University of Notre Dame

Final Project Report

Team Baja

Jung Whan (Stephen) Kim, Patrick Whalen, Michael Manno, Matt Creehan
5/7/2014

Table of Contents

1 Introduction page 3

1.1 Problem Description page 3

1.2 High Level Solution Description page 4

1.3 Results Overview page 5

2 Detailed System Requirements page 7

3 Detailed Project Description page 8

3.1 System Theory of Operation page 8

3.2 System Block Diagram page 9

3.3 System Packaging page 10

3.4 Power Supply page 12

3.5 Board Design page 14

3.6 GPS Module page 17

3.7 RPM Sensors page 18

3.8 LCD Screen page 21

3.9 SD Card page 24

3.10 Interfaces page 27

4 System Integration Testing page 28

	4.1 Testing	page 28
	4.2 Results	page 31
5	Users Manual	page 33
	5.1 How to Install	page 33
	5.2 How to Setup	page 34
	5.3 How to Tell if Working/Troubleshoot	page 34
6	To-Market Design Changes	page 35
7	Conclusion	page 36
8	References	page 39
9	Appendix	page 40
	9.1 MDD File System Library Software	page 40
	9.2 Main Baja Project Code, Demonstration.c	page 41
	9.3 LCD Code, newmain.c	page 66

1 - Introduction

1.1 Problem Description

The Notre Dame Baja team is a Baja car design team staffed with Notre Dame students who design and race their own Baja vehicle against other colleges around the country. These competitions consist of a variety of scoring areas, including dynamic events, such as hill climbs and maneuverability tests, and static events, such as design evaluations and presentations. Historically the Notre Dame team has performed well, finishing 32nd out of 118 teams in 2012 at their annual competition in Peoria, Illinois. The team is shooting for a top-10 finish this summer at the competition in Peoria, Illinois.

One feature that could potentially assist the Baja team in their efforts is an electrical design embedded within the Baja car. For this year's competition, the Notre dame Baja team wanted to install a system to improve the design score points by the judges before the race as well as the systematic analysis of their race through the electrical system data. The mechanical engineering Baja team proposed the cooperative design project with the electrical engineering major students to the Electrical Engineering Senior Design Professor Michael Schafer. As the electrical engineering team, we were responsible for the implementation of this electrical system, including two Hall effect sensors, a GPS (Global Positioning System) chip, an SD (Secure Digital) card, LCD display, several buttons and switches, and the box containing all the sub-components. The goal was to complete the comprehensive design process by May 2nd, the final demonstration day, and to help the Notre Dame Baja team achieve its top-10 finish in the competition this summer. The Baja Design

Process was a two-semester long project (Fall 2013, Spring 2014) to meet expectations outlined in discussions between the mechanical engineering “Notre Dame Baja Team” and the electrical engineering “Team Baja.”

1.2 High Level Solution Description

In order to meet the expectations of the mechanical engineering team, we designed a system that would be mounted inside the driver’s cabin of the Baja car for easy viewing and access for the driver. The system was designed to be protected from the harsh Baja environment yet compact and lightweight. The system includes an LCD screen on the front for the driver to view speed and lap data as well as an SD Card inside the module intended for storing and uploading vehicle speed, GPS, lap, and rpm data. Hall effect sensors would extend from within the box so that they could be mounted in the vehicle to record wheel and engine rpm using magnets mounted on the vehicle. The module includes an external power switch, a lap button for recording lap time, and is powered by a two-cell lithium ion battery.

The system has three modes. The first, “rest mode,” is the standard mode that is loaded upon startup in which the system simply displays velocity as the car drives. The second mode, “race mode,” allows the user to see vehicle speed and lap time and stores speed, GPS, rpm, and lap data to an SD card every second. The final mode, “test mode,” would allow the user to track rpm data on an SD card at a much faster rate to observe gear shifts during a 10-20 second test.

1.3 Results Overview

The final design of our electrical system met the original expectations in most of the areas. The details will be illustrated in the Detailed Systems Requirements Section . However, at the same time, the electrical design team abandoned some of the initial plans, including the SD card and several buttons, due to the time constraints at the end of project timeline.

First of all, the Mechanical Baja team requested the lightweight electrical system to maximize the vehicle's speed in the race. In our electrical system, we managed to use the light sub-components, such as a wooden-frame container and a Hall effect sensor mounted on a small size breadboard circuit. Moreover, our design focused on the durability of a system container to be mud resistant throughout the competition. Any holes or openings of a wooden container were filled with Silicon paste to avoid any water or mud leakage into the sub-components inside a wooden container of our electrical system.

The display output was a success and displays the data we originally planned to display, including the lap time, velocity, and GPS fix status. The UART connection with the GPS module worked very well, and we were successful in finding a fix outside. In addition, the demo version of our module was able to display RPS (Rotations Per Second) for the two rpm sensors. We chose to display this to demonstrate that they were functioning correctly since the SD data was not available.

Unfortunately, after getting it to work smoothly on the demo board, our SD card subsystem did not work adequately on our new board. The microcontroller consistently

sent strange signals on the SPI4 clock pin, which we believe caused the issue. Because this hardware problem was discovered so late in the process, we were unable to resolve it in time for the demonstration.

The system interface is also simple for the users to control as there are buttons to choose the data collection mode and to reset the data display on screen. Because we spent so much time trying to get the SD card to work, we did not get a chance to implement the button software which had worked sufficiently on our development board. Because the SD card did not work in the final demo, we decided to prioritize other things (like printing rpm data) over the buttons since their functions were tied to the SD card.

With the exception of the SPI4 interface, the rest of the electrical power system worked very well. The two-cell lithium ion battery was easily chargeable and powered the board as expected. All of the subsystems received the correct power, and after making some changes to the original design, the board behaved as we intended it to.

Finally, the electrical engineering design team developed the system under the budget requirement. Our overall spending was well below the maximum budge limit. One of the reasons was that some system parts were provided by Professor Schafer without extra charge. For example, a display screen was provided which has an estimated 30 to 40 dollar purchase cost. However, even after adding the possible additional costs for the parts our design team used for free, the final budget for electrical system was still below the maximum budget limit.

2 - Detailed System Requirements

The critical aspect of the judging that the Baja mechanical engineers hoped to improve was the car's electrical design. In order to best serve them, it was important for our team to meet with the mechanical engineers to clarify the project requirements. The descriptions below in Table 1 summarize the detailed system requirements for each sub-component.

Table 1. System Requirements

System Requirements		
Requirement	Description	Result
Hall Effect Sensors		
	- Must measure both wheel rpm and engine rpm - Must provide an updated information every second - Must be capable of also providing rpm data at a faster rate for "test mode"	Complete Complete Complete
GPS/SD Card		
	- Must provide the vehicle data, including velocity, rpm, and GPS coordinates every second - SD Card must be easily accessible/removable - Must save the GPS data into the SD card	Complete Complete Incomplete/ Hardware issues
Display		
	- Must display the key data on screen, including velocity, lap time, data collection mode, and fix - Must update the information every second - Must change the display format and lap information depending on the buttons pushed	Complete Complete Incomplete/ issues with buttons
Container		
	- Must contain all the sub-components within the container - Must be mud resistant - Must have the buttons placed outside of container for the display control	Complete Complete Complete
Battery		
	- Must provide sufficient voltage to power the electrical system - Must be rechargeable - Must be independent from the voltage supply from	Complete Complete Complete

	Baja car	
Buttons		
	- Must have the button for the power On/Off	Complete
	- Must have the button for lap time and split	Incomplete
	- Must have the button to switch between test and race mode	Incomplete
	- Must have a button to return to “rest mode”	Incomplete

One requirement not fully detailed in the System Requirements is how the module would be mounted in the car. Because design is not yet complete on the Baja car and the mechanical engineering team was not yet sure how they wanted to mount the module, we designed the module as a rectangular container which could then be mounted near the driver. Another idea raised by the mechanical engineers was to 3D-print a steering wheel to hold the screen while the rest of the module is mounted separately. It remains to be seen how the mechanical team plans to deal with mounting the device.

3 - Detailed Project Description

3.1 System Theory of Operation

The module we designed turns on with either the external power switch or the internal switch on the board. As soon as the module turns on it should display the velocity (00.00 if it does not have a GPS fix), the mode (“race”), current lap, (counting upwards each second), and previous lap (nothing displayed). GPS fix status should be displayed in the bottom right, displaying “Fix” if it has located satellites or “No Fix” if not enough satellites have been found to provide data. Revolutions per second (rps) for each rpm sensor will be displayed in the top right of the screen. Any time a magnet crosses one of the sensors (within a couple centimeters), the rps value should jump to

the corresponding value for how many times the magnet passed the sensor in that passed second. If the GPS receives a fix, the “No Fix” originally displayed on the screen should change to “Fix,” and the speed should begin to display on the screen in mph.

Unfortunately there are a few bugs in the current model. First of all there are some issues with cursor locations so the screen will often print in the wrong location. Second of all, none of the values will display on the screen until the “reset” button on the board is pressed, so although the power switches are capable of turning the device on, the “reset” button is needed to run the code. Lastly, the SD capabilities are not currently integrated on the board so those functionalities (and the corresponding buttons) are not in full working mode. The System Block Diagram and State Diagram in Figures 1 and 2 below will illustrate how the system would work had the SD card been integrated.

3.2 System Block Diagram

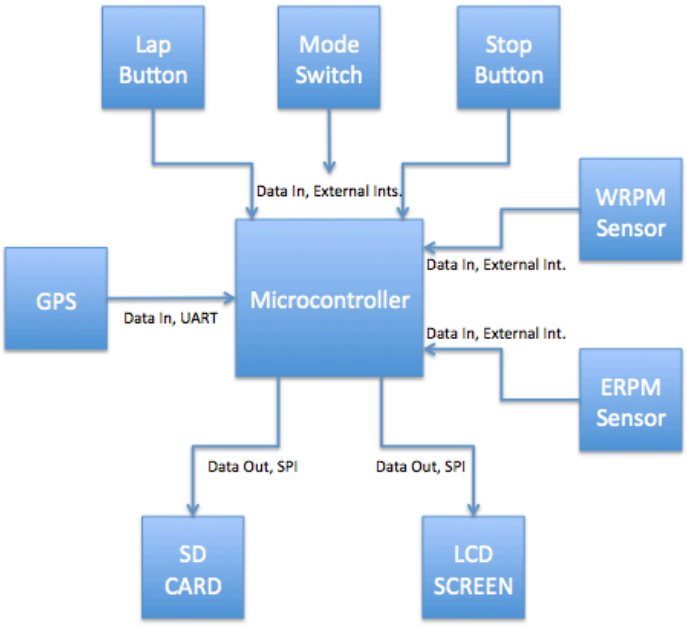


Figure 1. System Block Diagram of Proposed Design

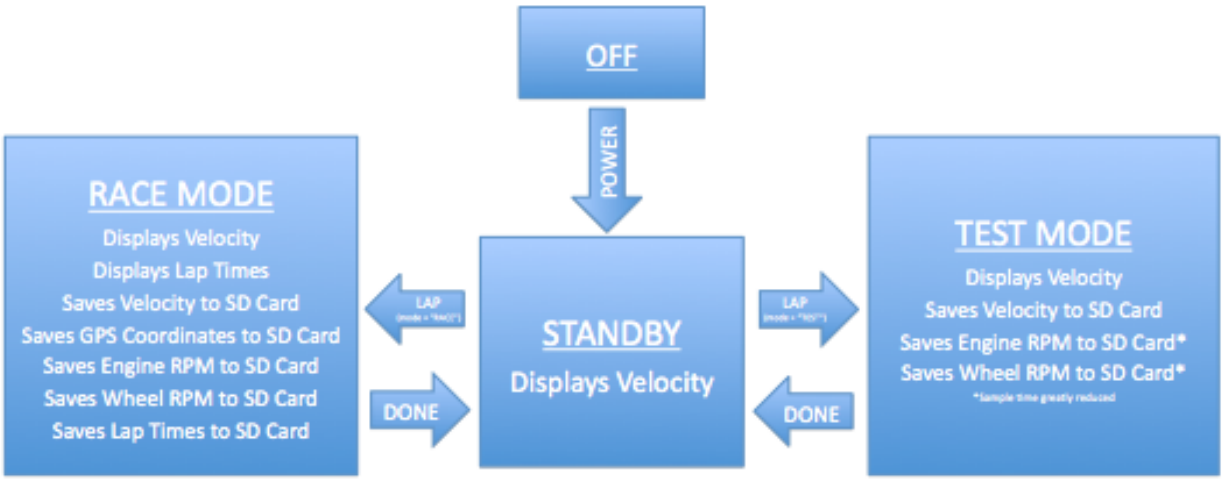


Figure 2. State Diagram of Proposed Design

3.3 System Packaging

There were several environmental concerns when designing the physical container that would house our system. Primarily, it needed to be water and mud resistant in order to protect the elements within from any dirt and debris during the course of the race. Secondly, the box needed to be sturdy and reliable, so as to not break or open during the rough race. Finally, the box needed to be large enough to safely house all of the necessary elements for our system. Accordingly, the box shown below in Figures 3 and 4 was selected and designed.

Figure 3: Container Exterior

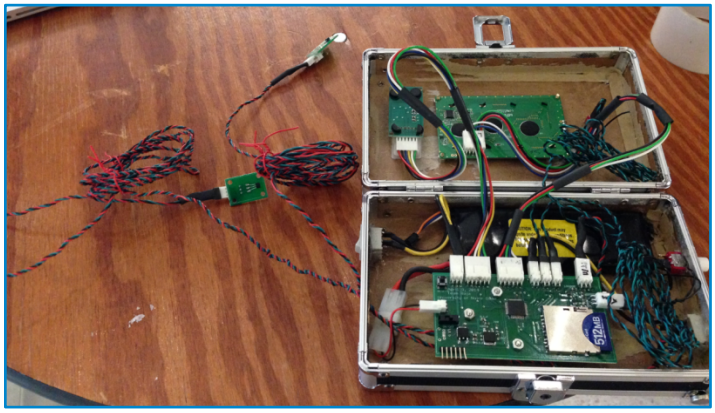


Figure 4: Container Exterior

Physically, this box meets all of the qualifications described above. The box is made of a composite wood, with a sturdy metal frame, assuring that it can hold up in the stressful race environment. Equally as important, the box has a locking mechanism, which can bee seen on its front. By having a locking mechanism, the box allows access to its interior, most notably the SD card, while maintaining securely closed when locked. The box was not completely waterproof, however, especially after holes were cut for the screen and buttons. Silicon sealant was used to waterproof these holes, helping to insure that no water could get into the box. This box was also large enough to house all

of our components, with enough room to hold the wires and battery. Also, being made of wood, the box allowed for the GPS module to be mounted within the box rather than externally, allowing for addition protection for that module. Furthermore, the box is easily opened and closed. This makes it easy for the Baja team to access the SD card when they're trying to analyze their data.

The buttons are all mounted in the side of the module and held in place with sealant, while the GPS and battery are held steady with Velcro strips. The board is elevated off of the base of the box with spacers and screwed tight to ensure that it does not move around during use. The LCD screen is also screwed into the box and is also sealed around the edges and screws to keep as much dirt out as possible. The rpm sensors extend from a hole in the board, out from which they can extend to wherever they would need to be placed on the Baja vehicle. The hole for these sensors was left unsealed to enable the mechanical engineering Baja team to easily pull those out in case they needed to be moved or adjusted. Eventually, the team will have to seal that hole with sealant as well once the final setup is determined.

3.4 Battery System

The system is powered by a two-cell lithium ion battery placed within the primary container. The battery supplies 7.9V when fully charged, more than enough to power all of the components on the board. The power system also contained two power switches, wired in parallel. One of these switches was placed directly on the board, and was used primarily during the testing and development of the system. The other switch was panel mounted on the box, and was used during operation of the system. In order for the

system to be powered off, both switches need to be in the off position. If either switch is on, the board receives power. The battery was attached to the board via a 2-pin Molex connector which ran to the switches, and to the regulators from there. Originally we had planned to also include a micro USB adapter to power the board as well, during the testing. This plan was eventually scrapped, as the battery worked just fine. We were able to abandon this feature of the design, as it was only included for testing purposes, and was not essential to the final design.

The system needed both 3.3V and 5.0V running to multiple locations across the board. In order to generate each of these voltages, two different voltage regulators were used, a 3.3V regulator and a 5.0V regulator. Two regulators, rather than a regulator and a DC-to-DC converter, were chosen for our voltage sources so as to insure that sufficient current was supplied to all the elements of the board. A DC-to-DC converter would have caused very low current on the 5.0V supply, possibly causing performance issues with the 5.0V components. A schematic for the voltage regulators and power source is included below in Figure 5:

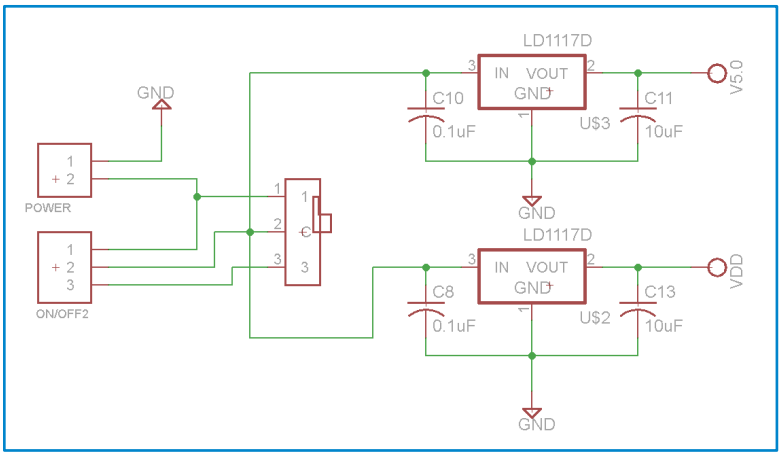


Figure 5: Power Supply Schematic

The microcontroller, the SD card, the GPS module all required 3.3V to operate, while the LCD screen and the Hall Effect sensors each required 5.0V to operate. In order to run the traces to the necessary locations, the 5.0V trace was run along the perimeter of the board, while the 3.3V traces ran along the interior of the board. This was primarily due to the location of the microcontroller in the center of the board.

The original schematic mistakenly had 5.0V running to a pin on the SD card adapter, a pin which should have been grounded. This mistake caused the board to overheat almost immediately when the SD card was inserted, however once this mistake was corrected the board power system functioned properly. It is worth mentioning that, once the voltage level of the battery drops below the 5.0V necessary for the 5.0V regulator, the components which require 5.0V to operate no longer function properly, causing the system to shut down. During our testing, we found that the battery would provide 8 to 10 hours of charge before this became an issue.

3.5 Board Design

The physical board design consisted of five essential elements: the microcontroller, the SD card adapter, the power source management, the circuit used by the Hall effect sensors, and the Molex connectors for various sensors and displays. In addition to these components, the board also had 3 mounting holes, which allowed for the board to be secured and mounted properly within the container. The board design is displayed below in Figure 6:

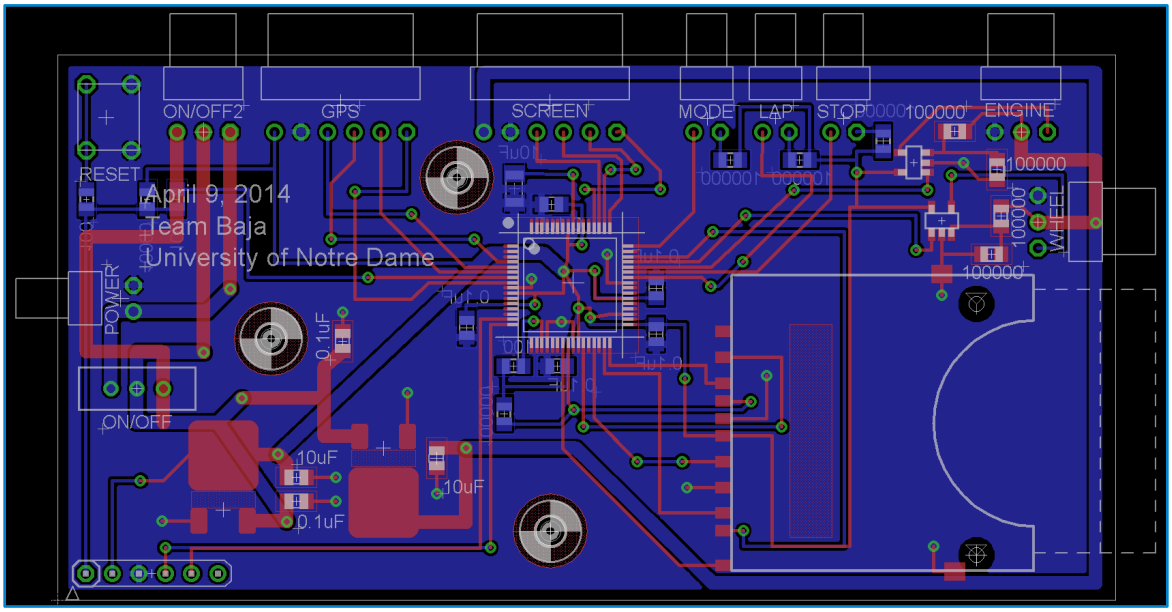


Figure 6: Board Design

The first of these elements, the microcontroller, can be clearly seen mounted near the center of the board. This location provided the maximum accessibility for all of the other elements, allowing for each element to access the necessary pins. A PIC32MX795 microcontroller was selected and used. This microcontroller was selected for two main reasons. It was possessed enough pins to support all of the elements necessary for our system. Also, it was the same microcontroller as was used on the development boards, which made for easier transition and implementation of our software from the development board to our board. The PicKit interface, in the extreme bottom left of the board, is also mirrored off of the development board, along with the reset button in the extreme top left.

The second component, the power supply management, can be seen in the left portion of the board design. This component consists of the two voltage regulators for 3.3V and 5.0V, the on off switch, and two Molex connectors. The first Molex connector,

on the left side of the board, was used to connect the battery to the board, providing the necessary power. The second Molex connector, labeled “ON/OFF2” on the top left of the board, ran to the external power switch. This section component will be described in greater detail in the “Power Supply” section.

Thirdly, the SD card adapter covers the bottom right portion of the board, seen above. This element was positioned such that, when the SD card was locked into the adapter, it would not stick over the edge of the board, but when it was released, it would hang slightly off the edge of the board. This was intended to provide maximum security for the SD card during the bumpy ride and to ensure that it remains locked in place during that time. Then, when the SD card is to be removed, it could easily be done, as the card hung off the edge of the board. The SD card adapter was wired to the microcontroller in an identical fashion to the way it was attached on the development board, with the one exception being that the chip select pin was moved from B8 to B12. This was done in an effort to ease the transition from the development board to our board.

The circuit used by the Hall effect sensors can be seen in the upper right portion of the board design. This circuit connects to the Hall effect sensors via two 3-pin Molex connectors. The sensors would then be mounted close to the wheel or engine, and send an analog signal back to the board. The circuit on the board would then generate a digital signal from that analog signal, and send it to the board, to two external interrupt pins. More detail on the operation of the Hall effect sensors can be found in the “RPM Sensors” section.

Finally, across the top of the board the various Molex connectors can be seen. These connectors connected the various elements to the microcontroller. Six-pin Molex connectors were used to connect the screen and GPS module, while two-pin connectors were used to connect the various panel-mounted buttons. These Molex connectors were selected for two main reasons. The actual connection is very secure, insuring that it would not shake loose during operation. Secondly, the way the connectors are designed, they can only connect when oriented the proper way. This was a valuable feature, as it insured that the connector could not be attached upside down, causing damage to both the microcontroller and the element.

3.6 GPS Module

A major source of data for this system is the GPS module. For the GPS data, we used a Mediatek-3329 that was provided by Professor Schafer. This system came with a board attached and hooked up to out board with a six-pin Molex connector. The GPS communicated with the microcontroller through UART pins. Below in Figure 7 is the schematic detailing the pins used by the GPS system:

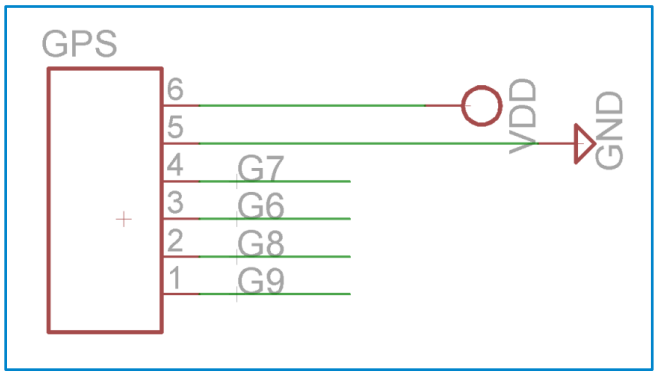


Figure 7. GPS Module Schematic

The mediatek-3329 runs off of 3.3V and has an enable pin that must be set high to V_{dd}. Once plugged in, the GPS will look for a fix, and when it finds enough satellites, begins transmitting data from those satellites. Outside on Notre Dame's campus, it typically took me 2-5 minutes to get a GPS fix after a long period with no fix, but it was usually much quicker when the fix had only been lost for a short period of time.

The GPS sends a multitude of data at a rate of one update per second. Initially, this data is sent as ascii strings. It is much easier, however, to use that data when it comes in binary. Therefore out code sends the GPS a command to put it into binary mode.

In binary mode, the messages still come once a second, but the format is much different. Some of the data available in this mode includes altitude, latitude, longitude, velocity, date, UTC time, as well as prefixes to identify GPS messages and checksums to confirm that messages have been read appropriately. Once the message comes in, the relevant values (latitude, longitude, and speed) are stored in variables as unsigned longs where they can then be sent to either the screen or SD card. Currently, the GPS system is successfully able to interface with the LCD, indicating whether a fix has been found and displaying velocity to the user.

3.7 RPM Sensors

The requirement for this subsystem is the ability to measure RPM. Together with the SD card subsystem, the rpm sensor subsystem must be able to record the measured rpm data at one second intervals for race mode and faster for test mode. In hardware, this is implemented by a simple Hall effect sensor and a series of magnets.

An SEC SS49E Linear Hall effect sensor was used in this project. The magnets are arranged on both the wheel and the engine such that as either the wheel or engine rotates, the magnets pass the associated Hall effect sensor. The Hall effect sensor must be placed close enough that the passing magnet changes the voltage over the sensor appreciably; this distance should generally be lower than a half an inch.

When a magnet passes the Hall Effect sensor, the voltage is changed from nominal to a higher value. On the board, this signal acts as the non-inverting input of an op-amp comparator. When not active, the voltage will be lower than the inverting input, so a logical low, 0 V, is the output. When active, the output will be a logical high, 3.3 V. The wheel sensor is connected to pin D11, external interrupt 4. The engine sensor is connected to pin D10, external interrupt 3. This interrupts are set to trigger off the rising edge of the input, the output of the comparator for the sensor.

In the code for the project, the interrupts cause a counter associated with the respective sensor to be incremented. It is this count that is stored to the SD card either every second, for race mode, or more frequently, for test mode. The counter is cleared after it has been written to the SD card in order to initialize the counter for the next RPM data measurement. Actual RPM is calculated by accounting for number of magnets used and recording period such that RPM = count/(number of magnets * the conversion between the recording period and one minute). Below in Figure 8 is the schematic for the rpm sensors.

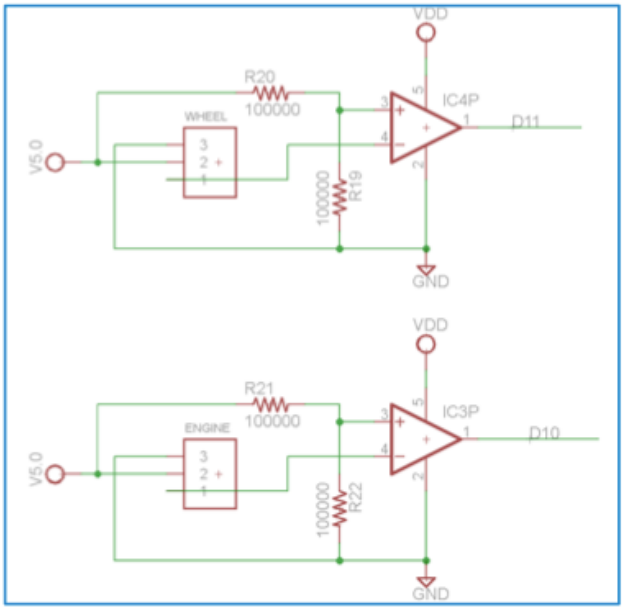


Figure 8. RPM Sensor Schematic

The SS49E Linear Hall Effect Sensor was chosen for this application because it possesses a high degree of sensitivity and a very low rise/fall time. High sensitivity is favorable because it allows a degree of variability in conditions for the strength of the magnets and their distance from the sensor. The low rise/fall time is important because it insures that the signal from one magnet will not affect the next magnet's signal. The sensor outputs near 2.5 V, so the op-amp comparators use 2.5 V as the inverting input. In this way, when the sensor detects a magnet, the voltage will rise and the op-amp will output 3.3 V. This subsystem was tested by placing an LED and a current limiting resistor in parallel to the input pin of the microcontroller. The LED lights up with the voltage change, ensuring that the physical circuit is functioning properly.

3.8 LCD Screen

The LCD display used for our project was a Newhaven 4x20 serial liquid crystal display module, part number NHD-0420D3Z-FL-GBW-V3. This screen was large enough to display all of the relevant information to the driver, while still remaining small enough to both fit properly in the box, and to properly interface with our selected microcontroller. The screen was mounted in the lid of the box with 4 screws, one through the mounting holes in each corner. It then connected to the board via a 6-pin Molex connector, which was in turn connected to the SPI3 interface on the microcontroller. We elected to use an SPI interface to program the screen, as we were most familiar with that interface. Our familiarity enabled the programming of the screen to go more smoothly than it would have had we been required to learn a new interface from scratch.

Pin assignments for each of the 6 pins are as follows. The last two pins, pin 6 and pin 5, were connected to 5.0V and ground, respectively. From there, pin 4 was wired to D3 on the microcontroller, and served as the MOSI pin for the LCD screen. This pin is used to send the display data to the screen. Pin 3 was connected to D1 on the microcontroller, and provided the clock signal for the screen to use. Pin 2 was connected to D2 and provided the MISO signal to the screen. This pin was ultimately unused, as the screen does not output data back to the microcontroller. Finally, pin 1 was connected to D4 on the microcontroller, and served as the chip select pin. This chip select pin needed to be set low prior to sending data to the screen, and then high again following the data transmission. These pin assignments were largely intended to mirror

those used by the development board, once again easing the software transition from the development board to our board. The assignments are shown below in Figure 9:

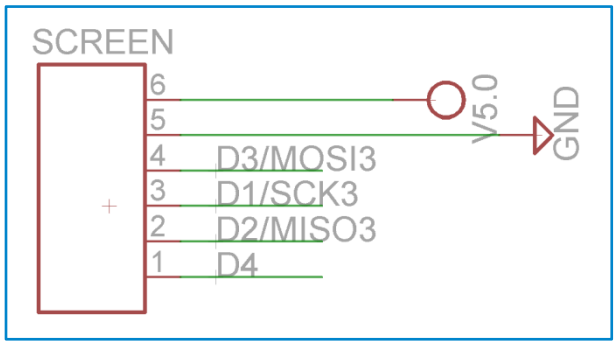


Figure 9: Screen Pin Assignments

The 6-pin Molex connector was again invaluable when creating the jumper cables to connect the screen and board. The connector insured that it was impossible to connect the pins upside down, wiring 5.0V to the chip select pin and so on. This provided an extra layer of security to our board, helping to further insure that no components were accidentally damaged or destroyed by a faulty connection.

In order to manipulate the screen and print in the correct places, we accessed the commands provided in the product data sheet. These commands are displayed below in Table 2:

Table 2: LCD Screen Commands

Prefix	Command	Parameter	Description	Execution time
-	-	1 Byte	Display Character Write (0x00 ~ 0xFF)	100uS
0xFE	0x41	None	Display on	100uS
0xFE	0x42	None	Display off	100uS
0xFE	0x45	1 Byte	Set cursor	100uS
0xFE	0x46	None	Cursor home	1.5mS
0xFE	0x47	None	Underline cursor on	1.5mS
0xFE	0x48	None	Underline cursor off	1.5mS
0xFE	0x49	None	Move cursor left one place	100uS
0xFE	0x4A	None	Move cursor right one place	100uS
0xFE	0x4B	None	Blinking cursor on	100uS
0xFE	0x4C	None	Blinking cursor off	100uS
0xFE	0x4E	None	Backspace	100uS
0xFE	0x51	None	Clear screen	1.5mS
0xFE	0x52	1 Byte	Set contrast	500uS
0xFE	0x53	1 Byte	Set backlight brightness	100uS
0xFE	0x54	9 Byte	Load custom character	200uS
0xFE	0x55	None	Move display one place to the left	100uS
0xFE	0x56	None	Move display one place to the right	100uS
0xFE	0x61	1 Byte	Change RS-232 BAUD rate	3mS
0xFE	0x62	1 Byte	Change I2C address	3mS
0xFE	0x70	None	Display firmware version number	4mS
0xFE	0x71	None	Display RS-232 BAUD rate	10mS
0xFE	0x72	None	Display I2C address	4mS

These commands were sent with the snd_cmd() function included in newmain.c. This function takes the command and parameter as inputs and sends those commands to the LCD screen. While for the most part, the screen commands were successful, there are some bugs with setting the cursor, as the cursor will jump to an incorrect location every once in awhile. While we did not have too much time to research this issue due to our SD issues, we believe this is an issue that can be fixed by tweaking the delays within the program to ensure that no errors can materialize.

As of our demonstration, the screen displays velocity data, current lap, as well as rpm sensors (in revolutions per second) every second. It also displays whether the GPS has received a fix or not. In the final product, the rpm would not necessarily need to be displayed, but since it could not be saved ad viewed we decided to display it. Once the buttons are integrated, the lap button will switch the current lap time to the previous lap section and restart the current lap time. The mode switch will be able to switch the mode from “race” to “test” and vice versa.

3.9 SD Card

The most tricky subsystem to work with on this project was the SD card. As the mechanical engineers wanted an easy way to save, track, and store data, we proposed an SD card system as a cheap and easy way to keep good records of their Baja tests and races.

In order to setup and write to an SD card, you must have an adapter. The initial adapter we worked with was provided by Professor Schafer and connected to the development board from its own board through a 12-pin connector. An image of the schematic can be seen below in Figure 10:

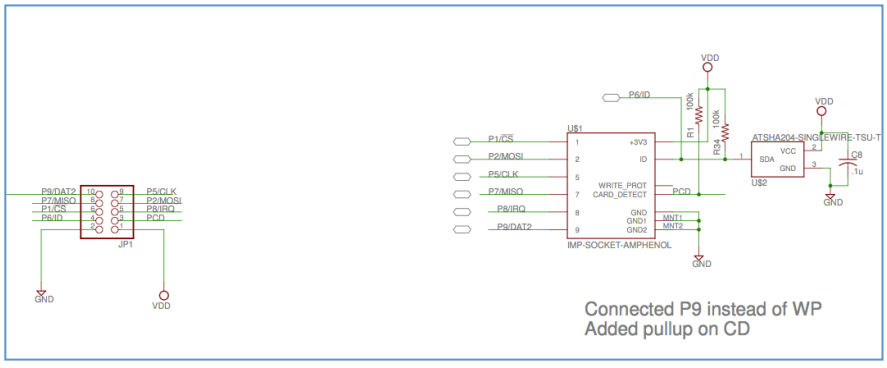
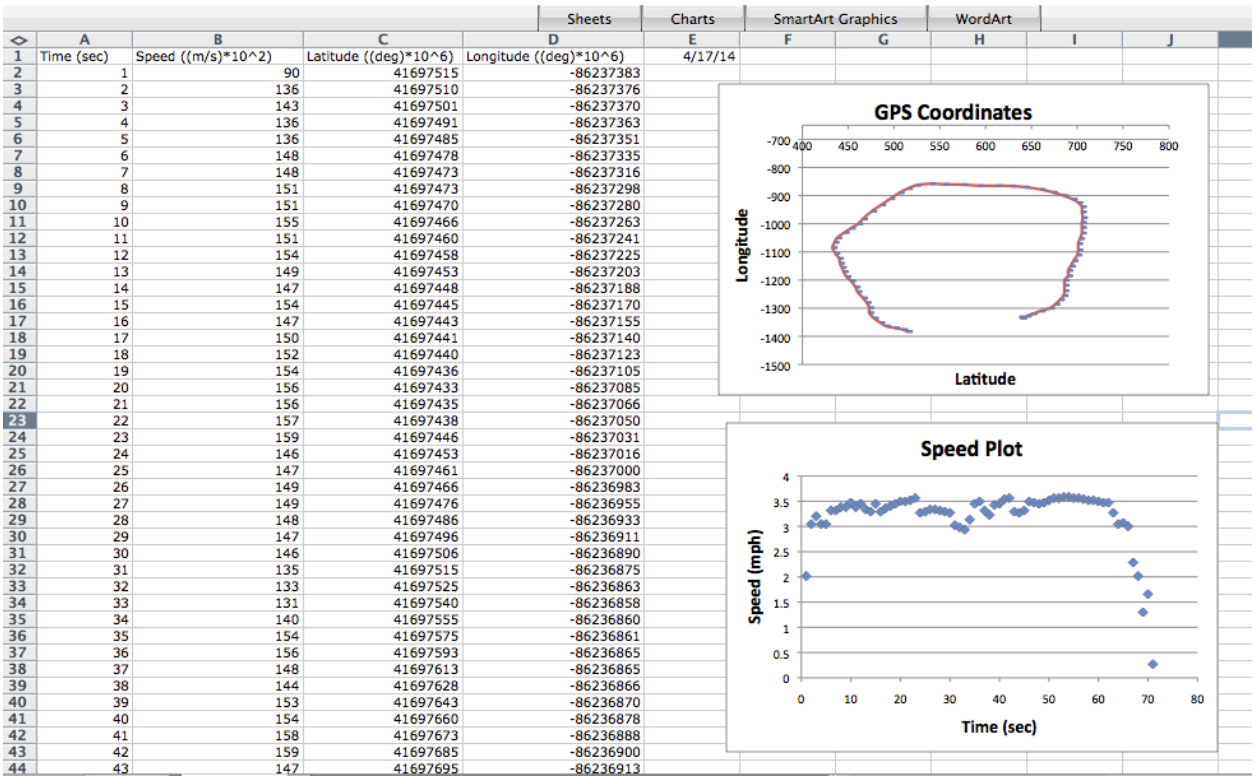


Figure 10. Development Board SD Card Adapter

For our board, however, we decided to skip having a separate board and put an SD card adapter right on the board. This allowed us to make a more compact design while still making it easy to open and remove the SD card.

The SD card operates on 3.3V and communicates through SPI. In order to setup and write to an SD card with a FAT library, the best way is to download Microchip's MDD library. This library contains a host of functions for initializing, configuring, and writing to an SD card. After successfully creating files using just the MDD library functions, last semester, the first step this semester was to create files using the GPS data from the mediatek3329.

On the development board, we had good success with linking the GPS and SD systems. After many initial issues, we effectively got the SD card printing consistant data every second to files created in the software. An example of this data can be seen below in Figure 11:



	A	B	C	D	E
1	Time (sec)	Speed ((m/s)*10^2)	Latitude ((deg)*10^6)	Longitude ((deg)*10^6)	4/17/14
2	1	90	41697515	-86237383	
3	2	136	41697510	-86237376	
4	3	143	41697501	-86237370	
5	4	136	41697491	-86237363	
6	5	136	41697485	-86237351	
7	6	148	41697478	-86237335	
8	7	148	41697473	-86237316	
9	8	151	41697473	-86237298	
10	9	151	41697470	-86237280	
11	10	155	41697466	-86237263	
12	11	151	41697460	-86237241	
13	12	154	41697458	-86237225	
14	13	149	41697453	-86237203	
15	14	147	41697448	-86237188	
16	15	154	41697445	-86237170	
17	16	147	41697443	-86237155	
18	17	150	41697441	-86237140	
19	18	152	41697440	-86237123	
20	19	154	41697436	-86237105	
21	20	156	41697433	-86237085	
22	21	156	41697435	-86237066	
23	22	157	41697438	-86237050	
24	23	159	41697446	-86237031	
25	24	146	41697453	-86237016	
26	25	147	41697461	-86237000	
27	26	149	41697466	-86236983	
28	27	149	41697476	-86236955	
29	28	148	41697486	-86236933	
30	29	147	41697496	-86236911	
31	30	146	41697506	-86236890	
32	31	135	41697515	-86236875	
33	32	133	41697525	-86236863	
34	33	131	41697540	-86236858	
35	34	140	41697555	-86236860	
36	35	154	41697575	-86236861	
37	36	156	41697593	-86236865	
38	37	148	41697613	-86236865	
39	38	144	41697628	-86236866	
40	39	153	41697643	-86236870	
41	40	154	41697660	-86236878	
42	41	158	41697673	-86236888	
43	42	159	41697685	-86236900	
44	43	147	41697695	-86236913	

Figure 11: SD-GPS System Integration Success

The created file is a .csv file which can easily be opened in excel and converted to an excel file. The file in Figure 11 was generated by walking around DeBartolo quad with the device in my hands. Looking at Figure 11, the SD card saved time in seconds, velocity, as well as GPS coordinates. With that data I was able to convert it to the correct units and plot a velocity versus time graph and plot my travelling course within only a few minutes.

Unfortunately the SD card software did not seem to cooperate with our new board. The SPI communication repeatedly failed, and we could not seem to diagnose what was wrong. Finally we found that the peak voltage of the SPI4 clock was only 1.2-1.3V. We believe that something must've happened to the microcontroller that messed

up the clock pin, and we were unable to resolve this issue in time. We are hopeful, however, that with a new microcontroller, the SD subsystem will succeed again.

3.10 Interfaces

Though our “lap” and “stop” buttons are not fully integrated onto the new board, once the SD system is fixed, they become very important. The “lap” button signals when a race starts and after that signals when a lap is complete. This signal tells the screen when to shift lap data and the SD system when to save lap data. This process continues until the “stop” button is pressed, sending the device back into its standard operating mode and signaling that the race is over.

Both buttons are implemented similarly from a physical circuit perspective. When the button is pressed, a logical high, 3.3V, is connected to the appropriate pin. When the button is not pressed, there is an open circuit on the pin. Below in Figure 12 is a schematic of the buttons.

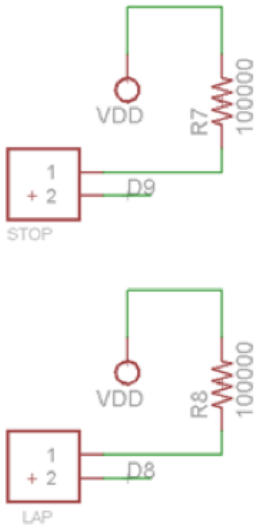


Figure 12. Button Schematic

The lap time button is connected to pin D8, external interrupt 1. The stop or done button is connected to pin D9, external interrupt 2. The interrupts themselves only set variables to indicate to the main program what should be done next. The lap button's interrupt ultimately causes the current lap time to be recorded to the SD card, displayed as the previous lap time, and finally cleared to set up for the next lap. The done button concludes operation of the device at the end of a race.

4 - System Integration Testing

4.1 Testing

After soldering all of the components onto the board as was shown on the original schematic, we began testing all of the essential subsystems, starting with the 3.3V and 5.0V regulators. During our initial testing, we found that the 3.3V regulator was functioning properly, but the 5.0V regulator was not functioning correctly, as the 5.0V trace was reading at 7.9V, the same voltage as the battery. This was a fortunate break, as the 5.0V trace did not run to any components directly, but rather to various Molex connectors, which in turn would run to the components. Because these connectors were open during this stage of the testing, no damage was done by the 7.9V on the board in place of 5.0V. Reviewing the Eagle schematic and board design, it came to our attention that two of the pads on the voltage regulator were connected to the same ground via, causing an effective short between these two pads and preventing the regulator from functioning properly. By removing a resistor connected to the faulty region, the issue was successfully resolved. Both the 3.3V and 5.0V regulators were functioning properly. While soldering the board it also came to our attention that there

was an extra 100000Ω resistor, causing two of these resistors to be wired in parallel. While this error did not cause any large issues with voltages, it was omitted from the board while soldering in an effort to maintain accuracy between our board and the development board.

After fixing the two voltage regulators and insuring that the proper voltages were being delivered to each component on the board, we turned our attention to the next issue. When the SD card was inserted into the SD card adapter, the board got very hot, and the problem could only be solved by the removal of the SD card. Once again turning to the schematic and board design, we again noticed that there was an incorrect trace on the board. One of the pins on the SD card adapter, the ID pin, was incorrectly wired to 5.0V, when it should have been in face wired to ground. Cutting the trace on the board between 5.0V and the ID pin on the SD card adapter resolved this issue, allowing the SD card to be safely inserted without causing the board to overheat.

Now that the board was wired properly, and the proper voltages were being delivered to each component, we began to test the various subsystems on our board. Several of these transitions went off without a hitch. The software and hardware for both the LCD screen and the Hall Effect sensors transferred from the development board to our board perfectly, with minimal to no changes necessary in the software. One thing which was off about the LCD screen, on both the development board and on our board, was that, after some time, the screen would print values to the wrong location. The time it took for this to occur would vary, but after playing with the delays in the code we managed to get it to last for upwards of one minutes before these errors would begin to

occur. There was no readily apparent cause for this in the software, but due to the consistency across both boards, we are confident that this was a software issue.

We also encountered issues when attempting to integrate both the GPS and SD card subsystems into our board. The SD card adapter, which was connected to the SPI3 interface, was getting a faulty clock signal. On the original development board, the voltage on the clock signal cycled between 0.0V and 3.3V, between 0 and V_{DD}, as was expected. On our board, however, the clock signal cycled between 0.0V and around 1.3V, which was well below the recommended voltage range for the clock signal in the SD card spec sheet. This problem we believe to be at the heart of our issues in getting the SD card to properly receive and send signals, ultimately resulting in the inability to use that subsystem on our board. We believe the root cause of this issue to be hardware related. The issues with the GPS however, were thankfully able to be resolved. Upon initial implementation of the GPS module into our board, we noticed that the module appeared to be triggering the MCLR pin every second, as the program seemed to reset every second. Through testing we were able to determine that the issue wasn't a short to the MCLR pin, but was instead was something within the SD functions was causing the problem. Once the SD card functions were commented out, the GPS module functioned properly, sending and receiving messages to and from the microcontroller.

Once we had determined which of the subsystems we could successfully integrate together, we assembled the system and mounted it on a bike, hoping to further test how well the functioning subsystems had been integrated together. Our bike test

was able to successfully demonstrate that, aside from the SD card subsystem, the system had been integrated successfully. The GPS and RPM modules were returning accurate data, and the information was displaying successfully on the screen. Similarly, the lap counter was accurately counting seconds, and displaying the cumulative count on the screen.

4.2 Results

After resolving as many of the implementation issues as possible, we were able to test the system as a whole. During this testing we were able to show that we met most of the design requirements. Beginning with the physical system, we met the environmental requirements, designing a container which was both sturdy and waterproof. The container was also of reasonable size, and could easily be mounted on the car as a dashboard of sorts. All the buttons and switches requested by the Baja team were mounted in reasonable locations, and all of the internal elements were able to fit within the container. The container could close securely during the race, but the interior could also be accessed easily after the race, courtesy of the locking mechanism on the box. The physical container met all of the designated design requirements.

The power system also met all of our design requirements. The two-cell lithium ion battery was able to provide sufficient voltage over an 8-10 hour timeframe, which successfully covered our initial design requirements. Similarly, our voltage regulators met our design requirements, successfully creating 3.3V and 5.0V signals on the board to power all of the various subsystems.

We also managed to meet all of the data collection requirements. The GPS module was properly initialized and put into binary mode. From there, it was able to return all of the requested data to the microcontroller, most notably velocity, longitude, and latitude. These values were then converted into reasonable units, primarily miles per hour for the velocity value. Both of the Hall effect sensors also functioned fully as expected. As was demonstrated when tested on a bicycle, the Hall effect sensors could successfully read the magnet on the wheel and trigger the interrupt in the code when mounted properly. By triggering this interrupt, the Hall effect sensors could generate an accurate rpm for both the engine and the wheels, with some unit conversion. The lap counter also kept track of the lap time successfully, accurately counting up with each passing second.

Aside from beginning to print values to the wrong locations after several minutes, the LCD screen also met all of our design requirements. It was large and bright enough to be easily read during the bike test. While this does not completely confirm that the screen will be able to be read once implemented on the Baja car, it does provide strong evidence that it will be sufficient. The screen also was able to display the collected data, displaying accurate velocity and fix information from the GPS module, and accurate rpm information from both of the Hall effect sensors. While displaying the rpm information was not an original design requirement, we chose to display the information as a means to prove that both the screen and the Hall effect sensors were functioning properly, which was an essential design requirement. Finally, the lap time was displayed

successfully on the screen, showing the proper time, and correctly updating it every second, as was necessary.

The SD card issues are where we failed to meet design requirements. Without the ability to interface with the SD card, we were not able to meet several design requirements. First, and most obviously, we could not generate files on the SD card for the Baja team to analyze later. Secondly, the various modes which the system could be put into were ultimately useless without the SD card functioning properly, as the modes dealt primarily with the data collection rate to the SD card. While we were able to meet most of our other design requirements, the shortcomings of the SD card subsystem inhibited the overall efficacy of the system.

5 - Users Manual

5.1 How to Install

In order for this module to be installed, the mechanical engineering Baja team will be mounting it near the driver's seat in their Baja vehicle. It is important that vibrations are minimized for the device, and that it is kept as high as possible to prevent any puddles or mud from splashing onto it. Furthermore, the rpm sensors need to be mounted near the wheel and engine of the Baja vehicle. On the vehicle, the team will have to place magnets where the engine and wheels revolve so that the magnets can trip the rpm sensors each revolution. Furthermore, the battery for the device should be charged overnight to minimize the risk of losing data due to a power failure.

5.2 How to Setup

In its current state there is not much setup involved with the module. As long as the battery is powered, the device must simply be turned on with one of its two on/off switches. Because of the weird issue with the program not fully resetting, the user currently has to press the reset button to get everything to display correctly. In order for the velocity to display, the user must go outside and wait for the screen to indicate that a GPS fix has been found. Once this has been found, all of the functions (seconds counter, rps, and velocity) should be working properly.

5.3 How to Tell if Product is Working

If the product is working, the screen will indicate that the GPS has found a fix. If this is not the case, open the device and look for the green LED indicator on the GPS module (mounted on the top of the box). The LED is on the opposite side so it might be best to un-velcro the module to check the light. If the LED light is flashing, no fix has been found, and it probably indicates that the location in which the user is trying to use the device does not have adequate GPS coverage. If, however, the light is off, that indicates that the GPS *is* receiving a fix, but there must be a software issue causing the screen to not recognize that a fix has been found.

Another thing that could happen is that the battery could run out. Usually the first indicator of a low battery is that the LCD screen gets more faint. This occurs because the LCD runs on 5V. If this occurs, turn the module off, unplug the battery, and plug it into the charger for a couple hours.

6 - To-Market Design Changes

Most of the sub-components are functioning as our design team originally planned. However, a few more improvements are needed for this to be a market-ready product. The list below briefly summarizes the possible improvements, followed by a detailed description.

The prototype version of the electrical system contains a LCD screen displaying the key data values. However, the electrical system must have capability not only to display an instant key data values, but also to collect the data and save it into the SD card as the mechanical engineering Baja team plans to use the data to estimate the CVT ratio to enhance its maximum driving performance. As mentioned earlier, the current version of microcontroller board has an irregular voltage supply issue, which determines the clock signal on the SPI. Before introducing the electrical system to the market, the SD card issue will be resolved so that the future users can collect and save the key data, including GPS information and rotations per second, into the SD card for the vehicle performance analysis.

In the more distant future, another option that would solve the storage problem is to transition to flash memory. With flash memory, the Baja team would not have to deal with the complicated MDD filesystem library, and they would also not have to open the device and access the board to get data. With a cord for outputting data saved in flash memory, the board would be much safer, as hands reaching in to grab the SD card all the time could cause issues.

The electrical system currently includes four buttons placed on the surface of the system container. In order for the module to be ready for the Baja team, these buttons need to be properly integrated. With the code we have already developed working on its own, button integration to the system should not be a big challenge.

Another change that might be beneficial is the location of the cords for the buttons and sensors. If we could go back in time, we would have planned out better where the board would be in the box and where the different buttons and switches plug into the board before implanting them in the box. Because we sort of planned as we went along with the container, some of the cords are cluttered because their corresponding sensor, button, or device, is not in an optimal location in the box.

One final idea for change would be to use a different screen. One issue that we had with this screen was that the cursor would not always end up exactly where we told it to go every time. Consequently, the screen becomes more and more cluttered with mistakes as time passes. The issue with our current screen is that it is unable to send data to the microcontroller in order to describe its state. It would be nice if before every print to the screen we would be able to check to see if the cursor is exactly where it is supposed to be. If we could find a screen with these capabilities, I think we would see much fewer issues with printing in the wrong location.

7 - Conclusion

The requirements of the electrical system are based on the request from the mechanical engineering Baja team. The electrical engineering design team followed a necessary process to meet these requirements for the Baja team to have a successful

result in this year's Baja car competition. Our completed prototype demonstrates most of the criteria suggested by the mechanical engineering Baja team. However, there are some sub-components which require further improvements to be fully functional and be combined into an overall electrical system. After this revision and additional development, the electrical system will be ready for Baja use.

The initial proposal of our design suggests the electrical system with a goal of receiving the GPS and rpm sensor data to be displayed on screen and be saved into the SD card for vehicle performance analysis. The final prototype of our electrical system meets most of the fundamental functionality requirements initially proposed by the mechanical engineering Baja team. The SD card data storage is currently not fully functioning due to the irregular voltage issues on the SPI clock, but our design team demonstrated a successful data storage using the developmental microcontroller board provided by Professor Schafer.

The additional suggestions are made in the Market Design Changes section to propose the possible changes in our electrical system to be a market-ready product. Our design team believes that the potential improvements with buttons, better data storage, better space management and different screen types gives us a lot of room for continuous improvement of our product

Overall, the electrical system designed by 2014 electrical engineering "Team Baja" met most of the expectations set aside at the beginning of the project. With a few improvements and changes, our electrical device will be on its way toward helping the

mechanical engineering Baja team achieve its goal of top-10 finish in the Baja car competition this year.

8 - References

SD Card Spec Sheet:

https://www.sdcard.org/downloads/pls/simplified_specs/part1_410.pdf

Newhaven LCD Display Spec Sheet:

http://www.newhavendisplay.com/specs/NHD-0420D3Z-FL-GBW-V3.pdf

PIC32 Spec Sheet:

http://ww1.microchip.com/downloads/en/DeviceDoc/61156H.pdf

Hall Effect Sensor Spec Sheet:

http://sensing.honeywell.com/index.php?ci_id=50359

GPS Module Spec Sheet:

http://inmotion.pt/documentation/diydrones/MediaTek_MT3329/mediatek_3329.pdf

9 – Appendix

9.1 MDD File System Library Software

In addition to the following code which the Baja team developed, out project also included several header and source files that we downloaded from Microchip within the MDD File System library. Because these files are very long and are accessible on the internet, we have chosen to simply list these files rather than including the full code. The included software files from Microchip are as follows:

- ***HardwareProfile.h
- SD-SPI.h
- struct-queue.h
- TimeDelay.h
- GenericTypeDefs.h
- FSIO.h
- FSDefs.h
- FSconfig.h
- Compiler.h
- debug_ram_buffer.c
- ***FSIO.c
- ***SD-SPI.c
- TimeDelay.c

*****These files were slightly altered for use in our system. Their full documentation is posted on the Team Baja website under "Files"*

9.2 Main Baja Project Code, Demonstration2.c

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

                Microchip Memory Disk Drive File System


********************************************************************
*******
 FileName:        Demonstration1.c
 Dependencies:    FSIO.h
 Processor:       PIC32
 Compiler:        C32
 Company:         Microchip Technology, Inc.

 Software License Agreement

 The software supplied herewith by Microchip Technology Incorporated
 (the ìCompanyî) for its PICmicroÆ Microcontroller is intended and
 supplied to you, the Companyís customer, for use solely and
 exclusively on Microchip PICmicro Microcontroller products. The
 software is owned by the Company and/or its supplier, and is
 protected under applicable copyright laws. All rights are reserved.
 Any use in violation of the foregoing restrictions may subject the
 user to criminal sanctions under applicable laws, as well as to
 civil liability for the breach of the terms and conditions of this
 license.

 THIS SOFTWARE IS PROVIDED IN AN ìAS ISî CONDITION. NO WARRANTIES,
 WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED
 TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
 PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. THE COMPANY SHALL NOT,
 IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL OR
 CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.


   Note:  This file is included to give you a basic demonstration of
how the
           functions in this library work.  Prototypes for these
functions,
           along with more information about them, can be found in
FSIO.h
********************************************************************
*******/
```

```
//DOM-IGNORE-BEGIN
/*******************************************************************************
 Change History:
  Rev            Description
  ----           -----------------------
  1.3.0          Initial Revision
  1.3.4          Cleaned up the unnecessary part of main() function.
*******************************************************************************/
//DOM-IGNORE-END

/*******************************************************************************
**********
//NOTE : DISABLE MACRO "SUPPORT_LFN" IN "FSconfig.h" FILE TO WORK WITH
THIS DEMO
         EFFECTIVELY. DISABLING "SUPPORT_LFN" WILL SAVE LOT OF MEMORY
FOR THIS
         DEMO.
*******************************************************************************
**********/

#include "FSIO.h"
#include <xc.h>
#include <stdio.h>
#include <stdlib.h>
#include <plib.h>
#include <sys/attribs.h>

unsigned char step =0;
unsigned char ind = 0;
unsigned char complete=0;
unsigned char message[];
unsigned char ck_a=0;
unsigned char ck_b=0;
unsigned long mode;
unsigned long test;
unsigned char gps_file_created;
unsigned char test_file_created;
unsigned long numEg;  //counter for engine rpm
unsigned long numWh;  //counter for wheel rpm
int butpre1 = 0;  //variable to be set to 1 when button is pressed,
debouncing purposes
int butpre2 = 0;  //variable to be set to 1 when button is pressed,
debouncing purposes
unsigned char NOFIX[] = {'N','o',' ','F','i','x'};
unsigned char FIX[] = {' ',' ',' ','F','i','x'};
unsigned char RACE[] = {'R','a','c','e'};
unsigned char BUG[] = {'B','U','G'};
```

```
unsigned char TEST[] = {'T','e','s','t'};



#if defined (__PIC32MX__)
    #pragma config FPLLMUL  = MUL_20        // PLL Multiplier
    #pragma config FPLLIDIV = DIV_2         // PLL Input Divider
    #pragma config FPLLODIV = DIV_1         // PLL Output Divider
    #pragma config FPBDIV   = DIV_2         // Peripheral Clock
divisor                                 ****cahnged from DIV_2
    #pragma config FWDTEN   = OFF           // Watchdog Timer
    #pragma config WDTPS    = PS1           // Watchdog Timer
Postscale
    #pragma config FCKSM    = CSDCMD        // Clock Switching & Fail
Safe Clock Monitor
    #pragma config OSCIOFNC = OFF           // CLKO Enable
    #pragma config POSCMOD  = OFF            // Primary Oscillator
****changed from HS
    #pragma config IESO     = OFF           // Internal/External
Switch-over
    #pragma config FSOSCEN  = OFF           // Secondary Oscillator
Enable (KLO was off)
    #pragma config FNOSC    = FRCPLL        // Oscillator Selection
*****changed from PRIPLL
    #pragma config CP       = OFF           // Code Protect
    #pragma config BWP      = OFF           // Boot Flash Write
Protect
    #pragma config PWP      = OFF           // Program Flash Write
Protect
    #pragma config ICESEL   = ICS_PGx1      // ICE/ICD Comm Channel
Select                                  ****changed from ICS_PGx2
    #pragma config DEBUG    = ON            // Background Debugger
Enable
#endif

void serial_init1(unsigned long value)
{
    U3MODEbits.ON=1;
    U3MODEbits.BRGH=1;
    U3STAbits.UTXEN=1;
    U3STAbits.URXEN=1;
    unsigned long Fpb = 40000000;
    unsigned long num = Fpb/(4*value)-1;
    U3BRG = num;
}

void serial_init6(unsigned long value)
```

```
{
    //U6MODEbits.ON=1;
    //U6MODEbits.BRGH=1;
    //U6STAbits.UTXEN=1;
    //U6STAbits.URXEN=1;
    //unsigned long Fpb = 10000000;
    //unsigned long num2 = 4*Fpb/(4*value)-1;
    //U6BRG = num2;
}

unsigned char getu1(void)
{
    while(1)
    {
        if (U3STAbits.URXDA == 1)
        {
            char letter = U3RXREG;
            return letter;
        }
    }
}

unsigned char getu6(void)
{
    //while(1)
    //{
    //    if (U6STAbits.URXDA == 1)
    //    {
    //        char letter3 = U6RXREG;
    //        return letter3;
    //    }
    //}
}

void putu1(unsigned char letter4)
{
    while(1)
    {
        if (U3STAbits.UTXBF == 0)
        {
            U3TXREG = letter4;
            return;
        }
    }
}

void putu6(unsigned char letter2)
{
```

```
   // while(1)
   // {
   //     if (U6STAbits.UTXBF == 0)
    //    {
   //          U6TXREG = letter2;
    //         return;
   //     }
   // }
}

void enableInter(void)
{
    INTCONbits.MVEC=1;

    IEC0bits.INT1IE=0;
    IEC0bits.INT2IE=0;
    IEC0bits.INT3IE=0;
    IEC0bits.INT4IE=0;

    IFS0bits.INT1IF=0;
    IFS0bits.INT2IF=0;
    IFS0bits.INT3IF=0;
    IFS0bits.INT4IF=0;

    IPC1bits.INT1IP = 7;
    IPC2bits.INT2IP = 6;
    IPC3bits.INT3IP = 4;
    IPC4bits.INT4IP = 3;

    INTCONbits.INT1EP=1;
    INTCONbits.INT2EP=1;
    INTCONbits.INT3EP=1;
    INTCONbits.INT4EP=1;

    IEC0bits.INT1IE=1;
    IEC0bits.INT2IE=1;
    IEC0bits.INT3IE=1;
    IEC0bits.INT4IE=1;

    IEC1bits.U3RXIE=1;
    IFS0bits.INT3IF=0;
    U3STAbits.URXISEL1=0;
    IPC7bits.U3IP=111;
    U3STAbits.URXISEL0=0;
    IFS1bits.U3RXIF=0;
    asm("ei");                                  //clear interrupt flag
}
```

```
void spiInit()
{
   SPI4BRG = (40/(2*2))-1;

   TRISBbits.TRISB12=0;  //CS output
   TRISBbits.TRISB14=0; //CLK output
   TRISFbits.TRISF4=1;  //MISO input
   TRISFbits.TRISF5=0;  //MOSI output

   LATBbits.LATB12=1;    //CS high

   unsigned int left;
   left = SPI4BUF;

   SPI4STATCLR=0x40;
   SPI4CONbits.ON=1;
   SPI4CONbits.CKE=1;
   SPI4CONbits.CKP=0;
   SPI4CONbits.SMP=1;
   SPI4CONbits.MSTEN=1;
   SPI4CONbits.SRXISEL=00;
}

unsigned char swap(unsigned char info)
{
    SPI4BUF=info;
    //while(!SPI4STATbits.SPIRBF);
    while(!SPILCD_INT);
    return SPI4BUF;
    SPILCD_INT=0;
}

void sndCMD (unsigned char comd[], unsigned char reply[])
{
    int n; int i;


    //send zero
    swap(0xFF);
    //send six command bytes
    for (n=0;n<6;n++)
    {
        swap(comd[n]);
    }
    //wait until reply is heard


    do
```

```
    {
        reply[0] = swap(0xFF);
    } while (reply[0]==0xFF);

    //read reply
    for (i=1;i<6;i++)
    {
        reply[i] = swap(0xFF);
    }

    return;
}

void sdInit ()
{

    spiInit();
    int n;

    unsigned char comd[6];
    comd[0]=0x40;
    comd[1]=0x00;
    comd[2]=0x00;
    comd[3]=0x00;
    comd[4]=0x00;
    comd[5]=0x95;
    unsigned char reply[6];
    reply[0]=0x00;
    reply[1]=0x00;
    reply[2]=0x00;
    reply[3]=0x00;
    reply[4]=0x00;
    reply[5]=0x00;

    //wait
    for (n=0;n<25;n++)
    {
        swap(0xFF);
    }

    //set CS low
    LATBbits.LATB12=0;


    //wait
    for (n=0;n<25;n++)
    {
        swap(0xFF);
```

```
    }

    //send CMD0
    sndCMD(comd,reply);

    //send CMD8
    comd[0]=0x48;
    comd[1]=0x00;
    comd[2]=0x00;
    comd[3]=0x01;
    comd[4]=0xAA;
    comd[5]=0x87;
    sndCMD(comd,reply);

    do
    {
        //CMD55 inform SD of application specific command
        comd[0]=0x77;
        comd[1]=0x00;
        comd[2]=0x00;
        comd[3]=0x00;
        comd[4]=0x00;
        comd[5]=0x95;
        sndCMD(comd,reply);

        //ACMD41 to check if SD is out of idle state
        comd[0]=0x69;
        comd[1]=0x40;
        comd[2]=0x00;
        comd[3]=0x00;
        comd[4]=0x00;
        comd[5]=0x95;
        sndCMD(comd,reply);

    } while (reply[0]!=0x00);
}

void sdstuff ()
{
   FSFILE * pointer;
   char path[30];
   char count = 30;
   char * pointer2;
   SearchRec rec;
   unsigned char attributes;
   unsigned char size = 0, i;
   // Turn on the interrupts
//INTEnableSystemMultiVectoredInt();
```

```
//SYSTEMConfigPerformance(GetSystemClock());
//mOSCSetPBDIV(OSC_PB_DIV_2);
                //1) Initialize the RTCC
//RtccInit();
                //while(RtccGetClkStat()!=RTCC_CLK_ON);        // wait
for the SOSC to be actually running and RTCC to have its clock source
//DelayMs(1000);                                              // could
wait here at most 32ms
//RtccOpen(0x10073000, 0x07011602, 0);

                //2) Initialize SD Card
    sdInit();

   SPI4BRG = 9;

   //TRISBbits.TRISB10=0;     //WE
   //PORTBbits.RB10=1;        //WE
   //TRISBbits.TRISB11=1;     //CD
   //PORTBbits.RB11=1;        //CD

   TRISBbits.TRISB12=0;     //CS output
   TRISBbits.TRISB14=0;     //CLK output
   TRISFbits.TRISF4=1;      //MISO input
   TRISFbits.TRISF5=0;      //MOSI output

   SD_CS=1;

   unsigned int empty_buff;
   empty_buff = SPI4BUF;

   SPI4CONbits.CKP = 0;
   SPI4CONbits.CKE = 1;
   SPI4CONbits.SMP=1;
   SPI4CONbits.MSTEN=1;

   SPI4CONbits.ON=1;


   /**

    int n;

    unsigned char comd[6];
    comd[0]=0x40;
    comd[1]=0x00;
    comd[2]=0x00;
    comd[3]=0x00;
    comd[4]=0x00;
```

```
comd[5]=0x95;
unsigned char reply[6];
reply[0]=0x00;
reply[1]=0x00;
reply[2]=0x00;
reply[3]=0x00;
reply[4]=0x00;
reply[5]=0x00;

//wait
for (n=0;n<25;n++)
{
    swap(0xFF);
}


//set CS low
SD_CS=0;

//wait
for (n=0;n<25;n++)
{
    swap(0xFF);
}

//send CMD0

sndCMD(comd,reply);



//send CMD8
comd[0]=0x48;
comd[1]=0x00;
comd[2]=0x00;
comd[3]=0x01;
comd[4]=0xAA;
comd[5]=0x87;
sndCMD(comd,reply);

do
{
    //CMD55 inform SD of application specific command
    comd[0]=0x77;
    comd[1]=0x00;
    comd[2]=0x00;
    comd[3]=0x00;
    comd[4]=0x00;
```

```
        comd[5]=0x95;
        sndCMD(comd,reply);

        //ACMD41 to check if SD is out of idle state
        comd[0]=0x69;
        comd[1]=0x40;
        comd[2]=0x00;
        comd[3]=0x00;
        comd[4]=0x00;
        comd[5]=0x95;
        sndCMD(comd,reply);

    } while (reply[0]!=0x00);


                //3) Media Detect
                //while(!MDD_SDSPI_MediaDetect());                  //
Initialize the library
                //4) FSInit
//while (!FSInit());

    */
SD_WE_TRIS=0;
SD_WE=0x00;
}

void sd_ammend (char toSD[])
{
#ifdef ALLOW_WRITES
   char newline[]="\r";
   FSFILE * pointer;
   int length2 = strlen(toSD);
   pointer = FSfopen("GPSDATA.csv","a");
   FSfwrite(toSD,1,length2,pointer);
   FSfclose(pointer);
#endif
}

void sd_fammend (unsigned long tester)
{
#ifdef ALLOW_WRITES
   char newline[]="\r";
   FSFILE * pointer;
   pointer = FSfopen("GPSDATA.csv","a");
   FSfprintf(pointer,"%1d",tester);
   FSfclose(pointer);
#endif
}
```

```
void lap_ammend (char toSD[])
{
#ifdef ALLOW_WRITES
   char newline[]="\r";
   FSFILE * pointer;
   int length2 = strlen(toSD);
   pointer = FSfopen("LAPTIMES.csv","a");
   FSfwrite(toSD,1,length2,pointer);
   FSfclose(pointer);
#endif
}

void lap_fammend (unsigned long tester)
{
#ifdef ALLOW_WRITES
   char newline[]="\r";
   FSFILE * pointer;
   pointer = FSfopen("LAPTIMES.csv","a");
   FSfprintf(pointer,"%1d",tester);
   FSfclose(pointer);
#endif
}

void sd_try(float try)
{
   #ifdef ALLOW_WRITES
   FSFILE * pointer;
   pointer = FSfopen("GPSDATA.csv","a");
   FSfprintf(pointer,"%.3f",try);
   FSfclose(pointer);
#endif
}

void sd_format (unsigned long UTCtime)
{
#ifdef ALLOW_WRITES
   char newline[]="\r";
   char topline[]="Time (sec),Speed ((m/s)*10^2),Latitude
((deg)*10^6),Longitude ((deg)*10^6),Engine rpm,Wheel rpm";
   char lapline[]="Lap,Lap Time (sec)\r";
   FSFILE * pointer1;
   FSFILE * pointer2;

   int length1 = strlen(topline);
   int length2 = strlen(lapline);

   if (gps_file_created==0)
```

```
    {
        pointer1 = FSfopen("GPSDATA.csv","w");
        FSfwrite(topline,1,length1,pointer1);
        FSfclose(pointer1);

        pointer2 = FSfopen("LAPTIMES.csv","w");
        FSfwrite(lapline,1,length2,pointer2);
        FSfclose(pointer2);
        gps_file_created=1;
    }
    else
    {
        sd_ammend("New Data Set,UTC Time:,");
        sd_fammend(UTCtime);
        sd_ammend("\r");

        sd_ammend("New Data Set,UTC Time:,");
        sd_fammend(UTCtime);
        sd_ammend("\r");
    }
#endif
}

void __ISR(_UART_3_VECTOR, IPL7AUTO) u1interrupt(void)
{

    unsigned char disp1;
    disp1=getu1();                                  //get input from
UART1 buffer
    //putu6(disp1);
//send to UART6 for terminal
    IFS1bits.U3RXIF=0;                              //clear interrupt
flag



    switch (step)
    {
        case 0:

            if (disp1==0xD0)
            {
                step=1;
            }
            else
            {
                step=0;
            }
```

```
    break;
case 1:
    if (disp1==0xDD)
    {
        step=2;
    }
    else
    {
        step=0;
    }
    break;
case 2:
    if (disp1==0x20)
    {
        ck_a=disp1;
        ck_b=ck_a;
        ind=0;
        step=3;
    }
    else
    {
        step=0;
    }
    break;
case 3:
    message[ind]=disp1;
    ck_a=ck_a+disp1;
    ck_b=ck_b+ck_a;
    ind=ind+1;
    if (ind==32)
    {
        step=4;
    }
    break;
case 4:
    if (ck_a==disp1)
    {
        step=5;
    }
    else
    {
        step=0;
    }
    break;
case 5:
    if (ck_b==disp1)
    {
        complete=1;
```

```
            }
            step=0;
            break;
        default:
            step=0;
    }
}

void __ISR(_EXTERNAL_1_VECTOR, IPL7AUTO) INT1Interrupt(void)    //LAP
{
    //whatever the lap button does
    butpre1 = 1;
    IEC0bits.INT1IE=0;
    IFS0bits.INT1IF=0;
}

void __ISR(_EXTERNAL_2_VECTOR, IPL7AUTO) INT2Interrupt(void)    //DONE
{
    //whatever the stop button does
    butpre2 = 1;
    IEC0bits.INT2IE=0;
    IFS0bits.INT2IF=0;
    test=1;
}

void __ISR(_EXTERNAL_3_VECTOR, IPL7AUTO) INT3Interrupt(void) //ENGINE
RPM
{
    numEg=numEg+1;
    IFS0bits.INT3IF=0;
}

void __ISR(19, IPL7AUTO) INT4Interrupt(void) //WHEEL RPM
{
    numWh=numWh+1;
    IFS0bits.INT4IF=0;
}

void excel_setup (unsigned long fix, unsigned long date, unsigned long
UTCtime)
{
    unsigned long yr;
    unsigned long day;
    unsigned long month;

    day = date/10000;
    date = date - day*10000;
    month = date/100;
```

```
    date = date - month*100;
    yr = date;

    sd_format(UTCtime);
    if (fix==3)
    {
        sd_ammend(",");
        sd_fammend(month);
        sd_ammend("/");
        sd_fammend(day);
        sd_ammend("/");
        sd_fammend(yr);
        sd_ammend(",");
    }
    else
    {
        sd_ammend(",DATE UNAVAILABLE");
    }
    sd_ammend("\r");
}

void gps_setting (unsigned char setting[])
{
    int n=0;
    while(setting[n]!='\0')
    {
        putu1(setting[n]);
        n=n+1;
    }
}

unsigned long value(unsigned int index)
{
    unsigned long value1;
    value1 = message[index];
    value1 = (value1<<8)+message[index-1];
    value1 = (value1<<8)+message[index-2];
    value1 = (value1<<8)+message[index-3];
    return value1;
}

int main(int argc, char** argv)
{

    numWh = 0;
    numEg = 0;

    //initialize UART for GPS and corresponding interrupts
```

```
    TRISGbits.TRISG6=1;
    TRISGbits.TRISG8=0;
    TRISGbits.TRISG7=1;
    TRISGbits.TRISG9=0;
    TRISDbits.TRISD8=1;
    TRISDbits.TRISD9=1;
    TRISDbits.TRISD10=1;
    TRISDbits.TRISD11=1
    enableInter();

    serial_init1(38400);

    gps_setting("$PGCMD,16,0,0,0,0,0*6A\r\n");


    //initialize SD card and SPI
    LCD_init();
    //sdstuff();

    //Delayms(30);
//    int g=0;
//
//    SD_CS =0;
//    while(g<15)
//    {
//       swap('V');
//       g=g+1;
//    }
//    SD_CS =1;

    //initialize gps and set in binary mode


    //variable declarations
    unsigned long latit;
    unsigned long longit;
    unsigned long date;
    unsigned long speed;
    unsigned long fix;
    unsigned long t=0;
    unsigned long laptime=0;
    unsigned long laps=0;
    unsigned char formatted=0;
    unsigned long time_of_last_press;
    unsigned long butdec1 = 0;  //counter to debounce button
    unsigned long butdec2 = 0;  //counter to debounce button

    char PREVLAP[4];
```

```
char SPEED[3];
char testspd[3];
char DIG0[1];
char DIG1[1];
char DIG2[1];
char DIG3[1];
unsigned long UTCtime;
mode=1;
test=0;
gps_file_created=0;
unsigned long prevLAP=0;
unsigned long speed2 = 2000;
unsigned long dig0;
unsigned long dig1;
unsigned long dig2;
unsigned long dig3;
unsigned int fixset=0;
unsigned int nofixset=0;
unsigned int modeset=0;
unsigned int velocityset=0;



while(1)
{
    //snd_cmd(0x51,0x00);
    //string_snd(tester,3);
    if (butpre1 == 1)  //resets counter if button is pressed
    {
        butdec1 = 0;
        butpre1 = 0;
        test=2;
    }
    if (butpre2 == 1)  //resets other counter if button is pressed
    {
        butdec2 = 0;
        butpre2 = 0;
    }
    if(butdec1 == 30000)  //restarts the lap button interrupt
    {
        IFS0bits.INT1IF=0;
        IEC0bits.INT1IE=1;
        butdec1 = 0;
    }
    if(butdec2 == 30000)  //restarts the stop button interrupt
    {
        IFS0bits.INT2IF=0;
        IEC0bits.INT2IE=1;
```

```
            butdec2 = 0;
        }
        ++butdec1;
        ++butdec2;



        //lap button activated
        if ((test==2)&&(mode==1))
        {
            //snd_cmd(0x45,0x51);
            //string_snd(tester,3);

            prevLAP=t;
            t=0;

            sprintf(PREVLAP, "%ld", prevLAP);
            snd_cmd(0x45,0x5E);
            //Delayms(10);
            if(prevLAP < 10)
            {
               string_snd(PREVLAP,1);
            }
            else if(prevLAP < 100)
            {
                string_snd(PREVLAP,2);
            }
            else if(prevLAP<1000)
            {
                string_snd(PREVLAP,3);
            }
            else if(prevLAP>1000)
            {
                string_snd(PREVLAP,4);
            }
            else
            {
                string_snd(PREVLAP,0);
            }


//            laps++;
//            laptime=t-time_of_last_press;
//            time_of_last_press=t;
//
////            lap_fammend(laps);
////            lap_ammend(",");
////            lap_fammend(laptime);
```

```
////            lap_ammend("\r");
////            //change screen display

           test=0;
        }

        //test mode activated
        if (test==1)
        {
            //formatted=0;
            //display speed on screen

            snd_cmd(0x45,0x46);
            string_snd(TEST,4);
            //when lap is pressed
                //create file if first time
                //delay 100ms
                //calculate rpm
                //output time and rpm to file
        }

        //race mode activated
        if (test==0)
        {
                if (modeset==0)
                {
                    //Delayms(5);
                    snd_cmd(0x45,0x46);
                    //Delayms(10);
                    string_snd(RACE,4);
                    modeset=1;
                }

                if (complete==1)
                {

                    //output rps
                    char ERPS[3];
                    //Delayms(5);
                    sprintf(ERPS, "%ld", numEg);
                    snd_cmd(0x45,0x52);
                    //Delayms(10);
                    if(numEg < 10)
                    {
                       string_snd(ERPS,1);
                    }
                    else if((numEg < 100)&&(numEg>9))
                    {
```

```
    string_snd(ERPS,2);
}
else
{
    string_snd(ERPS,3);
}
numEg=0;

char RPS[3];
//Delayms(5);
sprintf(RPS, "%ld", numWh);
snd_cmd(0x45,0x12);
// Delayms(10);
if(numWh < 10)
{
   string_snd(RPS,1);
}
else if((numWh < 100)&&(numWh>9))
{
    string_snd(RPS,2);
}
else
{
    string_snd(RPS,3);
}
numWh=0;


//output time
t++;
//sendValue(t,0x1D);
char T[4];
//Delayms(5);
sprintf(T, "%ld", t);
snd_cmd(0x45,0x1D);
//Delayms(20);
if(t < 10)
{
   string_snd(T,1);
}
else if(t < 100)
{
    string_snd(T,2);
}
else if(t < 1000)
{
    string_snd(T,3);
}
```

```
else
{
    string_snd(T,4);
}


//extract values from message
speed = value(15);
latit = value(3);
longit = value(7);
date = value(25);
UTCtime = value(29);
fix = message[21];


if (fix!=3)
{
    if (nofixset==0)
    {
        //Delayms(5);
        snd_cmd(0x45,0x22);
        //Delayms(10);
        string_snd(NOFIX,6);
        nofixset=1;
        fixset=0;
    }
}
else if (fix==3)
{
    if (fixset==0)
    {
        //Delayms(5);
        snd_cmd(0x45,0x22);
        //Delayms(10);
        string_snd(FIX,6);
        fixset=1;
        nofixset=0;
    }
}

if (mode==0)
{
    snd_cmd(0x45,0x06);
    //Delayms(10);
    string_snd(BUG,3);
}

if (mode==1)
```

```
                    {

                        //snd_cmd(0x45,0x46);
                        //string_snd(RACE,6);

                        //if this is the first message received,
format the two excel sheets
//                          if (formatted==0)
//                          {
////                              excel_setup(fix,date,UTCtime);
//                            formatted=1;
//                          }

                        //output data if gps fix exists
                        if (fix==3)
                        {
                            //calc erpm and wrpm based on counters

//                            sd_fammend(t);
//                            sd_ammend(",");
//                            sd_fammend(speed);
//                            sd_ammend(",");
//                            sd_fammend(latit);
//                            sd_ammend(",");
//                            sd_fammend(longit);
//                            //sd_ammend(",");
//                            //sd_fammend(erpm);
//                            //sd_ammend(",");
//                            //sd_fammend(wrpm);
//                            sd_ammend("\r");

                            //screen displays speed and lap times
                            //
                            long speed3=speed;
                            speed3=speed3*224;

                            //sprintf(testspd,"%ld",speed3);
                            //Delayms(5);
                            snd_cmd(0x45,0x05);
                            //Delayms(10);
                            dig0=(speed3/100000)%10;
                            dig1=(speed3/10000)%10;
                            dig2=(speed3/1000)%10;
                            dig3=speed3%1000;
                            sprintf(DIG0,"%ld",dig0);
                            sprintf(DIG1,"%ld",dig1);
                            sprintf(DIG2,"%ld",dig2);
                            sprintf(DIG3,"%ld",dig3);
```

```
                        if (DIG0!=0)
                        {
                            string_snd(DIG0,1);
                        }
                        string_snd(DIG1,1);
                        LCD_snd('.');
                        string_snd(DIG2,1);
                        string_snd(DIG3,1);
                        //snd_cmd(0x45,SPEED[0]);
                        //snd_cmd(0x45,'.');
                        //snd_cmd(0x45,SPEED[1]);
                        //snd_cmd(0x45,SPEED[2])
                        velocityset=0;
                    }
                    else
                    {
                        if (velocityset==0);
                        {
                            //Delayms(5);
                            snd_cmd(0x45,0x05);
                            //Delayms(10);
                            dig0=0;
                            dig1=0;
                            dig2=0;
                            dig3=0;
                            sprintf(DIG0,"%ld",dig0);
                            sprintf(DIG1,"%ld",dig1);
                            sprintf(DIG2,"%ld",dig2);
                            sprintf(DIG3,"%ld",dig3);
                            string_snd(DIG0,1);
                            string_snd(DIG1,1);
                            LCD_snd('.');
                            string_snd(DIG2,1);
                            string_snd(DIG3,1);
    //                        sd_fammend(t);
    //                        sd_ammend(",");
    //                        sd_ammend("NO GPS FIXXX0");
    //                        sd_ammend("\r");
                        }
                        velocityset=1;
                        //screen displays zero for speed and
normal lap time
                    }

                    complete=0;
                }
            }
```

```
        }
    }
    //return (EXIT_SUCCESS);
}
```

9.3 LCD Code, newmain.c

```
/*
 * File:   newmain.c
 * Author: mmanno
 *
 * Created on September 30, 2013, 7:23 PM
 */

#include <xc.h>
#include <stdio.h>
#include <stdlib.h>

#define SPIBRG              SPI3BRG
#define SPIBUF              SPI3BUF
#define SPICLOCK            TRISDbits.TRISD1
#define SPIOUT              TRISDbits.TRISD3
#define CS_TRIS             TRISDbits.TRISD4
#define SPIIN               TRISDbits.TRISD2
#define CS                  LATDbits.LATD4
#define SPILCD_INT          IFS0bits.SPI3RXIF

unsigned char Velocity[] = {'V','e','l',':'};
unsigned char Mode[] = {'M','o','d','e',':'};
//unsigned char Test[] = {'T','e','s','t'};
//unsigned char Rest[] = {'R','e','s','t'};
//unsigned char Race[] = {'R','a','c','e'};
unsigned char Fix[] = {'F','i','x'};
unsigned char Lap[] = {'L','a','p'};
//unsigned char No[] = {'N','o'};
//unsigned char NA[] = {'N','/','A'};
//unsigned char Space[] = {' ',' '};
unsigned char Current_Lap[] = {'C','u','r','r','e','n','t',':'};
unsigned char Previous_Lap[] = {'P','r','e','v','i','o','u','s',':'};
unsigned char rps[] = {'R','P','S',':'};

void LCD_SPI_init()
{
    SPIBRG = 85;
//SCK to 59kHz
    //SPIOUT = 0;
//MOSI output
    //SPIIN = 1;
//MISO input
    //SPICLOCK = 0;
//CLK output
```

```
    CS_TRIS = 0;
//CS output

    int rData = SPI3BUF;

    SPI3CONbits.CKP = 1;
    SPI3CONbits.CKE = 0;
    SPI3CONbits.SMP=0;
    SPI3CONbits.MSTEN=1;

    SPI3CONbits.ON=1;

    SPILCD_INT=0;
    CS=1;
}



unsigned char LCD_snd(unsigned char info)
{
    CS=0;
    SPI3BUF = info;
    while(!SPI3STATbits.SPIRBF);
    return SPI3BUF;
    CS=1;
}

delay_millisec(int input)
{
    int i=0;
    while (i<input*1000)
    {
        i=i+1;
    }
}

string_snd(unsigned char info[],int length)
{
    int i;
    for(i=0;i<length;i++)
    {
       LCD_snd(info[i]);
       delay_millisec(1);
    }

    delay_millisec(10);
}
```

```
void snd_cmd(unsigned char cmd, unsigned char location)
{
 LCD_snd(0xFE);
 delay_millisec(8);
 LCD_snd(cmd);

 if(cmd == 0x51)
 {
 delay_millisec(4);
 }


 if(cmd == 0x45)
 {
 LCD_snd(location);

 }

 else
 {
 delay_millisec(4);
 }

}


void LCD_init()
{

//    int test = 0;
//    int mode = 0;
//    int fix = 3;
//
//    unsigned long num;
//    char Vel[10];
//    num = 42424242;
//    sprintf(Vel, "%ld", num);

    //initialize SPI
    LCD_SPI_init();
    delay_millisec(100);

    snd_cmd(0x51,0x00);
    string_snd(Velocity,4);

    snd_cmd(0x45,0x0D);
    string_snd(rps,4);
```

```
    snd_cmd(0x45,0x40);
    string_snd(Mode,5);

    snd_cmd(0x45,0x25);
    string_snd(Fix,3);

    snd_cmd(0x45,0x14);
    string_snd(Current_Lap,8);

    snd_cmd(0x45,0x54);
    string_snd(Previous_Lap,9);

//    snd_cmd(0x45,0x0A);
//    string_snd(Vel,2);  //Will eventually be velocity
//
//    if(test == 1){
//        snd_cmd(0x45,0x46);
//        string_snd(Test,4);
//    }
//
//    if(test == 0){
//        if(mode == 0){
//        snd_cmd(0x45,0x46);
//        string_snd(Rest,4);
//        }
//        if(mode == 1){
//        snd_cmd(0x45,0x46);
//        string_snd(Race,4);
//        }
//    }
//
//
//
//    if(fix == 3){
//        snd_cmd(0x45,0x4E);
//        string_snd(Space,2);
//    }
//    if(fix != 3){
//        snd_cmd(0x45,0x4E);
//        string_snd(No,2);
//    }
//
//    snd_cmd(0x45,0x1D);
//    string_snd(NA,3); // Will eventually be current lap time
//
//    snd_cmd(0x45,0x5E);
//    string_snd(NA,3);  //Will eventually be previous lap time
}
```