These classes will be explained in more detail below.

**Dependency Relationships** (classes that this class requires in order to operate properly):
The class also is dependent on two static classes called InterfaceUART and FileRW. Static classes are classes that cannot be instanced; these classes have only "1 working copy" in the computer. As part of the implementation of the DeviveUsageData functions, calls will be made to InterfaceUART in order to retrieve data from the device. When data needs be stored to the disk, DeviceUsageData will use FileRW functions in order to store and retrieve data under the computer's file system. The interaction between DeviceUsageData and these other classes is specified by two interfaces, which are simply like a contract between the modules. This ensures each module is providing the correct functions for the module it is interacting with. (The implementers of these modules must make sure their modules complies with the interface, or in other words, make sure they hold up "their end of the contract").

### 4.2.2 TripLog

Provides an array of tripPacket structures and access functions to access them. Each tripPacket contains the data for a single trip. Together, this trip data forms a trip log.

**Inter-class relationships:** This class contains a simple C/C++ data structure, and is dependent only on this structure (described in more detail in next section).

### 4.2.3 GasLog

Provides an array of gasPacket structures and access functions to access them. Each gasPacket contains the data for a single gas purchase. Together, this gas data forms a log of gas purchases.

**Inter-class relationships:** This data structure is a simple C/C++ structure, and is not dependent on any other software element.

### 4.2.4 DeviceConfigData

This class defines the data structure that holds device configuration data, and provides functions for working with this data. The configuration data includes the VSS signal pulse rate, time updates, and destination names.

**Dependency relationships** (classes that this class requires in order to operate properly): Like DeviceUsageData, this class also is dependent on the two static classes InterfaceUART and FileRW. DeviceConfigData will be required to send to the device the configuration data, mentioned above. It will do this by making function calls to InterfaceUART. Configuration data also needs to be stored to the disk (so users don't need to continually re-enter configuration data). To do this, DeviceConfigData will use FileRW functions in order to store data under the computer's file system, as well as retrieve it. The interaction between DeviceConfigData and these other classes is specified by two interfaces, which act like a contract between the modules. This ensures each module is providing the correct functions for the module it is interacting with.

### 4.2.5 InterfaceUART

This class implements the interface to the device. It provides functions that will be used to read and write data to and from the device, once it has been connected to the computer through the USB port.

**Dependency relationships** (classes that this class requires in order to operate properly): The class is dependent on the interface requirements between itself and the DeviceUsageData and DeviceConfigData classes. This interface describes the functions that the InterfaceUART provides to the data classes in order to facilitate the transfer of information between the computer and the device. This is a static class, in that it is never instanced (has only 1 "working copy" in the computer at time). It serves as a "toolbox" of communication functions.

### 4.2.6 FileRW

This class is used to implement the reading and writing of data to the file system. It is used by other modules to record both configuration data as well as vehicle usage data to the hard disk, and is then later used to retrieve that data.

**Dependency relationships** (classes that this class requires in order to operate properly): The class is dependent on the interface requirements between itself and the DeviceUsageData and DeviceConfigData classes. This interface is what defines the functions that FileRW must provide to these other classes. Like InterfaceUART, this class is a static class that is not instanced (has only 1 "working copy" in the computer at time). It serves as a "toolbox" of file reading and writing functions.

### 4.2.7 PngApp

This class serves as the core application. It coordinates all application operations and is driver by user (through the GUI). It is the only part of the program the GUI interacts with, so all input from the user goes through PngApp. Additionally, the GUI, using information that has been provided by PngApp, displays the application's output on the screen.

**Dependency Relationships** (classes that this class requires in order to operate properly): PngApp is dependent on DeviceUsageData and DeviceConfigData which provide PngApp with the functions needed to carry out program functions (DeviceUsageDate and DeviceConfigData use other classes, in turn, as described above). PngApp also has a close relationship with the GUI. A major change to the GUI may require changes to PngApp, so the PngApp is dependent on the GUI.

### 4.2.8 GUI

To provide a graphical interface with which the user can use to interact with the program. While for design purposes the GUI is one unit of the program, due to practicality sake, it will be implemented in numerous different classes in the manner that is common among windows user interfaces.

**Dependecy Relationships** (classes that this class requires in order to operate properly): The GUI is ultimately dependent on PngApp, for should PngApp be rewritten, it would mean that the GUI would likely have to be modified to conform to the new version of the core program. In this way the GUI "needs" PngApp.

## 4.3 Data Structure Detail

As mentioned above, a tripPacket data structure is used to hold the data for a single trip. Figure 13 details the tripPacket data structure (note C/C++ primitives are used).

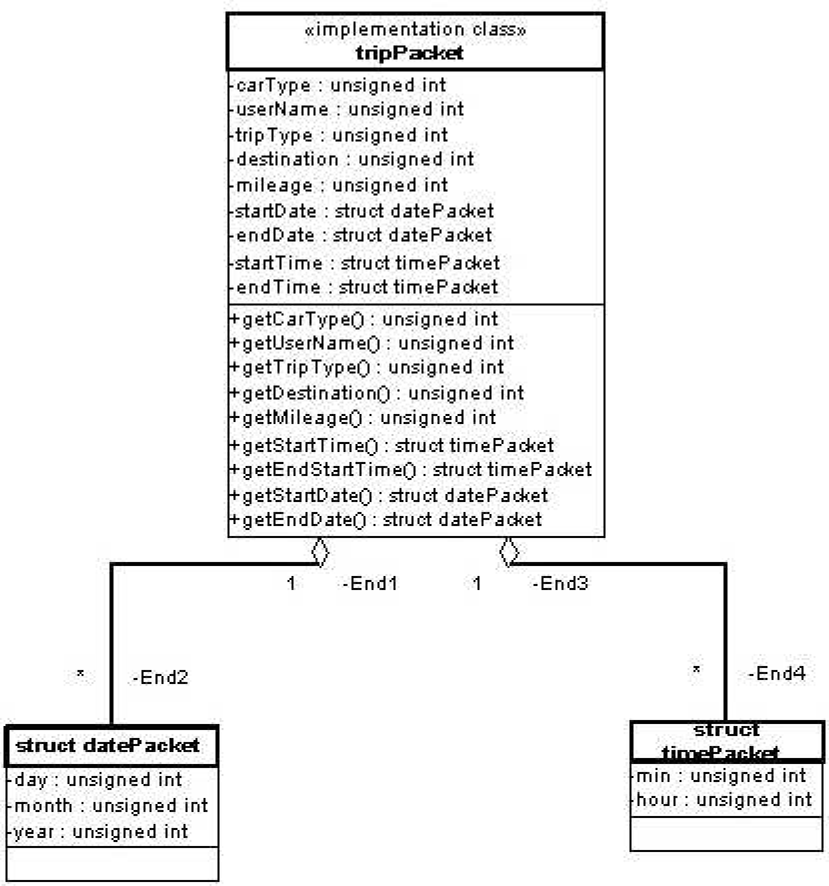

«implementation class»
tripPacket
-carType : unsigned int
-userName : unsigned int
-tripType : unsigned int
-destination : unsigned int
-mileage : unsigned int
-startDate : struct datePacket
-endDate : struct datePacket
-startTime : struct timePacket
-endTime : struct timePacket
+getCarType() : unsigned int
+getUserName() : unsigned int
+getTripType() : unsigned int
+getDestination() : unsigned int
+getMileage() : unsigned int
+getStartTime() : struct timePacket
+getEndStartTime() : struct timePacket
+getStartDate() : struct datePacket
+getEndDate() : struct datePacket
1
-End1
1
-End3
*
-End2
*
-End4
struct datePacket
-day : unsigned int
-month : unsigned int
-year : unsigned int
struct timePacket
-min : unsigned int
-hour : unsigned int

**Figure 13: tripPacket Data Structure**

The trip packet holds the following data about a particular trip:

- carType – identifies which vehicle the trip was taken in
- userName – identifies who was driving on this trip
- destination – specifies which destination message represents the destination of this trip
- mileage – specifies the total distance traveled on this trip

It also holds four more pieces of data about the trip:

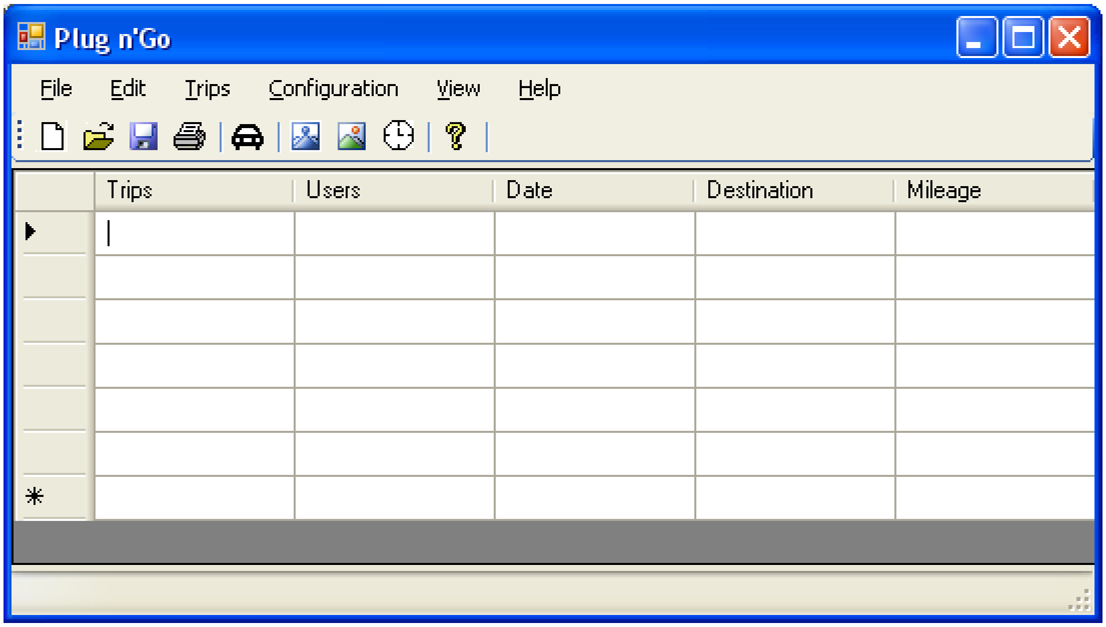
- startDate – struct contains the date the trip was started
- endDate – struct contains the date the trip ended
- startTime – struct contains the time the trip was started
- endTime – struct contatines the time the trip was ended

These last four data attributes are themselves structures, as shown in the diagram above, which in turn hold data and time data. A structure is a data structure made of simpler data primitives (containers).

The data structure also provides access functions to retrieve data values.

## 4.4 Windows Application: Graphical User Interface

The Windows software GUI has a general look and feel of a Windows application in order for it to be user friendly and intuitive to the average user. It has a drop down menu where users can perform functions such as open files, save files, and access help. A tool bar is present for quick access of commonly used functions. The center of GUI is a spreadsheet showing all trip data downloaded from the MRD EEPROM. All data is to be displayed using metric units. Also a status bar is placed at the bottom to provide feedback to users. Figure 14 shows the design of the Windows GUI as described above.



| | Trips | Users | Date | Destination | Mileage |
|---|---|---|---|---|---|
| ▶ | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| * | | | | | |

**Figure 14: Windows Software GUI**

### 4.4.1 Drop Down Menu

The menu tree is organized as shown in Figure 15.

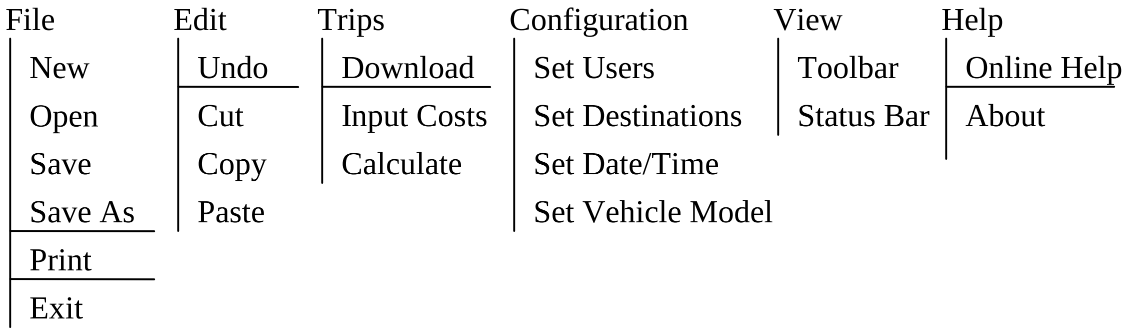


**Figure 15: Windows Software Menu Tree**

Under the "File" menu, users can perform typical functions that are present in every Windows application. Users can create a new file, open an existing file, save the current file, print the current file and exit the program. The application saves the spreadsheet data into the Comma Separated Value (CSV) file format. The CSV file format has been chosen because it can be opened with Microsoft Excel.

Under the "Edit" menu, users can perform various functions to the spreadsheet data. Users can undo changes, cut out incorrect data, copy data, and paste data. The spreadsheet data can be changed by user because the fields such as user and destination are limited. Users can correct data entry if an error occurred.

Under the "Trips" menu, users can download data off the MRD. Users can also input costs of their trips such as gas purchases, vehicle maintenance costs, and insurance. Users can also use calculate function to calculate their income tax deduction. Also users can calculate vehicle usage and fuel purchase statistics. (i.e. mileage totals, business to pleasure trip ratios, total fuel purchases, etc.) All this will be done using pop-up windows which allow users to enter the appropriate data.

Under the "Configuration" menu, users can configure MRD lists. Users will be able to set the user list, destination list, date/time, and speed sensor. All this will be done using pop-up windows with forms that users can fill out.

Under the "View" menu, users can toggle on and off the toolbar and status bar.

Under the “Help” menu, users can view the online help website which will contain user manual and tutorial to help users. Users will also be able to view any information about the software such as version number.

### 4.4.2 Toolbar

The toolbar icons are shown in Figure 16 above and reflect the drop down menu. It is to be used by users for quick access to different functions they might use often. The icons are grouped into four groups. The first group of icons corresponds to functions under “File” menu. The icons from left to right are new file, open file, save file, and print file. The second group of icons corresponds to the icons under “Trips” menu. Users can bring up the pop-up window for different trip functionalities by clicking on the icon. The third group of icon corresponds “Configuration” menu. Users can set user list, destination list, and date/time by clicking on the icons. The last icon is the help icon. The user can bring up the online help website by clicking on it.

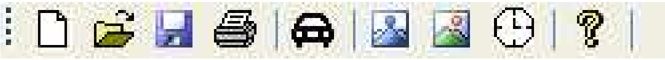
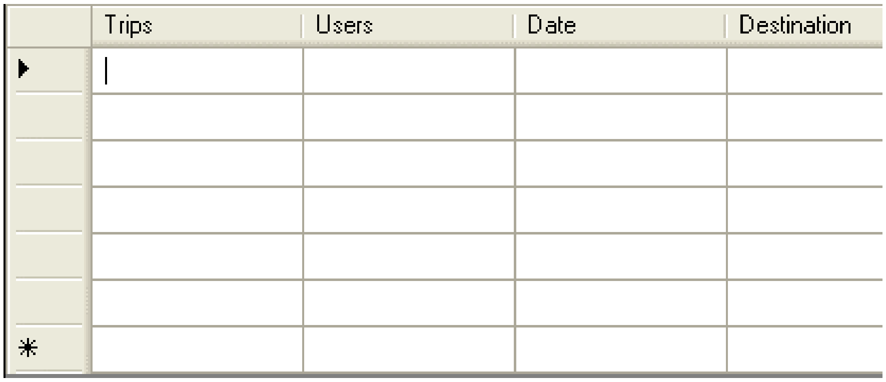


**Figure 16: Windows Software Toolbar**

### 4.4.3 Spreadsheet

The spreadsheet is to be very flexible, much like Microsoft Excel. Users can modify and organize the data to their liking. Users can click on the column titles to reorganize the data by that attribute. Users can also reorder columns by dragging the columns to desired place. Users can also hide a column by dragging the column border to hide it. Figure 17 illustrates the spreadsheet.



| | Trips | Users | Date | Destination |
|---|---|---|---|---|
| ▶ | \| | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| * | | | | |

**Figure 17: Windows Software Spreadsheet**

### 4.4.4 Status Bar

The status bar is placed at the very bottom of the application window. It is used to provide basic feedback to users.

### 4.4.5 Future Functionalities

Additional functionalities will be added on later. These functionalities include viewing the spreadsheet data in graphical formats such as pie and bar graphs. Users will also be able to convert data into imperial measurements from metric.

# 5 Test Plan

## 5.1 Hardware Testing

The hardware testing process will for the most part occur on an ongoing basis. As the hardware development will precede the firmware and software development processes, hardware issues may only present themselves upon execution of software and firmware applications. Should any hardware issues arise during the development of firmware and software, testing will be conducted to trace the source of the issue and corrective engineering will be executed.

### 5.1.1 Modular Design Testing

Individual hardware modules have been bread-boarded and tested to ensure that they operate as they are intended to by design. Many modules have already been redesigned such that they now appear to meet their intended function. Should any current design prove to provide unsatisfactory result, alternative designs will be considered, tested, and possibly implemented.

### 5.1.2 Circuit Board Implementation Testing

When the entire MRD and VCD designs have been implemented onto circuit boards each and every signal will be tested to ensure that the board is built as specified in the schematic diagram. The testing process will include:

1. **Continuity testing –** Each signal will be tested to ensure that connections between each IC are made to the correct pins and no pins have been inadvertently shorted together.
2. **Power testing –** Power will be applied to the MRD and VCD and voltage will be measured at appropriate nodes to ensure they meet the design voltage level. Applying power will help ensure that each IC has been correctly connected, overheated IC’s may indicate improper connections. IC’s which “blow-up” will also likely indicate a wiring issue and these IC’s will require replacement.

3. **Current draw testing –** A amp meter will be connected in series with the power source to ensure that the MRD and VCD draw power as specified in the Functional Specification[2] document.
4. **Impedance Testing –** Resistive elements of the circuitry will be measured with an ohmmeter to ensure that their impedance is on par with the design levels.
5. **Frequency Testing –** Periodic signals (i.e. clock signals) will be measured with an oscilloscope to ensure their frequency meet design intensions.

## 5.2 Device Firmware Testing

### 5.2.1 Connectivity Testing

Connections to the PC and VCD will be tested separately. The MRD will be connected to several cars and test strings will be displayed to the LCD to indicate the connection status. Also, the MRD will be connected to a digital function generator and signals with various amplitudes, frequencies, and duty cycles will be introduces to gauge the MRD's response to possible variations in VSS signals from the car. Noise will also be introduced to the generated signals to test the system's susceptibly to noisy VSS input.

The MRD will also be connected to the PC with a USB cable and the appropriate communication and software modules will be tested. The MRD will display specific LCD strings during initialization and usage of the connection to help debug any issues.

### 5.2.2 Data Handling Testing

Various sets of trip parameters will be input to the MRDUI and results on the PC GUI software will be tested carefully. The MRD data record will be also saved, with the help of the PC software, in the form of ASCII and binary data to make sure that the PC software does not corrupt the data.

### 5.2.3 MRDUI and Interrupt Testing

The LCD will be tested by displaying unique strings to both lines at different speeds to check the LCD response. The LCD backlight will also be toggled for testing.

The test for pushbuttons is primary a test for the interrupt algorithm utilized by the microcontroller. Multiple-button handling, pressing speeds and button sequence tests will be conducted to ensure the interrupt functions correctly. The timer interrupts and RTC will also be carefully tested by displaying time and date strings to LCD throughout the test duration to assure RTC and timer accuracy and proper timer interrupt handling.

## 5.3 Windows Software Testing

### 5.3.1 Module Testing

Windows software testing will be done first using module testing. Each class is considered a module. Each function will be implemented and tested before integrated into complete application. Refer to section 4.2 for information on the different modules/classes that are present in Windows software.

### 5.3.2 Functionality Testing

After module testing, Windows software will be tested for complete application functionality. All modular functions will be thoroughly tested to ensure operation within the complete application. Basically, this is a test ensures that integration of the modules does not adversely affect the functionality of the modules.

### 5.3.3 User Testing

Once the Windows software is functioning as desired, the GUI will undergo user testing. User testing is important because the software is to be user friendly and intuitive. Users will be asked to perform a selected list of functions without any help. The users' actions will be observed and recorded. Users will also be asked questions to provide feedback on the GUI. The GUI will be modified accordingly. This step will be done multiple times until users can use the software with ease the first time without aid.

# 6 References

[1] Canada Revenue Agency, *Motor Vehicle Expenses Claimed by Self-Employed Individuals*, December 16, 1996, http://www.cra-arc.gc.ca/E/pub/tp/it521r/it521r-e.html, Viewed: Jan 16, 2006

[2] OnBoard Travel Technology, *Functional Specification*, February 24, 2006

**Relevant IC Datasheets:**

Atmel, *ATmega16(L) – 8 bit microcontoller with 16KBytes In-System Programmable Flash*, http://www.atmel.com/dyn/resources/prod_documents/doc2466.pdf

Dallas Semiconductor, *DS1307 – 64x8 Real Time Clock*, http://pdfserv.maxim-ic.com/en/ds/DS1307.pdf

Delcom Engineering, *USB I/O Data Sheet*, http://www.delcom-eng.com/downloads/USBIODS.pdf

Microchip, *24AA512 - 512K I2C CMOS Serial EEPROM*, http://rocky.digikey.com/WebLib/Microchip/Web%20Data/24AA512,24LC512,24FC512.pdf

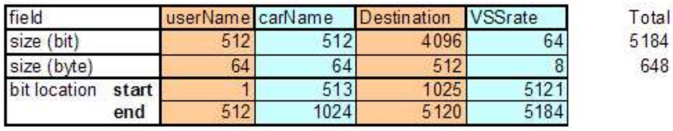
Texas Instruments, *TPS7301Q, TPS7325Q, TPS7330Q, TPS7333Q, TPS7348Q, TPS7350Q - Low-Dropout Voltage Regulators With Integrated Delayed Reset Function*, http://www-s.ti.com/sc/psheets/slvs124f/slvs124f.pdf

Samsung Electronics, *S6A0069 - 40 SEG / 16 COM Driver & Controller For Dot Matrix LCD*, http://www.displaytech-us.com/pdf/application/Character_Module/Samsung/S6A0069_V00_KS0066U.pdf

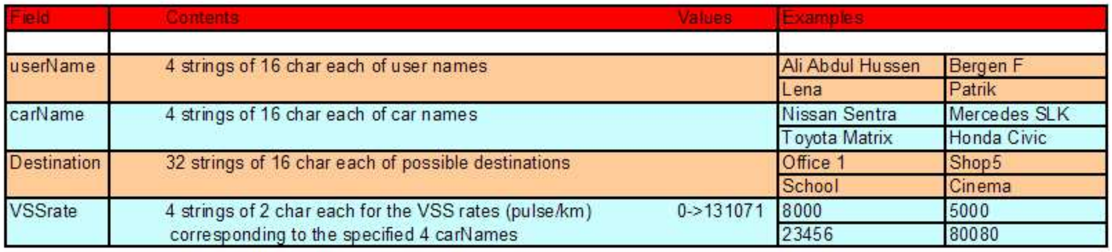
# Appendix A.1 (Different Trip Word Formats)

| Field | Possibilities | Values | Bits Needed | | | | |
|---|---|---|---|---|---|---|---|
| | | | Type I | Type II | Type III | Type IV | Type V |
| Car Type | | | 2 | 2 | 2 | 2 | 2 |
| | Type A | 00 | | | | | |
| | Type B | 01 | | | | | |
| | Type C | 10 | | | | | |
| | Type D | 11 | | | | | |
| User | | | 2 | 2 | 2 | 2 | 2 |
| | User A | 00 | | | | | |
| | User B | 01 | | | | | |
| | User C | 10 | | | | | |
| | User D | 11 | | | | | |
| Trip Type | | | 1 | 1 | 1 | 1 | 1 |
| | Business | 0 | | | | | |
| | Personal | 1 | | | | | |
| trip start/end time | | | 34 | 22 | 14 | 22 | 14 |
| 2x | hh:mm:ss | | 2x(5\|6\|6) | | | | |
| 2x | hh:mm | | | 2x(5\|6) | | 2x(5\|6) | |
| 2x | 1/4 hr | | | | 2x(5\|2) | | 2x(5\|2) |
| trip start/end date | | | 32 | 32 | 18 | 0 | 0 |
| 2x | yy:mm:dd | | 2X(7\|4\|5) | 2X(7\|4\|5) | | no date | no date |
| 2x | mm:dd | | | | 2x(4\|5) | | |
| Destinations | | | 5 | 5 | 5 | 5 | 5 |
| | 32 dest. | | | | | | |
| Mileage | | | 10 | 11 | 11 | 11 | 11 |
| max | 1023 | | 10 | | | | |
| max | 2047 | | | 11 | 11 | 11 | 11 |
| | | | | | | | |
| Total bits | | | 86 | 75 | 53 | 43 | 35 |
| Total Bytes | | | 11 | 10 | 7 | 5 | 4 |
| Message volume with 512Kbit | | | 6096 | 6554 | 9892 | 12193 | 14980 |
| 512Kbit = | 65536 | bytes | | | | | |

# Appendix A.2 (Configuration Word Format)

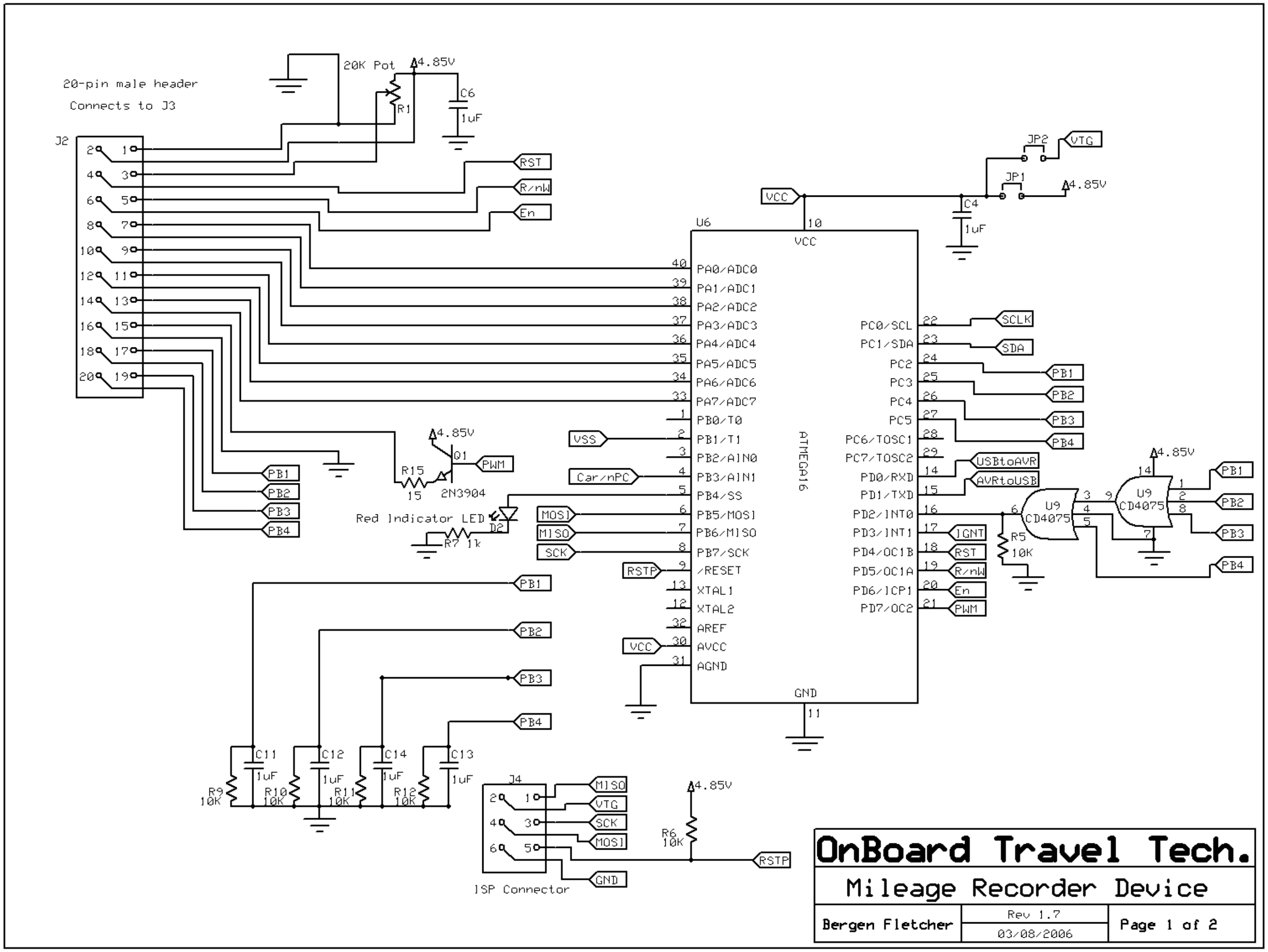


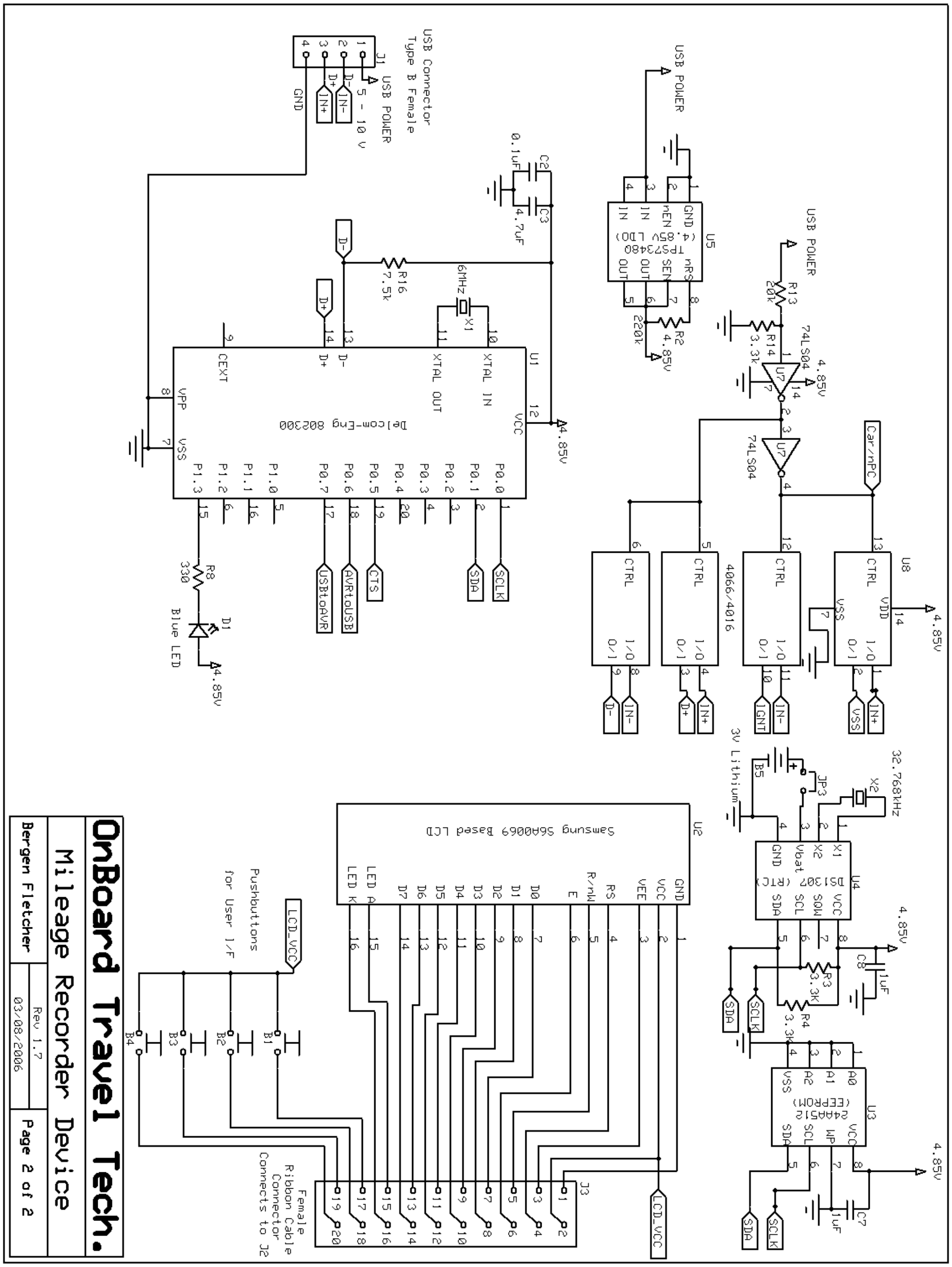
| field | | userName | carName | Destination | VSSrate | Total |
|---|---|---|---|---|---|---|
| size (bit) | | 512 | 512 | 4096 | 64 | 5184 |
| size (byte) | | 64 | 64 | 512 | 8 | 648 |
| bit location | **start** | 1 | 513 | 1025 | 5121 | |
| | **end** | 512 | 1024 | 5120 | 5184 | |



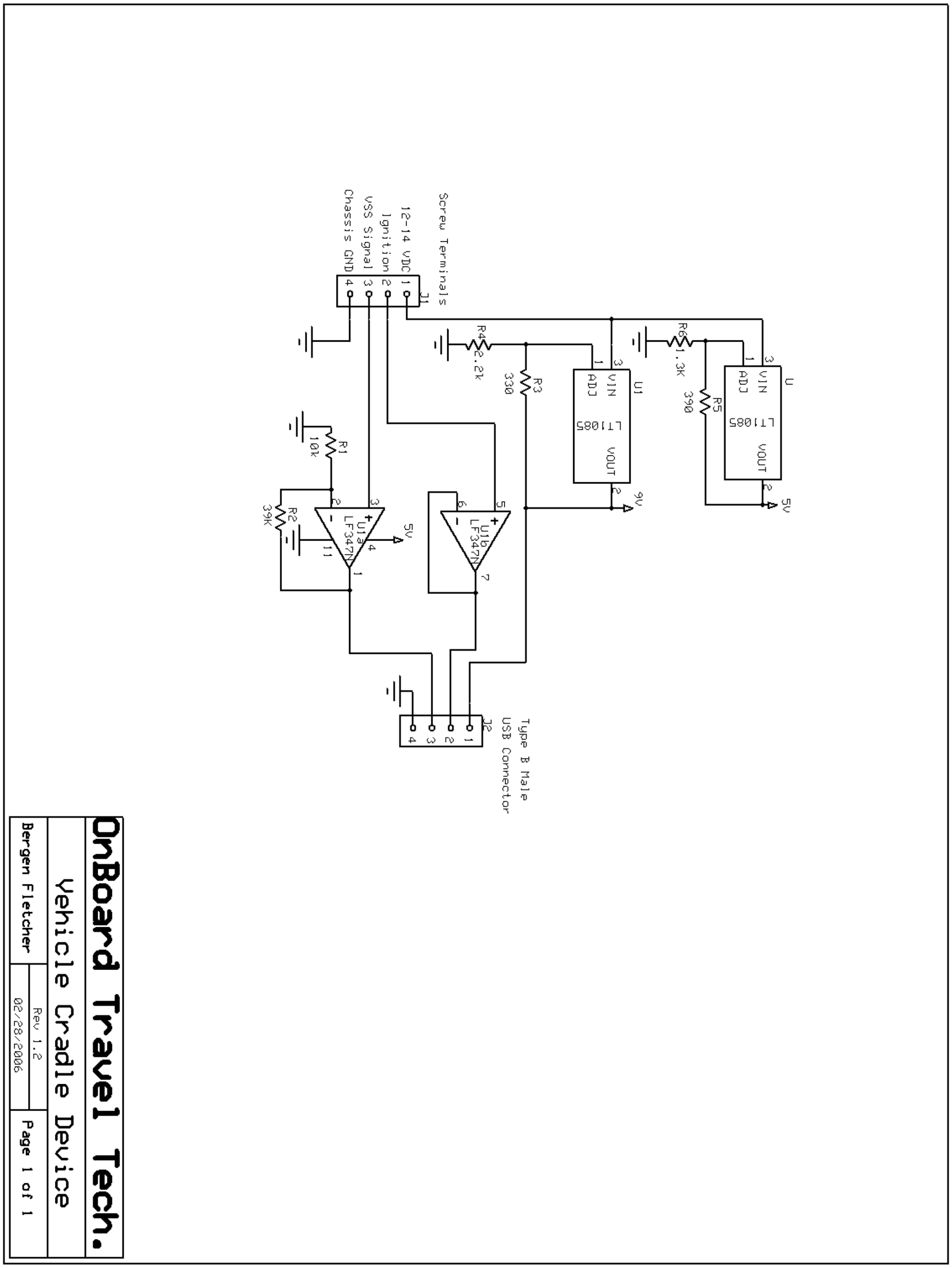
| Field | Contents | Values | Examples | |
|---|---|---|---|---|
| | | | | |
| userName | 4 strings of 16 char each of user names | | Ali Abdul Hussen | Bergen F |
| | | | Lena | Patrik |
| carName | 4 strings of 16 char each of car names | | Nissan Sentra | Mercedes SLK |
| | | | Toyota Matrix | Honda Civic |
| Destination | 32 strings of 16 char each of possible destinations | | Office 1 | Shop5 |
| | | | School | Cinema |
| VSSrate | 4 strings of 2 char each for the VSS rates (pulse/km) corresponding to the specified 4 carNames | 0->131071 | 8000 | 5000 |
| | | | 23456 | 80080 |

# Appendix B.1 (MRD Schematic)




OnBoard Travel Tech.
Mileage Recorder Device
Bergen Fletcher
Rev 1.7
03/08/2006
Page 2 of 2
USB Connector Type B Female
J1
USB POWER
5 - 10 v
D+
D-
IN+
IN-
GND
USB POWER
U5
TPS73480
(4.85V LDO)
GND
nEN
IN
IN
nRS
SEN
OUT
OUT
R2
220k
4.85V
C2
0.1uF
C3
4.7uF
R16
7.5k
6MHz
X1
U1
Delcom-Eng 802300
XTAL IN
XTAL OUT
CEXT
VPP
VSS
VCC
D+
D-
P0.0
P0.1
P0.2
P0.3
P0.4
P0.5
P0.6
P0.7
P1.0
P1.1
P1.2
P1.3
SCLK
SDA
CTS
AVRtoUSB
USBtoAVR
R8
330
D1
Blue LED
4.85V
R13
20k
R14
3.3k
74LS04
U7
Car/nPC
U8
4066/4016
CTRL
VDD
VSS
I/O
O/I
IN+
VSS
IN-
IGNT
D+
D-
32.768kHz
X2
JP3
B5
3V Lithium
U4
DS1307 (RTC)
X1
X2
Vbat
GND
VCC
SQW
SCL
SDA
C8
1uF
R3
3.3k
R4
3.3k
SCLK
SDA
U3
24AA512
(EEPROM)
A0
A1
A2
VSS
VCC
WP
SCL
SDA
C7
1uF
U2
Samsung S6A0069 Based LCD
GND
VCC
VEE
RS
R/nW
E
D0
D1
D2
D3
D4
D5
D6
D7
LED A
LED K
LCD_VCC
Pushbuttons for User I/F
B1
B2
B3
B4
J3
Female Ribbon Cable Connector Connects to J2
LCD_VCC

# Appendix B.2 (VCD Schematic)


Screw Terminals
J1
12-14 VDC 1
Ignition 2
VSS Signal 3
Chassis GND 4
R4 2.2k
R3 330
R6 1.3K
R5 390
U1
VIN
ADJ
LT1085
VOUT
U
9V
5V
R1 10k
R2 39K
U1a LF347N
U1b LF347N
J2
Type B Male
USB Connector
OnBoard Travel Tech.
Vehicle Cradle Device
Bergen Fletcher
Rev 1.2
02/28/2006
Page 1 of 1