DragAid-MK
Crash Detection Device
For Race Cars

Project Report

A Major Qualifying Project
submitted to the faculty of
WORCESTER POLYTECHNIC INSTITUTE
this 30th of April, 2009
in partial fulfillment of the requirements for the
Degree of Bachelor of Science

Submitted by:

Allison Smyth

Advised by:

Professor Stephen John Bitar

Table of Contents

Table of Figures

Table of Tables

1 Introduction

Since the foundation of drag racing and motorsports events in general, it has been the goal of the sanctioning bodies, drivers, and racing crews to create the fastest moving vehicle that can be operated as safely as possible. Different types of racing events feature different types of cars, which each have different vulnerabilities and strengths in terms of safety and performance. While this is true, it can be recognized that all racing vehicles have two similarities that are also vulnerabilities. The first susceptibility is their ignition or electrical system, and the second is their fuel. The combination of these two systems leads to potential conditions for fire and explosion.

Over the past few years, efforts have been made, especially in the professional categories of drag racing, to reduce the number of injuries and fatalities sustained in the sport. Last year, Ford Blue Boxes were mounted in all cars of the professional classes in order to monitor the g-force levels that the car and driver sustain during a race. Enclosed professional vehicles (such as the National Hot Rod Association – NHRA – funny car class) also incorporate a Halon system, which is intended to quickly extinguish fires from the cockpit of the race vehicle.

After the most recent tragedy in the NHRA funny car class, the Electrimotion Safety Shutoff Controller was also mandated in professional categories. This controller senses a manifold burst panel failure and simultaneously activates the fuel shut off, shuts off ignition, and deploys parachutes of the vehicle. While these efforts to create safer race cars have been good initial steps to making the sport safer, more work remains to be done. The area where the most vulnerability currently lies is in the area of sportsmen drag racing.

In general, there are two divisions in drag racing: sportsmen and professionals. The professionals are those seen at televised events and include the well known drivers of the sport. The sportsmen are those individuals who race for fun on the weekends at their local track or at NHRA and IHRA sanctioned events. While rules exist to ensure the safety of sportsmen drivers, very little has been done in terms of electrical safety devices to aid in their protection. Although these racers do not reach the same speeds as professional drivers, their cars are still fast enough

to warrant the use of crash detection and fire prevention devices. Currently, no such devices are mandated in the sportsmen categories.

For this reason, it was decided that a crash detection system, which combines the better features of systems used by professional racers with features that are necessary for sportsmen, would be a very beneficial product. Instead of focusing on detecting part failures, which is the leading cause of accidents in professional categories, this device would focus on accident detection and fire prevention, which are the two leading causes of injury and death in sportsmen racing. In general, it aims to protect the driver in accidents involving two cars striking one another, a car striking the wall of the race track, or a car rolling over during a race. It is hoped that the added safety of such a device will reduce the number of injuries in sportsmen racing and also reduce the number of injuries to safety personnel who risk their lives trying to save racers who have been in accidents.

In this way, the design for the DragAid-MK crash detection system was first created. The DragAid-MK is a data analyzer and master kill switch for race cars. It is designed to monitor and record the g-force levels that a race car sustains throughout the course of a race. In this way, it is similar to a Ford Blue Box. While it is recording data, the device is also comparing the g-force levels received to pre-specified g-force safety thresholds. When the g-force levels exceed the safety thresholds, the DragAid-MK will automatically shut off the ignition system and fuel pump of the vehicle (as does the new Electrimotion device for professional categories). These actions will eliminate the sources of spark and fuel that could lead to a fire in the race vehicle.

Although the DragAid-MK is designed for use in the sportsmen categories of drag racing, its unique design from other crash detection systems would also make it a valuable addition to the safety systems of professional drag racing and other types of racing teams as well. It would most likely be beneficial in NASCAR and formula one racing where collisions with the wall and roll-overs often incapacitate drivers and prevent them from properly shutting down their vehicles. Once it is proved in the racing industry, it could also be adapted to passenger vehicles for daily use. Although driver incapacitation is less likely in passenger car collisions, the risk of fire is still great. Often drivers do not realize the damage sustained by their vehicle following a collision, which results in them leaving their car idling or potentially continuing to drive. This

behavior makes fires more likely. A safety device such as the DragAid-MK would prevent this kind of post-accident behavior.

As of right now, the DragAid-MK is a completely functional data acquisition system and g-force monitor. It has the capability to turn off the ignition and fuel system of a vehicle when g-forces exceed thresholds set by the user. Although all functionality requirements set forth for the DragAid-MK during this project were met, there remains to be significant work before this device is suitable to be sold in the racing industry. Most improvements involve making the product easy to use for racers. These adjustments will be easy to make, since the basic functionality of the device has already been proven. After these enhancements have been made, the device should also be thoroughly tested on an actual race vehicle. This type of testing is not possible until late spring or early summer when race tracks reopen for the year.

Overall, through research it was found that a crash detection device such as the DragAid-MK would be extremely beneficial to race car drivers in all divisions of motorsports. Although the current device is targeted toward sportsmen racers of drag racing, it could easily be adapted to almost any type of racing or even passenger vehicles. As of right now, the DragAid-MK is completely operational. All project goals were met, and the device could technically be mounted on a race car and function as intended. In the future, work will be conducted in order to make the DragAid-MK PC software more user-friendly such that racers will be more willing to use the DragAid-MK to enhance their safety.

2 Prior Art and General Rules

Before designing the DragAid-MK, research was conducted in order to determine if products that involved ideas similar to the DragAid-MK existed. It was known that a device exactly like the DragAid-MK or with the same purpose of the DragAid-MK was not available at the time; however, it was hoped that similar devices could be found in order to gain ideas for the construction of the DragAid-MK. From the prior art research, several devices were found that could be used as models for the DragAid-MK. These devices are as follows: fuel shut off systems in motorcycles and racing bikes, the Ford Blue Box, a fuel shut off system designed by an Australian inventor, air bag deployment systems, and the Electrimotion safety device. Research was also conducted regarding the rules of the two major sanctioning bodies of drag racing in order to determine if there would be any restrictions on an electronic safety device.

Before research was conducted on systems that would make suitable models for the DragAid-MK, the current operation of race car shutdown controllers was reviewed. As of right now, a sportsmen race car contains one master kill switch on the outside of the vehicle as well as a control panel within the race car. The control panel within the car contains individual switches for each device electrically powered in the race car. Figure 1 shows a diagram of a standard control panel.

Figure 1: Race Car Control Panel[1]

The kill switch on the outside of the vehicle is not accessible to the driver within the race car. It is meant for use by the starting line crew who can shut off the car if they detect a problem with the vehicle before the race begins. Although this is a useful safety device, it does not aid a racer if a problem occurs once the race has begun.

[1] http://www.moroso.com/catalog/categorydisplay.asp?catcode=77191

The control panel on the inside of the car contains the same ability as the master kill switch; however, as can be seen in Figure 1, there are often several switches that must be pressed to turn off the entire car. During an accident, a racer may be incapacitated and therefore unable to shut off all items on the control panel. In the excitement of an accident, it is also possible for a racer to forget or be unable to press the switches that need to be toggled to turn off the vehicle. For these reasons, an electronically controlled master kill switch such as the DragAid-MK, which could automatically control the shut off of all sources of spark and liquid ejection within the vehicle, would be a beneficial improvement to the current control systems in race cars.

The first item researched in regard to electronic safety systems available on the market was the fuel shut off systems in motorcycles. This was suggested for study by Frank, one of the co-developers in the project, who owns a Suzuki motorcycle. He stated that in bikes designed for high speed racing and cornering there are often sensors to detect if the rider has fallen off the bike. When the system detects an accident it shuts off the bike's fuel system so that it will eventually run itself out of fuel and shutoff. While this safety feature is relatively well known, it was very difficult to find information regarding its actual operation.

Through various articles and motorcycle forums, it was eventually found that the product operated through the use of a tilt sensor, which is mounted in the front headlight of most bikes.[2] When the sensor detects that the bike is at a certain angle from the ground, it assumes the driver has fallen from the vehicle and shuts down the fuel injection system. For motorcycles, a tilt sensor is a very proper choice for accident detection, due to the fact that when the rider falls, the motorcycle will always tip to one side or the other. While this device works well for motorcycle crash detection, it would be more difficult to adapt to a race car. Although it would provide detection for roll-over accidents, it would not detect accidents that involved collisions between two cars or with a wall. For this reason, the tilt sensor method of accident detection was not chosen for implementation of the DragAid-MK.

[2] http://www.fireblades.org/forums/honda-rc51/48982-tilt-sensor.html

Research on safety in racing motorcycles led to increased research in the online forums of the National Hot Rod Association (NHRA). The NHRA is the largest sanctioning body in drag racing; therefore, it was believed that they would provide the most information on safety devices for race cars. From this forum, information on the Ford Blue Boxes, which are now mandatory in all professional categories, was obtained.

In March of 2007, Eric Medlen, a young professional driver of John Force Racing, was involved in a fatal accident at Gainesville Raceway Park in Gainesville, Florida. The accident was initiated by a tire of his vehicle going flat during practice. The equilibrium of the car was disturbed, and it resulted in severe tire shake. This shake caused Medlen's head to be pounded at enormous forces against the roll cage of his car. He died from irreversible brain damage. After this tragedy, John Force put Ford Blue Boxes in all of his vehicles and studied the information gathered from these devices after every round. He was determined to prevent an accident such as this one from occurring ever again.

Ford Blue Boxes were originally designed by the Ford Motor Company for use in the Champ Car World Series. While developing the Blue Boxes, Ford's racing division further created a Safety Research and Development group that is responsible for aiding teams in analyzing the data obtained from the Blue Boxes. A Ford Blue Box operates by collecting data through sensors actually in the Blue Box, but also by recording data from accelerometers that are placed in the ears of the driver. From this information, computer modeling is completed in order to determine the forces the driver and chassis are under during a race.[3]

When a crash occurs, raw data from the Blue Box are analyzed along with medical reports, photographs, and video. From this information, a CAE model of the driver inside the racecar cockpit is developed in order to recreate the accident through computer simulation. This research was used to predict responses and injury potentials to drivers in high g-force impacts. The results are then compared to the actual response and physical condition of the driver from the medical reports. Figure 2 shows a picture of the Ford Blue Box as well as a simulation conducted in the Funny Car Class of the NHRA.

[3] http://www.zercustoms.com/news/Ford-Blue-Box-On-All-2008-NHRA-Nitro-Cars.html

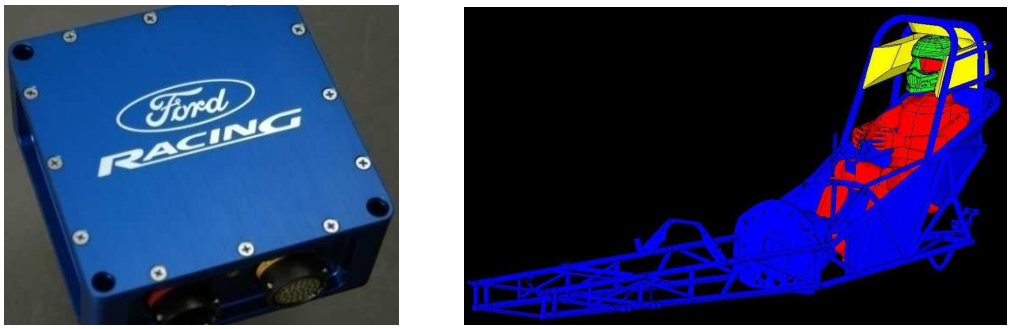


Figure 2: Ford Blue Box and Resulting Chassis Simulation[4]

Although the Ford Blue Box is a good example of a data acquisition system, it would not work as a crash detection system, which is what the DragAid-MK is being designed for. However, the Ford Blue Box did lead to further research into accelerometers, which were determined to be a very reliable method of accident detection.

Another device that was found through searching the forums of the NHRA was the new Electrimotion device invented for Top Fuel Funny Cars. When research was first conducted into the prior art of crash detection, the Electrimotion safety device did not exist. It was developed in early September of this year due to an accident that occurred in late June 2008 at Englishtown Raceway Park. During this event, Funny Car driver, Scott Kalitta's car exploded at the 1000 ft mark of the track causing the parachutes and body of the car to disintegrate. The Halon system was activated after the explosion, but since the body of the vehicle was no longer attached to the car, it was completely ineffective. It is assumed that Kalitta was knocked unconscious by the explosion for the car never slowed as it approached the end of the track. A final explosion when the car shot off the track is believed to have ended Kalitta's life.

The Electrimotion Safety Shutoff Controller was designed by Dave Leahy of Electrimotion, a company that specializes in electronic control systems. As stated earlier, the controller senses manifold burst panel failure and simultaneously activates the fuel shutoff, shuts off ignition, and deploys parachutes of the vehicle. The device was tested by several Nitro Funny Car teams and made mandatory by the NHRA within two races after its introduction. Although this device will aid in fire prevention in the professional nitro-methane classes, it will not aid sportsmen racers

[4] http://www.zercustoms.com/photos/Ford-Blue-Box-On-All-2008-NHRA-Nitro-Cars/Ford-Blue-Box-2008-NHRA-Nitro-Cars-1.jpg.html
http://www.myrideisme.com/Blog/ford-and-nhra-team-up-on-safety/

whose cars do not have a manifold burst panel. Therefore, the general idea of this safety controller is what is desired for sportsmen racers; however, its design will have to be altered to better suit the needs of a sportsmen vehicle. A diagram of the Electrimotion device has recently become available. This is currently the only figure available of the device. It can be seen in Figure 3.[5]

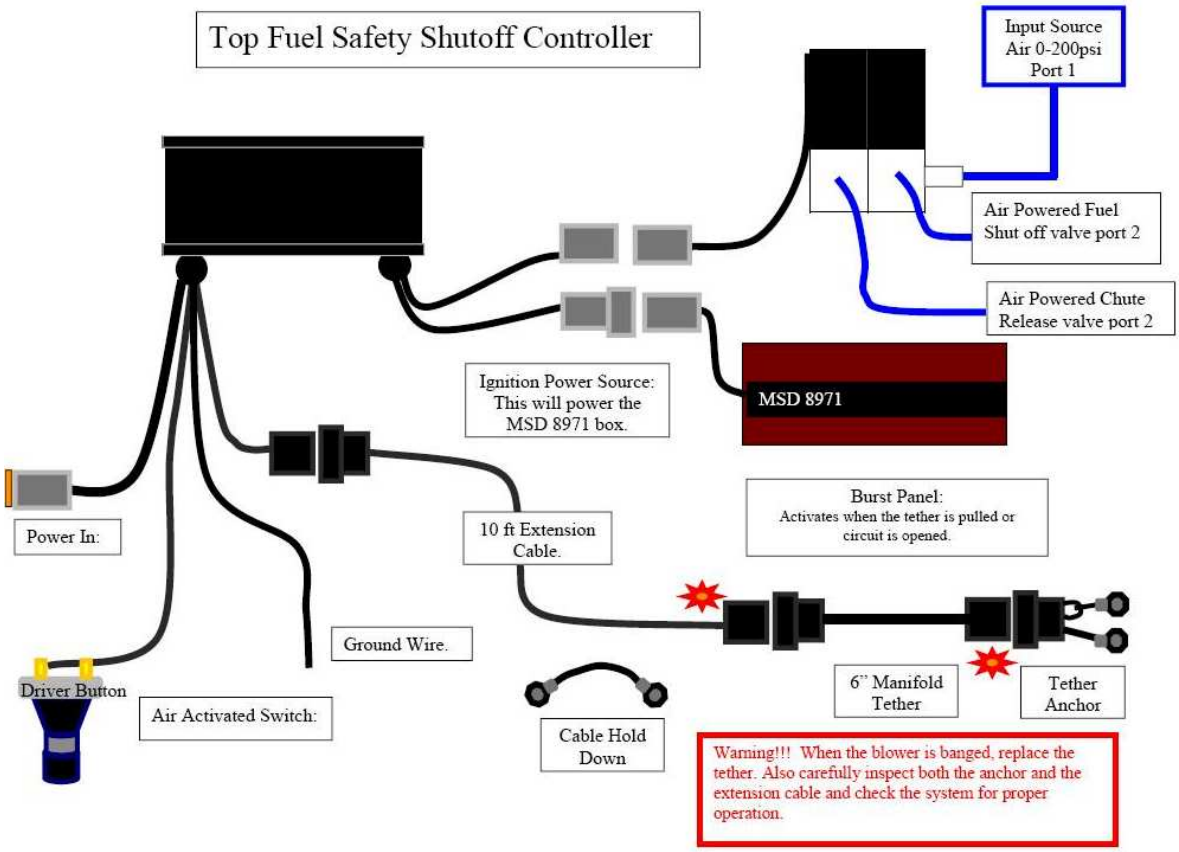


Figure 3: Electrimotion Safety Shutoff Controller[6]

After searching through the NHRA archives, research was also conducted outside the racing industry in order to determine if a device existed for passenger vehicles that would be a good model for the DragAid-MK. The first system found was the Vehicle Safety Shutdown System invented by John Quee of Australia.[7] This system is composed of two devices that are intended to reduce the risk of fire in a vehicle during an accident. Quee recognized that the two major causes of fire during an accident are the ignition system continuing to run and the fuel injection

[5] http://www.nhra.com/story/
[6] http://www.zoomerdaily.com/?tag=shutoff-controller
[7] http://www.carcrashfires.com/index.html

system continuing to pump fuel to the motor. He designed the Vehicle Safety Shutdown System to eliminate the risk from these two systems.

The Vehicle Safety Shutdown System created by Quee is composed of two devices, which are designed to stop fuel and spark from becoming united after an accident has occurred. The device is electromechanical in nature. Its diagnostic system is composed of accelerometers (electric sensors) which are designed to detect impact from any side of the vehicle. After an impact has been detected, the diagnostic system sends signals to the two mechanical pieces of the vehicle shutoff device.

The first mechanical device of the shutdown system is the fuel shutoff valve (FSV). The FSV is responsible for blocking the flow of the fuel from the fuel pump, zeroing the working pressure of the fuel system from the fuel pump to the engine, and converting the fuel system from high pressure to a vacuum, which then creates negative pressure. Therefore, if the pressurized fuel system is ruptured, the fuel will be drawn back rather than sprayed over a hot engine.

The second device of the Vehicle Safety Shutdown System is the battery isolation unit (BIU). This is responsible for cutting the supply of electricity to the vehicle's ignition system and severing the power supply (battery) from the entire vehicle. As of October 2008, Quee's Vehicle Safety Shutdown System has not been adapted to any vehicles manufactured in the United States or abroad; however, it appears to be a viable solution to preventing fires in automotive accidents. A diagram of his device is not available at this time since a patent has not been awarded to him, and he wishes to keep his design private.

The DragAid-MK will operate similarly to Quee's device; however, there are a few differences that will hopefully make it more reliable for race cars. The first change is that it will use electronic shutoffs rather than the mechanical shutoffs that are used in Quee's system. Although this will not create a vacuum in the fuel lines as does Quee's system, it has other advantages in terms of response time, which are necessary in a racing impact. Furthermore, the DragAid-MK will also be a data acquisition system, which will allow racers to adjust the shutoff thresholds to best meet the needs of their vehicle.

Another system that was reviewed was air bag deployment systems, which are now mandatory on all vehicles manufactured or sold in the United States. These systems currently use a combination of accelerometers and gyros in order to determine if the car has been in an accident. The gyros are used to detect vehicle rollovers. They sense change in tilt over time. The major manufacturer of accelerometers for car companies is Analog Devices, whose iMEMS technology is far advanced over the competition. These sensors are able to quickly detect and send signals to a controlling system after an accident has occurred. A general block diagram of a crash detection system that relies on Analog Devices' sensors can be seen in Figure 4.

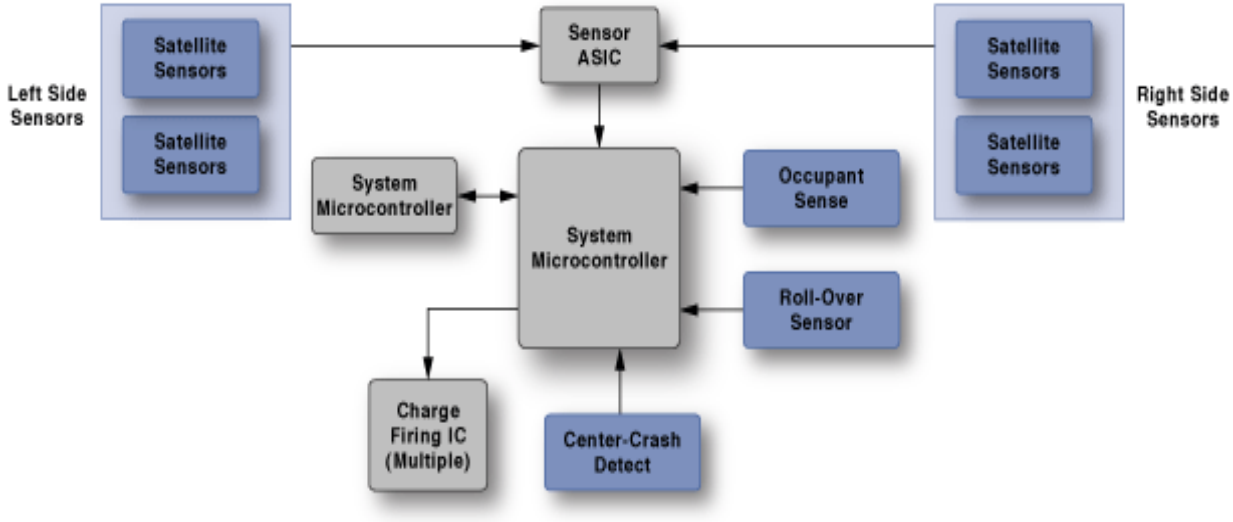


Figure 4: Analog Devices Crash Sensors[8]

Experiments are currently being conducted into new accident detection techniques for air bag deployment systems since these systems are extremely critical to time. One new method being investigated is detection of accidents through observing the metal flexing in the vehicle through magnetic plates.[9] This system is very promising for air bag systems, since it will give more time for air bag inflation, which will reduce injuries to occupants of the vehicle. This system of accident detection may not be as effective in race cars as in passenger vehicles for several reasons.

[8] http://www.analog.com/en/automotive-solutions/crash-detection/applications/index.html
[9] www.mathworks.com

The first reason is that this system of accident detection relies on detecting stresses or crushing of the chassis of the vehicle. Race cars, in general, undergo more stress than normal passenger vehicles. This would put them at risk of falsely triggering sensors that rely on chassis stress. For this reason, the new techniques of accident detection would not be appropriate for race cars. Furthermore, detection of this type would require drivers to completely redesign their cars in order to add the stress detection sensors. Racers tend to reject changes that appear too large or complex. They would especially be against a product that required major changes to their race car.

Although the new methods of air bag deployment are not appropriate for the DragAid-MK, the older methods of using accelerometers and gyros was very beneficial information to obtain. From this research, it was decided that Analog Devices accelerometers would be the most appropriate sensors to use. Research was also conducted into the gyros of Analog Devices; however, it was decided that these sensors would not be necessary for the DragAid-MK. It is believed that rollover detection can also be completed through information collected from the accelerometers.

After the prior art of accident detection was fully reviewed, the rules of the IHRA (International Hot Rod Association) and NHRA were observed in order to make sure that there were no restrictions present on electronic safety control systems. The rulebooks of both sanctioning bodies were found to be very similar. A summary of the two is as follows. The rulebooks were mostly concerned about cheating via electronic systems. They reject any electronic system that in some way connects to the track or the track timing system. All data recorders must be activated by a separate switch and cannot display information to the driver during a run. In order for a new device to be accepted it must be presented to the technical officials of both sanctioning bodies before an event. In general, both bodies state that they will allow a device that promotes safety with few questions as long as it can be proved to be fully functional.

In summary, research was conducted into many areas of accident detection and prevention. From this research, it was possible to show that an electronic safety shut off system for sportsmen race vehicles would be a very beneficial addition to the simple control panel that is

currently used. Further research unveiled several ideas that would be helpful in the design of such a system. As could be seen from the research, a device does not exist in racing or general automobile manufacturing that combines the features that will be included in the DragAid-MK, electronic shutoff device. For this reason, it is believed that the DragAid-MK will be a unique yet appropriate solution to the problems faced by sportsmen racers.

3 Objectives

In order to ensure that the DragAid-MK would meet the needs of sportsmen racers, an objective or goal list was created. The list was separated into two parts: general objectives and technical objectives. The sections below provide a summary of each of the objective lists.

3.1 General Objectives

The first objective created was that the DragAid-MK should be an electronically controlled master kill switch that also incorporates a data acquisition system. The device is intended to detect accidents that include vehicle roll-overs as well as side, front, or rear impacts. The data acquisition system of the device is intended to help racers determine appropriate g-force thresholds for proper accident detection in their vehicles. In order to review the data gathered through the acquisition system, the device should have a PC interface, which will allow the user to download the race data from the device.

The software for the DragAid-MK should be a user-friendly program that will easily allow a racer to adjust the shutoff thresholds for his car and then send them to the DragAid-MK. It is hoped that this program will eventually aid a driver by suggesting possible thresholds that would suit his vehicle. In the future, it will also prevent inappropriate thresholds from being selected.

The device should be easily testable by National Hot Rod Association (NHRA) and International Hot Rod Association (IHRA) technical officials. The device should provide a simple way for the NHRA and IHRA to verify its operation. This process must be quick and reliable because technical officials have limited time to review all race vehicles before the start of an event. It may be decided in the future to create simulation software for technical officials, which will also test the operation of the DragAid-MK.

A very important objective is to make the device adaptable to all types of cars. This includes all possible models of vehicles in the sportsmen categories (dragsters versus door cars), but it also means professional race cars as well as race vehicles outside of drag racing. This device was found to be unique in all types of racing; therefore, there is potential that it could be adapted beyond sportsmen categories of drag racing to other types of racing in the future.

The device should be manufactured with a durable case. This will protect the electronic circuitry and sensors from any type of accident that the vehicle may undergo. The device in general should be capable of shutting down the ignition system and all electronic systems of the vehicle when an unsafe condition is detected. It should also shut off liquid pumps including the fuel and water pump of the vehicle when an accident is detected. Although the DragAid-MK is designed to automatically shut down the vehicle it is mounted on, it should also have a manual control that will allow the driver to shut down all critical systems in case of an emergency.

The final specification that was identified for the master kill switch is that it is not a wireless device. All connections within the device and to the vehicle the device is controlling should be physical connections. Although wireless is very reliable, there are many sources of interference in a race car that could cause problems to a wireless system. Furthermore, wireless systems are more subjective to external tampering, which would be extremely detrimental in a device that controls the operation of a racecar. It is important that all signals sent to shut off the vehicle are sanctioned by the device, and not external signals sent by one racer as vengeance toward another. It may not be possible to prove that such tampering occurred within the device, which would lead to doubts as to the functionality of the device. This would undermine the racer's confidence in the system as well as potentially cause the racers to stop using the system.

The specification for no wireless in the system also includes data transfer to a PC. In general, racers are mechanical minded individuals. They do not understand electronics and prefer to find mechanical solutions to their problems when possible. For this reason, they often have trouble setting up electrical systems of any type. It is believed that setup of a wireless system would cause them more trouble than benefit. It would also be more difficult to troubleshoot problems that may occur if racers use wireless data transfer. With this in mind, wireless data transfer to a PC will not be included in this version of the DragAid-MK; however, in future versions, this idea may be considered.

3.2 Technical Objectives

The technical objectives provide more detail as to the operation and construction of the device. In general, they are an elaboration and usually quantitative description of the general objectives of the device described in the previous section.

The DragAid-MK should provide at least 5 auxiliary kill or shut off signals to the vehicle. It should also provide at least 1 master kill or master shut off signal to the vehicle. These lines should easily connect to the fuse panel or relay panel of the vehicle the DragAid-MK is mounted on, and lead to the shut off of the ignition system and fuel system of the vehicle.

The DragAid-MK should contain sufficient memory to record at least 3 runs worth of data. This is an improvement over the 1 run worth of data that it was originally expected to hold. The data the DragAid-MK collects is no longer required to have a time-stamp associated with it; however, the g-force information that it saves will be the maximum for each accelerometer axis over the .02 second interval that it monitors. It should be able to access and transfer this data to a PC via a USB connection.

Since the DragAid-MK is a peripheral device to a PC, it should contain a Type-B USB connection and be capable of interfacing with a PC using a standard Type-A to Type-B USB cable. High-speed communication is not necessary with the PC. The only requirement is that all data is transferred without error. USB was chosen since most racers carry laptops with them at a racetrack rather than a desktop computer. USB ports are more common on laptops than RS-232 serial communication ports; therefore, USB seemed to be the more suitable choice for PC communication.

The device should be able to run off a 9 volt battery or the 12 volt ignition system of the race car. As a user interface, it should include a series of switches (one of them being a master kill switch). By requirements of the NHRA and IHRA rulebooks, a master kill switch must be a positive action on/off switch. For this reason, momentary contact switches will be used for switches that select between different modes of operation of the DragAid-MK. A positive action

on/off switch will be used to initiate a master kill. The DragAid-MK should also contain several colored LEDs to inform the user of the operating mode the device is currently in.

The device should be capable of measuring the g-forces on the vehicle along the X, Y, and Z axes of the vehicle. Since side, frontal, and rear impacts will be the most severe (highest g-forces), the side, front, and rear accelerometers should be capable of detecting high g-forces (at least 50-g). The accelerometer used for measuring vehicle roll-over simply needs to detect -1-g of force; therefore, a low-g 3-axis accelerometer can be used for this operation.

An additional requirement that was added to the project is that the system should be triggered by the two-step or revolution limiter of the vehicle it is mounted on. This would mark a more accurate beginning of a race and allow the device to only record data that is relevant to the safety of the driver. It was also decided that the device should turn off completely (not only enter a low power mode) after the vehicle's ignition system has been shut down and the device has performed its shut down sequence. This will increase energy savings of the device, and also improve its operation on system startup.

A final requirement is that the technical verification system of the device be capable of testing the firmware of the system as well as the sensors. Both systems must be fully functional for the device to be completely operational.

4 Design Approach

The block diagram for the DragAid-MK is currently composed of 10 general modules with more specific blocks within each of these modules. The general modules that were identified for this project include a power supply module, a sensor module, a user input module, a signal conditioning module, a processing module, a memory module, a clock module, a programming module, an output module, and finally, a casing module. The block diagram can be seen in Figure 5 below. In this diagram, the dotted lines represent the general modules of the device and the more defined blocks within the modules represent the specific components that compose that module.

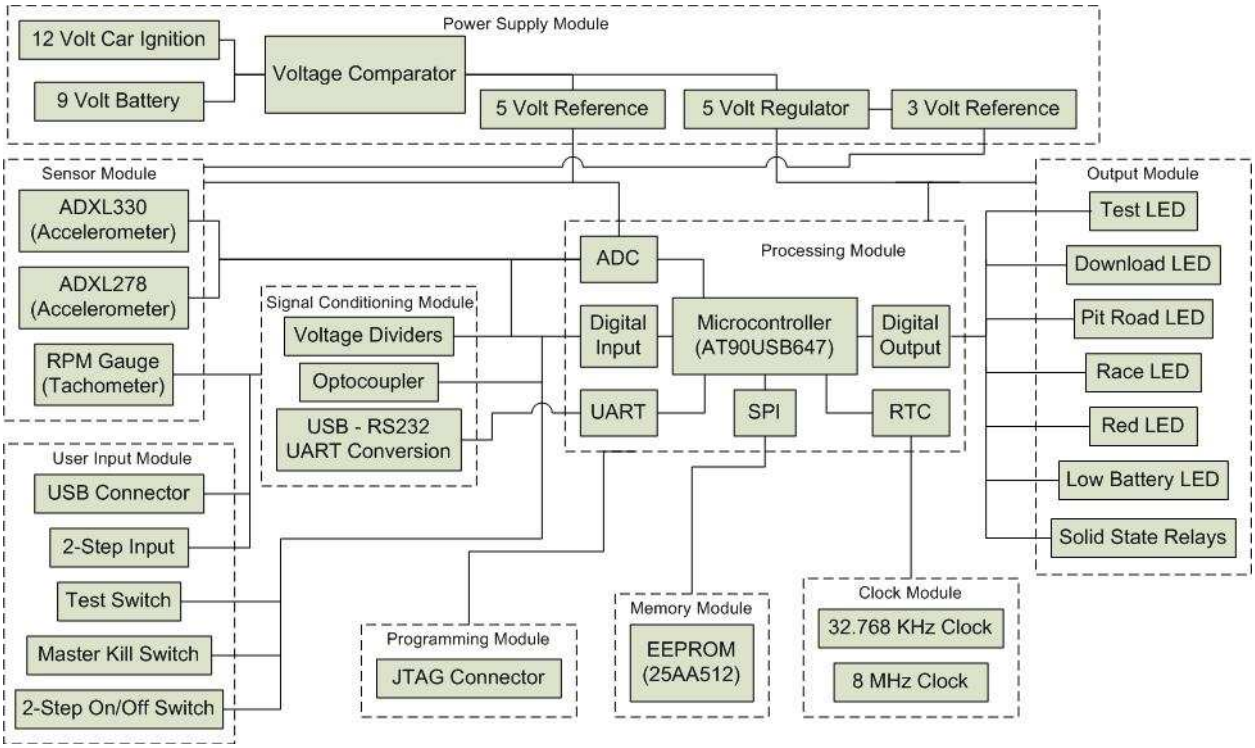


Figure 5: System Block Diagram

This section of the report will be broken into subsections based on each of the general modules in the block diagram of the DragAid-MK. Each subsection will aim to describe the purpose and function of the different modules in the DragAid-MK system. It will describe the input and outputs of each module as well as how each module interacts with the entire system.

4.1 Power Supply Module

The first module identified was the power supply module. This module is required to supply sufficient power to every module in the circuit that requires power to operate. The original

design for the power supply module only involved the use of voltage regulators to monitor the voltage across certain elements in the circuit. When the design of the system was observed more closely, it was decided that certain components such as the sensors and analog to digital converter should be supplied a more exact voltage. For this reason, the modified power supply module was constructed using a 5V voltage regulator, a 5V voltage reference, and a 3V voltage reference.

The processing module, memory module, output module, signal conditioning module, and programming module all receive 5 volts from the voltage regulator. The voltage supplied to these modules does not need to be exactly 5 volts, which is the reason that the voltage regulator is used to power these modules. The sensor module receives 5 volts and 3 volts from the two voltage references of the power supply module. Voltage references are specifically designed to provide more precise voltages to their load. They are not designed to handle as large of a load as voltage regulators; however, for the purpose of this project, they are appropriate. Two voltages are necessary for the sensor module, since one of the accelerometers requires a lower voltage to operate properly.

The voltage reference of the analog to digital converter within the processing module also requires 5 volts from the power supply's voltage reference. It is imperative that the analog to digital converter has a near exact voltage across it; otherwise, the base value for the conversions will fluctuate. This will cause the accuracy of the conversions to be questionable, which would undermine the entire operation of the DragAid-MK.

The input to the power supply module is received from two sources: the 12 volt car ignition and a 9 volt battery incorporated in the device. The power supply module is designed so that the input with the highest voltage is used to run the system. Therefore, when the vehicle ignition is on, the 9 volt battery will not be used. If the vehicle is turned off suddenly, the 9 volt battery will power the system and be used to complete shut down procedures. The voltage comparator was created using a series of diodes. It will be explained in more detail in Section 5, System Construction. Once the two voltages are compared, the higher voltage source is fed into the voltage regulators and references, which are used to power the remainder of the circuit.

4.2 Input Module

The input module is required to handle all user interaction with the device. In other words, it is the user interface of the DragAid-MK. It consists of two momentary contact switches that are used for switching the DragAid-MK into test mode or for turning off the input for the two-step. The two-step is a common device in race cars that is used as a motor revolution limiter. When the racer is in the start position for a race, he pushes down on some type of activator for the two-step. He then proceeds to press the throttle pedal of the race car to the floor. The engine will then limit itself to a preset revolution per minute (RPM) value that is controlled by the two-step. When the racer is ready to leave the starting line, he releases the two-step activator. The outputs of both switches are active low and will be fed directly into the processing module of the circuit.

The input module will also consist of a connector designed to allow a racer to easily connect his vehicle's two-step to the DragAid-MK. This will allow the DragAid-MK to monitor activation and deactivation of the two-step. It can use this information in order to decide when the car is actually being raced versus when it is simply warming up or preparing for a race. In this way, it can be used as a signal for a start of race, and therefore, a beginning point for recording the data received by the sensor module. The two-step input can be either active low or high (this will be a setup choice for the user). The signal received by the input module for the two-step will be fed into the signal conditioning module of the circuit before passing into the processing module.

The final two inputs to the input module are from the USB connector and the master kill switch. The USB connector is used for transferring data between the DragAid-MK and a PC. The signals received or sent by the USB connector are processed by the signal conditioning module of the DragAid-MK.

The master kill switch input is a positive action on/off switch as specified by the IHRA and NHRA rule books. It will be used to manually turn off the ignition and fuel pumps of the vehicle if the driver senses a dangerous situation. The output from this switch is also active low and is fed directly into the processing module of the circuit.

4.3 Sensor Module

The next module that was created was the sensor module. This module is responsible for detecting the different conditions that the vehicle being monitored by the DragAid-MK is experiencing. It currently consists of two accelerometers (an ADXL278 and an ADXL330) as well as an input from a RPM gauge (tachometer). The input from the tachometer may be removed in the final revision of the board. It was included in this revision since the initial functional design of the DragAid-MK required a RPM gauge for changing between user modes. In this revision, changes are controlled by the two-step and buttons; however, the RPM input was kept in case it was deemed necessary in future modifications to the device.

The ADXL330 accelerometer requires 3 volts to operate. It is a 3-axis, low-g accelerometer used for measuring z-axis g-force on the DragAid-MK. It will be used for identifying vehicle roll-overs in the DragAid-MK circuit. The output signal from the ADXL330 is fed directly into the processing module of the circuit. It produces an output analog signal between 0 and 3 volts; therefore, it is not necessary for this signal to pass through a signal processing module.

The ADXL278 accelerometer requires 5 volts to operate properly. It is a 2-axis, high-g accelerometer used for measuring x and y-axis g-forces on the vehicle. It will be used for sensing frontal, rear, or side impacts on the DragAid-MK. It produces an output analog signal between 0 and 5 volts. It will also be able to pass directly into the processing module of the circuit without signal conditioning.

4.4 Signal Conditioning Module

The signal conditioning module is required to process all signals that cannot be fed directly into the processing module of the circuit. It is required to perform several different operations in order to ensure that all data passed to the processing module can be handled. For this reason, it is composed of several different components.

The first portion of the signal processing module consists of 3 voltage dividers, which are represented as one block in the block diagram shown in Figure 5. The voltage dividers are used to process signals that have voltages greater than 5 volts. This is done to prevent them from

damaging the processing module, which can only handle signals with a maximum value of 5 volts.

Two of the voltage dividers receive signals from the power supply module of the circuit. They receive input directly from the battery and the ignition input of the power supply module; therefore, the two signals received by the signal conditioning module are around 9 volts and 12 volts. Once these signals are reduced to 5 volts, they are passed to the processing module of the circuit. The final voltage divider is used to reduce the RPM input signal to 5 volts. All systems in a race car generally run off of 12 volts; therefore, the signal received from the tachometer of the vehicle will also be around 12 volts when inputted to this voltage divider.

The next portion of the signal conditioning module consists of an optocoupler. This portion of the signal conditioning module takes as input the two-step signal form the user input module. The signal from a vehicle's two-step will range from 0 to 12 volts, and have a high value when activated and low value otherwise. In order to isolate the two-step from the processing module of the circuit (which will need to process whether the two-step is activated or deactivated), an optocoupler was used. This device will limit the two-step voltage to 5 volts when the two-step is activated and 0 volts when the two-step is off. Furthermore, it will prevent current from flowing from the two-step to the processing module, which could potentially damage the processing module of the circuit. This is added protection in case the two-step is not connected properly to the DragAid-MK.

The final portion of the signal conditioning module is the USB to UART conversion chip. The input to this chip is received from the USB connector of the input module as well as from the UART (universal asynchronous receive transmit) portion of the processing module (which will be described in more detail later in the report). The chip converts the information received from the USB connector to a format understood by a UART interface and sends this data to the processing module of the circuit. It also is able to accept information in UART format and convert data to standard USB 1.1 format, which it then sends to the USB connector of the input module of the circuit. The signals sent to the processing module from the USB to UART

conversion chip are between 0 and 3 volts. Although this is low for inputs to the processing module, it is able to function properly; therefore, a voltage amplifier is not necessary.

4.5 Memory Module

The memory module was created as an aid to the processing module. It is used to store the data received by the processing module from the sensing module of the circuit. The memory module is required to input and output data to the processing module. It operates from 5 volts; therefore, signals sent between the memory module and the processing module do not require signal conditioning.

The chip used for the memory module is an external EEPROM manufactured by Microchip. It is capable of holding 64 Kbytes of data, which is equivalent to 65536 bytes. The EEPROM communicates to the processing module using a serial peripheral interface (SPI). This interface is synchronous, which makes it faster than an UART interface. The EEPROM is capable of transferring data at a rate of 20 MHz. Although this speed is not possible with the current processor, it proves that data transfer between the processing module and the EEPROM will not be a source of system latency. The processing module should be able to store data as fast as it receives it.

4.6 Clock Module

The clock module of the circuit is required to supply all clock sources to the processing module of the circuit. Although the processing module has its own internal oscillators, these oscillators are known for being inaccurate and at times a-periodic. For this reason, it is better to run the system off of an external 8 MHz clock when accuracy in timing is desired. Although, instruction timing is not of critical importance for this circuit, it was found that using the external oscillator improved the UART communication. During testing of the initial board design, the UART receive was found to be missing a bit in transfer. When the 8 MHz clock was added to the board, the UART no longer dropped a bit, and all data was found to be received properly. This improvement may be due to the increase of speed to 8 MHz, or due to the greater accuracy of the external oscillator.

The second oscillator in the clock module of the circuit is a 32.768 kHz clock used for calculations in real-time. The frequency 32.768 kHz is generally used for real time clock calculations, since when divided by 256 in an 8-bit system, roll-over of the 8-bit register occurs every 1 second. This oscillator is extremely accurate, and will be used for timing in the processing circuit.

4.7 Programming Module

The programming module of the circuit consists of a JTAG header used for connecting a JTAG cable from a PC to the DragAid-MK board. This interface is currently used to program the firmware written in assembler to the DragAid-MK processing module. Since the SPI bus of the DragAid-MK's processing module is being used for storing and receiving data from the memory module, the JTAG interface will be used to program DragAid-MK processing modules even after the test phases of the project are complete.

Since its initial development, JTAG has frequently been used as an in-circuit debugger for embedded systems. The most essential benefit of a JTAG debug (or boundary scan) is that virtual breakpoints can be set. Although debugging an embedded system through JTAG appears to be similar to debugging PC software, this is not true. JTAG only has access to boundary values of the embedded system. This excludes all registers that are part of the internal core of the processor. For this reason, a JTAG debug is limited; however, it still provides more information than was available before the creation of the JTAG interface. It also allows embedded programmers to step through their code line by line, which is extremely helpful.

The JTAG interface will be used whenever a problem is found with the firmware created for the processing module. It will also be used to verify the correct logical operation of the code, which would be difficult to do without the ability to simulate system states using the debugger.

4.8 Processing Module

The processing module is the most important module of the DragAid-MK circuit. It is responsible for organizing all the data received by the sensor and input modules of the circuit and then providing the proper output signals to the output module. The processing module consists of an AT90USB647 microcontroller manufactured by Atmel Corporation. The main features of

the microcontroller that will be used in this project can be seen as the separate blocks within the processing module in Figure 5. They include the analog to digital converter (ADC), digital input and output ports, serial peripheral interface (SPI), universal asynchronous receive transmit (UART), and real time clock (RTC).

The ADC portion of the processing module will receive input from the accelerometers of the sensor module as well as the 9 volt battery and 12 volt ignition voltage dividers from the signal conditioning module. The signals from the accelerometers, once converted to digital signals, will be used to control whether the system should send off signals to the vehicle or continue operating normally. The conditioned input from the 9 volt battery and 12 volt ignition system are meant to determine if the battery life of the system is low or if the ignition system of the vehicle has been shut down.

The digital input and output ports of the system are connected to many modules of the entire system. They are used to allow a user to interface with the DragAid-MK. The main function of the digital input ports of the processing module are to take in user input. These ports are used to determine whether the user wishes to put the device in test mode, bypass the input signal from the two-step, start racing, or shut down the entire vehicle. The digital output ports are used for two main purposes. First they provide signals to the output module of the circuit, allowing the current mode of the DragAid-MK to be displayed to the user. They are also responsible for sending shut down signals to the solid state relays of the vehicle the device is mounted on.

The SPI portion of the processing module is responsible for coordinating data transfers with the memory module of the device. This is where the digital data that result from the analog to digital conversion are sent after they have been processed. They are transferred to the memory module over the SPI bus where they are stored until a user wishes to review the data recorded during the run. When this occurs the data can be read and erased from the memory module using commands sent over the SPI bus.

Another key system of the processing module is the UART. This module is used to send the data collected during a run to the user. It is also used to receive important data from the user such as

threshold values for vehicle shut off, car information, and two-step polarity. Other values that will be transferred between the PC and DragAid-MK through the UART include the serial number, version number, and version date of the DragAid-MK as well as the date the thresholds were last modified.

The final individual module within the processing module is the RTC interface. The RTC port of the AT90USB647 is basically designed to allow asynchronous data input. This makes the port ideal for sensing a clock input, which may not run synchronously with the main clock of the processor. The RTC will be used as the timing base for all action occurring in the processing module. It will be used to set a time limit for the device being in race mode. Furthermore, it will be used to create delays in UART data sends.

4.9 Output Module

The final module displayed in the block diagram seen in Figure 5 at the beginning of the section is the output module. This module is responsible for informing the user of the state of the device. Furthermore, it is responsible for sending "off-signals" to the vehicle it is attached to. The output module consists of a series of LEDs, which each have a different meaning in regard to the system state.

The LEDs require 15 milliamps of current for operation. They are designed to be active low, which means a low signal from the processing module will activate them. They receive a signal to turn on and off from the processing module of the circuit. As of right now there are 6 LEDs on the DragAid-MK board. Their purposes are as follows: Test Mode LED, Download Mode LED, Pit Road Mode LED, Race LED, System Test LED, and Two-Step On/Off LED. Therefore, each LED will give the user information on the current state of the system.

The final portion of the output module is connections that will eventually attach to solid state relays of the vehicle the DragAid-MK is mounted on. These connections will receive signals from the digital output port of the processing module. They will be used to send vehicle shut down signals to the car controlled by the DragAid-MK. The exact layout for this portion of the output module was not considered as part of this project and is open for future investigation.

4.10 Casing Module

Another module, which cannot be seen in the block diagram, is the case module. This module is responsible for protecting the circuitry of the device in everyday use as well as in a crash situation. It is required to be sturdy so that if the device is dropped the circuitry will not be affected. Furthermore, it must have sufficient padding to eliminate the vibrations from the ordinary movement of the car while still allowing the accelerometers to pick up major events. This is basically an attempt to reduce noise from the accelerometer signals in a physical manner, since the type of the noise is physical in nature.

The current base design for the case module is a metal electrical box. It is hoped that a box of this type will reduce electrical and magnetic interference that may occur within a race car. Very little research has been done on the interference produced by race cars; therefore, we are not sure what to expect. However, we do not want our device to get faulty readings due to environmental noise. For this reason, a metal protection cage around the essential sensors and circuitry of the DragAid-MK seems to be a good solution. More work on the aesthetics of the case should be completed before the DragAid-MK goes into production.

5 System Construction

The creation of the final prototype for the DragAid-MK took place in 4 basic stages. The first stage involved deciding how the system should function and then creating software flowcharts and system block diagrams that represented this functionality. The second stage involved selecting components. This included selecting sensors, memory, and the microprocessor as well as resistors, capacitors, and diodes that would best suit the needs of the device. The third stage involved the schematic design, which included properly arranging the components in order to create a functioning device. The final stage was the actual construction of the system, which included the printed circuit board (PCB) layout and the soldering of the components to the PCB.

In this section of the report, the construction of the DragAid-MK will be described through observing the construction process of each of the individual modules described in Section 4. The only module that will not receive individual description is the signal conditioning module, which will be incorporated in the description of the input module and the sensor module of the circuit. Since the case module will not be completed as part of this project, this module will be omitted from this section of the report.

Additional subsections will be added to this portion of the report to discuss the design and creation of the firmware and software for the DragAid-MK. The final subsection will contain a description of the design of the printed circuit board. This will include information on slight modifications that should be made before the product is released as an actual safety device.

A complete schematic for the DragAid-MK can be viewed in Appendix L. A complete parts list for the final PCB can be seen in Appendix M.

5.1 Power Supply Circuit

The first module that was designed was the power supply module. Figures 6 and 7 are schematic representations of the power supply module. Figure 6 shows the 5 volt voltage regulator, the 5 volt voltage reference, as well as the two input power sources to the circuit. Figure 7 shows the 3 volt voltage reference of this module.

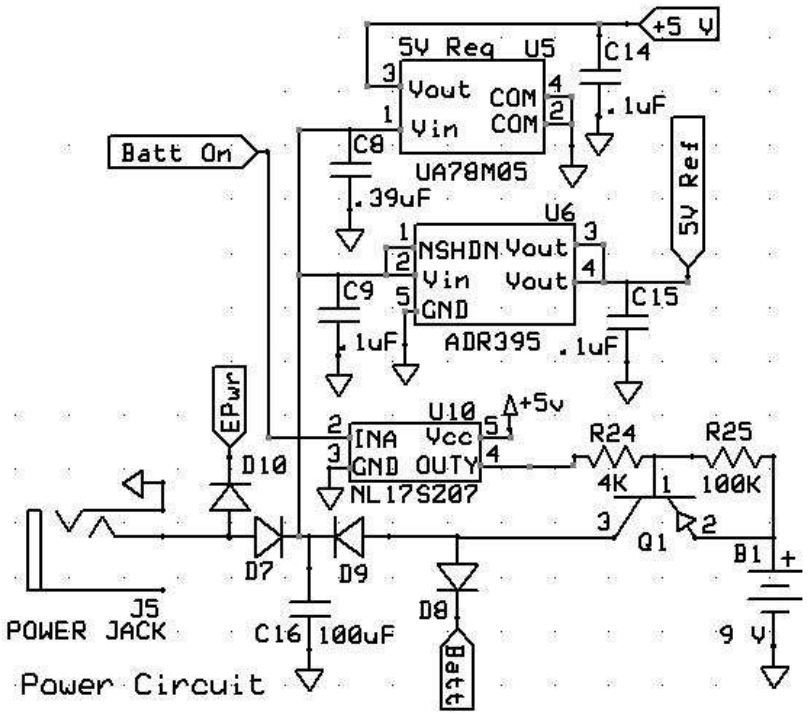


Figure 6: Power Supply Module – Power Input Circuitry

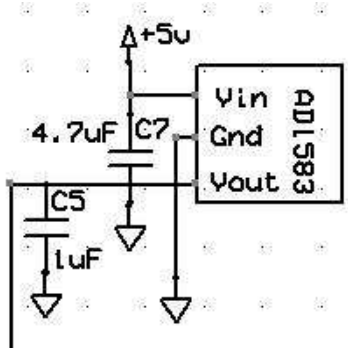


Figure 7: Power Supply Module - 3 Volt Reference

As can be seen from the figures, the power supply module receives input from the 12 volt ignition system of the vehicle (represented as the center-pin positive power jack in Figure 6) as well as from a 9 volt battery used as a backup power source (B1). This module is designed such that the 12 volt car ignition can power the DragAid-MK system while the car is running. If the vehicle is shut down, either by the operator or by the DragAid-MK, the device will continue to run from the 9 volt battery for a certain length of time. Once all final system operations are completed, the microprocessor of the circuit will have the ability to cut-off the battery from the DragAid-MK circuit; thereby, completely turning off the DragAid-MK system.

From the figures it can also be seen that the power supply module of the circuit is required to provide three outputs to the remainder of the circuit. The first output is a 5 volt power signal that is meant to run the portions of the circuit that are not voltage sensitive. The remaining two outputs are precision voltage signals of 3 and 5 volts that are meant to provide accurate reference voltages for the sensors and analog to digital converters in the circuit.

The power supply module was designed to handle 500 milliamps of current; however, in normal operating conditions, the current draw from the circuit should not exceed 100 milliamps. This calculation was made by observing the current draw of the key circuit elements in the DragAid-MK. The resulting calculation can be seen in Table 1 below.

Table 1: Table of Circuit Current Consumption

Device	Quantity Running Concurrently	Single Component Current (mA)	Total Current (mA)
LEDs	2	15	30
Microcontroller	1	18	18
EEPROM	1	10	10
ADXL278	1	2.2	2.2
ADXL330	1	0.18	0.18
CP2102	1	26	26
Total	-	-	**86.38 ≈ 100**

The power supply module has a complex design due to a unique requirement (mentioned briefly above) that it was designed to meet. Under normal conditions, when the vehicle is running properly and no crash has occurred, the DragAid-MK is expected to be powered by the ignition system of the vehicle. When a crash occurs, and the DragAid-MK shuts down the ignition system of the vehicle, the device must continue to run from batteries included on the DragAid-MK board. After the DragAid-MK has finished recording data, the device should then send a signal to the power circuitry in order to eliminate the input from the battery and thus completely shut down the system. Although it is possible for systems to have completely electrical shutdowns, it is not common for a system to attempt to shut off its own power. For this reason, many attempts were made at creating this circuit. The operation of the power supply circuit seen in Figure 6 operates as described below.

When power is provided by the ignition system of the vehicle, a 12 volt power signal is received at the D7, D9 diode junction. This signal is greater than the maximum 9 volt signal that could be supplied to this junction from the battery circuit. This will result in the D7 diode being in the forward biased condition and the D9 diode being in the reversed biased condition. Diodes only allow current to flow when they are forward biased; therefore, current will be drawn from the ignition system of the vehicle and not from the battery backup circuit. When the signal from the ignition system is removed, the diodes will be biased in the opposite directions allowing current to be drawn from the battery circuit.

After the signal from the power source passes through the diode junction, it enters into the UA78M05 five volt regulator and the ADR395 five volt reference. The 5 volt regulator is designed to take in a signal between 7 and 20 volts. It is guaranteed to output a signal between 4.8 and 5.2 volts. It is also able to handle a current draw of 500 milliamps, although it is unlikely that the DragAid-MK circuitry will draw this level of current. The voltage regulator will supply power to the microprocessor, EEPROM, and USB to UART conversion chip. These devices do not require precision voltage levels and only require power signals to operate. The power signal provided to the circuit from the 5 volt regulator is also passed into the 3 volt reference (see Figure 7).

The 5 volt reference requires an input voltage between 5.3 and 15 volts. It is guaranteed to output a voltage between 4.995 and 5.005 volts as well as supply 5 milliamps of current. The output from the 5 volt reference is used as a reference signal for the analog to digital converter of the microcontroller. This circuitry requires negligible current as stated in the microcontroller's user's guide. It is also used to supply power to the ADXL278 accelerometer, which relies on an accurate and precise power input in order to make accurate g-force readings. Since the ADXL278 only requires 2.2 milliamps for proper operation, the current draw from the voltage reference will not exceed the maximum allowable current.

The final 3 volt reference is used to supply power to the ADXL330 accelerometer, which requires a voltage less than 3.6 volts and nominally 3 volts to operate properly. As with the ADXL278 accelerometer, the ADXL330 requires an accurate and precise voltage input in order

to guarantee accurate sensor readings. For this reason, it was necessary that a 3 volt reference be used. The 3 volt reference requires a 5 volt input signal, which is why its input signal is supplied from the 5 volt regulator of the circuit. It will output a voltage signal between 2.997 and 3.003 volts. It also can provide up to 5 milliamps of current. This is sufficient since the ADXL330 only draws .18 milliamps.

The final components of the power supply circuit were added to support the battery cut-off feature described above. The main component used to achieve this function was a PNP transistor, which was operated as a switch to the battery in the circuit. This transistor was a 2SB0779 surface mount component manufactured by Discrete Semiconductor Products. It was chosen due to its ability to handle 500mA of current through the collector of the transistor. This is the maximum current that the power supply module was designed to withstand.

As can be seen in Figure 6, the emitter of the transistor is connected to the output of the battery and the collector is connected to the input of the DragAid-MK circuit. The base of the transistor is connected to the microprocessor through a base resistance as well as a NL17SZ07 open-drain, non-inverting buffer.

Since the transistor used in this circuit is a PNP type, a low signal on the base is necessary to turn it on, and a high signal is necessary to turn it off. When a low voltage signal is applied to the base of the transistor, the voltage from the emitter to the base is greater than .7 volts. As stated above, this turns on the transistor allowing current to flow through the transistor and power the DragAid-MK circuit.

When the base voltage of the transistor is set high, the transistor is turned off, thereby eliminating the battery source from the circuit. Since the microcontroller will not be able to sustain a high voltage once the battery has been removed from the circuit, a pull-up resistor is used to keep the transistor base tied to a high voltage (9 volts). A large pull-up resistor value is used in order to ensure that little current flows in the resulting loop that is created. Technically no current should flow since the transistor is cut-off. In order for this circuit to work properly, it was assumed that the microcontroller ports were open-drain. In other words, they are able to

properly pull a signal low; however, in the high-state the circuit has high impedance rather than an actual high voltage level.

Before this circuit was implemented on the printed circuit board of the project, it was setup and tested using a protoboard. It was found that the assumption that the microcontroller had open-drain outputs was false. In other words, the transistor would turn on when the microcontroller sent a low voltage level; however, it would not turn off when the microcontroller sent a high voltage level. In order to rectify this, an open drain buffer was obtained and placed between the base of the transistor and the output from the microcontroller. With this buffer in place, the circuit was found to work properly. The buffer used was a NL17SZ07 manufactured by ON semiconductor. It can be seen in the completed circuit of Figure 6.

The 100 micro Farad capacitor placed between diodes D7 and D9 is charged during the normal operation of the circuit. When power is completely removed from the circuit (the battery and ignition circuits are shut down), this capacitor discharges power to the circuit allowing the processor a few extra milliseconds of power to ensure complete shut down.

Another aspect of the circuit in Figure 6 that should be mentioned is the power lines EPwr and Batt, which leave the page. These signals are used to perform voltage checks on the 12 volt ignition and 9 volt battery circuit. The circuit that performs this can be seen in Figure 8 below.

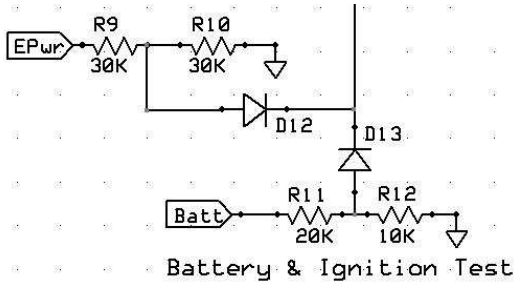


Figure 8: Ignition and Battery Voltage Check

As can be seen from the circuit above, the battery and ignition checks are conducted using the same analog to digital converter pin of the microcontroller (PF3). Due to both the ignition and battery circuits providing voltages greater than 5 volts, voltage dividers were necessary in order to limit the signals being passed to the microcontroller. The voltage divider connected to the input from the ignition system was designed to reduce the signal to 5 volts. The voltage divider

connected to the input from the battery circuit was designed to reduce its input to 2.5 volts. In this way, the processor is able to detect when the 12 volt ignition power source is removed from the circuit by observing that the analog to digital conversion from the power input line drops from 5 volts to 2.5 volts.

On the first revision and current revision of the prototype boards, a center-pin positive power jack was used to simulate the input from the 12 volt car ignition. This was done to make it easy to power the device for testing. In the boards that are created to be marketed, this will be replaced with positive and negative through-hole insertion points for input from the ignition system.

5.2 Sensor Circuit

The sensor circuitry of the device contains two accelerometers and an input line for input from the tachometer of the race vehicle. The first accelerometer is an ADXL278 2-axis high g-force accelerometer. It is manufactured by Analog Devices and was chosen since it was specifically designed for detecting vehicle collisions.

The ADXL278 is capable of monitoring the x and y-axes (front, rear, and sides) of the vehicle that the device is mounted on. It is capable of detecting g-forces of 50-g's on each of the axes. This meets the requirement that was originally identified in Section 3.2 of the report. Figure 9 shows a diagram of the ADXL278 accelerometer, and its connections to the microcontroller (processing module of the circuit).

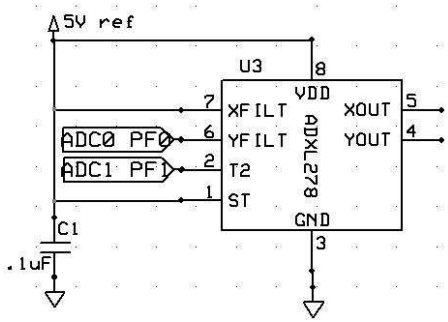


Figure 9: ADXL278 Circuit

The ADXL278 operates on 5-volts of power provided by the 5 volt reference of the power supply circuit. As stated in Section 5.1 above, this voltage needs to be precise in order to ensure accurate g-force measurements. The ADXL278 outputs 0-5 volts on pins 6 and 2 (Pin 6 is the x-

axis output and Pin 2 is the y-axis output). The output from the ADXL278 is fed directly into the microcontroller used for the project. The microcontroller is able to handle 0-5 volt input; therefore, no signal processing is necessary on the output from the ADXL278. The capacitor seen in Figure 9 is a bypass capacitor used for noise reduction on the ADXL278 power lines. The placement of this capacitor was suggested by the ADXL278 datasheet.

Capacitors are not necessary on the X and Y output lines of the ADXL278. In some Analog Devices components, capacitors are needed for bandwidth selection of the output signal; however, the ADXL278 comes equipped with a 400 Hz Bessel Filter. With this filter it is necessary to sample output from the accelerometer at greater than 800 Hz, which is equivalent to about once every millisecond.

The second accelerometer selected was the ADXL330, which is capable of monitoring the x, y, and z-axes of the vehicle. It was chosen to fulfill the objective of detecting vehicle rollovers. In a vehicle rollover, a car will experience 1-g of force on the underside of the vehicle. This is equivalent to experiencing -1-g of force on the roof of the vehicle. Therefore, with the ADXL330, it will be possible to detect vehicle rollovers by monitoring the z-output of the accelerometer and determining if it reaches -1-g of acceleration. The ADXL330 can only accurately handle 3-g of acceleration; however, since only 1-g is in question this will suffice. Figure 10 shows the circuitry design for the ADXL330.

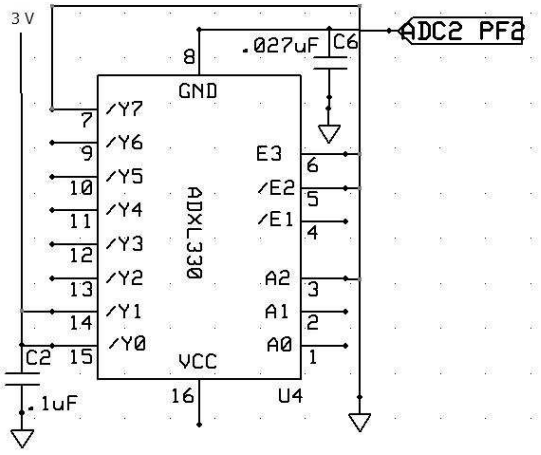


Figure 10: ADXL330 Circuit

The ADXL330 requires 3 volts of power for normal operation. It outputs 0-3 volts on Pin 8, which is the z-axis output pin. As with the ADXL278, this output can be fed directly into the microcontroller without signal processing. Since this part is also manufactured by Analog Devices, it also requires a 0.1 micro-Farad bypass capacitor between the input power lines and ground for noise reduction. Unlike the ADXL278, the ADXL330 does not have an internal bandwidth selection filter. For this reason, a capacitor is needed on the z-axis data output line in order to ensure the proper bandwidth is selected. Since the processor needs to sample the ADXL278 accelerometer every millisecond, it was decided that the ADXL330 should also have new data every millisecond. For this reason, the bandwidth selection chart which can be seen in Table 2 was reviewed, and a capacitor value of 0.027 micro-Farads was selected.

Table 2: Bandwidth Capacitor Selection

Bandwidth (Hz)	Capacitor (uF)
1	4.7
10	0.47
50	0.10
100	0.05
200	0.027
500	0.01

It may be observed from the description above that the ADXL330 has more features than is necessary for detecting vehicle rollovers. In other words, it measures g-force for more than just the z-axis. Even though it was a more powerful sensor than was needed, the ADXL330 proved to be the best choice for the DragAid-MK. The primary reason that the ADXL330 was chosen was due to it being capable of measuring forces perpendicular to the chip. This allows the chip to be mounted solidly on the PC board without inhibiting the chip from measuring z-axis data. Another reason the ADXL330 was selected is due to it being manufactured by Analog Devices. This chip was found to be more reliable than other 1-axis, z-axis accelerometers on the market and it could be obtained at a relatively low price. Therefore, it was decided that the ADXL330 should be used to sense vehicle rollovers in the project.

The final portion of the sensor module was the tachometer input. The circuit for the tachometer can be seen in Figure 11. This circuit will remain in the final design of the DragAid-MK although it is currently not being used. The original design for the DragAid-MK required that

the RPM level of the vehicle be used to determine the operation mode of the device. In B-term, the operational design of the DragAid-MK was altered, rendering RPM monitoring unnecessary. Despite this fact, the sponsors of the DragAid-MK project did not want to remove the ability to monitor RPM levels from the DragAid-MK boards.

The signal from the tachometer is a series of pulses with amplitude of 12 volts. The frequency of the pulses depends on the revolutions of the motor or the RPMs. It also depends on the number of cylinders of the motor. Therefore, a motor with two cylinders running at 5000 RPMs will generate a different number of pulses than a motor with eight cylinders running at 5000 RPMs. It was due to this aspect of calculating RPM that the methods mentioned in previous reports for determining how the RPM signal was encoded failed.

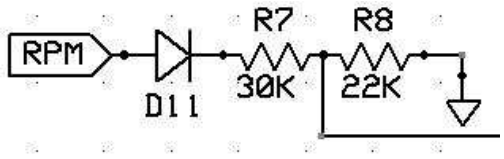


Figure 11: RPM Circuit

Since the signal from the tachometer fluctuates between 0 and 12 volts, it could not be fed directly into the processing module of the circuit without damaging an input port. For this reason, the signal from the tachometer is passed through a diode and a voltage divider. The voltage divider is designed to reduce the input signal from the tachometer such that the maximum value of the signal is 5 volts. The diode will reduce the signal by .7 volts and the actual voltage divider will reduce the input voltage by about 2/5.

The conditioned tachometer signal is then passed into a digital input port of the processor, which is responsible for calculating the frequency of the motor revolutions. As stated before, the number of cylinders in the engine must be known in order for the processor to properly calculate the RPM value. This variable will have to be specified by the user in the PC software created for the device. This variable can be added to the firmware and software of the DragAid-MK system, if it is decided that RPM calculations are necessary.

5.3 Input Circuit

The input module of the DragAid-MK is composed of a series of switches as well as the USB interface to the circuit. Two of the switches of the DragAid-MK are momentary contact switches. The third switch is a positive action on/off switch. The final switch is the input from the two-step of the vehicle the DragAid-MK is mounted on. This is considered a switch since it behaves similar to a switch (it is either activated or deactivated).

A diagram showing one momentary contact switch as well as the positive action on/ off switch can be seen in Figure 12. The positive action on/off switch is the master kill switch of the circuit. It is connected to Pin 25 of the microcontroller due to this pins ability to generate interrupts. When this switch is closed, an interrupt will be generated that will proceed to send shutoff signals to the vehicle the DragAid-MK is mounted on.

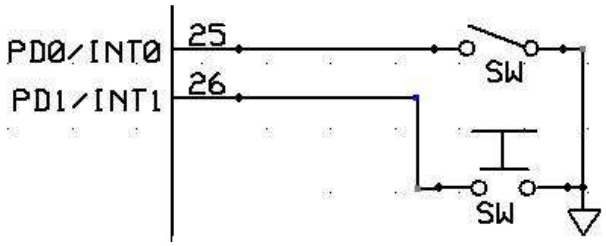


Figure 12: Momentary Contact and Positive Action Switches

The momentary contact switch connected to Pin 26 of the processor is also capable of generating an external interrupt in the microcontroller. This switch is used to control the two-step bypass feature of the device. When the user does not want the two-step to cause the device to start recording data, this button can be pressed. The two-step will not be monitored until this button is pressed again.

Figure 13 shows the second momentary contact switch of the circuit as well as the two-step input circuitry. This switch is connected to Pin 17 of the processor and is used to put the device into test mode (the mode where track officials can test the operation of the device). Although this pin is capable of generating an interrupt, it is multiplexed with the other pins of Port B. In other words, 8 interrupt sources (the 8 pins of the port) share one interrupt vector in the processor.

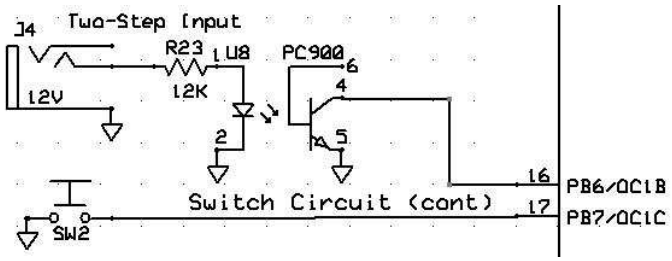


Figure 13: Momentary Contact Switch and Two-Step Input

Pin 16 of the processor is connected to the two-step input from the vehicle. This pin is also on Port B, which could cause a problem if both the two-step and test mode button were monitored using interrupts. This was the original plan for the switch input circuitry; however, once programming began, it was decided that the two-step input should be polled rather than monitored through interrupts. Therefore, no problems are expected from connecting both switches to Port B of the processor.

As described above, the two-step is a form of a momentary switch. It is held on and then released. The rising edge of the two-step is monitored in order to signal the device to start recording data. As can be seen in Figure 13, an optocoupler is used to isolate the two-step input from the input port of the microcontroller. Since the two-step is basically a switch directly across the battery of the vehicle the device is mounted on, the optocoupler was added as a precaution against damaging current levels flowing from the vehicle battery into the input port of the microcontroller. In order to prevent damage to the optocoupler, a 12 KΩ resistor was added to limit current through the input of the optocoupler. This resistor reduces current flow through the optocoupler to about 1mA, which was within the acceptable input current range of the device.

All of the switch signals, except for the two-step input through the optocoupler, are active low, which means that internal pull-up resistors of the microcontroller are activated. When a button is pressed or a switch closed, a low signal is received by the corresponding microcontroller pin. This action generates an interrupt, which changes the operating mode of the DragAid-MK. If the test button is pressed a check on the sensors and software of the DragAid-MK is performed. If the master kill switch is toggled, shut off signals will be sent to the vehicle. If the two-step

bypass button is pressed, a two-step signal will not activate race mode of the device, and if the two-step input changes level, race mode will be entered.

In the previous input circuit of the DragAid-MK, a button was also included for initiating USB downloads in the circuit. Although USB is still used in the new schematic, the download button was removed since a UART interrupt is now used for signaling data transfer between the DragAid-MK and a PC.

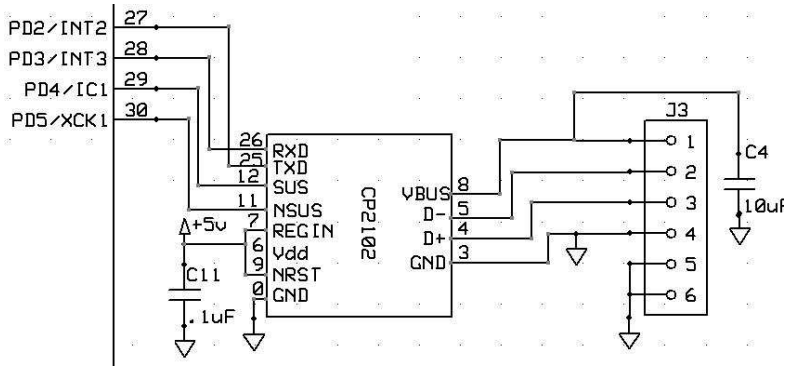


Figure 14: USB Circuit

As can be seen in Figure 14, the USB circuit of the DragAid-MK was altered from the original schematic design of A-term. The previous USB circuit relied on the built-in USB function of the Atmel processor. When testing began with this feature of the Atmel processor, it was found to be difficult to use. There was a lot of ambiguity regarding the proper way to setup the USB interface to the PC and very little documentation. For this reason, the circuit was modified to the version seen in Figure 14.

This circuit uses the CP2102, USB to UART conversion chip manufactured by Silicon Laboratories. This chip connects to the universal asynchronous receive transmit (UART) of the microcontroller and transmits USB data to a PC. A PC board purchased from Spark Fun Electronics was originally used to test the operation of the CP2102. Figure 15 shows this breakout board. The breakout board was used as a model for the USB circuitry seen in Figure 14. It could not be used exactly since the CP2102 breakout board is designed to use power from the USB bus to power the circuit it is attached to. This operation is not desired in the DragAid-MK.

Figure 15: CP2102 Breakout Board[10]

In general, the USB circuit operates as follows. Data is passed from Pins 27 and 28 of the microcontroller to the CP2102 conversion chip. Pin 27 of the processor is the UART transmit pin and Pin 28 is the UART receive pin. Pins 29 and 30 of the microcontroller are connected to the suspend lines of the CP2102 conversion chip. These lines are controlled by the conversion chip. They are used to alert the processor when the USB bus enters the suspend state. This will signal that data transfer has ended.

The remaining lines of the conversion chip are the power lines, which require 5 volts for proper operation and the USB port connections. A USB port consists of two data lines, a voltage line, and a ground line. USB data transfer is a complicated procedure of alternating the signals on the two data lines. An explanation of this will not be given, since the USB conversion chip is able to handle all the details of communication.

5.4 Memory Module

The memory module of the DragAid-MK was constructed using the 25AA512 EEPROM chip manufactured by Microchip. This chip is capable of holding 512 Kbit of data. It connects to a microcontroller using a serial peripheral interface (SPI). This type of interface is a synchronous interface that requires a common clock between the microcontroller and the chip. The clock signal must be provided by the microcontroller. Figure 16 shows the EEPROM circuit, which includes the pins that the 25AA512 must connect to on the microcontroller.

[10] http://www.sparkfun.com/commerce/images/products/00198-03-L.jpg

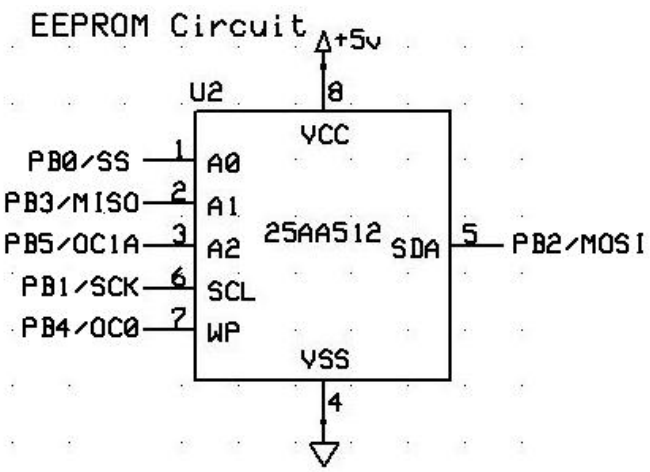


Figure 16: EEPROM Circuit

As can be seen in the figure, the EEPROM circuit contains the standard lines found in an SPI interface. Since the EEPROM is a slave in this circuit, it only has one output signal. This is Pin 2, the master in slave out line. All data sent to the microcontroller will be sent over this pin. Pin 5 is the master out slave in line, which will receive all data from the microcontroller. Pin 1 is the chip select line, which will remain high until the microcontroller wishes to initiate a data transfer.

The 25AA512 allows for rapid read and write of data. It has a maximum write time of 5 ms per page, which will allow many data bytes to be stored in a very short period of time. The exact read time of the chip is not specified in the EEPROM's datasheet, which means that it is very small compared to the write time. The read time is not of major concern, since it will only limit the data transfer rate to the PC.

The EEPROM also has built in write protection and high reliability (endurance of 1 million erase/write cycles). It operates off of 5 volts and stores data in byte format. It is capable of storing a total of 65,536 bytes of data. This is equivalent to 7 runs worth of data. The calculation for this last value can be seen in Figure 17.

$$run = 15s$$
$$save\_rate = .02s$$
$$saves/run = \frac{15}{.02} = 750$$
$$bytes/save = 12$$
$$bytes/run = 9000$$
$$bytes/chip = 65536$$
$$runs/chip \approx 7$$

Figure 17: Data Run Calculation

Despite this ability, it was decided that the DragAid-MK should only be designed to store 3 runs of data. This decision was made due to the chip erase scheme of the 25AA512 EEPROM. Although 7 runs of data could technically fit into the EEPROM, it would be difficult to erase or manage this data if it was not aligned on pages or sectors of the EEPROM. It was found that data could only be erased from the 25AA512 EEPROM by page, sector, or entire chip. It was also found that most EEPROM manufactures suggest erasing data from a chip prior to a write. For these reasons, it was decided that an easy organization of data on the chip would be necessary to ensure fast erase. This led to the conclusion that data for each run should be stored within a sector of the 25AA512 EEPROM chip.

Since the first sector of the chip is designated to hold the necessary system data (race number, threshold values, threshold date, car number, serial number, version number, and version date), which should never be erased, only 3 sectors remained for saving race data. Each sector contains 16384 bytes of storage, which is more than enough to store 1 run of data. When a new run is made, the sector designated to store the new run can be easily erased before the run using the sector erase command of the EEPROM. This will allow the new data to be stored with no difficulties or errors.

Overall, the 25AA512 EEPROM was an appropriate choice for the memory module of the circuit. It is capable of storing sufficient data within the time constraints of the system.

5.5 Clock Module

The clock module of the circuit consists of two oscillators: a 32.768 kHz real time clock oscillator and an 8 MHz external oscillator. The real time circuit oscillator (32.768 kHz crystal)

is connected to the TOSC2 and TOSC1 pins of the microcontroller as can be seen in Figure 18 below. The 18pF capacitors that can be seen in the figure were added to the circuit after testing was conducted on the board. It was found that the clock was 6 times faster than specified before the capacitors were added, which is due to aliasing. Although data sheets for both the 32.768 kHz clock and microcontroller were reviewed to ensure that no capacitors would be necessary, it appears that an error existed in the documents. Once the capacitors were added, the oscillator operated accurately at 32.768 kHz.

The pins that the oscillator is connected to were chosen purposely, since these pins are able to detect changes asynchronously of the microcontroller clock. This feature allows these pins to be used to detect clock pulses. The microcontroller must then count the oscillations of the crystal. All real time clock calculations are performed in firmware.

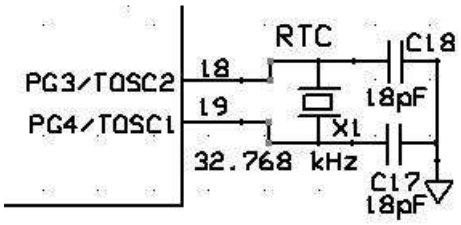


Figure 18: Real Time Clock Circuit

The reason a 32.768 kHz clock was used for the real time clock circuit is as follows. A 32.768 kHz clock pulses at 32,768 cycles per second. When this signal enters the microcontroller the clock frequency is divided by 128. This results in the clock signal appearing to be 256 Hz. Therefore, after 256 pulses, one second has passed. The microcontroller must then only count to 256 in order to observe 1 second passing. The number 256 also has special significance in an 8-bit microcontroller. It represents the maximum value that can be held in a standard 8-bit register. Therefore, when counting to 256, the microcontroller must simply wait for a register overflow to occur. At this time, it knows that 1 second has passed. The processor can then keep track of minutes and seconds as needed.

The 8 MHz clock circuit can be seen in Figure 19. It is connected to the XTAL1 and XTAL2 pins of the microcontroller, which are specifically designed for an external clock input signal. When an oscillator is connected to these pins, it is possible to setup the processor to use the external clock source as the master clock for the processor.

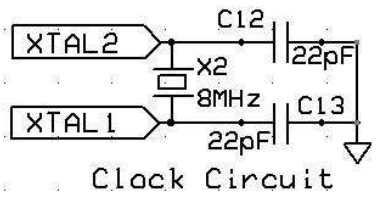


Figure 19: 8 MHz External Clock

Capacitors are also included in the clock circuit above. They are required to ensure that the oscillator operates at the intended frequency (8 MHz) rather than a harmonic of this frequency. Although it is possible to use the 8 MHz internal clock as the master clock for the processor, the accuracy of this clock is in question. When conducting tests between the first revision PC boards (operating from the internal oscillator) and the second revision PC boards (operating from the external oscillator), noticeable timing improvements were observed.

The area where noticeable improvement was made was in the UART communication between the microcontroller and the PC. The new external oscillator allowed the UART to operate with a more accurate baud rate, which resulted in less error in data transfers. When using the first revision DragAid-MK boards, it can be found that the most significant bit of data is dropped on nearly every data transfer. In the second revision boards with the new clock, this situation has not been reproduced even after extensive testing.

5.6 Processing and JTAG Module

The processing module is composed of the AT90USB647 microcontroller, which is manufactured by Atmel Corporation. This microcontroller was originally chosen due to its sufficient memory, many input/output ports, accurate analog to digital converter, serial peripheral interface, and USB capability. Although it was eventually decided that the Atmel processor's USB documentation was insufficient to create a functioning, bug-free device, the processor was not discarded for a new one. This processor is still relatively new as is USB technology. It is believed that when USB processors become more common, Atmel will have improved USB documentation, and it will be possible to use the processor as it is intended.

A pin-out of the AT90USB647 can be seen in Figure 20. The pins that were used in the construction of the DragAid-MK have been marked in the diagram. As can be seen in the pin-out, a majority of the pins of the AT90USB647 were used.

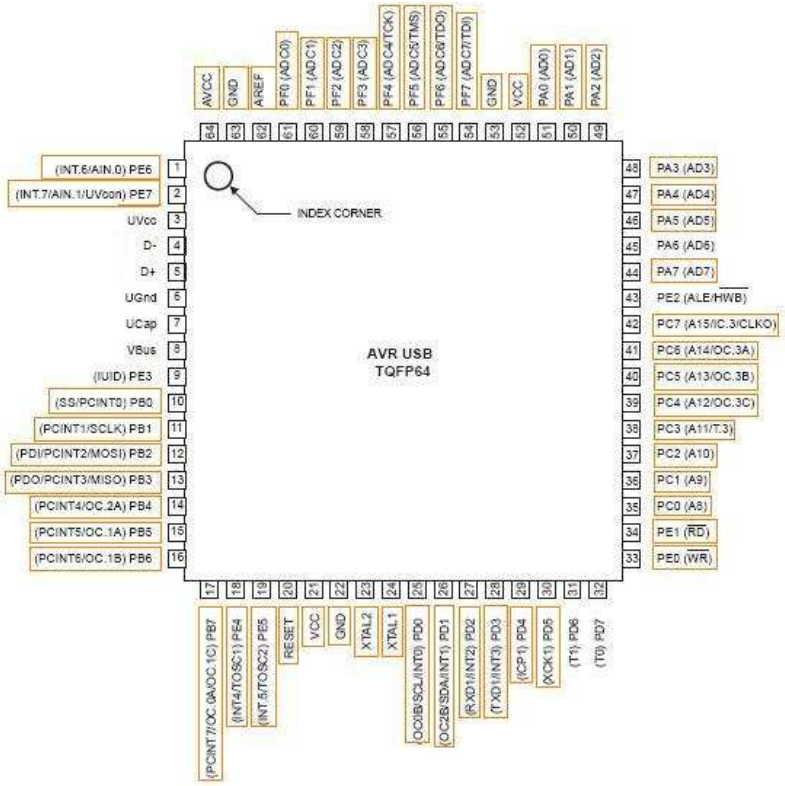


Figure 20: AT90USB647 Pin Out

The main features of the microcontroller that were used in the DragAid-MK are as follows. Pin 2 is used to completely shut down the circuit by disabling the battery backup system. This was described in more detail in Section 5.1. Pins 10 through 15 are used to interface with the EEPROM or memory module of the device. Four of the 6 pins are specifically designated in the microcontroller for use in SPI data transfers. The remaining two pins are used to control the write protect and hold (other control signals) of the EEPROM.

Pins 16 and 17 are connected to the two-step input circuitry and the test mode momentary contact switch, respectively. They were placed on this port of the microcontroller due to the interrupt capability of the port pins. Since the signals received from the momentary contact switch is active low, the internal pull-up resistor feature of the processor will also need to be used on this pin.

Pins 18 and 19 of the microcontroller are connected to the 32.768 kHz real time clock, while pins 23 and 24 are connected to the 8 MHz external oscillator. Pins 25 and 26 are connected to the

remaining two switches of the input module. Pin 25 is connected to the positive action on/off switch allowing it to be used as the input to the processor for the master kill switch, and pin 26 is connected to the momentary contact switch that represents the two-step bypass switch. Both pins have individual interrupt capability. The interrupt of pin 25 has higher priority than all other button interrupts; therefore, the master kill switch was assigned to this port pin.

Pins 27-30 are used to connect to the UART to USB conversion chip. The first two pins (27 and 28) are the receive and transmit lines of the UART, which are connected directly to the conversion chip. Pins 29 and 30 are general purpose input ports of the processor. They are connected to the suspend lines of the conversion chip. They are used to determine when data transfer is initiated or terminated over the USB bus of the circuit.

Pins 33 through 42 are used as shut off signals to the vehicle. They will eventually be connected to solid state relays that will be capable of actually shutting down the vehicle. Pin 44 is used for the RPM input signal. Pins 46 through 51 are general output ports that are used to connect to the LEDs of the DragAid-MK. Finally, pins 58 through 61 connect to the various parts of the DragAid-MK that require analog to digital conversion. This includes the accelerometers and the battery and ignition of the DragAid-MK.

Four remaining pins that were not mentioned in the description above are pins 54-57. These are designated for the JTAG interface to the DragAid-MK. They each have a specific purpose as specified by the joint test action group (JTAG) standard. The first pin corresponds to TDI (test data in), the second to TDO (test data out), the third to TMS (test mode select), and the final to TCK (test clock). The JTAG interface allows debugging of the circuit. In reality, it uses a boundary scan to review all registers that are not part of the internal core of the processor. Breakpoints can be inserted in the embedded code in order to halt the processor when a certain boundary condition is found. In this way, boundary scan has become very popular for in circuit debugs of complex embedded systems. The JTAG is also commonly used for processor programming.

5.7 Output Module Construction

The final module that was constructed is the output module of the circuit. The output module is composed of a series of LEDs that are used to inform the user of the mode the DragAid-MK is currently in. The output module also consists of shutoff signals that are connected to the vehicle the DragAid-MK is mounted on.

The circuit for the LED user interface can be seen in Figure 21 below. As can be seen, the LEDs are active low, which means the microcontroller must provide a low voltage level in order to turn on each of the LEDs. They run on 15 milliamps; however, this will not be a problem, since at most 2 LEDs will be on at one time. All of the LEDs on the current prototype board are yellow; however, the final product will most likely include red LEDs as well.

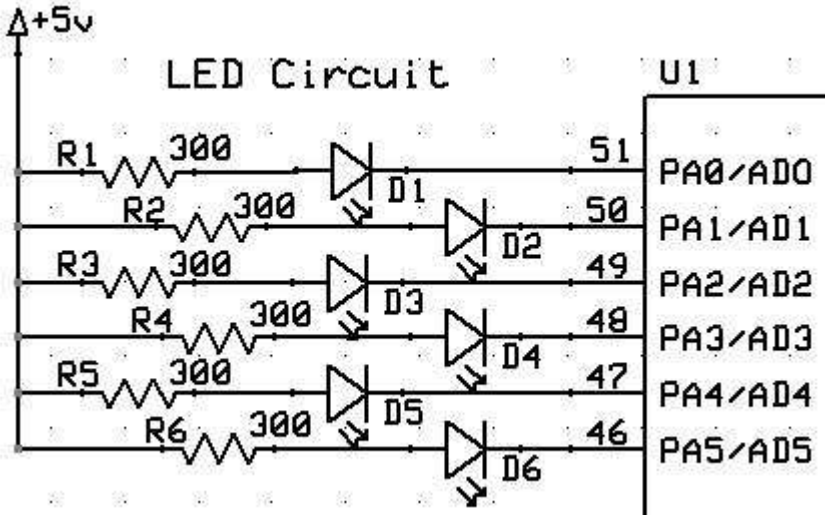


Figure 21: LED Circuit

The purpose of the LEDs is as follows. The first LED will inform the user that the device is in pit road mode (a run will not be made). The second LED will signal that the device is in race mode (prepared to make a run). If the device is in pit road mode, the fifth LED will inform the user that the two-step has been bypassed. If the user is in race mode, it will inform him that the master kill switch has been pressed or g-forces have exceeded their threshold levels. The third LED will signal that the device is in download mode. The fourth LED will inform the user that the device is in test mode, and the final LED will be used, along with the other LEDs on the board, to test the device while in test mode.

The second part of the output module is the shutoff signals to the vehicle. The design of this circuitry was not completed as part of this project; however, it will eventually involve connecting general output ports of the processor to solid state relays. These relays will either be mounted in the device or in the race vehicle. A member of the design team, who works for the project sponsor and is familiar with race cars, will be designing this circuit this spring. Figure 22 shows the design for the shutoff signals on the prototype board. This design is very basic; however, it should allow us to test if the device is sending the proper signal to the relays, which is all that is necessary in a prototype board.

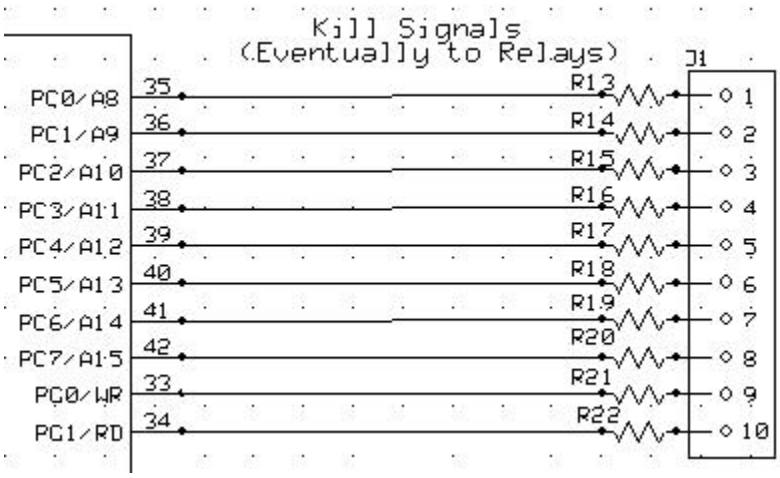


Figure 22: Shutoff Signal Circuitry

5.8 Firmware Design

The functionality of the DragAid-MK, or in other words, how the mechanism works, is controlled by the firmware of the device. The firmware is written in Atmel Assembly, and consists of 5 code modules. Each code module will be described separately in the following sections. Assembly was chosen as a programming language for two reasons: timing requirements of the DragAid-MK and specifications of the project sponsor.

The firmware of the system is all code that executes within the embedded processor (AT90USB647). When creating this code, the circuitry described in the sections above were carefully considered in order to determine the proper way to interact with all systems in the project. The desired functionality of the device was also reviewed. From the information gathered, it was decided that 1 main module and 4 auxiliary modules (a total of 5 as described above) should control the operation of the DragAid-MK.

Before programming began on each module, firmware flow charts were designed in order to create guidelines to promote the completion of each module. This technique worked well especially when problems were identified in programming. It allowed the project to stay on task and, most importantly, maintain the desired functionality of the device even when errors occurred. Each subsection of this report will describe a particular module, and demonstrate the functionality of this module based on a flowchart. All flowcharts have been modified to represent the final functionality of the device.

5.8.1 RollOverDevice (Main Function)

The first module to be described is the RollOverDevice module. This is the main module of the DragAid-MK and describes the main functionality of the device. It contains the main loop of the DragAid-MK and describes the operation of the device in both pit road mode and race mode. A flowchart for the main module can be seen in Figure 23. The firmware for RollOverDevice.asm (actual code) can be viewed in Appendix G.

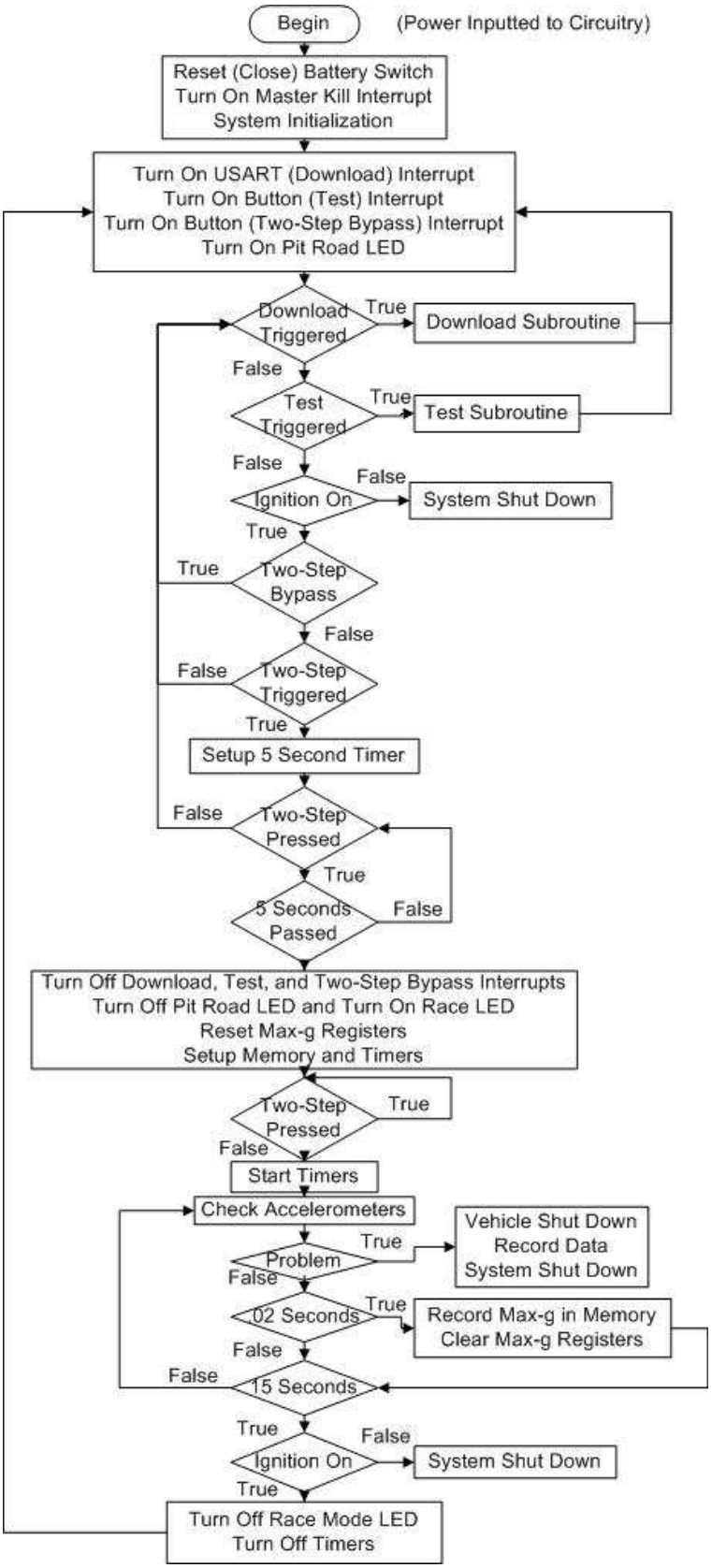


Figure 23: Main Function Flowchart

The main function of the DragAid-MK operates as follows. The program begins when the microprocessor receives 12 volt input from the ignition system of the vehicle. The first action of the processor is to output a low signal to the battery on/off switch (transistor). This will result in the battery circuit being enabled; thereby, allowing it to power the DragAid-MK if the ignition system of the vehicle shuts down.

After this is completed, the program performs initialization of the main systems to be used. This includes tasks such as declaring the stack pointer and setting the data direction of the LED port. The initialization process must also set the initial values to high for the LED port (all LEDs off). It further includes declaring the switch ports to be input ports and enabling the pull-up resistors for these ports. The power reduction register is also setup at this time. In order to reduce power consumption of the processor, all unnecessary systems are shut down through the power reduction register. At this time, the system functions that are disabled throughout the program include the two-wire synchronous serial interface, timer/counter 0, the USB controller, and timer/counter 3.

During system initialization, the SPI interface to the EEPROM is also setup. This system is set to operate as the master of the interface with a transfer speed of 4 MHz. All control values of the EEPROM are setup properly such that the EEPROM is not activated; however, it is not write protected either. A preliminary data read is then conducted such that system variables can be received from the EEPROM.

After this is completed, the ADC is initialized so that each following section of the code must only declare a channel in order to make an ADC reading. The x-axis accelerometer is connected to channel 1, the y-axis is connected to channel 2, the z-axis is connected to channel 3, and the power input circuitry is connected to channel 4 of the analog to digital converter input port. The ADC initialization involves enabling the ADC and setting it to operate at a clock speed of 125 kHz (this is a suggested clock rate for the ADC in order to ensure accurate conversions).

After system initialization is completed, the processor will enable the interrupt for the master kill switch as well as global interrupts. It is necessary to enable global interrupts, since this feature

allows the interrupt controller of the processor to function. Otherwise, no interrupts will be received. When enabling the master kill switch interrupt, it is necessary to specify the edge of the interrupt signal that will produce an interrupt. It was decided that the falling edge of the interrupt signal (since the switches are active low) should produce both a master kill and a two-step bypass interrupt.

At this point, the initialization process for pit road mode begins. The first item initialized in pit road mode is the universal asynchronous receive transmit (UART). This system is setup to operate asynchronously (it has the option to operate synchronously as well), with no parity bits, 1-stop-bit, and 8-bit (1-byte) transfers. Both transmit and receive systems are enabled in the controller with a baud rate set to 9600 bits per second. Now that testing has been completed this rate could be increased without any difficulties. The receive interrupt of the UART controller is also enabled such that the DragAid-MK system is able to detect a PC request to transfer data.

During pit road initialization, the test button and two-step bypass interrupts are also setup. A variable, which monitors the switches that are currently pressed, is cleared before entering the actual pit road mode program. The final initialization step will be to turn on the pit road LED, to inform the user that the DragAid-MK is on, but not in race mode. In order to do this, a low signal is sent to the pit road LED pin of the LED port.

After initialization is complete, a series of checks are made. Before describing the checks, an important concept regarding the interrupts should be explained. Although the UART, test, and two-step interrupt handling routines will occur immediately when the proper trigger signals are received, the interrupt subroutines will not perform the actual operations that the processor is being interrupted to recognize. As an example, the interrupt subroutine for the test button will not perform the entire test sequence of the processor. Instead the interrupt subroutines will set flags that can be checked later to determine if any interrupts have occurred. The only interrupt that contains its entire sequence within the handling routine is the master kill interrupt. This is due to the fact that the master kill switch requires immediate handling upon being pressed. The actual master kill routine will be described in a following section.

Almost all checks that occur after system initialization are on the flag register described above, which holds the information as to which interrupt has occurred. The first interrupt checked via the flag register is the UART interrupt. If it is found that a data transfer or download is pending, the download subroutine is entered. Upon return from this subroutine, the pit road initialization sequence (beginning after the setup of the master kill interrupt) must be repeated. The next interrupt checked is the test interrupt. If this was found to be triggered, the system will enter the test mode subroutine. It will perform the same action as the download subroutine upon return.

The next check performed is a voltage check on the ignition system of the vehicle. It is performed using the ADC of the microcontroller. In order to complete this check, the ADC channel must be set to 4 in order to tell the system that it wishes to read the level of the power input circuitry. If the reading from the power input circuitry is above 3.7 volts, it can be assumed that the ignition system of the vehicle is on. A voltage of 3.7 corresponds to an ADC value (in hex) of 0x300. Therefore, if the high-byte of the ADC conversion is greater than or equal to 0x03, it is known that the ignition system of the vehicle is operating; otherwise, the DragAid-MK will be running from the battery system. If the ignition system is found to be off, the shut down routine of the microcontroller is called, which will cut-off the battery from the system. If the ignition system is running, the check sequence of the main function continues.

The next check is to determine if the two-step of the vehicle has been bypassed. When the bypass interrupt occurs, the bypass LED of the circuit will be turned on. If the two-step is found to be bypassed at this check, the two-step status will be cleared and the series of checks will begin again. If the two-step is not bypassed, the series of checks will continue by checking the two-step flag. If the two-step has not been triggered, the series of checks will begin again; otherwise, a connection sequence between pit road mode and race mode will be entered.

The connection sequence between pit road mode and race mode involves setting up a 5 second timer. It was decided that if the two-step is held for 5 seconds or longer, the driver is most likely on the starting line preparing to make a run. Otherwise, the driver may only be testing the two-step to ensure that it is functioning properly. If this is occurring, race mode should not be entered. Therefore, if the two-step is held for longer than 5 seconds, race mode initialization

should begin. If it is released before the 5 second time mark, the checks of pit road mode should begin again.

The initialization sequence of race mode begins by turning off the UART interrupt, test interrupt, and two-step interrupts. It turns off the pit road LED, and replaces it with the race mode LED, which will signal to the user that the system is ready to collect data. At this point a 15 second and .02 second timer is setup, which will be used to time events in race mode. The first timer represents the time limit of race mode and the second represents the amount of time between each data save in race mode. Furthermore, registers, which will hold maximum g-force readings in a .02 second interval, will be cleared in preparation for data collection. A memory read is also conducted during race mode initialization, in order to prepare and obtain the memory location that data should be written to during a run. Once all this is complete, the system waits in order to determine if the two-step has been released. If it has, the race has started, and race mode is finally entered.

The representation of race mode seen in the flowchart of Figure 23 is much generalized compared to the actual operation of the code. This was done in order to make the flow chart easy to read and the general operation of race mode easy to understand. A more accurate representation of this mode will now be given. It can also be seen in the actual code in Appendix G.

The first action completed in race mode is the checking of the x-axis accelerometer. In order to do this, the ADC must be set to read from channel 1, which is the x-axis input. The value received is a 10-bit value; therefore, a double comparison scheme must be conducted in order to test both the high-byte and low-byte of the conversion. This scheme will not be described in detail, but if interested can be seen in Appendix G.

Once the conversion value is received, it is compared to determine if it is higher than an upper threshold specified for the x-axis. Since g-force can be positive or negative depending on the direction an impact occurred, it is necessary to have both upper and lower thresholds for each axis of the DragAid-MK. The actual g-force values read from the vehicle should be between the

two threshold values for that particular axis. If the actual g-force reading is outside of the limits, a problem has occurred and the vehicle should be shut down.

Therefore, after a conversion result is received for the x-axis, the firmware continues by comparing the conversion result to the maximum upper g-force value received from the x-axis during this .02 second interval (this value is stored in two registers for comparison purposes). If the new value is greater than the old, the maximum upper g-force register is updated with the new value. If the new value is greater, a check is also completed to determine if the new g-force value is greater than the x-axis upper threshold of the device. If no problems are found a series of lower tests are conducted. If the new value is not greater than the old values, then the threshold check is skipped, and the device continues by checking the lower g-force boundaries.

If the g-force value is lower than the minimum lower g-force value received from the x-axis during the current .02 second interval, the minimum lower g-force registers are updated. As before, if this is true, a comparison of the new g-force values to the actual device thresholds is conducted. If this check finds a problem (the g-force values are lower than the low thresholds), the vehicle is shut down, the g-force values received are stored, and the DragAid-MK system shuts down. Otherwise, race mode continues to operate.

Race mode continues by setting up the ADC for a y-axis conversion and completing the same comparisons as above with y-axis maximum and minimum values as well as y-axis upper and lower thresholds if necessary. Once the y-axis is complete, the z-axis is also completed in the same manner. After the z-axis checks are completed, the .02 second timer interrupt flag register is checked in order to determine if .02 seconds has passed. If this time has passed, an EEPROM memory write is initialized. The current values of the maximum upper g-force registers and minimum lower g-force registers (12 registers or bytes in total) are written into EEPROM. When this is completed, the memory pointer is updated, and the maximum and minimum registers are reset to default values.

Finally the system checks to see if 15 seconds has passed. If this is false, race mode repeats by once again checking the accelerometers of the system. Otherwise, the race is complete in the

eyes of the DragAid-MK. The ignition is checked in order to determine if the car is still running. If it is not, a system shut down occurs; otherwise, the race mode LED is shut down and all interrupts initialized for this mode are turned off. The system returns to the initialization sequence of pit road mode, which occurs after the initialization of the master kill interrupt. This concludes the operation of the main firmware of the DragAid-MK.

5.8.2 Common.asm

The next code module written was common.asm. The code for this module can be viewed in Appendix H. This module contains all auxiliary functions for the DragAid-MK system. These include functions needed to interact with the serial peripheral interface (SPI) and analog to digital converter (ADC). It also includes the functions that were designed to perform special tasks for the DragAid-MK, such as system shut down. Flowcharts were only created for two of the functions in this module. In general, the functions in this module have designs that only involve the processor executing a sequence of actions, which would result in simple flowcharts.

The first set of functions created was for the SPI interface between the microcontroller and EEPROM. In communications with the EEPROM, it is necessary to begin by sending a command to the EEPROM. There are several commands that can be sent such as read, write, write enable, page erase, sector erase, and chip erase. Each command is specified by sending a certain byte long command to the EEPROM. Some commands require that an address be sent immediately following the command. These commands include read, write, page erase, and sector erase (the EEPROM must know where to read, write, or erase from).

The first function created for the SPI interface was named *SPIComSend* and was designed to take in a command and an address (if necessary). The function began by loading the command into the SPI data register and calling *SPIWait*, which is another function in this module. *SPIWait* operates by looping until the transmit data complete flag of the SPI status register is set. Therefore, *SPIWait* does not return until the data transmit is complete. After this is completed, *SPIComSend* checks the command that was sent in order to determine if it is necessary to send a 16-bit address as well. If the one of the commands that requires an address is being sent, two more data sends are completed with the address bytes. Once this is complete, the function returns.

Another function in this section of the module is the *SPIRead* function. In order for the microcontroller to receive data from the EEPROM, it must send dummy data to the EEPROM after the read command and read address have been sent. A dummy data send is necessary, since in an SPI interface, the slave device is not able to initiate a data transfer. Therefore, the EEPROM may have data to send to the microcontroller, but it is not able to send this data unless the microcontroller sends data, thereby, activating the clock that controls the SPI interface.

Therefore, the *SPIRead* function is designed to begin by sending out $00 to the EEPROM, which will give the EEPROM the opportunity to send data back to the microcontroller. After loading the SPI data register with the dummy value, *SPIRead* calls *SPIWait* in order to loop until the transfer is complete. Upon return from this function, the data read from the EEPROM will be in the SPI data register. This value is returned to the calling function.

The next function created was the *SPIWrite* function. This function simply stores the data to be sent to the EEPROM, which is the input to the function, into the SPI data register. It then calls *SPIWait*. Upon completion of *SPIWait*, it returns to the calling function.

The final function created for SPI communication was a function specifically designed to wait for a chip or sector erase to complete (*SPIStatusWait*). This function should be called after a chip or sector erase command is sent to the EEPROM. It begins by delaying firmware execution by 16 ms, which is the approximate time of a sector or chip erase. It then requests to read the EEPROM status register by sending the status read command through the SPI. If the status data shows that the erase is complete, the function exits. Otherwise, it loops and continues to check the status register until the erase is complete.

The next set of functions created was for the ADC of the microcontroller. The first function written was *ADCWait*, which, as its name implies, is designed to loop until an ADC conversion is complete. This function should only be called after an ADC conversion has been started. It checks the conversion complete flag (ADIF) of the ADC status register (ADCSRA) and exits once this flag is set.

The second and final function created to work with the ADC was *ADCStart*. This function is designed to initiate and complete an ADC conversion. The function begins by starting the ADC conversion and clearing the ADC conversion complete flag, which may still be set from previous conversions. It accomplishes this by writing the correct bits of the ADC status register (ADCSRA). After this is completed, *ADCWait* is called. Upon completion of this function, it is known that the result of the ADC conversion has been written into the ADC high and low data registers. This data is then transferred from these registers into user controlled registers and returned to the calling function.

The next set of functions created were the special functions for the DragAid-MK. The first of these functions was the *SystemShutDown* function. Upon being called, it is expected that the ignition system of the vehicle the device is mounted on is no longer running. It is designed to send a high signal to the transistor acting as the battery cut-off switch. The high signal should shut down the transistor and cut-off power from the DragAid-MK circuit. The *SystemShutDown* function then loops until the power is removed from the circuit (which should be nearly instantaneous). A flowchart for the *SystemShutDown* function was created. As stated above, it is very simple and merely summarizes the above paragraph. It can be seen in Figure 24.

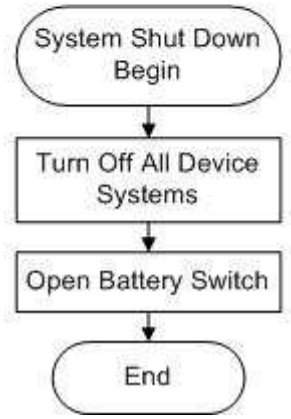


Figure 24: System Shut Down Function Flowchart

The next function of this set is the *VehicleShutDown* function, which as its name implies, is designed to send off-signals to the vehicle the DragAid-MK is mounted on. This is equivalent to the operation of the master kill switch interrupt. It should only be called after the race mode code decides that the device thresholds have been exceeded. The function operates by sending

high signals to all port pins designated to be used for shutting down the ignition and fuel system of the vehicle. Upon completing this operation, the function returns. A flowchart for this function can be seen in Figure 25.

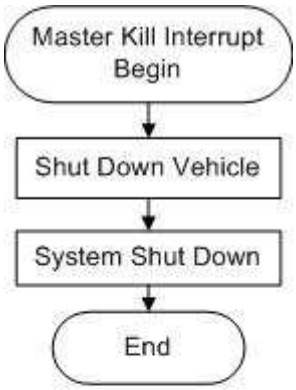


Figure 25: Vehicle Shut Down Flowchart

The final function of this section was the *StoreValues* function. This function is designed to complete a g-force value save to EEPROM. It will simply store the 12 bytes of data in the maximum upper g-force registers and minimum lower g-force registers into the EEPROM. This function was written to be called after g-force values have surpassed the thresholds of the device; therefore, it does not need to update the memory pointer or maximum and minimum g-force registers, since upon returning to the calling function, the system will be shut down.

5.8.3 Download.asm

The firmware written for the download subroutine or data transfer subroutine can be seen in Appendix I. A flowchart for this routine can be seen in Figure 26.

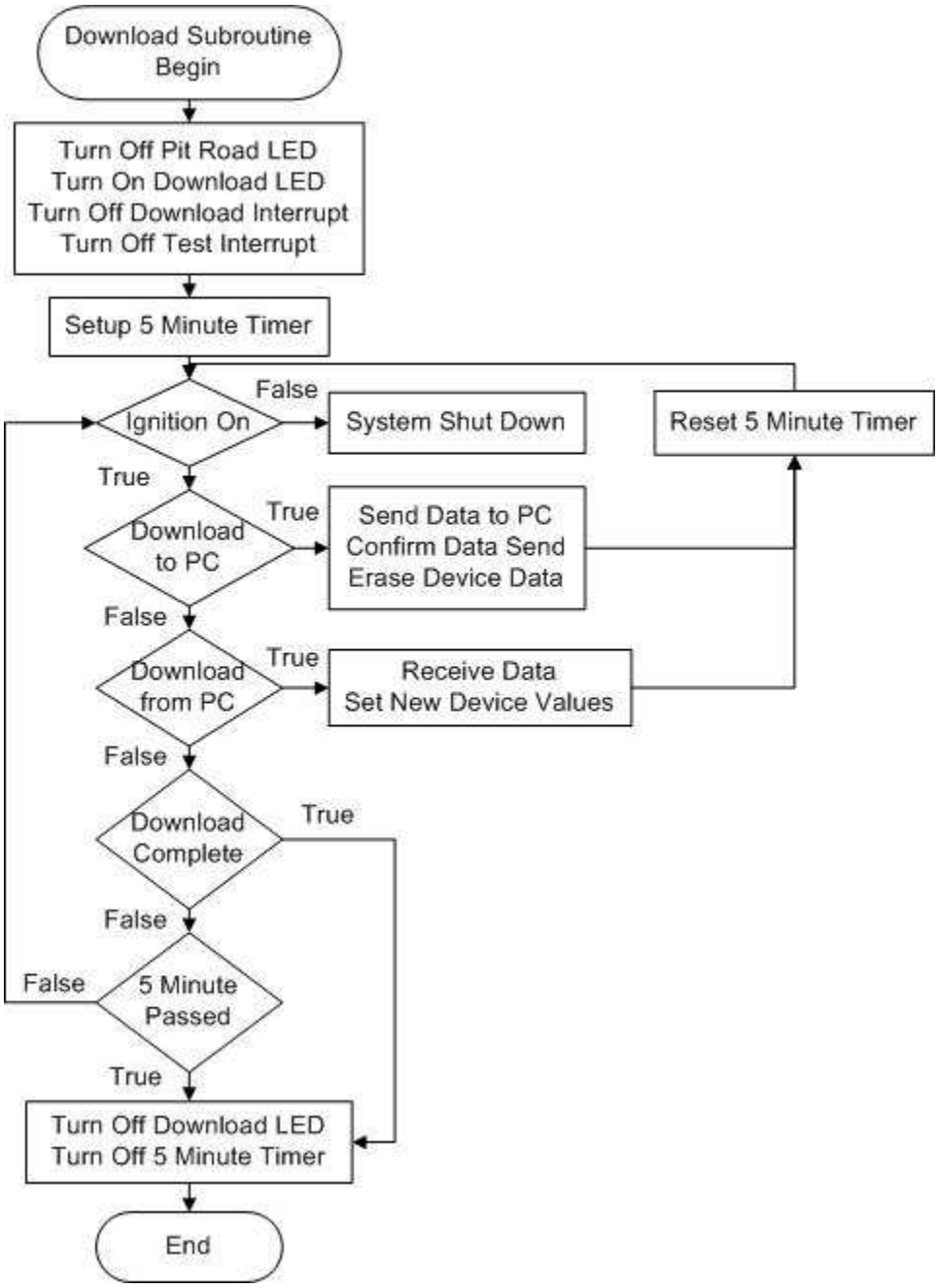


Figure 26: Download Subroutine Flowchart

The download function of the DragAid-MK operates as follows. It begins by turning off the pit road LED and turning on the LED to signal that the device is in download mode. It then performs initialization to prepare the device for operating in download mode. The main steps taken to initialize the device for download mode include turning off the download, test, and two-step interrupts as well as setting up a 5 minute and 1 millisecond timer. The 5 minute timer is a timeout for download mode (when 5 minutes pass the device will reenter pit road mode). The 1

millisecond timer is necessary as a buffer on data sends. The final step is to set the ADC of the device such that conversions are taken from channel 4. This is done so that the ignition system or power input of the vehicle can be monitored.

As with the code for the main function, once initialization has occurred, a series of checks are performed. The first test conducted is to see if the vehicle ignition is on. If this is found to be false, download mode is exited and system shut down occurs. If the ignition is still on, the system checks to see if data was received from the PC. If this is true it further checks to see if the PC wishes for the microcontroller to send or receive data. If the microcontroller is to receive data, the system calls the function *DataRecieve.* If the microcontroller is to send data, the function *DataSend* will be called. Otherwise, the function checks to see if data transfer is complete. If this is true the system returns from the download subroutine to the calling function.

Upon returning from the *DataSend* and *DataReceive* routines, a time out check is completed in order to see if 5 minutes has passed. If this is true, the system returns from the download subroutine to the calling function. Otherwise, the system jumps to the beginning of the loop of checks in order to see if a new data transfer has occurred.

The two main functions of the download module are the *DataSend* and *DataReceive* functions. These functions are responsible for all major data transfer in the DragAid-MK system. The first function (*DataSend)* has two main operations. It can either send basic device data back to the PC, or it can send race data that was collected. The information sent to the PC depends on a command that is received by the DragAid-MK from the PC. If the PC only wants to receive information about the DragAid-MK, the *DataSend* routine proceeds by sending the car #, upper and lower threshold values, and date the thresholds were last modified. If the PC wants to receive the run data, the DragAid-MK will respond with the car # followed by the run data. It is capable of sending up to 3 runs of data back to the PC. When the data send is complete a ‘C’ will be sent to the PC before the *DataSend* function exits. The *DataSend* function will also clear the time out variable before returning to the main download loop.

The *DataReceive* function also has two main operations. It will either receive initialization data for the DragAid-MK, which resets all key variables in the device, or it will receive threshold data. If the PC sends an initialization command, the DragAid-MK responds with 'OK' and continues by loading default values for car #, threshold values, threshold date, version number, version date, and serial number into the EEPROM of the device. If the PC sends a threshold command, the DragAid-MK responds with 'OK' and continues by waiting for the thresholds, the threshold date, and the car # from the PC. These are the only values that are updatable by the user of the DragAid-MK. The initialization command is for the manufacturer of the DragAid-MK. When these values are received, they are stored into temporary variables before a final data save is completed, which will store these values into the EEPROM of the device. When the data save is complete, the DragAid-MK will respond with 'C' to let the PC know that everything was completed successfully. As with the *DataSend* command, the *DataReceive* function will reset the time out variable on exitting.

The final two functions written for the download module were *ReceiveWait* and *TransmitWait*. These functions were designed to improve the readability of the *DataSend* and *DataReceive* functions. The first function (*ReceiveWait*) is designed to read the UART status register (UCSR1A). It checks the receive-complete flag of this register. While this flag is not set, the function loops inside the *ReceiveWait* routine. When this flag is set, the data from the UART data register will be loaded into a user register and the function will return with the received data.

The final function of the download module, *TransmitWait*, takes as input the data to send to the PC. It stores this value in the UART data register, which will initiate the UART data transmit. The function then loops continuously checking the UART status register (UCSR1A). When the transmit-complete flag is set, the loop ends. At this point, the function enters another loop, which is designed to delay the system for 1 millisecond. This delay is necessary between transmits; otherwise, the PC will not obtain all data transferred in a consecutive data transfer. It is uncertain whether this error is due to a receive buffer in the PC or in the CP2102 chip; however, this small modification to the transmit function fixes the problem. This problem and solution were found through experimental data transfers using the device.

As an end note, before the processor returns from download mode, the data transfer LED is turned off and the 5 minute and 1 millisecond timers are disabled.

5.8.4 Interrupts.asm

The next module constructed was the interrupt module. This module contains all interrupt subroutines used in the DragAid-MK firmware. The code for this module can be viewed in Appendix J. Flowcharts were not created for the functions written in this module due to their simple design. Good program design requires that interrupt subroutines are as short as possible. Therefore, the interrupt routines found in this module have no looping and very little branching. They mostly consist of a short sequence of actions that the microcontroller must perform.

The first interrupt subroutine found in this module is the *MasterKill* routine. This function performs the same operation as the *VehicleShutDown* function presented in section 5.8.2 of this report. The main difference between this function and the function described earlier is that this function is triggered by pressing the master kill switch of the DragAid-MK. Since it is an interrupt handling routine, it must also store all registers used in the routine onto the stack. Atmel processors also require the user to store the microcontroller status register onto the stack. Otherwise, this register is not preserved during an interrupt subroutine call.

The next interrupt subroutine is *TwoStepInt*. This routine is entered when the two-step bypass button is pressed. This function operates by checking the flag of the switch variable corresponding to the two-step bypass. If the two-step bypass flag is set, the function knows that the two-step is currently being bypassed in the program. Since the button has been pressed again, the two-step bypass flag should be cleared and the LED representing the two-step bypass should be turned off. If the flag is not set on entering the routine, the flag should be set during the routine and the two-step bypass LED should be turned on. After one of these two operations is completed, the *TwoStepInt* will return to the function that it interrupted.

Another interrupt subroutine in the module interrupts.asm is *TestInt*. This function has a simple operation. When the test button of the DragAid-MK board is pressed, the flag in the switch variable corresponding to the test switch is set. After this is completed, the interrupt subroutine exits. The interrupt routine *Download*, which follows *TestInt* in the interrupt module performs

the same operation as *TestInt*. The two differences between the routines are that the *Download* routine occurs on a UART receive interrupt (rather than a test button press) and modifies the download flag of the switch variable (rather than the test flag).

The final interrupt subroutine in the interrupt module, which performs an actual function, is *RTCInt*. This module simply increases the value of a variable named secs. It is used as a counter that can be set to count at different intervals. The final interrupt handling routine in this module is *Unused*. If an interrupt not being used by the DragAid-MK firmware is triggered by accident, the firmware is setup to call the *Unused* interrupt. This interrupt simply consists of a return statement that will cause the execution of the program to return to the interrupted function.

5.8.5 Test.asm

The final module written to complete the functionality of the DragAid-MK was the test module. The code for this module was written in test.asm and can be seen in Appendix K. The flowchart for the test module can be seen in Figure 27.

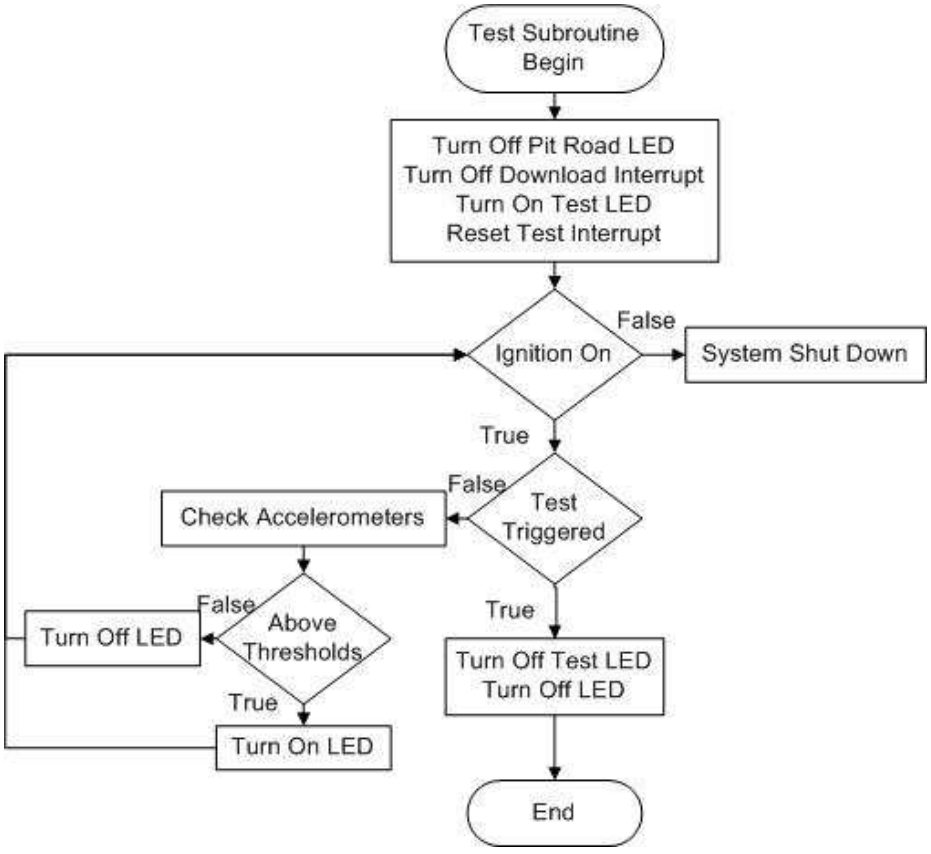


Figure 27: Test Subroutine

As with the main routine and download routine, the test function subroutine begins with its own initialization sequence. The first step is to turn off the pit road LED and turn on the LED for test mode. It proceeds by turning off the download interrupt and resetting the test mode interrupt. When the user wishes to exit test mode, the test button should be pressed again, which will signal to the test function that it should return to the calling function.

After the initialization sequence for test mode is completed, the main loop of test mode is entered. In this loop, the function begins by using the analog to digital converter to check the voltage level of the power input circuitry. If it is found that the ignition system is off, the system shut down function will be called, which will shut down the DragAid-MK device. If the ignition system is still running, test mode will continue by checking the switch variable in order to see if the test button has been pressed again. If it was once again pressed, the test mode LED will be turned off and the system will return to the calling function.

If the test mode button has not been pressed, test mode continues by conducting an accelerometer check. This accelerometer test sequence is setup in the same way as the tests conducted in race mode. The only difference is that the x, y, and z thresholds are set to low values (+/- 1 g). When the g-force levels surpass these thresholds, instead of vehicle shut down occurring as in race mode, an LED on the DragAid-MK turns on. If the threshold surpassed is the upper x-threshold, the first LED on the device will turn on. If the lower x-threshold is surpassed, the second LED on the device will turn on. Surpassing the upper y-threshold results in the third LED turning on while the fifth LED will turn on when the lower y-threshold is surpassed. The final LED will light when the device is held upside down in order to test the z-axis accelerometer. The accelerometer check loop of test mode will continue until the test mode button is pressed or the ignition system of the vehicle is shut down.

5.9 Software Design

The DragAid-MK software is the program to be run on the user's PC in order to view the data collected by the actual embedded system. The software was written in Delphi, which is a specialized program for creating windows applications. The software interface of the DragAid-MK was not a major feature of this project (it was not a goal set forth for the completion of the MQP); however, it was started in order to lay a base for future continuation of this project.

The basic operation of the software has been decided; however, more work is required before the software is released to the racing market. The major work conducted in the software design was the screen layouts and screen expectations. These can be seen in the following figures below.

Figure 28: Software Main Screen

Figure 28 is the main screen of the PC interface software for the DragAid-MK. As can be seen above, when the software is complete, it will have four major functions. It will allow the user to download data from the DragAid-MK, view data that was downloaded in the past, adjust threshold values and update the DragAid-MK, and in the far future, it will also use the data received by the DragAid-MK to simulate what occurred to the car during the run.

The screen shot in Figure 29, shows the help screen of the DragAid-MK.

Figure 29: Software Help Screen

This screen will explain the different features of the software to the user. The intent of this page is to be easy to use as well as helpful. Right now it operates as follows. The user must select one of the 6 choices in the upper left corner of the help screen. When a new choice is selected, the help available for that selection is displayed in the main portion of the help screen.

If the "View Run" button of the main screen is selected, the dialog box seen in Figure 30 is displayed to the user.

Figure 30: Software Car Select Screen

It requests the user to enter the car number and run number for the car that the user wishes to view. In this way, it is able to search through available runs and car numbers in order to find the run the user is requesting.

The actual view run screen is not complete. Its current condition can be seen in Figure 31.

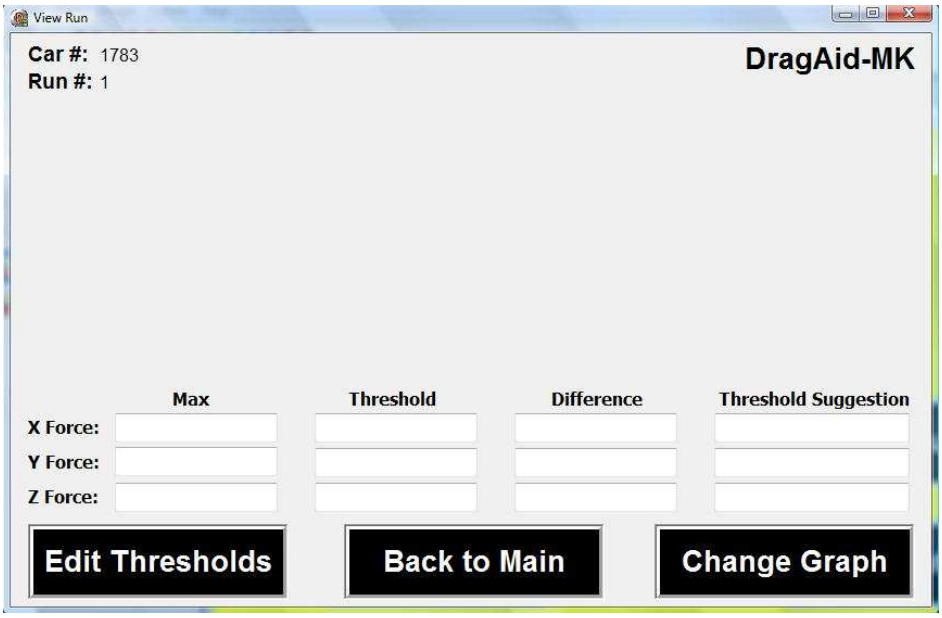


Figure 31: View Run Screen

The view run screen will eventually display a graph of the g-force data collected during the particular run that the user requested. It will also display the maximum g-forces for each axes recorded during the run. Furthermore, it will eventually suggest possible g-force thresholds for the vehicle based on the run.

The final screen is the threshold update screen. This can be seen in Figure 32.

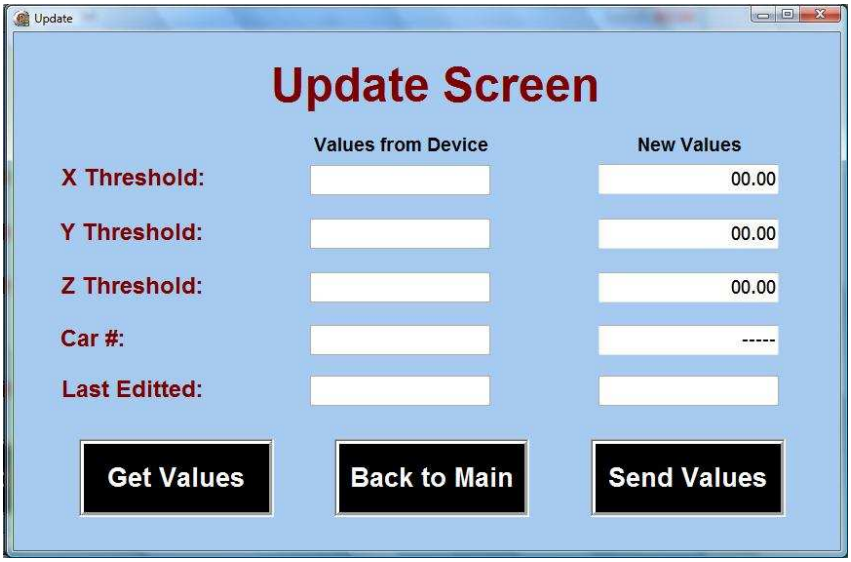


Figure 32: Update Screen

This screen will eventually be able to send new threshold values to the microcontroller and receive the current threshold values from the microcontroller.

Overall, a good start has been made to the software screens shown above. It is believed that the screen layouts will remain the same when additional functionality is added to them and the project is continued by the sponsor.

5.10 PC Board Design

In this project, two revisions of printed circuit boards (PCBs) were made. The first revision can be seen in Figure 33.

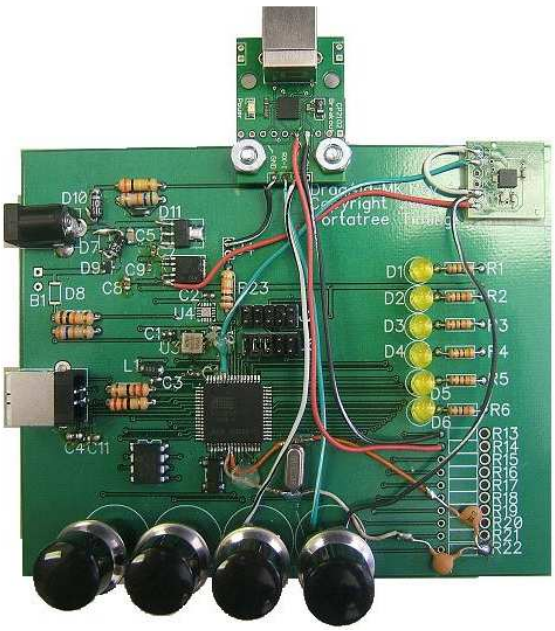


Figure 33: Initial PCB

As can be seen from the image, many wires and extra components were added to this board as various system tests were conducted and modifications were needed. The board served its main purpose, which was to test the main systems of the DragAid-MK board. Using this board, it was possible to test the accelerometers interaction with the analog to digital converter. Furthermore, it was possible to test the LEDs, switches, 32.768 kHz clock, SPI interface, and USART. Therefore, this board served the purpose of a development board. It worked very well in this application, and provided the necessary basis for developing the second revision board.

Several improvements were identified for the second revision PCB during the testing performed with the initial board. The most important improvement was that the traces for the surface mount parts should be made larger. As noted before, several additional components were required on the PCB in order to progress the board into the final stages of the project. A picture of the second revision prototype board can be seen in Figure 34.

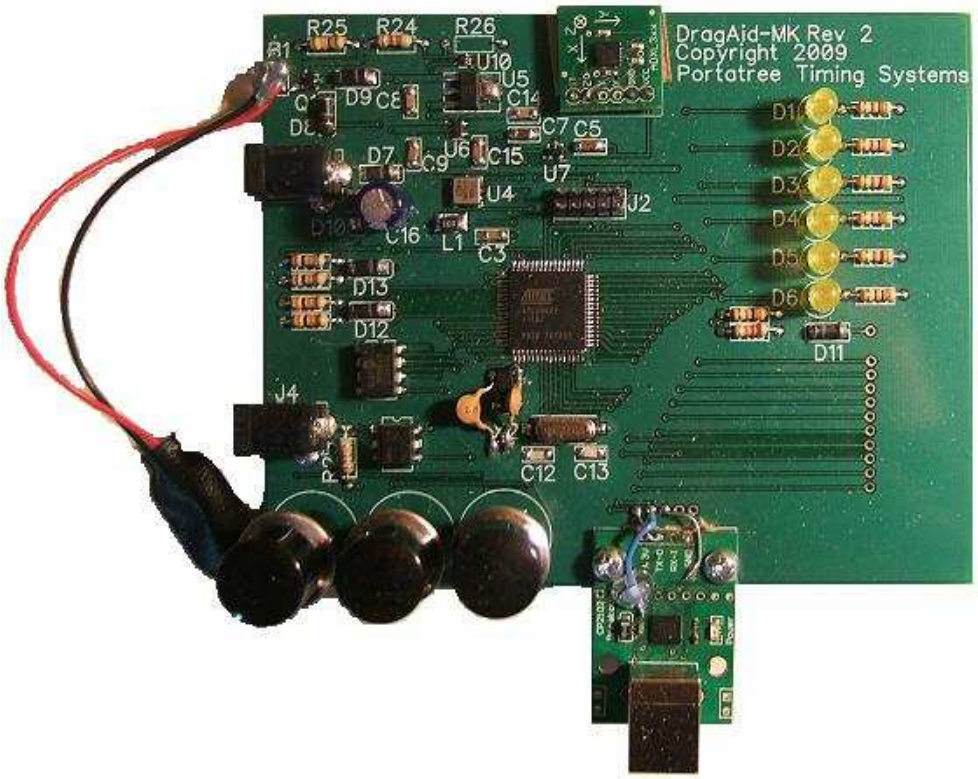


Figure 34: Final Project PCB

In order for the project to be sold on the market some slight changes would be required to this board. I notified the sponsor of these necessary changes before producing this board, and they agreed that two more board passes could be made before production.

This PCB revision was designed with stuffing the boards in mind. For this reason, the components with pins underneath (MLP and LFCSP packages) were not included on the board. Samples of the two components with these package types (ADXL330 Accelerometer and CP2102 USB to UART Bridge) were purchased from SparkFun Electronics on breakout boards. These boards were fully functional and required no changes to be used in conjunction with the DragAid-MK circuitry. This was also proven in the initial design revision seen in Figure 33.

Therefore, to make the soldering process easier, positions were set aside on the printed circuit board in order to mount the breakout boards from SparkFun. Connection holes were also provided on the PC board in order to make the process of connecting the breakout boards to the PC board easy and neat. A layout for the printed circuit board before the boards were produced can be seen in Figure 35.

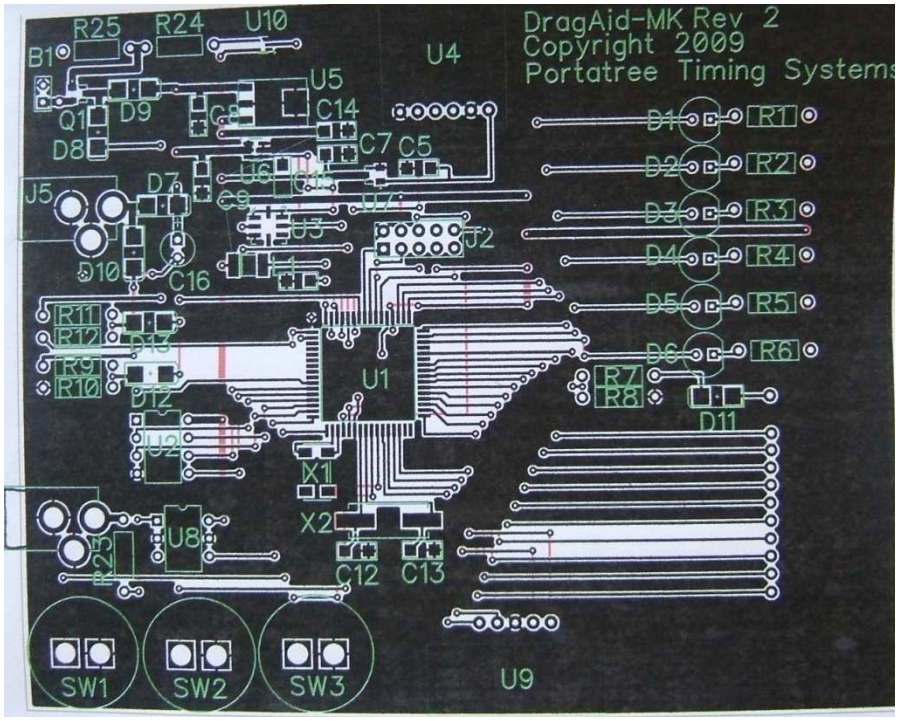


Figure 35: Printed Circuit Board Layout

Once the PC boards were designed, they were sent to Advanced Circuits in Colorado. This company provides the manufacture of two-sided PC boards for only $33. Four boards were ordered and a 5th board was sent as a bonus for the purchase. The components for the boards were also ordered from DigiKey around the same time. Upon arrival, two boards were stuffed immediately in order to be tested. This was done due to the possibility of component failure on a single board, which occurred when stuffing the first revision PC boards in September.

The most noticeable revision that would be required in a final PCB design is that the breakout boards would have to be removed from the PCB. This would save on cost and space of the PCB. The second revision is that capacitors must be added to the 32.768 kHz clock. It was noticed during testing that this clock ran faster than expected. When an oscilloscope was connected to the clock in order to test the frequency, the clock ran at the proper rate. It was quickly deduced

that capacitors were needed to cause the oscillator to oscillate at the correct rate and prevent it from running at a harmonic. At this time, tests have been conducted and no other board changes appear to be necessary; therefore, this board revision was a success in that it can support the main functionality of the DragAid-MK.

6 System Testing

Before the firmware design for the DragAid-MK was created, each key system of the DragAid-MK board was tested on the initial and final prototype boards. This was done to ensure that any problems detected in the final program were truly due to firmware and not the DragAid-MK circuitry. As of now six preliminary tests and a final system test were conducted on the DragAid-MK board. They are summarized in each of the sections below.

6.1 LED Circuit Test

The first test that was conducted was to ensure the operation of the LEDs of the DragAid-MK prototype board. In order to do this a simple light routine was created using the AVR assembler. The Assembly code for this program can be seen in Appendix A. The light program was designed to perform a simple loop. It would turn on one light and then wait a short amount of time. The delay was created by a simple software loop that performed no operations. After the delay, the light currently on was switched off and a new light was turned on. It was found that all the LEDs on the board functioned properly and also functioned as expected (active low). It was not necessary to change any aspect of the LED circuitry.

6.2 Switch Circuit Test

The next circuit tested was for the switches of the prototype board. Appendix B shows the complete test code used for the switches. This code is designed to poll for an input from the switches and then light up a different LED when each switch is pressed. Therefore, four LEDs and all four of the switches on the initial prototype board are used for this program. This program could not be mapped to the final prototype board due to port changes made between board revisions. However, this test program showed that the circuitry design for the switches was correct, and allowed the circuitry on the final board to be designed properly.

When the program was downloaded to the board, it was found to function properly. As each switch was pressed a different LED was lit on the board. There were no problems identified with this program; therefore, further tests were not conducted on the switch circuitry.

6.3 EEPROM (SPI) Circuit Test

The 25AA512 EEPROM chip is connected to the AT90USB647 through the serial peripheral interface (SPI) of the microcontroller. The User's Guide of the AT90USB647 was consulted in order to learn about the operation of the SPI interface. Through the user's guides of both the AT90USB647 and the 25AA512, it was eventually possible to create the test code seen in Appendix C.

The test code operates as follows. First it allows the user to enter 8 values onto the test board. What is meant by entering values is that the board allows the user to enter any combination of 8 button presses into the board. When a button is pressed, a corresponding LED is lit on the board in order to inform the user that the button was actually pressed. As each button is pressed, the information regarding the buttons pressed is stored in the EEPROM chip. After 8 buttons are pressed, the information is recovered from the EEPROM chip and the LEDs corresponding to the buttons that were pressed are lit in the correct order. It is similar to a primitive Simon game.

The EEPROM (SPI) test code and circuitry worked as intended. It stored the information gathered from the buttons of the DragAid-MK prototype board flawlessly. Since this program was also setup to use the switches of the initial prototype board, it could not be loaded into the final prototype board; however, as with the last test program, it verified the proper operation of the SPI circuitry. Therefore, this test program allowed the circuitry on the final prototype board to be designed properly.

6.4 Accelerometer (ADC) Circuit Test

The accelerometer or analog to digital converter (ADC) circuitry of the prototype board was tested using the ADC test code seen in Appendix D. This code tests two accelerometers: the ADXL278 and the ADXL330. There are two output lines coming from the ADXL278 representing the x and y-axes of the device. There is one output line coming from the ADXL330 corresponding to the z-axis of the device.

The ADC test code seen in Appendix D operates by lighting different LEDs on the board when g-force thresholds are passed. The thresholds are set to $\pm$1-g for each of the axes. Therefore,

no matter which side the test board is tilted to, an LED will light. When the firmware was downloaded to the board, it was found to work properly. The circuitry for each of the accelerometers was correctly designed; therefore, no circuit changes were needed in the final prototype. Furthermore, due to the proper design of this test program, the basic code from it was used in the race mode function, which can be seen in section 5.8.1, and the test mode function, which can be seen in section 5.8.5.

6.5 Real Time Clock Circuit Test

The real time clock circuit was tested to ensure that the 32.768 kHz clock was operating properly. The test code created can be seen in Appendix E. It basically consists of a real time clock interrupt that updates the number of seconds that have passed on each interrupt. The number or seconds is then displayed using binary and the LEDs available on the board. After 60 seconds pass, the variables reset and the count begins again from 0.

When downloaded to the board, the firmware was found to work properly. The 32.768 kHz clock was then tested for accuracy. This was done by carefully synchronizing the board clock to a stopwatch and determining if both clocks reached 60 seconds at the same time. The 32.768 kHz clock was found to be accurate in 60 seconds. It was due to this initial accuracy test that the circuitry from the initial prototype board was reproduced exactly on the final prototype board. Unfortunately the crystal was found to operate too quickly on the new boards, which required additional capacitors to be added to the board design.

It is uncertain whether the original tests conducted on the initial prototype board were completed improperly, or if the crystal was simply a fluke; however, since the problem has been identified and solved, this will not be discussed further at this time.

6.6 USB Circuit Test

The final test code was designed to achieve USB communication between the DragAid-MK and a PC. After two weeks of testing, successful data transfer was achieved. Transferring data between the DragAid-MK and a PC required a program to be written in both software and firmware. The firmware code can be seen in Appendix F.

The software designed to test USB communication had a very simple user interface. Before it is described in detail, information needs to be understood about the USB conversion chip on the actual DragAid-MK board. When a USB peripheral is first attached to a PC, the PC requests information about the type of device that was connected. The USB conversion chip on the DragAid-MK is designed to tell the PC that a RS232 device has been attached. This will allow PC applications to communicate with the DragAid-MK board as if they were communicating to a COM Port or in other words a normal RS232 Port.

Since the DragAid-MK board appears as an RS232 device to the software application, the software must properly setup the virtual COM port the DragAid-MK is attached to in order to properly communicate with the device. For the test, it was decided that a baud rate of 9600 should be used along with no parity bit, 8-bit data transfer, and 1 stop bit. Since the virtual COM port that the DragAid-MK will appear on is unknown at the beginning of the software (it is not known until the DragAid-MK is connected to the computer), it is necessary to have a COM port selection field in the test software. A screen shot of the test software can be seen in Figure 36.

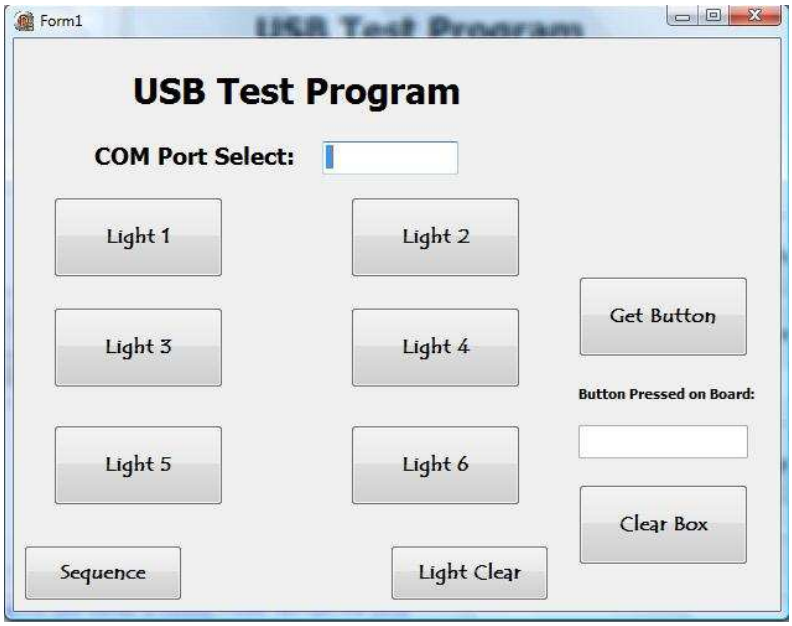


Figure 36: USB Test Software

The actual USB test software operates quite simply. When one of the light buttons is pressed, the software will send the hex code corresponding to the light number to the DragAid-MK board. Windows API functions, which are designed specifically for communicating with computer hardware, are used to send data to the virtual COM Port. Therefore, sending hex codes to the

DragAid-MK board is made simple by calling one Windows API function. The "Sequence" button will result in several LEDs being turned on and off on the DragAid-MK board. The "Light Clear" button will result in all the lights turning off.

Since USB is the underlying communication protocol of the system (despite the RS232 interface), the DragAid-MK board can never send data to the PC without the PC's permission. This is part of the USB communication protocol, which states that the master (the PC) always has control of the data bus. Therefore, in order to test the DragAid-MK's ability to send data to the PC, it was necessary for the PC to first request data. This was done with the "Get Button" button. The processor will return a value representing one of the buttons on the DragAid-MK board. This value will be displayed as text in the edit box underneath the "Get Button" button.

The USB test firmware was designed to specifically interface with the test software. After setting up the UART receive interrupt, which is used to receive data from the USB conversion chip, the processor enters a wait loop. The receive interrupt is designed to take in the 8-bit data received over the USB bus from the PC. If the data is a hex code between 1 and 6, a light on the DragAid-MK board will be lit. If the hex code received is $46, the lights on the device will be cleared. If the hex code is $47, it is known that the software wishes for the DragAid-MK to send data. The board then waits for a button to be pressed and sends information on the button press to the USB conversion chip.

The USB transfer was found to be very reliable in the tests conducted following the creation of the test software and firmware. At first an intermittent problem was found with transfer from the PC to the DragAid-MK board. Every so often, the most significant bit (MSB) of the data transfer would be misread. It was soon found that the baud rate generated by the Atmel processor was inaccurate to 7%. The system clock of the Atmel processor was raised causing the baud rate accuracy to improve such that there was only 0.2% error. When the test program was once again run, the program received the correct data on every transfer.

Due to the design of this test program, it could be loaded into the final prototype board in order to test the UART communication. It ran properly on the new boards, and confirmed the operation of the USB data transfer circuitry.

6.7 Final Firmware Test

Upon completion of the firmware for the DragAid-MK, it was downloaded into the final prototype board and tests to ensure its proper operation were conducted. The first test completed was to ensure that the device entered race mode. In order to conduct this test, it was necessary to use a second 12 volt power supply, which would simulate the input from the two-step. The firmware was designed such that the two-step must be activated (high) for at least 5 seconds before the device would enter race mode. Therefore, the 12 volt power supply was connected to the two-step input and a 5 second count was begun using a stopwatch. It was found that the race mode LED lit at approximately 5 seconds after the two-step was activated. Therefore, the two-step trigger of race mode worked properly.

The second test conducted was to ensure that race mode lasted the proper amount of time (15 seconds from the release of the two-step). In order to do this, the system and stop-watch were reset. Once everything was ready, the two-step was activated again using the 12 volt power supply. After 5 seconds passed, the two-step was deactivated to signal a race start and the stop watch was started. At approximately 15 seconds according to the stop watch, the race mode LED shut off, signaling the end of race mode; therefore, it was found that the system worked as intended.

The next portion of the system that was tested was the two-step bypass feature of pit road mode. In order to do this, the two-step bypass button was pressed. It was immediately noticed that the two-step bypass LED was lit as expected. The 12 volt power supply was then applied to the two-step input to symbolize activating the two-step. It was found that the race mode LED did not come on even after 5 seconds passed with the two-step activated. This showed that the system was properly bypassing two-step input.

The next test conducted was on test mode. To begin this test, the test mode button of the DragAid-MK board was pressed. As expected the test mode LED came on. The board was

tilted to the front and back first. When tilted forward, the first LED of the board lit (exceeded upper x-threshold), and when titled backward, the second LED lit. When tilted to the left, the third LED of the board lit, and when tilted to the right the fifth lit. Finally the board was held upside down, and it was found that the 6^{th} LED lit. Therefore, test mode was found to work as expected.

The test of the master kill switch required a digital multimeter to be used. Before the test was conducted, the voltage levels of all ports designated to send off signals were tested. It was found that all these signals had low outputs, as expected. The master kill switch was then pressed, and the values were once again recorded through the digital multimeter. It was found that each output pin designated to shut down the vehicle had a voltage level of 4.96 volts. This is as expected, a high voltage signal. Therefore, the master kill switch of the device works as intended.

The next feature of the device was slightly harder to test. This is the system shut down feature, or in other words, the ability of the DragAid-MK to shut itself down after the ignition system of the vehicle has been removed. It was decided that the best way to do this would be to connect a battery to the DragAid-MK system along with a 12 volt power source representing the ignition system. The device should then be entered into race mode using another 12 volt power source to represent the two-step. After race mode has begun, the 12 volt ignition should be removed. The system is designed such that race mode will be completed before the device will shut down. Therefore, even with the 12 volt power source representing the ignition removed, the DragAid-MK system should continue to run. This test resulted in proof of the correct operation of the system shut down feature of the DragAid-MK.

The only areas that remained to be tested at this point were the data saving ability of the DragAid-MK, and the USB transfer or download module. It was decided that these two features of the DragAid-MK would be easiest to test together. A simple test program was created that would send commands to the DragAid-MK and determine if the DragAid-MK produced the proper response. The layout for this test program can be seen in Figure 37.

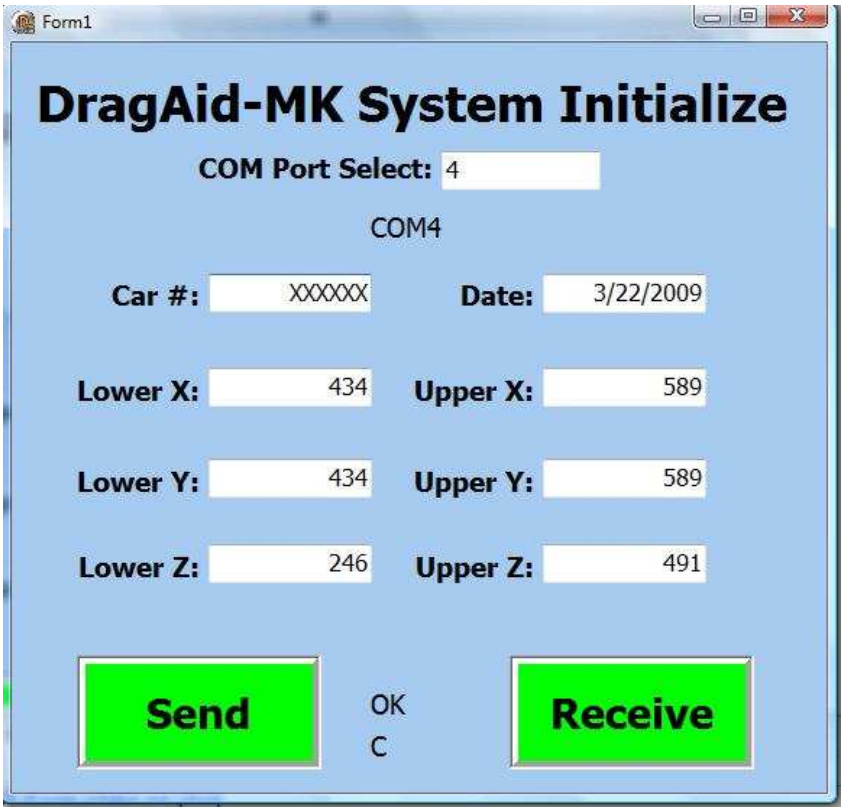


Figure 37: Initialize Test Program

This test program was designed such that different responses from the DragAid-MK could be viewed. As of right now, it tests the initialize portion of the *DataReceive* command in the download module of the DragAid-MK as well as the threshold send portion of the *DataSend* command. When the program is started, the COM port that the DragAid-MK is connected to must be selected in order for the system to run properly. As can be seen in Figure 37, when this is set in the program, a label appears that confirms the port number the device is connected to.

The first button that was pressed is the "Send" button. This button will send the initialize command to the DragAid-MK causing it to write initial threshold values to its variables and EEPROM. When the DragAid-MK receives the command it sends "OK" back to the PC. The initialization program created shows that the PC received these characters from the DragAid. After all values are written to the EEPROM of the DragAid-MK a 'C' is sent to the PC. This value is also displayed upon being received as can be seen by the third label on the screen of the initialization program.

Once the system initialization operation appeared to be working properly, the DragAid-MK was shut down and then turned back on. This was done to erase all the RAM values of the system. On system startup, all data held in RAM during the standard operation of the DragAid-MK is read from the EEPROM of the DragAid-MK (which is nonvolatile). Therefore, through cycling power, an attempt was made to ensure that the SPI interface of the DragAid-MK was working properly. The device was reconnected to the PC through a USB cable, and the "Receive" button of the initialization program was pressed. As expected, all values written to the DragAid-MK during initialization were received by the initialization software.

The values received by the software may seem to be rather large and, therefore, incorrect upon first looking at the initialization software figure (Figure 37). This is due to the fact that the DragAid-MK only deals with values that are understandable by the analog to digital converter of the processor. The analog to digital converter cannot deal in decimal points or negatives and only understand values from 0 to 1023. Therefore, all voltage values produced by the accelerometers (which themselves represent g-force values) are mapped to a value in the range above in order to be understood by the ADC. In a final program, these values would have to be converted back to g-force values in order to allow the user to easily work with them.

Of course, all these systems did not work properly on the first try; however, they operated close enough to the desired function that only minor adjustments were necessary. Some areas where initial error was found included preventing switch bounce. Eventually extra precaution was taking by disabling the interrupts connected to the switches for a certain period after the switches were pressed. This effectively eliminated switch bounce. Another minor mistake was a reversal of the two-step input polarity. This error was quickly and easily fixed.

The most difficult problem to identify was a data transfer error with the USB interface. It was found that the PC software would only receive the first two bytes transferred to it from the DragAid-MK firmware. When the code in the DragAid-MK was stepped through, it was found that the PC received all data sent by the DragAid-MK. If the code in the DragAid-MK was partially stepped through, the PC would receive all bytes that were stepped through and two bytes following this point. It was eventually determined that the DragAid-MK, when

transferring data consecutively to the PC, transferred the data too quickly for the PC to handle. A 1 millisecond delay was added after each transmit in the DragAid-MK firmware, and it was found that all data was transferred perfectly to the PC.

Overall, the firmware of the system withstood initial testing. The functionality of the device was successfully confirmed, and it was decided that no major changes would be necessary to the logic of the operation of the device. The tests also confirmed that the hardware of the DragAid-MK was once again designed properly. Only minor adjustments, all described in the sections above, will be needed in the final prototype board that will be designed by the project sponsor.

7 Future

This project has extraordinary potential for future improvement, which is also the intention of the project sponsor. The area where the most focus will be applied is in software development for the DragAid-MK. The project sponsor has recognized that the firmware for the device is completed and functional and will most likely only require minor adjustments as the project design comes to a close. They, therefore, wish to dedicate their resources to an area where considerable improvement can still be made. They approve of the software layouts that were created for the program during this project and plan on expanding upon these layouts in the future.

As stated in section 5.10 of the report, another revision of the printed circuit board will be required before the product can be sold. Furthermore, the design of the solid-state relay vehicle shut-offs will have to be completed. This was not part of the project conducted for this MQP due to the complexity of these devices. A project engineer that works for the project sponsor has considerable experience with solid state relays as well as race cars. He is in charge of the design of the solid state relay system shutoffs. All data that he needs to interface them with the DragAid-MK should be available in this report.

A design for the case of the DragAid-MK must also be completed in the near future. Mechanical engineers who will be continuing the project have already come up with several ideas that will protect the delicate circuitry of the device from all crash situations. The case will also be required to prevent electrical and magnetic noise in the race car from interfering with the sensors of the crash detector. A final requirement of the case is that it is rigid enough to detect impacts on the vehicle, as well as designed to be easy to mount in the race vehicle.

Due to the timeline of this project, it was not possible to test the DragAid-MK on an actual race car. Racing season does not begin until late April, and most racers with cars that reach reasonable speeds do not begin racing until early to mid-May for safety reasons. Therefore, considerable testing on actual race vehicles was not conducted during this project. Test data would be extremely beneficial to this project. The more test data received, the more the

DragAid-MK can be improved. The device will not be able to be released to racers until sufficient test data has been received and studies on this data have been conducted.

A final improvement to the DragAid-MK that should be considered in the future of the project is the use of multiple systems. Since the DragAid-MK is a safety device, it is extremely important that it does not receive false accelerometer readings that result in a system shut down or a failure of the system to shut down. For this reason, multiple systems would be extremely beneficial. The complete system would have 3 sets of accelerometers working in parallel. The processor would be required to read data from each set of accelerometers and compare the results among all three sets.

The general idea is to create a system of checks and balances. A system shut down would only occur when 2 out of the 3 sets of accelerometers show that g-force thresholds have been exceeded. This will reduce the chances of receiving an erratic reading that causes problems in the operation of the system.

Overall, it seems like all improvements mentioned for the DragAid-MK are possible. Hopefully with continued work they will become part of the device in the near future.

8 Conclusion

In the end, this project resulted in the construction of the DragAid-MK, crash detection device for race cars. The device is able to meet many requirements specified for the entire product as well as all requirements specified for the MQP. The DragAid-MK was designed due to the increasing number of fatalities in drag racing that result from the ignition system and fuel system of race vehicles continuing to run even after a crash has occurred. This often causes fire or explosion in the race vehicle, which endangers the life of both the racer and safety crew.

The DragAid-MK, designed in this project, is a data analyzer and master kill switch for race cars. It is designed to collect and record g-force data that a drag race vehicle experiences during a run. G-force data is collected through two accelerometers operating in parallel in the DragAid-MK system. The first accelerometer, an ADXL278, is able to record x-axis and y-axis g-forces up to 50g. The second accelerometer, an ADXL330, is able to record z-axis g-force data up to 3g. The device saves g-force data to a serial EEPROM every .02 seconds of a race. After a race has been triggered in the DragAid-MK system, the device will record 15 seconds of g-force data.

A unique feature of the DragAid-MK, which makes it adaptable to different types of race vehicles, is the adjustable vehicle shut off thresholds. The DragAid-MK is designed such that it shuts down the ignition system and fuel system of a vehicle when g-force levels on the vehicle surpass certain thresholds. Different race vehicles experience different g-force levels depending on the setup of the car; therefore, the DragAid-MK system allows thresholds to be adjusted by the user of the vehicle. This will allow a racer to adapt the DragAid-MK to his car, making the device safer and more reliable overall.

In order to allow a racer to make reasonable adjustments to the thresholds of the DragAid-MK, it is necessary for the driver to know all g-force levels recorded by the DragAid-MK during a run. For this reason, the DragAid-MK is also a data acquisition system. At the end of a race, a racer can extract g-force data from the DragAid-MK using a standard Type-A to Type-B USB cable, which will connect the DragAid-MK to a PC. In the future, this data will be displayed to the racer through a series of graphs.

Although this project comes to a close, the plans for future development of the DragAid-MK are clear. Focus must first be set on the software interface for the DragAid-MK, the solid-state relay shut-offs, and the case of the device. Once these aspects have been completed, extensive testing can be done using actual race vehicles. Multiple systems should also be considered, although this is not necessary for the completion or initial tests of the DragAid-MK.

Overall, this project resulted in a crash detection device for race cars that has the potential of saving many lives in the sport of drag racing. With extended work on the project, the DragAid-MK system could be applied to other divisions of motorsports and passenger vehicles as well. Working on the DragAid-MK during this project has been extremely rewarding. I have learned many new concepts, which will aid me in my professional career. Most importantly, I feel that I have been part of a crucial development in safety systems for sportsmen drag racers, which until now was an area that was extremely underdeveloped.

References

1 http://www.moroso.com/catalog/categorydisplay.asp?catcode=77191

2 http://www.fireblades.org/forums/honda-rc51/48982-tilt-sensor.html

3 http://www.zercustoms.com/news/Ford-Blue-Box-On-All-2008-NHRA-Nitro-Cars.html

4 http://www.zercustoms.com/photos/Ford-Blue-Box-On-All-2008-NHRA-Nitro-Cars/Ford-Blue-Box-2008-NHRA-Nitro-Cars-1.jpg.html
http://www.myrideisme.com/Blog/ford-and-nhra-team-up-on-safety/

5 http://www.nhra.com/story/

6 http://www.zoomerdaily.com/?tag=shutoff-controller

7 http://www.carcrashfires.com/index.html

8 http://www.analog.com/en/automotive-solutions/crash-detection/applications/index.html

9 www.mathworks.com

10 http://www.sparkfun.com/commerce/images/products/00198-03-L.jpg

11 ADXL278 Data Sheet Rev. A 5/2005

12 ADXL330 Data Sheet Rev. A 9/2006

13 AT90USB647 Device User's Guide

14 CP2102 USB to UART Bridge Data Sheet

15 25AA512 EEPROM Data Sheet

Appendix A: LED Test Code

```
.include "usb647def.inc"
.cseg
.org  0
      jmp SetupLEDTest
.org  $04C
SetupLEDTest:
      ldi r16, $10      ; Initialize the Stack Pointer to the end of
      out SPH, r16      ; the internal SRAM 0x10FF
      ldi r16, $FF
      out SPL, r16
      in r16, DDRA            ; Set PORTA as output
      ori r16, $3F
      out DDRA, r16
      in r16, PORTA           ; Turn all the lights off to start
      ori r16, $3F
      out PORTA, r16
StartLEDTest:
      cbi PORTA, 0            ; Turn LED 0 on
      call SWDelay            ; Delay
      sbi PORTA, 0            ; Turn LED 0 off
      cbi PORTA, 1            ; Turn LED 1 on
      call SWDelay            ; Delay
      sbi PORTA, 1            ; Turn LED 1 off
      cbi PORTA, 2            ; Turn LED 2 on
      call SWDelay            ; Delay
      sbi PORTA, 2            ; Turn LED 2 off
      cbi PORTA, 3            ; Turn LED 3 on
      call SWDelay            ; Delay
      sbi PORTA, 3            ; Turn LED 3 off
      cbi PORTA, 4            ; Turn LED 4 on
      call SWDelay            ; Delay
      sbi PORTA, 4            ; Turn LED 4 off
      cbi PORTA, 5            ; Turn LED 5 on
      call SWDelay            ; Delay
      sbi PORTA, 5            ; Turn LED 5 off
      jmp StartLEDTest  ; Loop again
SWDelay:
      clr r17
SWDloop:
      clr r16
      inc r17
SWDloop1:
      inc r16
      cpi r16, $FF
      brne SWDloop1
      cpi r17, $FF
      brne SWDloop
      ret
.exit
```

Appendix B: Switch Test Code

```
.include    "usb647def.inc"
.org  $0
      jmp SetupSwitchTest
.org  $4C
SetupSwitchTest:
      ldi r16, $10      ; Initialize the Stack Pointer to the end of
      out SPH, r16      ; the internal SRAM 0x10FF
      ldi r16, $FF
      out SPL, r16
      in r16, DDRA            ; Setup PORTA pins 0-3 as output
      ori r16, $0F
      out DDRA, r16
      in r16, PORTA           ; Turn off the 4 LEDs
      ori r16, $0F
      out PORTA, r16
      in r16, DDRD            ; Setup PORTD pins 0-3 as input
      andi r16, $F0
      out DDRD, r16
      in r16, PORTD           ; Turn on the pull-up resistors for
      ori r16, $0F            ; PORTD pins 0-3
      out PORTD, r16
StartSwitchTest:
      in r16, pind
      ori r16, $F0
      cpi r16, $FF
      breq StartSwitchTest
      cpi r16, Button0
      brne StartSwitchTest1
      ldi r16, $0F
      out PORTA, r16
      cbi PORTA, 0
StartSwitchTest1:
      cpi r16, Button1
      brne StartSwitchTest2
      ldi r16, $0F
      out PORTA, r16
      cbi PORTA, 1
StartSwitchTest2:
      cpi r16, Button2
      brne StartSwitchTest3
      ldi r16, $0F
      out PORTA, r16
      cbi PORTA, 2
StartSwitchTest3:
      cpi r16, Button3
      brne StartSwitchTest
      ldi r16, $0F
      out PORTA, r16
      cbi PORTA, 3
      jmp StartSwitchTest
.equ  Button0     = $FE
.equ  Button1     = $FD
.equ  Button2     = $FB
.equ  Button3     = $F7
.exit
```

Appendix C: EEPROM (SPI) Test Code

```
.include "usb647def.inc"
.org  $0
      jmp   SetupSPITest
.org  $4C
SetupSPITest:
; Setup Stack Pointer
      ldi r16, $10      ; Initialize the Stack Pointer to the end of
      out SPH, r16      ; the internal SRAM 0x10FF
      ldi r16, $FF
      out SPL, r16
; Setup LEDs
      in r16, DDRA      ; Make Port A an output port
      ori r16, $0F
      out DDRA, r16
      in r16, PORTA     ; Output high to all of Port A
      ori r16, $0F
      out PORTA, r16
; Setup Switches
      in r16, DDRD      ; Make Port D an input port
      andi r16, $F0
      out DDRD, r16
      in r16, PORTD     ; Activate the pull-up resistors
      ori r16, $0F
      out PORTD, r16
; Setup SPI & Memory Pointer
      in r16, DDRB      ; Make Port B Pins 5-4,0-2 output and 3 input
      andi r16, $F7
      ori r16, $37
      out DDRB, r16
      sbi PORTB, 0
      sbi PORTB, 4
      sbi PORTB, 5
      ldi r16, (1<<SPE)|(1<<MSTR)
      out SPCR, r16     ; Setup the SPI
StartSPITest:
      in r16, PORTA
      ori r16, $0F
      out PORTA, r16
      clr r17
      sts MemPtL, r17
      sts MemPtH, r17
SPITest1:
      in r16, PIND      ; Put the button pressed into r16
      ori r16, $F0
      cpi r16, $FF
      breq SPITest1
      cpi r16, $FE
      breq SPITest2
      cpi r16, $FD
      breq SPITest2
      cpi r16, $FB
      breq SPITest2
      cpi r16, $F7
      brne SPITest1
SPITest2:
```

```
      inc r17
      sts Data, r16     ; Store the button in Data
      call StoreData    ; Save it in EEPROM
      andi r16, $0F     ; Turn on the corresponding LED
      out PORTA, r16
SPITest3:
      in r16, PIND
      ori r16, $F0
      cpi r16, $FF
      brne SPITest3     ; Wait until the button is no longer pressed
      call SWDelay
      ldi r16, $0F      ; Turn Off all LEDs
      out PORTA, r16
      cpi r17, $08      ; See if 4 choices made
      brne SPITest1     ; If not continue
      clr r17
      sts MemPtL, r17   ; Clear the memory
SPITest4:
      call ReadData     ; Read the Data at the First Memory Position
      lds r16, Data     ; Load r16 with the Data
      andi r16, $0F
      out PORTA, r16    ; Turn on the corresponding LED
      call SWDelay      ; Slight Delay
      inc r17
      cpi r17, $08
      brne SPITest4
      jmp StartSPITest
ReadData:
      cbi PORTB, 0      ; Set CS low
      ldi r16, $03
      out SPDR, r16     ; Send READ instruction
      call TransmissionComplete
      lds r16, MemPtH
      out SPDR, r16     ; Send High Byte of Memory Pointer
      call TransmissionComplete
      lds r16, MemPtL
      out SPDR, r16     ; Send Low Byte of Memory Pointer
      call TransmissionComplete
      clr r16
      out SPDR, r16     ; Send Nothing to Receive the Data
      call TransmissionComplete
      sts Data, r16     ; Store the Data into r16
      sbi PORTB, 0      ; Set CS high
      lds r16, MemPtL
      inc r16
      sts MemPtL, r16
      ret
StoreData:
      push r16
      cbi PORTB, 0      ; Set CS low
      ldi r16, $06
      out SPDR, r16     ; Send WREN instruction
      call TransmissionComplete
      sbi PORTB, 0      ; Set CS high
      nop
      nop
      nop
```

```
      nop
      nop                     ; Slight Delay to give EEPROM time
      cbi PORTB, 0      ; Set CS low
      ldi r16, $02
      out SPDR, r16     ; Send WRITE instruction
      call TransmissionComplete
      lds r16, MemPtH
      out SPDR, r16     ; Send High Byte of Memory Pointer
      call TransmissionComplete
      lds r16, MemPtL
      out SPDR, r16     ; Send Low Byte of Memory Pointer
      call TransmissionComplete
      lds r16, Data
      out SPDR, r16     ; Send Data Byte to Memory
      call TransmissionComplete
      sbi PORTB, 0      ; Set CS high
      lds r16, MemPtL
      inc r16
      sts MemPtL, r16
      pop r16
      ret
TransmissionComplete:
      in r16, SPSR      ; Load r16 with the Status Register
      andi r16, $80
      cpi r16, $00      ; See if transmission is complete
      breq TransmissionComplete
      in r16, SPDR      ; Load data into r16
      ret
SWDelay:
      clr r18
SWDelay1:
      clr r16
SWDelay2:
      inc r16
      cpi r16, $FF
      brne SWDelay2
      inc r18
      cpi r18, $FF
      brne SWDelay1
      ret
.equ  MemPtH      = $100
.equ  MemPtL      = $101
.equ  Data  = $102
.exit
```

Appendix D: Accelerometer (ADC) Test Code

```
.include "usb647def.inc"
.org  $0
      jmp   SetupADCTest
.org  $4C
SetupADCTest:
      ldi r16, $10      ; Initialize the Stack Pointer to the end of
      out SPH, r16      ; the internal SRAM 0x10FF
      ldi r16, $FF
      out SPL, r16
      in r16, DDRA      ; Set Port A as an output port
      ori r16, $3F
      out DDRA, r16
      in r16, PORTA     ; Output high to keep all the LEDs off
      ori r16, $3F
      out PORTA, r16
      clr r16
      sts ADCSRB, r16
      ldi r16, (1<<ADC0D)|(1<<ADC1D)|(1<<ADC2D)
      sts DIDR0, r16
StartADCTest:
; X Axis
      clr r16
      sts ADMUX, r16    ; Select X- axis of accelerometer
      ldi r16, (1<<ADEN)|(1<<ADSC)
      sts ADCSRA, r16   ; Start the conversion
ADCTest1:
      lds r16, ADCSRA
      andi r16, $10
      cpi r16, $00
      breq ADCTest1     ; Wait here until the conversion is complete
      lds r16, ADCL
      lds r17, ADCH     ; Take the results
      sts XAxisH, r17   ; Store the results
      sts XAxisL, r16
      lds r16, ADCSRA   ; Clear the ADC flag
      ori r16, $10
      sts ADCSRA, r16
; Y Axis
      ldi r16, (1<<MUX0)
      sts ADMUX, r16    ; Select Y-axis of accelerometer
      ldi r16, (1<<ADEN)|(1<<ADSC)
      sts ADCSRA, r16   ; Start the conversion
ADCTest2:
      lds r16, ADCSRA
      andi r16, $10
      cpi r16, $00
      breq ADCTest2     ; Wait here until the conversion is complete
      lds r16, ADCL
      lds r17, ADCH     ; Take the results
      sts YAxisH, r17   ; Store the results
      sts YAxisL, r16
      lds r16, ADCSRA   ; Clear the ADC flag
      ori r16, $10
      sts ADCSRA, r16
; Z Axis
```

```
      ldi r16, (1<<MUX1)
      sts ADMUX, r16    ; Select Z-axis of accelerometer
      ldi r16, (1<<ADEN)|(1<<ADSC)
      sts ADCSRA, r16   ; Start the conversion
ADCTest3:
      lds r16, ADCSRA
      andi r16, $10
      cpi r16, $00
      breq ADCTest3     ; Wait here until the conversion is complete
      lds r16, ADCL
      lds r17, ADCH     ; Take the results
      sts ZAxisH, r17   ; Store the results
      sts ZAxisL, r16
      lds r16, ADCSRA   ; Clear the ADC flag
      ori r16, $10
      sts ADCSRA, r16
; Tests
      lds r16, XAxisL   ; Load r16 and r17 with the ADC conversion results
      lds r17, XAxisH
      ldi r18, $0D      ; Load r18 and r19 with upper threshold
      ldi r19, $02
      cp  r17, r19
      brlo ADCTest5     ; If r17 is larger than r19 then Light On
      brne ADCTest4     ; If r17 is not equal to r19 then below Threshold
      cp    r16, r18
      brlo ADCTest5     ; If r16 is larger than r18 then Light On
ADCTest4:
      in r16, PORTA     ; All lights off
      ori r16, $3F
      out PORTA, r16
      cbi PORTA, 0      ; Light on
      jmp StartADCTest; Go back to start
ADCTest5:
      ldi r18, $F8      ; Load r18 and r19 with the lower threshold
      ldi r19, $01
      cp    r19, r17
      brlo ADCTest7     ; If r19 is lower than r17 then above threshold
      brne ADCTest6     ; If r19 is not equal to r17 then below threshold
      cp  r16, r18      ; If r16 is the same or higher then above threshold
      brsh ADCTest7
ADCTest6:
      in r16, PORTA     ; All lights off
      ori r16, $3F
      out PORTA, r16
      cbi PORTA, 1      ; Light on
      jmp StartADCTest; Go back to start
ADCTest7:
      lds r16, YAxisL   ; Load r16 and r17 with the ADC conversion results
      lds r17, YAxisH
      ldi r18, $07      ; Load r18 and r19 with upper threshold
      ldi r19, $02
      cp r17, r19
      brlo ADCTest9     ; If r17 is lower than r19 then below threshold
      brne ADCTest8     ; If r17 is not equal to r19 then above threshold
      cp    r16, r18
      brlo ADCTest9     ; If r16 is lower than r18 then below threshold
ADCTest8:
```

```
      in    r16, PORTA  ; All lights off
      ori r16, $3F
      out PORTA, r16
      cbi PORTA, 2      ; Light on
      jmp StartADCTest; Go back to start
ADCTest9:
      ldi r18, $F8      ; Load r18 and r19 with the lower threshold
      ldi r19, $01
      cp r19, r17
      brlo ADCTest11    ; If r19 is lower than r17 then above threshold
      brne ADCTest10    ; If r19 is not equal to r17 then below threshold
      cp r16, r18       ; If r16 is the same or higher then above threshold
      brsh ADCTest11
ADCTest10:
      in r16, PORTA     ; All lights off
      ori r16, $3F
      out PORTA, r16
      cbi PORTA, 3      ; Light on
      jmp StartADCTest; Go back to start
ADCTest11:
      lds r16, ZAxisL   ; Load r16 and r17 with the ADC conversion results
      lds r17, ZAxisH
      ldi r18, $70      ; Load r18 and r19 with upper threshold
      ldi r19, $01
      cp r17, r19
      brlo ADCTest13    ; If r17 is lower than r19 then below threshold
      brne ADCTest12    ; If r17 is not equal to r19 then above threshold
      cp r16, r18       ; If r16 is lower than r18 then below threshold
      brlo ADCTest13
ADCTest12:
      in    r16, PORTA  ; All lights off
      ori r16, $3F
      out PORTA, r16
      cbi PORTA, 4      ; Light on
      jmp StartADCTest; Go back to start
ADCTest13:
      ldi r18, $F6      ; Load r18 and r19 with the lower threshold
      clr r19
      cp r19, r17
      brlo ADCTest15    ; If r19 is lower than r17 then above threshold
      brne ADCTest14    ; If r19 is not equal to r17 then below threshold
      cp r16, r18       ; If r16 is the same or higher then above threshold
      brsh ADCTest15
ADCTest14:
      in    r16, PORTA  ; All lights off
      ori r16, $3F
      out PORTA, r16
      cbi PORTA, 5      ; Light on
ADCTest15:
      jmp   StartADCTest; Go back to start
.equ  XAxisH      = $100
.equ  XAxisL      = $101
.equ  YAxisH      = $102
.equ  YAxisL      = $103
.equ  ZAxisH      = $104
.equ  ZAxisL      = $105
.exit
```

Appendix E: Real Time Clock Test Code

```
.include "usb647def.inc"
.org $0
      jmp   SetupRTCTest
.org OVF2addr
      jmp RTCtick
.org $4C
SetupRTCTest:
; Setup Stack Pointer
      ldi r16, $10      ; Initialize the Stack Pointer to the end of
      out SPH, r16      ; the internal SRAM 0x10FF
      ldi r16, $FF
      out SPL, r16
; Enable Interrupts
      sei
; Setup LEDs
      in r16, DDRA      ; Make Port A an output port
      ori r16, $3F
      out DDRA, r16
      in r16, PORTA     ; Output high to all of Port A
      ori r16, $3F
      out PORTA, r16
; Setup RTC Clock
      ldi r16, (1<<AS2)
      sts ASSR, r16
      clr r16
      sts secs, r16
      sts TCCR2A, r16
      sts TCNT2, r16
      ldi r16, (1<<TOIE2)
      sts TIMSK2, r16
      ldi r16, (1<<CS22)|(1<<CS21)
      sts TCCR2B, r16
StartRTCTest:
      jmp StartRTCTest
RTCtick:
      push r16
      in r16, sreg
      push r16
      lds r16, secs
      inc r16
      sts secs, r16
      cpi r16, $3C
      brne RTCtick1
      clr r16
      sts secs, r16
      ldi r16, $FF
RTCtick1:
      com r16
      out PORTA, r16
      pop r16
      out sreg, r16
      pop r16
      reti
.equ secs = $100
.exit
```

Appendix F: USB Test Code

```
.include "usb647def.inc"
.org  $0
      jmp SetupUSBTest
.org  URXC1addr
      jmp ReceivedData
.org  $4C
SetupUSBTest:
      cli
; Setup the Stack Pointer
      ldi r16, $10      ; Initialize the Stack Pointer to the end of
      out SPH, r16      ; the internal SRAM 0x10FF
      ldi r16, $FF
      out SPL, r16
; Setup the LEDs
      in r16, DDRA      ; Setup PORTA pins 0-5 as output
      ori r16, $3F
      out DDRA, r16
      in r16, PORTA     ; Turn off the 6 LEDs
      ori r16, $3F
      out PORTA, r16
; Setup the Switches
      in r16, DDRD      ; Setup PORTD pins 0-1 as input
      andi r16, $FC
      out DDRD, r16
      in r16, PORTD     ; Turn on the pull-up resistors for
      ori r16, $03      ; PORTD pins 0-1
      out PORTD, r16
; Setup the USART
      in r16, DDRD
      andi r16, $FB
      ori r16, $08
      out DDRD, r16     ; Properly set data direction on UART pins
      ldi r16, (1<<UCSZ11)|(1<<UCSZ10)
      sts UCSR1C, r16   ; Set parity, # bits, and # stop bits
      clr r16
      sts UBRR1H, r16
      ldi r16, 6
      sts UBRR1L, r16   ; Set the baud rate
      ldi r16, (1<<RXEN1)|(1<<RXCIE1)|(1<<TXEN1)
      sts UCSR1B, r16   ; Setup UART to transmit and receive
      clr r16
      sts UDR1, r16
      sts UCSR1A, r16
      sei               ; Enable Interrupts
      clr r17
USBTest:
      cpi r17, $47      ; Wait for $47 (code for device to transmit)
      brne USBTest      ; to be received
WaitforButton:
      in r16, pind      ; Loop until user presses a button
      ori r16, $FC
      cpi r16, $FF
      breq WaitforButton
      andi r16, $33
WaittoTransmit:
```

```
      lds r17, UCSR1A   ; Wait for the transmit line to be available
      andi r17, $20
      cpi r17, $20
      brne WaittoTransmit
      sts UDR1, r16     ; Send the button pressed to the PC
      clr r17
      jmp USBTest       ; Return to the loop waiting for command $47
ReceivedData:           ; Interrupt handling routine for receiving data
      push r16          ; Store r16 and the status register
      in r16, sreg
      push r16
      in r16, PORTA     ; Turn off all LEDs
      ori r16, $3F
      out PORTA, r16
      lds r16, UDR1     ; Load the value received into r16
      andi r16, $7F
      cpi r16, $46      ; Compare the value to $46
      breq EndTest      ; $46 = command for turning off LEDs, done, so end
      cpi r16, $31      ; If '1' received turn on LED 1
      brne Test2
      cbi PORTA, 0
Test2:
      cpi r16, $32      ; If '2' received turn on LED 2
      brne Test3
      cbi PORTA, 1
Test3:
      cpi r16, $33      ; If '3' received turn on LED 3
      brne Test4
      cbi PORTA, 2
Test4:
      cpi r16, $34      ; If '4' received turn on LED 4
      brne Test5
      cbi PORTA, 3
Test5:
      cpi r16, $35      ; If '5' received turn on LED 5
      brne Test6
      cbi PORTA, 4
Test6:
      cpi r16, $36      ; If '6' received turn on LED 6
      brne Test7
      cbi PORTA, 5
Test7:
      cpi r16, $47      ; If $47 received load the value into r17
      brne EndTest      ; so the main program will know
      ldi r17, $47
EndTest:                ; Exit sequence
      clr r16           ; Clear the receive flag to show it has been handled
      sts UCSR1A, r16
      pop r16           ; Return saved registers
      out sreg, r16
      pop r16
      reti              ; Return from the interrupt
.exit
```

Appendix G: RollOverDevice.asm (Main Code)

```
.include "usb647def.inc"
; Vector Table (Beginning of memory)
.org  $0
      jmp MainEntry     ; On Reset go to the start of the program
.org  INT0addr
      jmp MasterKill    ; Master Kill Switch Interrupt
.org  INT1addr
      jmp TwoStepInt    ; Two Step On/Off Interrupt
.org  INT2addr
      jmp Unused
.org  INT3addr
      jmp Unused
.org  INT4addr
      jmp   Unused
.org  INT5addr
      jmp Unused
.org  INT6addr
      jmp Unused
.org  INT7addr
      jmp Unused
.org  PCI0addr
      jmp TestInt       ; Test Interrupt or Two-Step Pressed
.org  USB_GENaddr
      jmp Unused
.org  USB_COMaddr
      jmp Unused
.org  WDTaddr
      jmp Unused
.org  OC2Aaddr
      jmp RTCInt
.org  OC2Baddr
      jmp Unused
.org  OVF2addr
      jmp RTCInt        ; Real Time Clock Interrupt
.org  ICP1addr
      jmp Unused
.org  OC1Aaddr
      jmp Unused
.org  OC1Baddr
      jmp Unused
.org  OC1Caddr
      jmp Unused
.org  OVF1addr
      jmp Unused
.org  OC0Aaddr
      jmp Unused
.org  OC0Baddr
      jmp Unused
.org  OVF0addr
      jmp Unused
.org  SPIaddr
      jmp Unused
.org  URXC1addr
      jmp Download      ; Download Interrupt Triggered
.org  UDRE1addr
```

```
      jmp Unused
.org  UTXC1addr
      jmp Unused
.org  ACIaddr
      jmp Unused
.org  ADCCaddr
      jmp Unused
.org  ERDYaddr
      jmp Unused
.org  ICP3addr
      jmp Unused
.org  OC3Aaddr
      jmp Unused
.org  OC3Baddr
      jmp Unused
.org  OC3Caddr
      jmp Unused
.org  OVF3addr
      jmp Unused
.org  TWIaddr
      jmp Unused
.org  SPMRaddr
      jmp Unused
; Beginning of the Program Code
MainEntry:
; Setup the Stack Pointer
      ldi r16, $10      ; Initialize the Stack Pointer to the end of
      out SPH, r16      ; the internal SRAM 0x10FF
      ldi r16, $FF
      out SPL, r16
; Set the Global Interrupt Flag in the SREG
      sei
; Reset the battery on/off switch
      in    r16, DDRE   ; Make port E pins 6 and 7 output pins
      ori r16, $C0      ; Pin 7 = battery on/off
      out DDRE, r16     ; Pin 6 = battery volt check enable
      in    r16, PORTE  ; Turn on the battery and Enable battery check
      andi r16, $3F     ; Pin 7 & 6 = 0
      out   PORTE, r16
; Reset the Master Kills
      ldi r16, $FF      ; Set port C as an output port
      out DDRC, r16
      in r16, DDRE      ; Set pins 0 & 1 of port E as an output port
      ori r16, $03
      out DDRE, r16
      clr r16           ; Clear all outputs
      out PORTC, r16
      cbi PORTE, 0
      cbi PORTE, 1
; Initialize the LEDs
      in    r16, DDRA   ; Set the Data Direction Register so Pins 0-5
      ori r16, $3F      ; are output pins
      out DDRA, r16
      in    r16, PORTA  ; Set Pins 0-5 High to turn off the LEDs
      ori r16, $3F
      out PORTA, r16
; Initialize the Switches
```

```
      in r16, DDRD      ; Make Port D pins 0 and 1 input pins
      andi r16, $FC     ; Pin 0 = Master Kill Switch
      out DDRD, r16     ; Pin 1 = Two-Step On/Off Input
      in r16, DDRB      ; Make Port B pins 6 and 7 input pins
      andi r16, $3F     ; Pin 6 = Two-Step Input
      out DDRB, r16     ; Pin 7 = Test Input
      in r16, PORTD     ; Turn on pull-up resistors for ports B and D
      ori r16, $03
      out PORTD, r16
      in r16, PORTB
      ori   r16, $C0
      out PORTB, r16
      clr r16           ; Set Switch variable initially to 0
      sts Switch, r16
; Initialize the Power Reduction Registers
      ldi r16, (1<<PRTWI)|(1<<PRTIM0)
      sts PRR0, r16
      ldi r16, (1<<PRUSB)|(1<<PRTIM3)
      sts PRR1, r16
; Initialize the SPI Interface
      in    r16, DDRB   ; Setting the data direction of Port B for SPI
      ori   r16, $37    ; Pins 0-2, 4 and 5 are output
      andi r16, $F7     ; Pin 3 is input
      out DDRB, r16
      in    r16, PORTB  ; Set the initial values of SS, Hold, and WP
      ori   r16, $31    ; All set high - off
      out PORTB, r16
      ldi   r16, (1<<SPE)|(1<<MSTR)
      out SPCR, r16     ; SPI on, set for master
      ldi r16, (1<<SPI2X)
      out SPSR, r16     ; Set for 4MHz speed
      cbi   PORTB, 0    ; Pull CS low
      ldi r16, $03      ; Send Read Command
      clr r17
      clr r18
      call SPIComSend
      call SPIRead
      sts races, r16          ; Store data in # races counter
      call SPIRead
      sts racecount, r16      ; Store total number of races made since
download
      call SPIRead
      sts LThresXH, r16 ; Store data for X Threshold
      call SPIRead
      sts LThresXL, r16
      call SPIRead
      sts UThresXH, r16
      call SPIRead
      sts UThresXL, r16
      call SPIRead
      sts LThresYH, r16 ; Store data for Y Threshold
      call SPIRead
      sts LThresYL, r16
      call SPIRead
      sts UThresYH, r16
      call SPIRead
      sts UThresYL, r16
```

```
      call SPIRead
      sts LThresZH, r16 ; Store data for Z Threshold
      call SPIRead
      sts LThresZL, r16
      call SPIRead
      sts UThresZH, r16
      call SPIRead
      sts UThresZL, r16
      call SPIRead
      sts ThresD0, r16
      call SPIRead
      sts ThresD1, r16
      call SPIRead
      sts ThresD2, r16
      call SPIRead
      sts ThresD3, r16  ; Store Date
      call SPIRead
      sts Car0, r16
      call SPIRead
      sts Car1, r16
      call SPIRead
      sts Car2, r16
      call SPIRead
      sts Car3, r16
      call SPIRead
      sts Car4, r16
      call SPIRead
      sts Car5, r16     ; Store Car #
      sbi   PORTB, 0    ; Pull CS high
; Initialize Analog to Digital Converter
      ldi r16, (1<<ADC0D)|(1<<ADC1D)|(1<<ADC2D)|(1<<ADC3D)
      sts DIDR0, r16    ; Turn off Digital Register of ADC conversion pins
      clr r16
      sts ADCSRB, r16   ; Set control register
      ldi r16, (1<<ADEN)|(1<<ADPS2)|(1<<ADPS1)
      sts ADCSRA, r16   ; Turn on the ADC and set clock 125 KHz
; Turn on Master Kill Interrupt
      Ldi r16, (1<<ISC11)|(1<<ISC01)
      sts EICRA, r16    ; Falling edge produces interrupt for MKill and TStep
      clr r16
      sts EICRB, r16
      ldi r16, (1<<PCIE0)
      sts PCICR, r16    ; Falling edge produces interrupt for Test
      sbi EIMSK, 0      ; Enable Master Kill Interrupt
PitRoadInit:
; Setup ADC for Ignition Test
      ldi r16, (1<<MUX0)|(1<<MUX1)
      sts ADMUX, r16    ; Set for Ignition Test Pin
; Setup USART
      ldi r16, (1<<FE1)|(1<<TXC1)
      sts UCSR1A, r16   ; Set for No double speed and no multi-processor
      ldi r16, (1<<UCSZ11)|(1<<UCSZ10)
      sts UCSR1C, r16   ; asynchronous, no parity, 1 stop bit, 8-bit transfer
      ldi r16, (1<<RXCIE1)|(1<<RXEN1)|(1<<TXEN1)
      sts UCSR1B, r16   ; Set Receive interrupt, receive and transmit enabled
      clr r16
      sts UBRR1H, r16
```

```
      ldi r16, 51
      sts UBRR1L, r16   ; Set the baud rate to 9600kbps
; Setup Test and Two-Step On/Off Interrupts
      sbi EIFR, INTF1   ; Clear the interrupt flag for two-step interrupt
      sbi EIMSK, 1      ; Enable Two-Step Bypass Interrupt
      ldi r16, (1<<PCINT7)
      sts PCMSK0, r16   ; Enable Test Interrupt
      clr r16
      sts Switch, r16   ; Clear all switches
; Turn on Pit Road Mode LED
      cbi PORTA, 0
; Beginning of Pit Road Mode Tests
PitRoad:
      lds r16, Switch   ; Load the status of the Switch variable to r16
      andi r16, (1<<SDload)   ; Mask off Download bit
      cpi r16, $00            ; If 0 not set
      breq PitRoad1           ; Skip to Pit Road 1 if 0
      call DloadRoutine       ; Call Download Subroutine
      jmp   PitRoadInit       ; When return reinitialize pit road mode
PitRoad1:
      lds r16, Switch         ; Load the status of the Switch variable to r16
      andi r16, (1<<STest)    ; Mask off the Test bit
      cpi r16, $00            ; If 0 is not set
      breq PitRoad2           ; Branch if 0 to PitRoad2
      call TestRoutine        ; Otherwise call test subroutine
      jmp PitRoadInit         ; When return reinitialize pit road mode
PitRoad2:                     ; Ignition Test
      call ADCStart
      cpi r17, $03      ; See if high byte is greater than or equal to $03
      brsh PitRoad3     ; If greater than or equal to $03 then ignition is on
      call SystemShutDown
PitRoad3:
      lds r16, Switch         ; Load switch variable into r16
      andi r16, (1<<STOnOff)  ; Check if the two-step is bypassed
      cpi r16, 0              ; If 0 it is not bypassed
      breq PitRoad4           ; Not bypassed so go to PitRoad4
      jmp   PitRoad           ; Is bypassed go to PitRoad
PitRoad4:
      in    r16, PINB         ; Take in info from Port B
      andi r16, $40           ; Check if two-step pressed
      cpi r16, $40            ; Compare two-step input to 0
      brne TwoStepPressed     ; If 2-step pressed (1) go to TwoStepPressed
      jmp   PitRoad           ; Otherwise retry tests from PitRoad
TwoStepPressed:
      ldi r16, (1<<AS2)       ; Set to 32.768 kHz external clock
      sts ASSR, r16
      clr r16                 ; Setup the timer
      sts TCCR2A, r16
      sts secs, r16           ; Clear seconds variable
      sts TCNT2, r16          ; Clear counter
      ldi r16, (1<<TOIE2)
      sts TIMSK2, r16
      ldi r16, (1<<CS22)|(1<<CS20)
      sts TCCR2B, r16         ; Setup 5 second timer
TwoStepPressed1:
      in r16, PINB            ; Take in info from Port B
      andi r16, $40           ; Check if two-step pressed
```

```
      cpi r16, $40
      brne TwoStepPressed2    ; If it is pressed go to TwoStepPressed2
      clr r16
      sts TCCR2B, r16         ; Turn off timer
      jmp   PitRoad           ; Otherwise return to pit road mode
TwoStepPressed2:
      lds r16, secs           ; Load in the seconds variable
      cpi r16, $05            ; Compare to 5
      brlo TwoStepPressed1    ; If lower keep checking the two step
; Start of Race Mode - When two-step released will be treated as a race
RaceModeInit:                 ; Greater than 5 seconds
; Turn off download interrupts
      clr r16
      sts UCSR1A, r16   ; Set all the USART controls to 0 (will turn it off)
      sts UCSR1B, r16
      sts UCSR1C, r16
; Turn off the two-step and test interrupts
      cbi EIMSK, 1            ; Disable Two-Step Bypass Interrupt
      clr r16
      sts PCMSK0, r16         ; Disable Test Interrupt
; Set LEDs
      sbi PORTA, 0            ; Turn off Pit Road LED
      cbi PORTA, 1            ; Turn on Race Mode LED
; Setup .02 second timer
      clr r16
      sts TCCR1A, r16         ; Setup the timer control registers
      sts TCCR1B, r16
      sts TCNT1H, r16         ; Clear the timer counter registers
      sts TCNT1L, r16
      ldi r16, $4E
      sts OCR1AH, r16
      ldi r16, $20
      sts OCR1AL, r16         ; Set max count compare to 20000 (.02 seconds)
; Clear Maximum Lower Variables
      ldi LMaxXH, $FF
      ldi LMaxXL, $FF
      ldi LMaxYH, $FF
      ldi LMaxYL, $FF
      ldi LMaxZH, $FF
      ldi LMaxZL, $FF
; Clear Maximum Upper Variables
      clr UMaxXH
      clr UMaxXL
      clr UMaxYH
      clr UMaxYL
      clr UMaxZH
      clr UMaxZL
; Prepare memory variables
      lds r16, races    ; Increment the races variable (start of a new race)
      inc r16
      cpi r16, 4        ; If greater than 4 races must replace first race
      brlo RaceInit1
      ldi r16, 1
RaceInit1:
      sts races, r16
      ldi r17, $40
      ldi r18, $40
```

```
      cpi r16, $01
      breq RaceInit1a
      add r18, r17
      cpi r16, $02
      breq RaceInit1a
      add r18, r17
RaceInit1a:
      clr r17
      sts MemPtrH, r18
      sts MemPtrL, r17
      lds r16, racecount      ; Load the number of races made since last
      inc r16                 ; download into r16
      sts racecount, r16      ; Increment and store result
      cbi PORTB, 0            ; Clear bit to signal Memory Write
      ldi r16, $06
      call SPIComSend         ; Send Write Enable Command
      sbi PORTB, 0            ; Set bit for latch to take effect
      lds r17, MemPtrL        ; Load address for erase into r17:r18
      lds r18, MemPtrH
      ldi r16, $D8            ; Load command for erase into r16
      cbi PORTB, 0            ; Clear bit to activate Memory
      call SPIComSend         ; Send erase
      sbi PORTB, 0            ; Set bit to latch command
      call SPIStatusWait      ; Wait until erase complete
      cbi PORTB,0             ; Clear bit to activate Memory
      ldi r16, $06            ; Send Write Enable Command
      call SPIComSend
      sbi PORTB,0             ; Set bit to latch command
      ldi r16, $02            ; Send write command
      clr r17                 ; Address $0000
      clr r18
      cbi PORTB, 0            ; Clear bit to activate Memory
      call SPIComSend         ; Send command
      lds r16, races          ; Write races
      call SPIWrite
      lds r16, racecount      ; Write racecount
      call SPIWrite
      sbi PORTB, 0            ; Set bit to latch command
      lds r16, MemPtrL
      lds r17, MemPtrH        ; Load starting pointer to r16:r17
      ldi r18, 128
      clr r19
      add r16, r18
      adc r17, r19            ; Address of Start of Next Page is in r16:r17
      sts PageEndL, r16       ; Store the start of the next page in memory
      sts PageEndH, r17
; Check to see if two-step released
TwoStepWait:
      in r16, PINB            ; Take in info from Port B
      andi r16, $40           ; Check if two-step pressed
      cpi r16, $40
      brne TwoStepWait        ; It two-step pressed, loop   until release
      clr r16
      sts secs, r16
      ldi r16, (1<<WGM12)|(1<<CS10)
      sts TCCR1C, r16
RaceMode:
```

```
      clr r16
      sts ADMUX, r16          ; Set to X-axis ADC channel
      call ADCStart           ; Perform conversion - result in r16:r17
      cp    r17, UMaxXH       ; Compare conversion value to Max
      brlo RaceMode2          ; If lower then no problem
      brne RaceMode1    ; If not equal then new data is definitely higher
      cp  r16, UMaxXL         ; Compare conversion value to lower Max
      brlo RaceMode2          ; If lower then below max
RaceMode1:                    ; Above X Max Values
      mov UMaxXH, r17         ; Update Maximum Values
      mov UMaxXL, r16
      lds r18, UThresXL       ; Load Upper Thresholds into r18:r19
      lds r19, UThresXH
      cp    r17, r19          ; Compare upper bytes
      brlo RaceMode2          ; If conversion value lower then no problem
      brne RaceMode1a   ; If not equal conversion is higher than thresholds
      cp    r16, r18          ; Compare lower bytes
      brlo RaceMode2          ; If conversion lower then no problem
RaceMode1a:                   ; Above upper threshold
      cli
      call VehicleShutDown
      call StoreValues
      call SystemShutDown
RaceMode2:                    ; Below X Max Values or X Upper Threshold
      cp    LMaxXH, r17       ; Compare upper bytes
      brlo RaceMode4          ; If threshold value lower than conversion okay
      brne RaceMode3    ; If threshold not equal conversion lower - problem
      cp    r16, LMaxXL       ; Compare lower bytes
      brsh RaceMode4    ; If conversion is same or higher - threshold is okay
RaceMode3:                    ; Lower than previous max values
      mov LMaxXH, r17         ; Store new max values
      mov LMaxXL, r16
      lds r18, LThresXL       ; Load lower threshold into r18:r19
      lds r19, LThresXH
      cp    r19, r17          ; Perform same comparsion with thresholds
      brlo RaceMode4
      brne RaceMode3a
      cp r16, r18
      brsh RaceMode4
RaceMode3a:                   ; If problem then shut down system
      cli
      call VehicleShutDown
      call StoreValues
      call SystemShutDown
RaceMode4:                    ; Check Y Values
      ldi r16, (1<<MUX0)
      sts ADMUX, r16          ; Set to Y-axis ADC channel
      call ADCStart           ; Perform conversion - result in r16:r17
      cp    r17, UMaxYH       ; Compare conversion value to Max
      brlo RaceMode6          ; If lower then no problem
      brne RaceMode5          ; If not equal then new data is higher
      cp  r16, UMaxYL         ; Compare conversion value to lower Max
      brlo RaceMode6          ; If lower then below max
RaceMode5:                    ; Above Y Max Values
      mov UMaxYH, r17         ; Update Maximum Values
      mov UMaxYL, r16
      lds r18, UThresYL       ; Load Upper Thresholds into r18:r19
```

```
      lds r19, UThresYH
      cp    r17, r19          ; Compare upper bytes
      brlo RaceMode6          ; If conversion value lower then no problem
      brne RaceMode5a         ; If not equal then conversion is higher
      cp    r16, r18          ; Compare lower bytes
      brlo RaceMode6          ; If conversion lower then no problem
RaceMode5a:                   ; Above upper threshold
      cli
      call VehicleShutDown
      call StoreValues
      call SystemShutDown
RaceMode6:                    ; Below Y Max Values or Y Upper Threshold
      cp    LMaxYH, r17       ; Compare upper bytes
      brlo RaceMode8          ; If threshold is lower than conversion okay
      brne RaceMode7    ; If threshold not equal, conversion lower, problem
      cp    r16, LMaxYL       ; Compare lower bytes
      brsh RaceMode8          ; If conversion is same or higher – okay
RaceMode7:                    ; Lower than previous max values
      mov LMaxYH, r17         ; Store new max values
      mov LMaxYL, r16
      lds r18, LThresYL       ; Load lower threshold into r18:r19
      lds r19, LThresYH
      cp    r19, r17          ; Perform same comparsion with thresholds
      brlo RaceMode8
      brne RaceMode7a
      cp r16, r18
      brsh RaceMode8
RaceMode7a:                   ; If problem then shut down system
      cli
      call VehicleShutDown
      call StoreValues
      call SystemShutDown
RaceMode8:
      ldi r16, (1<<MUX1)
      sts ADMUX, r16          ; Set to Z-axis ADC channel
      call ADCStart           ; Perform conversion - result in r16:r17
      cp    r17, UMaxZH       ; Compare conversion value to Max
      brlo RaceMode10         ; If lower then no problem
      brne RaceMode9          ; If not equal, new data is higher
      cp  r16, UMaxZL         ; Compare conversion value to lower Max
      brlo RaceMode10         ; If lower then below max
RaceMode9:                    ; Above Z Max Values
      mov UMaxZH, r17         ; Update Maximum Values
      mov UMaxZL, r16
      lds r18, UThresZL       ; Load Upper Thresholds into r18:r19
      lds r19, UThresZH
      cp    r17, r19          ; Compare upper bytes
      brlo RaceMode10         ; If conversion value lower then no problem
      brne RaceMode9a   ; If not equal then conversion higher than thresholds
      cp    r16, r18          ; Compare lower bytes
      brlo RaceMode10         ; If conversion lower then no problem
RaceMode9a:                   ; Above upper threshold
      cli
      call VehicleShutDown
      call StoreValues
      call SystemShutDown
RaceMode10:                   ; Below Z Max Values or Z Upper Threshold
```

```
      cp    LMaxZH, r17       ; Compare upper bytes
      brlo RaceMode12         ; If threshold is lower than conversion okay
      brne RaceMode11   ; If threshold not equal conversion lower - problem
      cp    r16, LMaxZL       ; Compare lower bytes
      brsh RaceMode12         ; If conversion same or higher, threshold okay
RaceMode11:                   ; Lower than previous max values
      mov LMaxZH, r17         ; Store new max values
      mov LMaxZL, r16
      lds r18, LThresZL       ; Load lower threshold into r18:r19
      lds r19, LThresZH
      cp    r19, r17          ; Perform same comparsion with thresholds
      brlo RaceMode12
      brne RaceMode11a
      cp r16, r18
      brsh RaceMode12
RaceMode11a:                  ; If problem then shut down system
      cli
      call VehicleShutDown
      call StoreValues
      call SystemShutDown
RaceMode12:
      lds r16, TIFR1          ; Load r16 with timer flag register
      andi r16, (1<<OCF1A)    ; Isolate Compare bit
      cpi r16, 0              ; If 0 then .02 seconds has not passed
      brne RaceMode12a        ; Otherwise it has and time to store data
      jmp RaceMode12b         ; Keep checking accelerometers
RaceMode12a:
      lds r16, TIFR1          ; Clear the bit
      ori r16, (1<<OCF1A)
      sts TIFR1, r16
      lds r16, MemPtrL        ; Load MemPtr into r16:r17
      lds r17, MemPtrH
      ldi r18, 8
      clr r19
      add r16, r18
      adc r17, r19
      lds r18, PageEndL       ; Load address of end of page into r18:r19
      lds r19, PageEndH
      cp r19, r17             ; See if values are equal
      brne TimeUp1
      cp r18, r16
      brne TimeUp1            ; If not equal go to TimeUp1
      sts MemPtrL, r16        ; Store new address
      sts MemPtrH, r17
      ldi r18, 128
      clr r19
      add r16, r18
      adc r17, r19
      sts PageEndL, r16       ; Store new end of page
      sts PageEndH, r17
TimeUp1:
      cbi PORTB, 0            ; Clear bit to activate memory
      ldi r16, $06            ; Load write enable command
      call SPIComSend         ; Send the command
      sbi PORTB, 0            ; Set bit to latch command
      ldi r16, $02            ; Load write command
      lds r17, MemPtrL        ; Load write address into r17:r18
```

```
      lds r18, MemPtrH
      cbi PORTB, 0            ; Clear bit to activate memory
      call SPIComSend         ; Send write command and address
      mov r16, LMaxXH         ; Write 12 bytes of data
      call SPIWrite
      mov r16, LMaxXL
      call SPIWrite
      mov r16, UMaxXH
      call SPIWrite
      mov r16, UMaxXL
      call SPIWrite
      mov r16, LMaxYH
      call SPIWrite
      mov r16, LMaxYL
      call SPIWrite
      mov r16, UMaxYH
      call SPIWrite
      mov r16, UMaxYL
      call SPIWrite
      mov r16, LMaxZH
      call SPIWrite
      mov r16, LMaxZL
      call SPIWrite
      mov r16, UMaxZH
      call SPIWrite
      mov r16, UMaxZL
      call SPIWrite
      sbi PORTB, 0            ; Set bit to end write
      lds r16, MemPtrL
      lds r17, MemPtrH
      ldi r18, $12
      clr r19
      add r16, r18
      adc r17, r19
      sts MemPtrL, r16        ; Add 12 and store address for next save
      sts MemPtrH, r17
; Reset the Maximum Thresholds
      ldi LMaxXH, $FF
      ldi LMaxXL, $FF
      ldi LMaxYH, $FF
      ldi LMaxYL, $FF
      ldi LMaxZH, $FF
      ldi LMaxZL, $FF
      clr UMaxXH
      clr UMaxXL
      clr UMaxYH
      clr UMaxYL
      clr UMaxZH
      clr UMaxZL
; Check if 15 seconds is done
RaceMode12b:
      lds r16, secs           ; Load seconds into r16
      cpi r16, 15             ; Compare value in r16 to 15
      brsh RaceMode13         ; If greater than 15 go to RaceMode13
      jmp RaceMode            ; Otherwise go to RaceMode
RaceMode13:
      ldi r16, (1<<MUX0)|(1<<MUX1)
```

```
      sts ADMUX, r16          ; Set for Ignition Test Pin
      call ADCStart           ; Perform Conversion
      cpi r17, $03            ; See if high byte greater than or equal to $03
      brsh RaceMode14         ; If true then ignition is on
      call SystemShutDown     ; Turnoff DragAid-MK
RaceMode14:
      sbi    PORTA, 1         ; Turn off Race Mode LED
      clr r16
      sts secs, r16           ; Clear seconds variable
      sts TCCR1C, r16         ; Shut off Timer 1
      sts TCCR2B, r16         ; Shut off Timer 2
      jmp PitRoadInit
.include "common.asm"
.include "definitions.asm"
.include "interrupts.asm"
.include "download.asm"
.include "test.asm"
.exit
```

Appendix H: Common.asm

```
;***************************************************************************
;*                            SPI Functions                                *
;***************************************************************************
; Precondition: SPI command is in r16, address is in r17 & r18
; Postcondition: SPI command is sent over SPI interface
SPIComSend:
      out SPDR, r16           ; Output the command
      call SPIWait            ; Wait for transmission to complete
      cpi r16, $03            ; If read, write, page erase, or sector erase
      breq SendAddress  ; send address too
      cpi r16, $02
      breq SendAddress
      cpi r16, $42
      breq SendAddress
      cpi r16, $D8
      breq SendAddress
      ret
SendAddress:
      out SPDR, r18           ; Output address (high)
      call SPIWait
      out   SPDR, r17         ; Output address (low)
      call SPIWait
      ret
; Precondition: Read command must be sent first
; Postcondition: Result of read is in r16
SPIRead:
      ldi r16, $00      ; Send dummy data to receive info
      out SPDR, r16
      call SPIWait      ; Wait for data to be received
      in    r16, SPDR   ; Take in data
      ret
; Precondition: Write command has been sent
; Postcondition: Byte to write is in r16
SPIWrite:
      out SPDR, r16     ; Send data out to memory
      call SPIWait      ; Wait for data to be sent
      ret
; Precondition: SPI command has been sent
; Postcondtion: Returns when SPI transmission is complete
SPIWait:
      push r16
SPIWait1:
      in r16, SPSR
      andi r16, (1<<SPIF)
      cpi r16, $00
      breq SPIWait1
      pop r16
      ret
; Precondition: Erase Command has been sent
; Postcondition: Returns when sector erase is complete
SPIStatusWait:
      push r16
      push r17
      push r18
      clr r16
```

```
      clr r17
      clr r18
SPIStatus1:
      inc r16
      cpi r16, $FF
      brne SPIStatus1
      clr r16
      inc r17
      cpi r17, $FF
      brne SPIStatus1
      clr r16
      clr r17
      inc r18
      cpi r18, $2
      brne SPIStatus1         ; Delay for 16ms
SPIStatus2:
      cbi PORTB, 0
      ldi r16, $05
      call SPIComSend
      call SPIRead
      sbi PORTB, 0
      andi r16, $01
      cpi r16, $00
      brne SPIStatus2
      pop r18
      pop r17
      pop r16
      ret
;***************************************************************************
;*                            ADC Functions                                *
;***************************************************************************
; Precondition: ADC Start Conversion has been triggered
; Postcondition: When a conversion is complete this will return
ADCWait:
      push r16
ADCWait1:
      lds r16, ADCSRA         ; Load the status of the conversion into r16
      andi r16, (1<<ADIF)
      cpi r16, 0              ; While the conversion complete flag is not set
loop
      breq ADCWait1
      pop r16                       ; Otherwise return
      ret
; Precondition: None
; Postcondition: Triggers an ADC conversion, waits for the result, and puts
;     the result in r16 and r17
ADCStart:
      lds r16, ADCSRA
      ori r16, (1<<ADSC)|(1<<ADIF)
      sts ADCSRA, r16         ; Start the ADC conversion and clear results
flag
      call ADCWait
      lds r16, ADCL           ; Put the results in r16 and r17
      lds r17, ADCH
      ret                           ; Return
```

```
;*****************************************************************************
;                              Special Functions                                *
;*****************************************************************************
; Precondition: System operation is complete
; Postcondition: System is off (power eliminated from the circuit)
SystemShutDown:
      sbi PORTE, 7      ; Send off signal to the system
SystemShutDown1:
      jmp SystemShutDown1     ; Wait here while system shuts off
      ret                           ; Should never be reached
; Precondition: Thresholds have been exceeded
; Postcondition: "Off" Signals sent to Port C and E
VehicleShutDown:
      ldi r16, $FF      ; Set Pins 35-42 high
      out PORTC, r16
      sbi PORTE, 0      ; Set Pins 33-34 high
      sbi PORTE, 1
      ret
; Precondition: Thresholds have been exceeded
; Postcondition: Current maximum values are stored in memory at next memory
;     location, no memory pointers are updated
StoreValues:
      lds r16, MemPtrL        ; Load MemPtr into r16:r17
      lds r17, MemPtrH
      ldi r18, 8
      clr r19
      add r16, r18
      adc r17, r19
      lds r18, PageEndL       ; Load address of end of page into r18:r19
      lds r19, PageEndH
      cp r19, r17             ; See if values are equal
      brne StoreValues1
      cp r18, r16
      brne StoreValues1       ; If not equal go to TimeUp1
      sts MemPtrL, r16        ; Store new address
      sts MemPtrH, r17
StoreValues1:
      cbi PORTB, 0            ; Clear bit to activate memory
      ldi r16, $06            ; Load write enable command
      call SPIComSend         ; Send the command
      sbi PORTB, 0            ; Set bit to latch command
      ldi r16, $02            ; Load write command
      lds r17, MemPtrL        ; Load write address into r17:r18
      lds r18, MemPtrH
      cbi PORTB, 0            ; Clear bit to activate memory
      call SPIComSend         ; Send write command and address
      mov r16, LMaxXH         ; Write 12 bytes of data
      call SPIWrite
      mov r16, LMaxXL
      call SPIWrite
      mov r16, UMaxXH
      call SPIWrite
      mov r16, UMaxXL
      call SPIWrite
      mov r16, LMaxYH
      call SPIWrite
      mov r16, LMaxYL
```

```
        call SPIWrite
        mov r16, UMaxYH
        call SPIWrite
        mov r16, UMaxYL
        call SPIWrite
        mov r16, LMaxZH
        call SPIWrite
        mov r16, LMaxZL
        call SPIWrite
        mov r16, UMaxZH
        call SPIWrite
        mov r16, UMaxZL
        call SPIWrite
        sbi PORTB, 0            ; Set bit to end write
        ret
.exit
```

Appendix I: Download.asm

```
DloadRoutine:
; Set LEDs
      sbi PORTA, 0      ; Turn off Pit Road LED
      cbi PORTA, 2      ; Turn on Download LED
; Turn off Download Interrupt
      ldi r16, (1<<RXEN1)|(1<<TXEN1)
      sts UCSR1B, r16
; Turn off Button Interrupts
      cbi EIMSK, 1                  ; Disable Two-Step Bypass Interrupt
      clr r16
      sts PCMSK0, r16         ; Disable Test Interrupt
      ldi r16, (1<<AS2)       ; Set to 32.768 kHz external clock
      sts ASSR, r16
      ldi r16, (1<<WGM21)     ; Set to count to OCR2A
      sts TCCR2A, r16
      clr r16                 ; Setup the timer
      sts secs, r16           ; Clear seconds variable
      sts TCNT2, r16          ; Clear counter
      ldi r16, 32             ; Load r16 with 32
      sts OCR2A, r16          ; Store in OCR2A
      ldi r16, (1<<OCIE2A)    ; Set to interrupt on reaching OCR2A value
      sts TIMSK2, r16
      ldi r16, (1<<CS20)
      sts TCCR2B, r16
; Setup ADC for ignition test
      ldi r16, (1<<MUX0)|(1<<MUX1)
      sts ADMUX, r16          ; Set for Ignition Test Pin
DownloadLoop:                 ; Start Loop
      call ADCStart
      cpi r17, $03            ; See if high byte greater than or equal to $03
      brsh Dloop1             ; If true then ignition is on
      call SystemShutDown
Dloop1:
      lds r16, UCSR1A         ; Load USART flag register into r16
      andi r16, (1<<RXC1)     ; Isolate Receive Bit
      cpi r16, 0              ; If 0 nothing is in receive register
      breq Dloop2
      lds r16, UDR1           ; Load received command into r16
      cpi r16, $53            ; See if 'S'
      brne Dloop1a            ; If not check again
      call DataReceive        ; Otherwise computer sending data for DragAid
      jmp Dloop2              ; When done do a time out check
Dloop1a:
      cpi r16, $52            ; See if 'R'
      brne Dloop1b            ; If not check again
      call DataSend           ; Otherwise computer wants to receive data
      jmp Dloop2              ; When done do a time out check
Dloop1b:
      cpi r16, $44            ; See if 'D'
      brne Dloop2             ; If not, not recognized - go to 5 minute check
      lds r16, UDR1           ; Otherwise check again
      cpi r16, $44            ; If 'D' again then done
      breq DDone
Dloop2:                       ; Nothing is in the receive register
; Time out check
```

```
      lds r16, secs           ; Load counter into r16
      cpi r16, 255            ; Compare to 60
      brlo DownloadLoop       ; If lower than 60, 5 minutes has not passed
DDone:
      sbi PORTA, 2            ; Turn off Download LED
      clr r16
      sts TCCR2B, r16         ; Shut off time out timer
      ret                     ; Return to main program

; Precondition: Received 'R' to state that PC wishes to receive data from ;
      device
; Postcondition: Device either sends thresholds and car # or these as well as
;     Run Data
DataSend:
      call ReceiveWait
      sts Command, r16        ; Either 'T' or 'D'
      lds r16, Car0
      call TransmitWait
      lds r16, Car1
      call TransmitWait
      lds r16, Car2
      call TransmitWait
      lds r16, Car3
      call TransmitWait
      lds r16, Car4
      call TransmitWait
      lds r16, Car5
      call TransmitWait       ; Send Car # to PC
      lds r16, Command        ; Check to see if Data or Threshold Send
      cpi r16, $44
      brne DataSend1
      jmp DataSend1a
DataSend1:                    ; If 'D' then Data Send
      lds r16, LThresXH
      call TransmitWait
      lds r16, LThresXL
      call TransmitWait
      lds r16, UThresXH
      call TransmitWait
      lds r16, UThresXL
      call TransmitWait
      lds r16, LThresYH
      call TransmitWait
      lds r16, LThresYL
      call TransmitWait
      lds r16, UThresYH
      call TransmitWait
      lds r16, UThresYL
      call TransmitWait
      lds r16, LThresZH
      call TransmitWait
      lds r16, LThresZL
      call TransmitWait
      lds r16, UThresZH
      call TransmitWait
      lds r16, UThresZL
      call TransmitWait       ; Send Thresholds to the PC
```

```
      lds r16, ThresD0
      call TransmitWait
      lds r16, ThresD1
      call TransmitWait
      lds r16, ThresD2
      call TransmitWait
      lds r16, ThresD3
      call TransmitWait       ; Send Threshold Date to the PC
      jmp DataSendComplete
DataSend1a:
      lds r20, racecount
      cpi r20, $00
      breq DataSendComplete
      cbi PORTB, 0            ; Enable the memory device
      ldi r16, $03            ; Read command
      clr r17
      ldi r18, $40
      call SPIComSend         ; Send read from Race 1 Address
      clr r18
      clr r19
DataSendLoop:
      call SPIRead
      call TransmitWait       ; Must do this 9012 times
      inc r18
      cpi r18, $00
      brne DataSendLoop1
      inc r19
DataSendLoop1:
      cpi r19, $23
      brne DataSendLoop
      cpi r18, $34
      brne DataSendLoop
; When here have sent Race 1
      cpi r20, $01
      breq DataSendComplete
      clr r18
      clr r19
DataSendLoop2:
      call SPIRead
      call TransmitWait
      inc r18
      cpi r18, $00
      brne DataSendLoop3
      inc r19
DataSendLoop3:
      cpi r19, $23
      brne DataSendLoop2
      cpi r18, $34
      brne DataSendLoop2
; When here have sent Race 2
      cpi r20, $02
      breq DataSendComplete
      clr r18
      clr r19
DataSendLoop4:
      call SPIRead
      call TransmitWait
```

```
      inc r18
      cpi r18, $00
      brne DataSendLoop5
      inc r19
DataSendLoop5:
      cpi r19, $23
      brne DataSendLoop4
      cpi r18, $34
      brne DataSendLoop4      ; 3rd Race Sent
DataSendComplete:
      sbi PORTB, 0            ; Disable Memory Device
      clr r16
      sts racecount, r16
      sts races, r16          ; Should also save this data to memory!
      cbi PORTB, 0            ; Enable Memory Device
      ldi r16, $06
      call SPIComSend         ; Send Write Enable Command
      sbi PORTB, 0
      ldi r16, $02            ; Write Command
      clr r17                 ; Addres (Low Byte)
      clr r18                 ; Address (High Byte)
      cbi PORTB, 0            ; Enable Memory Device
      call SPIComSend         ; Send Command
      lds r16, races
      call SPIWrite
      lds r16, racecount
      call SPIWrite           ; Write races and racecount to memory
      sbi PORTB, 0            ; Disable Memory Device
      ldi r16, $43
      sts UDR1, r16           ; Send 'C' (Complete)
DataSend2:
      lds r16, UCSR1A
      andi r16, (1<<TXC1)
      cpi r16, 0              ; Check TXC1 to see if transmit complete
      breq DataSend2          ; Loop until transmit complete
      clr r16
      sts secs, r16           ; Reset the 5 minute timer
      sts Command, r16        ; Reset the command
      ret

; Precondition: Received 'S' to state that PC wishes to send data to device
; Postcondtion: Device accepts data and stores it in memory
DataReceive:
; Send O.K to say can receive data
      call ReceiveWait
      sts Command, r16        ; Either 'I' or 'T'
      cpi r16, $49
      breq DataReceivea
      jmp DataReceiveb
DataReceivea:
; Store thresholds
      ldi r16, ILXH
      sts LThresXH, r16
      ldi r16, ILXL
      sts LThresXL, r16
      ldi r16, IUXH
      sts UThresXH, r16
```

```
      ldi r16, IUXL
      sts UThresXL, r16
      ldi r16, ILYH
      sts LThresYH, r16
      ldi r16, ILYL
      sts LThresYL, r16
      ldi r16, IUYH
      sts UThresYH, r16
      ldi r16, IUYL
      sts UThresYL, r16
      ldi r16, ILZH
      sts LThresZH, r16
      ldi r16, ILZL
      sts LThresZL, r16
      ldi r16, IUZH
      sts UThresZH, r16
      ldi r16, IUZL
      sts UThresZL, r16
; Store date
      ldi r16, ID0
      sts ThresD0, r16
      ldi r16, ID1
      sts ThresD1, r16
      ldi r16, ID2
      sts ThresD2, r16
      ldi r16, ID3
      sts ThresD3, r16
; Store Version Number
      ldi r16, IVersionH
      sts VersionH, r16
      ldi r16, IVersionL
      sts VersionL, r16
; Store Serial Number
      ldi r16, ISerialH
      sts SerialH, r16
      ldi r16, ISerialL
      sts SerialL, r16
      ldi r16, $58            ; Store XXXXX as Car #
      sts Car0, r16
      sts Car1, r16
      sts Car2, r16
      sts Car3, r16
      sts Car4, r16
      sts Car5, r16
DataReceiveb:
      ldi r16, $4F
      sts UDR1, r16           ; Send 'O'
DataReceive1:
      lds r16, UCSR1A
      andi r16, (1<<TXC1)
      cpi r16, 0              ; Check TXC1 to see if transmit complete
      breq DataReceive1       ; Loop until transmit complete
      lds r16, UCSR1A
      ori r16, (1<<TXC1)      ; Reset the bit
      ldi r16, $4B
      sts UDR1, r16           ; Send 'K'
DataReceive2:
```

```
      lds r16, UCSR1A
      andi r16, (1<<TXC1)
      cpi r16, 0              ; Check TXC1 to see if transmit complete
      breq DataReceive2       ; Loop until transmit complete
      lds r16, UCSR1A
      ori r16, (1<<TXC1)      ; Reset the bit
      clr r16
      sts races, r16
      sts racecount, r16
      lds r16, command
      cpi r16, $49
      brne DataReceive2a
      jmp ReceiveStoreData
DataReceive2a:
; Get Threshold Data
      call ReceiveWait
      sts LThresXH, r16
      call ReceiveWait
      sts LThresXL, r16
      call ReceiveWait
      sts UThresXH, r16
      call ReceiveWait
      sts UThresXL, r16
      call ReceiveWait
      sts LThresYH, r16
      call ReceiveWait
      sts LThresYL, r16
      call ReceiveWait
      sts UThresYH, r16
      call ReceiveWait
      sts UThresYL, r16
      call ReceiveWait
      sts LThresZH, r16
      call ReceiveWait
      sts LThresZL, r16
      call ReceiveWait
      sts UThresZH, r16
      call ReceiveWait
      sts UThresZL, r16
; Get Threshold Date Modfified
      call ReceiveWait
      sts ThresD0, r16
      call ReceiveWait
      sts ThresD1, r16
      call ReceiveWait
      sts ThresD2, r16
      call ReceiveWait
      sts ThresD3, r16
      call ReceiveWait
      sts Car0, r16
      call ReceiveWait
      sts Car1, r16
      call ReceiveWait
      sts Car2, r16
      call ReceiveWait
      sts Car3, r16
      call ReceiveWait
```

```
      sts Car4, r16
      call ReceiveWait
      sts Car5, r16
ReceiveStoreData:
      cbi PORTB, 0            ; Enable the memory device
      ldi r16, $06
      call SPIComSend         ; Send write enable command
      sbi PORTB, 0            ; Set latch
      ldi r16, $02            ; Load write command and address
      clr r17
      clr r18
      cbi PORTB, 0            ; Enable the memory device
      call SPIComSend         ; Send Write command
      lds r16, races
      call SPIWrite           ; Clear races variable
      lds r16, racecount
      call SPIWrite           ; Clear racecount variable
      lds r16, LThresXH
      call SPIWrite
      lds r16, LThresXL
      call SPIWrite
      lds r16, UThresXH
      call SPIWrite
      lds r16, UThresXL
      call SPIWrite           ; Write X-thresholds
      lds r16, LThresYH
      call SPIWrite
      lds r16, LThresYL
      call SPIWrite
      lds r16, UThresYH
      call SPIWrite
      lds r16, UThresYL
      call SPIWrite           ; Write Y-thresholds
      lds r16, LThresZH
      call SPIWrite
      lds r16, LThresZL
      call SPIWrite
      lds r16, UThresZH
      call SPIWrite
      lds r16, UThresZL
      call SPIWrite           ; Write Z-thresholds
      lds r16, ThresD0
      call SPIWrite
      lds r16, ThresD1
      call SPIWrite
      lds r16, ThresD2
      call SPIWrite
      lds r16, ThresD3
      call SPIWrite           ; Write Modified Date
      lds r16, Car0
      call SPIWrite
      lds r16, Car1
      call SPIWrite
      lds r16, Car2
      call SPIWrite
      lds r16, Car3
      call SPIWrite
```

```
      lds r16, Car4
      call SPIWrite
      lds r16, Car5
      call SPIWrite           ; Write Car #
      lds r16, Command
      cpi r16, $49            ; Determine if initialize
      brne ReceiveComplete    ; If not then done
      lds r16, SerialH
      call SPIWrite
      lds r16, SerialL
      call SPIWrite           ; Write Serial Number
      lds r16, VersionH
      call SPIWrite
      lds r16, VersionL
      call SPIWrite           ; Write Version Number
      lds r16, ThresD0
      call SPIWrite
      lds r16, ThresD1
      call SPIWrite
      lds r16, ThresD2
      call SPIWrite
      lds r16, ThresD3
      call SPIWrite           ; Write Date Version Loaded
ReceiveComplete:
      sbi PORTB, 0            ; Disable the memory
; Transmit to PC to say receive is complete
      ldi r16, $43
      sts UDR1, r16           ; Send 'C' (Complete)
DataReceive3:
      lds r16, UCSR1A
      andi r16, (1<<TXC1)
      cpi r16, 0              ; Check TXC1 to see if transmit complete
      breq DataReceive3       ; Loop until transmit complete
      lds r16, UCSR1A
      ori r16, (1<<TXC1)      ; Reset the bit
; Done
      clr r16
      sts Command, r16        ; Clear the command received
      sts secs, r16           ; Reset the 5 minute timer
      ret
; Precondition: Data send expected
; Postcondition: Data received and put in r16
ReceiveWait:
      lds r16, UCSR1A         ; Load r16 with the receive register
      andi r16, (1<<RXC1)     ; Isolate the receive bit
      cpi r16, 0              ; If not set then no new data
      breq ReceiveWait        ; Wait for new data
      lds r16, UDR1           ; Put received data in r16
      ret
; Precondition: Data to send put in r16
; Postcondition: Data sent to PC
TransmitWait:
      sts UDR1, r16           ; Send Data in r16
TransmitWait1:
      lds r16, UCSR1A
      andi r16, (1<<TXC1)
      cpi r16, 0              ; Check TXC1 to see if transmit complete
```

```
      breq TransmitWait1      ; Loop until transmit complete
      lds r16, secs
      inc r16
WaitaSec:
      lds r17, secs
      cp r16, r17
      brne WaitaSec
      ret
.exit
```

Appendix J: Interrupts.asm

```
; Interrupt Subroutine File
;*********************************************************************
; Precondition: Master Kill Switch Pressed
; Postcondition: Pins 33-42 of the processor will be brought low to
;     shut down the vehicle
MasterKill:
      push r16          ; Save r16 and status register
      in r16, sreg
      push r16
      ldi r16, $FF      ; Set Pins 35-42 high
      out PORTC, r16
      sbi PORTE, 0      ; Set Pins 33-34 high
      sbi PORTE, 1
      pop r16           ; Return saved registers
      out sreg, r16
      pop r16
      reti
;*********************************************************************
;*********************************************************************
; Precondition: Two-step bypass button pressed
; Postcondition: Flag in the switch variable is set
TwoStepInt:
      push r16
      in r16, sreg
      push r16
TwoStepInt2:
      in r16, PIND
      andi r16, $02
      cpi r16, $00
      breq TwoStepInt2  ; Do not pass through until button released
      lds r16, Switch   ; Load Switch variable into r16
      sbrc r16, STOnOff ; If two-step bypass button is not cleared branch
      jmp   TwoStepInt0
      ori r16, (1<<STOnOff)
      cbi PORTA, 4      ; Turn on the two-step bypass LED
      jmp TwoStepInt1
TwoStepInt0:
      andi r16, ~(1<<STOnOff) ; Clear bit for two-step bypass
      sbi PORTA, 4      ; Turn off the two-step bypass LED
TwoStepInt1:
      sts Switch, r16   ; Store result back into switch
      sbi EIFR, INTF1
      pop r16           ; Return registers
      out sreg, r16
      pop r16
      reti
;*********************************************************************
;*********************************************************************
; Precondition: Test button pressed
; Postcondition: Test Flag in the switch variable is set
TestInt:
      push r16          ; Save r16 and status register
      in r16, sreg
      push r16
      lds r16, Switch   ; Load Switch register into r16
```

```
      ori r16, (1<<STest)     ; Set the Test Flag
      sts Switch, r16   ; Store r16 to Switch register
TestInt1:
      in r16, PINB
      andi r16, $80
      cpi r16, $00
      breq TestInt1     ; Do not return until button released
      pop r16           ; Return saved variables
      out sreg, r16
      pop r16
      reti
;*********************************************************************
;*********************************************************************
; Precondition: USART interrupt received
; Postcondition: Download Flag in Switch register is set
Download:
      push r16          ; Save r16 and status register
      in r16, sreg
      push r16
      lds r16, Switch   ; Load Switch register into r16
      ori r16, (1<<SDload)    ; Set the Download Flag
      sts Switch, r16   ; Store r16 to Switch register
      pop r16           ; Return saved variables
      out sreg, r16
      pop r16
      reti
;*********************************************************************
;*********************************************************************
;     Precondition: One Second has passed since last RTC interrupt
;     Postcondition: Variable secs will be incremented by 1
RTCInt:
      push r16          ; Save r16 and status register
      in    r16, sreg
      push r16
      lds r16, secs     ; Load secs into r16
      inc r16           ; Increment r16
      sts secs, r16     ; Store new value to secs
      pop r16           ; Return r16 and status register
      out sreg, r16
      pop r16
      reti              ; Return
;*********************************************************************
;*********************************************************************
; Safety interrupt
Unused:
      reti
;*********************************************************************
.exit
```

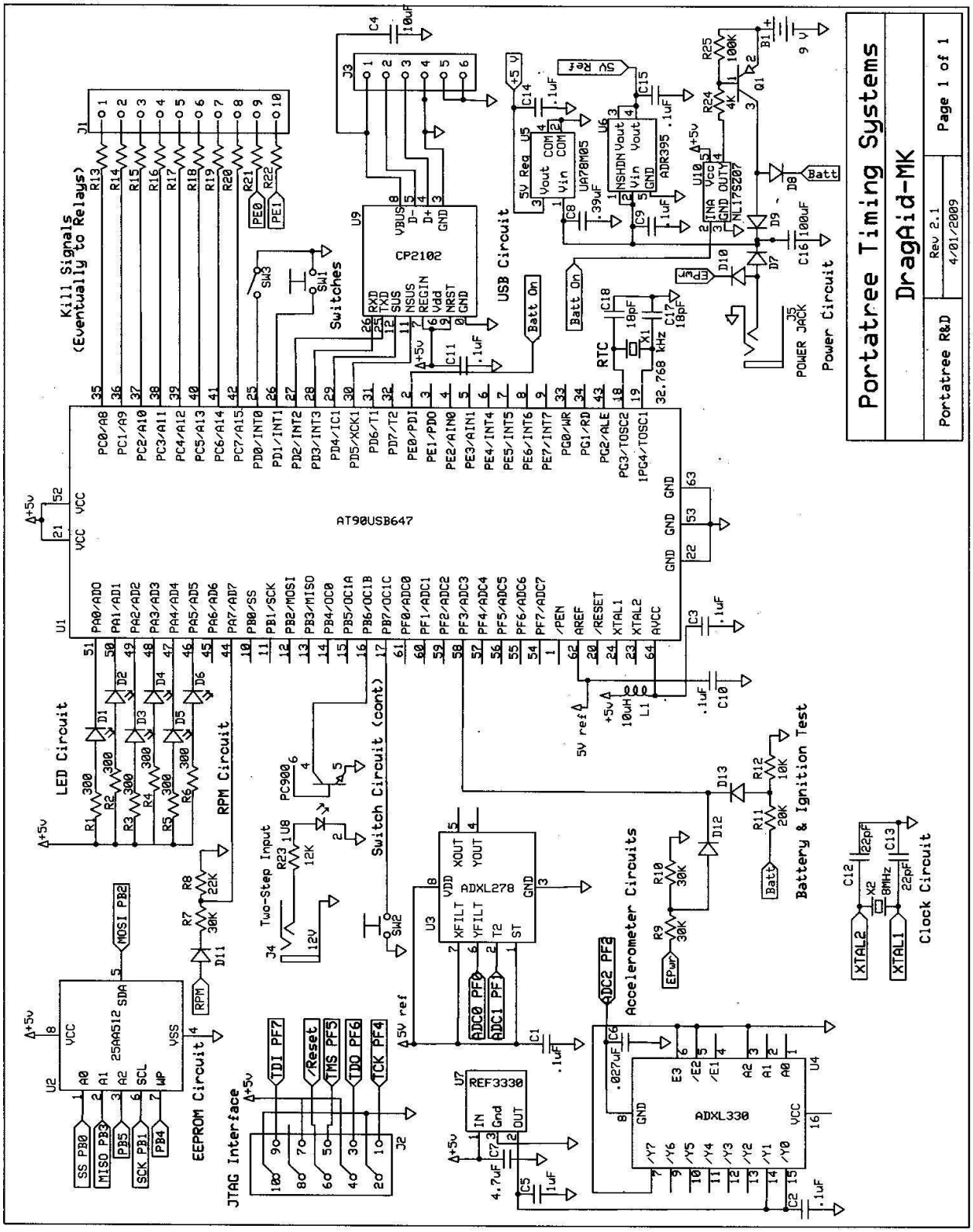
Appendix K: Test.asm

```
TestRoutine:
; Setup LEDs
      sbi PORTA, 0      ; Turn off Pit Road LED
      cbi   PORTA, 3    ; Turn on Test Mode LED
; Turn off download interrupts
      clr r16
      sts UCSR1A, r16   ; Set all the USART controls to 0 (will turn it off)
      sts UCSR1B, r16
      sts UCSR1C, r16
; Turn off two-step bypass interrupt
      cbi EIMSK, 1      ; Disable Two-Step Bypass Interrupt
      sbi PORTA, 4      ; Turn it off if on
; Reset Switch Interrupt
      lds r16, Switch   ; Load Switch register into r16
      andi r16, ~((1<<STest)|(1<<STOnOff))      ; Clear the Test Flag and
bypass flag
      sts Switch, r16   ; Store r16 to Switch register
TestLoop:
; Perform Ignition Test
      ldi r16, (1<<MUX0)|(1<<MUX1)
      sts ADMUX, r16    ; Set for Ignition Test Pin
      call ADCStart     ; Perform Conversion
      cpi r17, $03      ; See if high byte is greater than or equal to $03
      brsh TestLoop1    ; If greater than or equal to $03 then ignition is on
      call SystemShutDown
TestLoop1:
      lds r16, Switch   ; Load r16 with Switch data
      andi r16, (1<<STest)    ; Isolate switch test bit
      cpi r16, 0        ; See if 0
      breq TestLoop1a
      jmp TestComplete  ; If not then pressed and test is done
TestLoop1a:
      clr r16
      sts ADMUX, r16    ; Set to X-axis ADC channel
      call ADCStart     ; Perform conversion - result in r16:r17
      ldi r18, $03
      ldi r19, $02
      cp  r17, r19
      brlo TestLoop3    ; If r17 is larger than r19 then Light On
      brne TestLoop2    ; If r17 is not equal to r19 then below Threshold
      cp    r16, r18
      brlo TestLoop3    ; If r16 is larger than r18 then Light On
TestLoop2:
      in r16, PORTA
      ori r16, $37
      out PORTA, r16
      cbi PORTA, 0      ; Light on
      jmp TestLoop      ; Go back to start
TestLoop3:
      ldi r18, $F7      ; Load r18 and r19 with the lower threshold
      ldi r19, $01
      cp    r19, r17
      brlo TestLoop5    ; If r19 is lower than r17 then above threshold
      brne TestLoop4    ; If r19 is not equal to r17 then below threshold
      cp  r16, r18      ; If same or higher then above threshold
```

```
      brsh TestLoop5
TestLoop4:
      in r16, PORTA
      ori r16, $37
      out PORTA, r16
      cbi PORTA, 1      ; Light on
      jmp TestLoop      ; Go back to start
TestLoop5:
      ldi r16, (1<<MUX0)
      sts ADMUX, r16    ; Set to Y-axis ADC channel
      call ADCStart     ; Perform conversion - result in r16:r17
      ldi r18, $FC      ; Load r18 and r19 with upper threshold
      ldi r19, $01
      cp  r17, r19
      brlo TestLoop7    ; If r17 is larger than r19 then Light On
      brne TestLoop6    ; If r17 is not equal to r19 then below Threshold
      cp    r16, r18
      brlo TestLoop7    ; If r16 is larger than r18 then Light On
TestLoop6:
      in r16, PORTA
      ori r16, $37
      out PORTA, r16
      cbi PORTA, 2      ; Light on
      jmp TestLoop      ; Go back to start
TestLoop7:
      ldi r18, $F0      ; Load r18 and r19 with the lower threshold
      ldi r19, $01
      cp    r19, r17
      brlo TestLoop9    ; If r19 is lower than r17 then above threshold
      brne TestLoop8    ; If r19 is not equal to r17 then below threshold
      cp  r16, r18      ; If same or higher then above threshold
      brsh TestLoop9
TestLoop8:
      in r16, PORTA
      ori r16, $37
      out PORTA, r16
      cbi PORTA, 4      ; Light on
      jmp TestLoop      ; Go back to start
TestLoop9:
      ldi r16, (1<<MUX1)
      sts ADMUX, r16    ; Set to Z-axis ADC channel
      call ADCStart     ; Perform conversion - result in r16:r17
      ldi r18, $F8      ; Load r18 and r19 with the lower threshold
      clr r19
      cp    r19, r17
      brlo TestLoop11   ; If r19 is lower than r17 then above threshold
      brne TestLoop10   ; If r19 is not equal to r17 then below threshold
      cp  r16, r18      ; If same or higher then above threshold
      brsh TestLoop11
TestLoop10:
      in r16, PORTA
      ori r16, $37
      out PORTA, r16
      cbi PORTA, 5      ; Light on
      jmp TestLoop      ; Go back to start
TestLoop11:
      in r16, PORTA
```

```
      ori r16, $37
      out PORTA, r16
      jmp TestLoop      ; Go back to start
TestComplete:
      sbi PORTA, 3      ; Turn off Test Mode LED
      sbi PORTA, 5      ; Turn off Test LED
      ret
.exit
```

Appendix L: Schematic



Appendix M: Master Parts List

(Please see next page)

DragAid-MK – Drag Race Analyzer and Master Kill Switch
Master Parts List – Rev. B
Allison Smyth, Al Smyth & Frank Legassey
December 23, 2008

Design Team: Allison Smyth asmyth@wpi.edu
Frank Legassey frank@portatree.com
Al Smyth al@portatree.com

#	QTY	Ref	Value	Description	Dist.	Part No.	MFG	MFG Part No.	Unit	Sub
1	8	C1, C2, C3, C9, C10, C11, C14, C15	.1uF	CAP CER .1UF 50V 10% X7R 1206	Digi-Key	490-1775-2-ND	Murata Electronics	GRM319R71H104KA01D	0.02500	0.20000
2	1	C4	10uF	CAP 10UF 10V CERAMIC 1206 X5R	Digi-Key	PCC2178TR-ND	Panasonic-ECG	ECJ-3YB1A106M	0.19336	0.19336
3	1	C5	1uF	CAP CER 1UF 50V X7R 10% 1206	Digi-Key	445-1423-1-ND	TDK Corporation	C3216X7R1H105K	0.27500	0.27500
4	1	C6	.027uF	CAP 27000PF 50V CERAMIC X7R 1206	Digi-Key	311-1204-2-ND	Yageo	CC1206KRX7R9BB273	0.01750	0.01750
5	1	C7	4.7uF	CAP CER 4.7UF 50V X5R 1206	Digi-Key	399-5507-2-ND	Kemet	C1206C475K5PACTU	0.14900	0.14900
6	1	C8	.39uF	CAP .39UF 25V CERAMIC X7R 1206	Digi-Key	PCC1890TR-ND	Panasonic-ECG	ECJ-3YB1E394K	0.11081	0.11081
7	2	C12, C13	22pF	CAP CERAMIC 22PF 50V NP0 1206	Digi-Key	311-1154-1-ND	Yageo	CC1206JRNP09BN220	0.07700	0.15400
8	1	C16	100uF	CAP 100UF 50V ALUM LYTIC RADIAL	Digi-Key	P5182-ND	Panasonic-ECG	ECA-1HM101	0.29000	0.29000
9	6	D1, D2, D3, D4, D5, D6	_	LED OVAL NO FLNG ALINGAP AMBER	Digi-Key	160-1621-ND	Lite-On Inc.	LTL5V3SSS	0.18080	1.08480
10	7	D7, D8, D9, D10, D11, D12, D13	--	RECTIFIER GPP 100V 1A SMD MELF	Digi-Key	DL4002-FDICT-ND	Diodes Inc.	DL4002-13-F	0.49000	3.43000
11	1	J1	_	10x1 Inline header connector .025” contacts .100” spacing	_	_	_	_	_	_
13	1	J2	_	JTAG pin header connector 2x5 .025” contacts, .100” spacing	Phoenix	HWS1334	_	_	0.10800	0.10800
14	1	J3	_	CONN USB RECEPT R/A TYPE B 4POS	Digi-Key	A31725-ND	Tyco Electronics AMP	292304-1	0.67137	0.67137
15	2	J4, J5	_	DC Power Connectors 2mm PCB JACK POWER JACK	Mouser	806-KLDX-0202-A	Kycon	KLDX-0202-A	0.19000	0.38000
16	1	L1	10uH	INDUCTOR FIXED SMD 10UH 10%	Digi-Key	PCD1020CT-ND	Panasonic-ECG	ELJ-FA100KF	0.11646	0.11646
17	1	Q1	_	TRANS PNP 20VCEO 500MA	Digi-Key	2SB07790RLTR-	Panasonic-SSG	2SB07790RL	0.23500	0.23500

				MINI-3		ND				
18	6	R1, R2, R3, R4, R5, R6	300Ω	RES 300 OHM 1/4W 5% CARBON FILM	Digi-Key	300QBK-ND	Yageo	CFR-25JB-300R	0.00855	0.05130
19	3	R7, R9, R10	30K Ω	RES 30K OHM 1/4W 5% CARBON FILM	Digi-Key	30KQBK-ND	Yageo	CFR-25JB-30K	0.06400	0.19200
20	1	R8	22K Ω	RES 22K OHM 1/4W 5% CARBON FILM	Digi-Key	22KQBK-ND	Yageo	CFR-25JB-22K	0.06400	0.06400
21	1	R11	10KΩ	RES 10K OHM 1/2W 5% CARBON FILM	Digi-Key	10KH-ND	Yageo	CFR-50JB-10K	0.01222	0.01222
22	1	R12	20KΩ	RES 20K OHM 1/4W 5% CARBON FILM	Digi-Key	20KQBK-ND	Yageo	CFR-25JB-20K	0.06400	0.06400
23	10	R13, R14, R15, R16, R17, R18, R19, R20, R21, R22	_	_	_	_	_	_	_	_
24	2	R23, R26	1KΩ	RES 1.0K OHM 1/4W 5% CARBON FILM	Digi-Key	1.0KQBK-ND	Yageo	CFR-25JB-1K0	0.05400	0.10800
25	1	R24	4KΩ	RES 4.7K OHM 1/4W 5% CARBON FILM	Digi-Key	4.7KQBK-ND	Yageo	CFR-25JB-4K7	0.06400	0.06400
26	1	R25	100KΩ	RES 100K OHM 1/4W 5% CARBON FILM	Digi-Key	100KQBK-ND	Yageo	CFR-25JB-100K	0.06400	0.06400
27	2	SW1, SW2	_	Momentary Contact Switches (Not yet decided)	_	_	_	_	_	_
28	1	SW3	_	Positive Action On/Off Switch (Not yet decided)	_	_	_	_	_	_
29	1	U1	_	IC AVR MCU 64K 64TQFP	Digi-Key	AT90USB647-16AU-ND	Atmel	AT90USB647-16AU	6.00000	6.00000
30	1	U2	_	IC SRL EEPROM 512K 1.8V 8DIP	Digi-Key	25AA512-I/P-ND	Microchip Technology	25AA512-I/P	1.74000	1.74000
31	1	U3	_	IC ACCELEROMETER 3-AXIS 16LFCSP	Digi-Key	ADXL330KCPZ-RLTR-ND	Analog Devices	ADXL330KCPZ-RL	8.54145	8.54145
32	1	U4	_	IC ACCELER 50G DUAL-AXIS 8CLCC	Digi-Key	AD22285-R2CT-ND	Analog Devices	AD22285-R2	13.81900	13.81900
33	1	U5	_	IC VOLT REG FIXED POS SOT-223	Digi-Key	296-12290-1-ND	Texas Instruments	UA78M05CDCYR	0.18620	0.18620
34	1	U6	_	IC VREF W/SHUTDN 5V TSOT23-5	Digi-Key	ADR395AUJZREEL7CT-ND	Analog Devices	ADR395AUJZ-REEL7	1.95000	1.95000
35	1	U7	-	IC LDO V-REF 3.0V SOT23-3	Digi-Key	296-22643-1-ND	Texas Instruments	REF3330AIDB	2.38000	2.38000

								ZT		
36	1	U8	_	PHOTOCOUPLER OPIC DGTL VDE 6-DIP	Digi-Key	425-2205-5-ND	Sharp Microelectronics	PC900V0YSZXF	1.17000	1.17000
37	1	U9	-	IC USB-TO-UART BRIDGE 28MLP	Digi-Key	336-1160-ND	Silicon Laboratories	CP2102-GM	3.98000	3.98000
38	1	U10	-	IC BUFFER SGL OPEN DRAIN SOT353	Digi-Key	NL17SZ07DFT2GOSCT-ND	ON Semiconductor	NL17SZ07DFT2G	0.48000	0.48000
39	1	X1	32.768 KHz	CRYSTAL 32.768KHZ 6PF SMD	Digi-Key	728-1004-1-ND	Seiko Instruments	SPT2AF-6PF20PPM	0.35000	0.35000
40	1	X2	8MHz	CRYSTAL 8.000MHZ SERIES SMD	Digi-Key	XC1243TR-ND	ECS Inc.	ECS-80-S-5PX-TR	0.40500	0.40500
									Total:	**48.88**