On-board Instrumentation for an Electric Formula SAE Race car

THE UNIVERSITY OF WESTERN AUSTRALIA
Achieving International Excellence

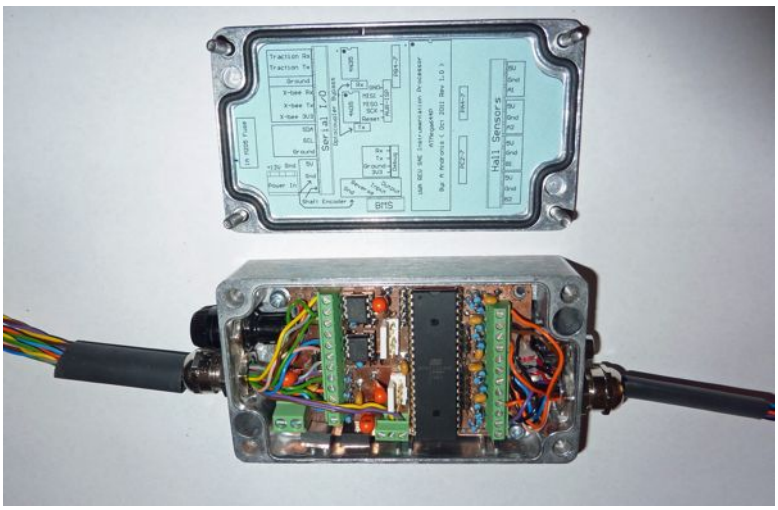
Alexandros Andronis

20262318

REV Project Group

Supervisor: Professor Thomas Braunl

Submitted: 4th November 2011



Abstract

In response to increasing petrol prices, researchers are constantly finding new ways of increasing the quality and efficiency of electric vehicles. The REV Project at the University of Western Australia is a team of undergraduate engineers and academics working together with the shared objective of building electric vehicles that are viable for the commercial market. Their most recent project is an Electric SAE race-car being built as an alternative to the traditional petrol engine Formula SAE race-cars. The focus of this dissertation is the electrical instrumentation, which is an essential component of the overall vehicle design. It involves the design and implementation of a system used to gather real-time information about the vehicle used for two main purposes: traction control and performance analysis. This dissertation is split into three major sections: Electrical design, hardware design and software design.

The electrical design section focuses on the electronics used to interconnect the system with an external power source, sensors and I/O devices. This includes noise filtering and circuit protection. A major component of the electrical system was the design and implementation of a Printed Circuit Board (PCB). Taking background research into consideration, the PCB and enclosure were designed to be Electro Magnetic Compatibility (EMC) compliant.

The hardware design component of this report focuses on the design and implementation of external sensors used in this vehicle. Three main sensors were developed as part of the instrumentation system: Pedal position sensors are used to determine the position of the accelerator and brake pedals. A rotary sensor is used to detect the angle of the steering wheel. An accelerometer/ gyroscope IMU is used to characterize the motion of the vehicle. The sensors are connected to a microcontroller (central computer) used to process raw data into useful information.

The software design section details the software framework used to interface these sensors with the system's microcontroller. This framework includes software for communicating with analog and digital sensors as well as communication protocols for communicating data to other I/O devices such as the traction control AVR (used for motor control) and an X-Bee transmitter, which is capable of transmitting data wirelessly to an on-site computer (for performance analysis)

Acknowledgments

I would formally like to acknowledge the following people that made this project possible: My supervisor Professor Thomas Bruanl, for giving me the opportunity to work on such an interesting and hands-on final year project. I would also like to thank the entire REV team for their support and in particular, Ian Hopper for all the technical advice he has given me throughout the year. Finally I would like to thank my family and friends for helping me through these tough times.

Nomenclature

Table 1 lists selected acronyms and abbreviations used in this document.

Table 1 Acronyms and Abbreviations

Term	Definition
ADC	Analog to Digital Convertor
ACK	Acknowledgment
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
AVR	Atmel Corporation 8-bit micro controller
BMS	Battery Management System
BPS	Bits Per Second
CAN	Controller Area Network bus
CE	Conformité Européene
COM	Communication device
CPU	Central Processing Unit
CRC	Cyclic Redundancy Code
DIL	Dial In Line
EF	Electric Formula
EMC	Electro Magnetic Compatibility
EMI	Electro Magnetic Emissions
EMS	Electro Magnetic Susceptance
ESD	Electro Static Discharge
FCC	Federal Communications Commission (USA)
GHz	Giga Hertz
GND	Ground
IMU	Inertial Measurement Unit
I^2C	Inter IC bus
IC	Integrated Circuit
I/O	Input / Output
kB	Kilo Byte
kHz	Kilo Hertz
kW	Kilo Watt
LED	Light Emitting Diode
LSB	Least Significant Bit
MHz	Mega Hertz
MSB	Most Significant Bit
NACK	Negative Acknowledgment
PCB	Printed Circuit Board
PWM	Pulse Width Modulation
REV	Renewable Energy Vehicle

Table 1 Acronyms and Abbreviations	
Term	**Definition**
RF	Radio Frequency
ROHs	Removal Of Hazardous substances
Rx	Receive
SAE	Society of Automotive Engineering
SCL	Synchronization Line
SPI	Serial Peripheral Interface bus
TTL	Transistor-Transistor Logic
TWBR	Two Wire Bit Rate
Tx	Transmit
USART	Universal Serial Asynchronous Receiver Transmitter
USB	Universal Serial Bus
UWA	University of Western Australia
V	Volts
Vcc	Power supply voltage
WiFi	Wireless Fidelity wireless local area network
X-Bee	Wireless communication protocol

Table of Contents

Figures

Tables

Equations

1. Introduction

1.1 The REV Project

The Renewable Energy Vehicle (REV) Project group is a research group run by the University of Western Australia's Electrical Engineering department. The team comprises of Electrical and Mechanical Engineering students and academics collaborating together with the shared objective of building electric vehicles that are viable for the commercial market, thus demonstrating the viability of sustainable energy sources. The project was started in 2008 as a response to rising fuel prices and the prospect of high efficiency electric motors eventually replacing petrol engines.

Since its establishment in 2008 the REV Project has converted three petrol engine vehicles into electric vehicles. The first of these was a 2008 Hyundai Getz, a 5-seater economy car fit for everyday use. The petrol engine on this vehicle has been replaced a 39kW electric motor as well as being fitted with 45 3.2V Lithium Ion Phosphate batteries and an on-board instrumentation system. The next car to be converted was a 2002 Lotus Elise, a 2-seater performance car, which was, fitted with a 54kW Electric motor and 83 3.2V Lithium Ion Phosphate batteries.

In 2010 the REV team developed an Electric Formula SAE race car is based on the 2001 UWA Motorsport (UWA Mechanical Engineering) SAE race car, which has been converted to run off a dual electric rear motor drive instead of its original petrol motor and has also been fitted with 15 Lithium Ion Phosphate batteries. The 2011 Electric Formula SAE Race car has been designed as a major improvement on its predecessor, it sports a modified chassis that is more compact and aerodynamic and uses 40 Lithium Ion Phosphate batteries. Another major improvement is that the 2011 version uses 4 hub motors (installed inside the wheel hubs) instead of using a dual motor drive to control the rear wheels. Each motor has a peak power rating of 15kW and a voltage of 48V.

1.2 The SAE-A Competition

UWA's 2011 Electric SAE car has been designed with the initial intent of entering it in the electric vehicle division of the Formula SAE-A competition in Melbourne,

during December 2011, competing with other universities throughout Australia. The competition imposes strict design guidelines for both the mechanical and electrical components of the vehicle. These restrictions are dictated by the US Formula SAE rules [1], the German Formula SAE Student Electric Rules [2], as well as the Australian Addendum to the US Formula SAE rules [3]. One of the main design restrictions that the SAE rules impose on the electrical instrumentation as described in section 4.1 of the German Formula SAE Student Electric Rules is that "*The entire high voltage and low voltage systems must be galvanically isolated*"[2]. The major implication behind this is the microcontroller reading data from low voltage sensors cannot be the same microcontroller that controls the high voltage motors. The approach that was taken to satisfy this constraint was to use separate microcontrollers for the instrumentation and motor control system and communicate data between these two microcontrollers by using opto-couplers to achieve galvanic isolation.

1.3 Objectives

The purpose of electrical instrumentation is to collect real-time information about a particular system and then use that information to control a certain variable of the system or process that raw information into something that can be used to measure the performance or current state of that system. The main objectives of this project were:

- To select an appropriate microcontroller to collect data from sensors,
- To integrate sensors into the vehicle to gather as much real-time information as possible about the current state of the vehicle,
- To develop a protocol to interface the microcontroller with the sensors to store useful information,
- To develop a protocol to transmit data to the traction control microcontroller (being developed independently by final year student Zac Brandstater)
- To develop a protocol to send all relevant data wirelessly to an onsite computer for performance analysis.
- To design and build a Printed Circuit Board (PCB) for the system

It was determined that the microcontroller would have to interface with the following sensors and communication lines

- Pedal position hall sensors:

- Measure the position of the position of the accelerator and brake pedals.

- Linear shaft encoder:
  - Measure the position of the steering wheel
- 6 degree of freedom Accelerometer/Gyroscope:
  - Measure the acceleration in the x, y and z directions.
  - Measure the yaw rate in the pitch, roll and yaw axis.
- Reverse switch:
  - Set the direction of the motors (forward or reverse)
- Battery Management System Control lines:
  - Turn the battery management system on/off.
  - Read the current state of the batteries.
- Serial communication lines connected to a galvanically isolated Traction-Control Microcontroller:
  - Communicate relevant feedback data from sensors
- Serial communication lines connected to a X-Bee wireless transmitter:
  - Send information to the onsite computer for performance analysis

Figure 1 illustrates the Instrumentation System vehicle installation location.

Figure 1 UWA REV Electric SAE Chassis

2. Literature Review

The author was unable to find relevant automotive instrumentation research papers with similar objectives to this project. This is mainly due to the electric car division of the SAE Australasian competition being fairly new. Little research has been done in this specific area except by other universities participating in the competition. Previous student theses relating to Electric SAE car projects from different universities have not been published due to these teams not wanting to lose their competitive edge by revealing design specifications. Paul Homes, who worked on the 2010 UWA REV SAE car instrumentation, pulled out of the REV project midyear therefore his unpublished work was unsuitable for use in this background review.

2.1 Hardware Design Considerations

Technical documentation on how to design printed circuit boards was thoroughly reviewed by the author. These documents described certain issues that must be taken account of when designing PCB boards and how to avoid/minimize their effects. The two main issues effecting PCB designs are electromagnetic interference (EMI) and electromagnetic susceptibility (EMS). The Electromagnetic Compatibility (EMC) design guide for ST Microcontrollers [4] defines EMI as the level of conducted or radiated noise sourced by a device and EMS as a devices level of resistance to electrical disturbances and conducted electrical noise. Therefore EMI is electromagnetic radiation generated from the system itself, which may interfere with other devices, and EMS is the system's resistance to EMI generated by other devices which may include power transients (sudden spike in current) or electro static discharges (ESD) which are a particularly large threat because ESD can be created by the human body coming in direct contact with electronics and can lead to permanent damage. The vehicle electric motors will be drawing large currents which will generate electromagnetic radiation that may interfere with the Instrumentation System.

2.2 Electromagnetic Interference

A major cause of EMI is high frequency digital switching circuits. This includes microcontrollers as they consist of tens of thousands of transistors switching in the MHz range [5]. EMI is radiation is measured between 30MHz to 6GHz but the AVR microcontroller internal RC oscillator operates at 8MHz with expected harmonics less than 30MHz and is unlikely to generate significant EMI. Although microcontrollers

are shielded from emitting EMI, the tracks leading from the I/O pins on the PCB act as antennas, which may transmit EMI from the microcontroller. This is because each loop and track includes parasitic inductances and capacitances, which absorb and radiate EMI signals [5]. Making the track length smaller and placing track loops on the same PCB layer and closer to each other (thus reducing the surface area of the tracks) can reduce inductance. As a rule of thumb interconnections should be no longer than 1/20 of the wavelength to minimize EMI. If required ferrites can be placed at the ends of cables that exit the electronics housing, to absorb emissions [6]. Ferrites convert electrical cables into lossy inductors where high frequency electrical noise is dissipated as heat.

2.3 Power Supply Filtering

Power Supply filtering is another method used to decrease the amount of EMI generated by the circuit. Because the main power loop is used by all parts of the circuit, it must be considered with special attention. Supply loops should be decoupled to ensure that signal levels and power currents do not cause interference [4]. Decoupling capacitors are used to reduce the EMI generated by the power supply; they are connected from the supply to ground and also across the voltage input pins of devices such as AVR's to shunt any input noise. Electrolytic capacitors of high impedance are used to filter low frequency noise, however they do not filter high frequency noise because electrolytic capacitors become inductive at high frequencies [4]. Therefore to filter high frequencies ceramic capacitors should be used. Capacitors should be placed as close as possible to the supply pins to minimize the surface area of the resultant loop [4], the larger the resultant loop, the more EMI that is emitted. At the power supply, using both types of capacitors is recommended but at the input pins of connecting device only the ceramic capacitors are required. High impedance low pass filters placed across sensor input pins on the AVR can also decrease the EMS of a system by making it less susceptible to transients.

2.4 Multilayer Printed Circuit Board Design

An effective way of decreasing the EMI of a PCB is by using a multilayer board, which is more beneficial than using a single layer board even if the components are only being mounted on one side. Firstly the power and signal ‘‘loop areas’’ are minimized, reducing emissions and decreasing the EMS of the board [6]. Secondly,

the power and ground impedance levels are lowered which reduces power and ground perturbations [6]. Third the presence of power and ground planes greatly minimize crosstalk between traces [6]. The characteristics of a multilayer board are caused to the "*image plane effect*": By placing a current carrying wire close to a metal surface, most of the high frequency current returns directly under the wire. A transmission line is formed by the wire's ''*mirror image*" located over the metal surface. With equal and opposite currents, these transmission lines do not radiate well, nor do they pick up external energy [6]. As in the case of this project, a double layer board can still utilize the image plane effect by having a dedicated ground plane layer. Having this ground plane reduces EMI and EMS of the board. A double layer board is almost as effective as a multilayer board and therefore is sufficient for the purpose of this project.

2.5 Electrical Shielding

Another way in which EMI and EMS can be reduced is by encasing the PCB in a material with high permeability and low resistivity, this is known as "shielding". Due to the characteristics of the material, only a very small amount electromagnetic radiation can enter the shield increasing the electromagnetic susceptibility and only a small amount can exit the shield minimizing the amount of electromagnetic radiation affecting other electronic devices. Shielding also protects the electronics from electrostatic discharge, as the electronics is no longer directly exposed to potential ESD from direct contact and the PCB is essentially isolated from all directions. Shielding the external electrical cables and bonding the shield to the case also improves the EMI and EMS performance.

2.6 ESD Precautions

When assembling the board, certain precautions need to be made to protect the electronics from ESD. Some of the recommended precautions that can be taken to prevent ESD related destruction are: use of a conductive mat to place the components on and a static control wrist strap should be used as well as conductive shoes to ground the human body. Conductive floor mats and a work suit with anti-static measure are other alternatives [1]. It is also recommended that components be kept in their anti-static bags until required to prevent any unnecessary exposure to ESD.

2.7 FCC and CE Certification

For an electronic device to be certified for consumer use it needs to pass certain electromagnetic compatibility (EMC) regulations. If a product does not pass tests conducted by authorities such as the FCC (U.S) or CE (Europe) then it cannot be sold on the market [7]. A device must pass certain tests including a test on the frequency and quantity of emitted EMI and a test ensuring that the device is immune to frequencies that are commonly used for wireless transmissions in particular countries. Australia follows the European Standards of EMC regulations [7].

3. Electrical Design

3.1 System Voltage

The System can accept an input voltage of 8-28V DC, however will only be running off a voltage of 12V DC through a step-down switch mode power supply connected to the vehicle's internal battery packs. This input voltage is then passed through 5V DC and 3.3V DC voltage regulators. Two regulators are used because components in the system run on different voltages. A secondary function of the regulators is they protect the system from accidental over-voltage. The linear regulators operate by dissipating any residual voltage when the input voltage is above the desired voltage level. Both voltage regulators are connected in series as illustrated by Figure 2. The output of the 5V regulator is connected to the input of the 3.3V so that less energy is wasted because a smaller voltage needs to be dissipated regulating a 5V supply to a 3.3V output than the alternative of connecting it directly to the 12V input which is a much larger voltage difference. With the two regulators connected in series less heat is generated by the system. Load testing indicates that the linear regulators run cool enough not to require a heat sink.

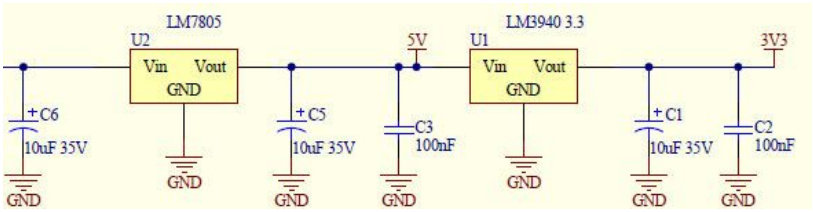


Figure 2 Power Supply

3.2 Power Supply Protection

The system is protected from reverse voltages (accidental reversal of the polarity of the power source) by use of a diode. This is based on the characteristic that diodes only allow the flow of current in one direction. When the power polarity is connected correctly the diode is operating in forward bias and allowing current to flow. When the power source is connected in reverse, the diode is operating in reverse bias preventing current to flow in the wrong direction, resulting in an open circuit. Thus the circuit is protected from any reverse voltages, which would normally damage the electronics.

The system is also protected from excessive amounts of current flowing in, by a fuse connected in series with the diode as illustrated by Figure 3. Fuses are designed to break the circuit if too much current is being drawn; in this system the fuse is rated at 1A (i.e. 12V = 12W maximum input power or 5W at the linear regulators). This offers external power source protection from overloading if the circuit fails or an electrical cable short circuit occurs.

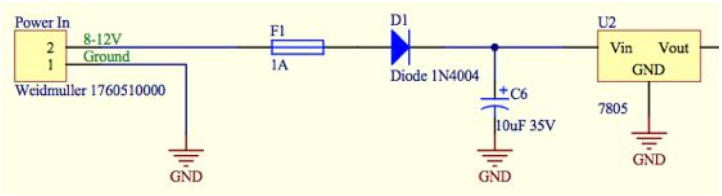


Figure 3 Input Power Protection Circuit

3.2 I/O Protection

All external sensors that are either not opto isolated or pass through a 5V to 3.3V attenuator are connected directly to the micro controller with no additional protection. The micro controller provided diode protection for all I/O lines as illustrated by Figure 4[8]. The diodes clamp the I/O lines to ground and 3.3V. The instrumentation system must be wired to the intended device otherwise an external voltage of sufficient magnitude will damage the micro controller.

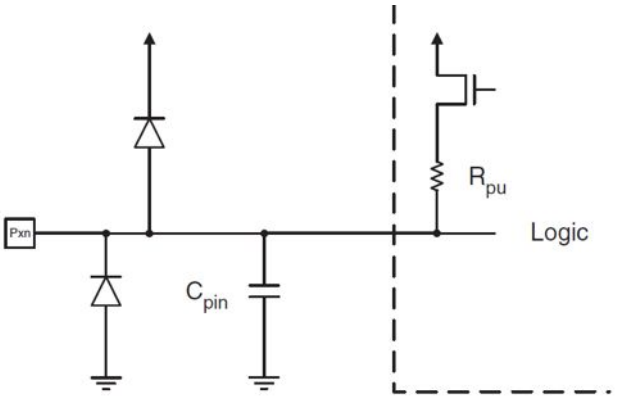


Figure 4 AVR Micro Controller I/O Diode Protection [8]

3.3 Digital Logic Levels

Although the microprocessor itself supports an input voltage between 2.7-5.5V[8] it was decided to run the system at 3.3V so that it is compatible with certain sensors such as the accelerometer/gyroscope board and the X-Bee wireless transmitter, both with a maximum input voltage of 3.3V. It would not be enough to run these sensors

themselves at 3.3V because the microprocessor must be run at the same voltage so that the logic levels are compatible for communicating between devices. For example, if a sensor is running at 3.3V and the microcontroller is running at 5V, when communicating, a logical high (1) corresponds to 3.3V for the sensor and 5V for the microcontroller, so since 3.3V is significantly less than 5V it could be interpreted as a logical low (0) by the microcontroller depending on the microcontroller's threshold voltage for a logical high. An alternative method of solving the logic level problem would be to connect the communication lines to CMOS logic level converters to convert the input to the sensor to 3.3V logic and the input to the AVR to 5V logic, however this would make the circuit board design unnecessarily more complex, so the former design was chosen for simplicity.

A consequence of the chosen design is that sensors such as the hall sensors and the linear shaft encoder do not operate at voltages as low as 3.3V therefore in order to maintain correct logic levels these sensors are run at a supply voltage of 5V and use a simple voltage divider to attenuate the voltage to a 3.3V logic level as illustrated by Figure 5.

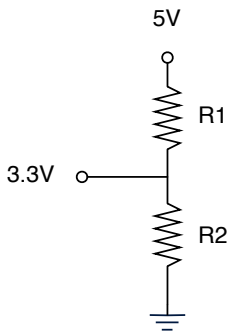


Figure 5 5V to 3.3V Attenuator

As shown by the simple voltage divider above, the resister values must be chosen such that:

$$\frac{R_2}{R_1 + R_2} \times 5V = 3.3V$$

Equation 1 5V to 3.3V Attenuation

R_1 was chosen as 10kΩ and R_2 was chosen as 22kΩ. These resister values were chosen because they are common components and readily available at local electronics suppliers such as JAYCAR and ALTRONICS. These resister values were also chosen to provide high 32kΩ load impedance, to ensure that the voltage drop across the sensor output is minimal.

3.4 Noise Filtering

The output of each sensor has been connected to a low pass filter; the purpose of this is to produce a more stable output signal by filtering out any high frequency noise. As a consequence of this the bandwidth of the signal is limited to the bandwidth of the low pass filter thus limiting the sample rate. This is a fair trade off, as the sensors don't necessarily need to be sampled at the maximum sample rate of the sensor. A 16Hz low pass RC Filter was designed, this consists of a 100kΩ resister and 100nF capacitor as illustrated by Figure 6. Nyqueist's theory requires the maximum sample rate of a sensor is equal to twice the bandwidth. In this case the maximum sample rate of a sensor connected to a 16Hz low pass filter is 32Hz, that's 32 samples per second which is more than enough for both the pedal sensors and rotary sensor.

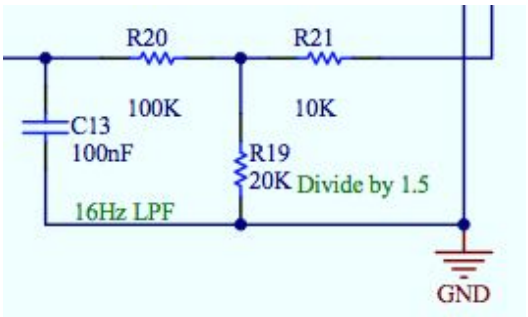


Figure 6 16Hz Low Pass Filter

3.5 Power Supply Filtering

Capacitors are placed across the power supply output pins as well as on either side of voltage regulators. 10μF electrolytic and 100nF ceramic capacitors are used to filter out low and high frequency noise as described in the background review. At the voltage input pins of the AVR's 100nF ceramic capacitors have also been used to filter out high frequency noise.

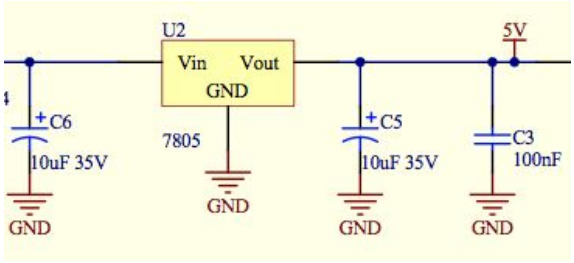


Figure 7 Linear Regulator Power Supply Filtering

3.7 PCB Design

A conceptual diagram was drawn up as an early design of the system. This was used to show how the microcontroller would interact with the sensors and external IO devices (other AVR's, X-Bee module). It was a simplified model of the system but was used as a basis for the final design.

A final circuit diagram and PCB for the system was designed in Altium Designer 10 (this can be found in Appendix A). The decision to use this software as opposed to simpler software such as Eagle, was based on the author's familiarity with this software suite, through vacation work at an Electronics/Software Engineering company. The program itself is quite complex and difficult to use however it is the industry standard for PCB design. All connectors, capacitors resisters and diodes mounted directly onto the PCB via through holes. No surface mount components are used. The microcontroller and opto-couplers are not directly soldered to the board, they are connected via 6 and 40 pin dual inline (DIL) turned pin sockets, enabling these components to be easily removed and replaced. All external sensors, power lines and communication lines are connected to the board via screw terminals making it easy to connect/disconnect external connections when required. The external wiring diagram is attached to the lid of the housing as illustrated by Figure 8.

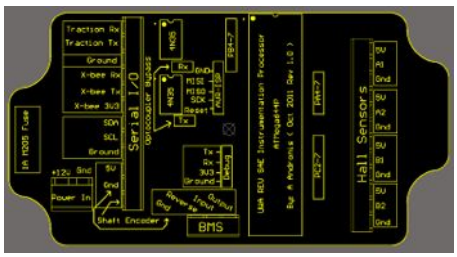


Figure 8 External Wiring Diagram (Attached to lit)

The fuse is connected to the circuit using an externally accusable M205 fuse holder, which allows for fuses to be replaced as illustrated by Figure 9.

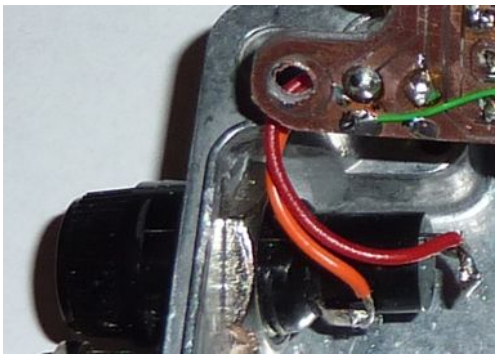


Figure 9 Externally Accessible M205 Fuse Holder

A double layer board is used for the PCB; components are mounted to the top layer, while interconnections between components are on the bottom layer. Minimum Track width is ½mm (IPC standards minimum track width is ¼mm). Minimum clearance between tracks is ¼mm (IPC standards minimum clearance is.15mm). Minimum Track widths and clearances where calculated by Altium Designer based on IPC standards, for a maximum current is 1A. For the purpose of this PCB track widths and clearances were increased slightly to accommodate for the board being fabricated by a hobbyist rather than professionally manufactured. As described in the literature review the top layer *(Figure 10)* acts as a ground plane to reduce the amount of EMI and reduce the PCB's EMS.

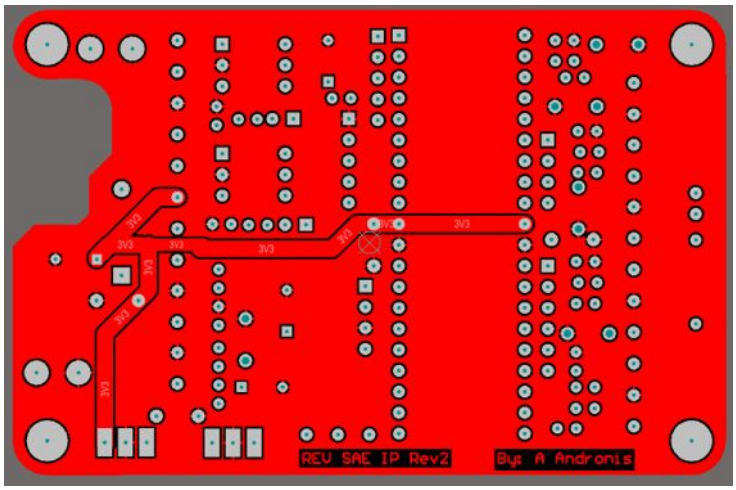


Figure 10 PCB Top Layer (Ground Plane)

The handmade PCB does not contain electrically plated through holes and require the components to be soldered on the top and bottom layer if a components connection is connected to ground as illustrated by Figure 11.

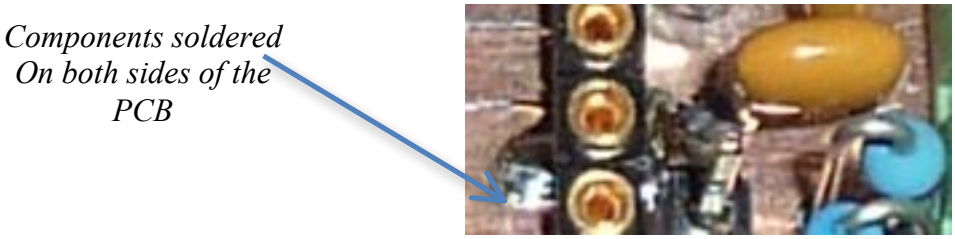


Figure 11 Double Sided PCB (No through Hole Plating) Ground Plane Link

The instrumentation system was not tested for EMI and EMS as testing facilities were not easily available. If subsequent EMI and EMS identify a problem it will most likely be due to external wiring electromagnetic radiation. Clip on ferrites or ferrite bead can be added to the wiring harness or individual wires as illustrated by Figure 12.

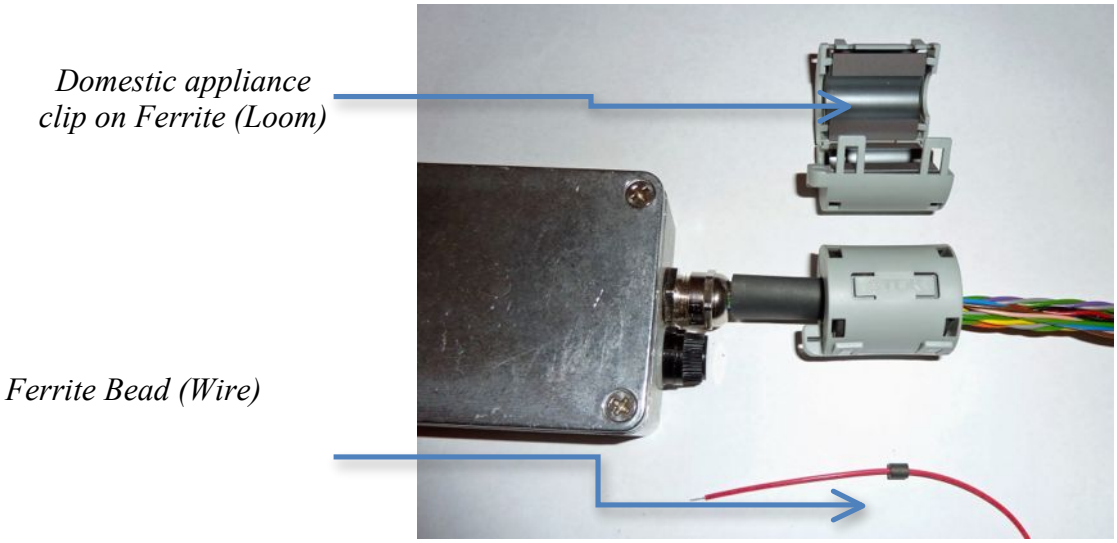


Figure 12 EMI/EMC Filtering

The housing chosen for the PCB is an aluminum diecast box, which is fire proof and watertight as illustrated in Figure 13. This makes the system more robust and suitable for harsh weather conditions such as rain, which is quite likely to be a problem when the Formula SAE vehicle is out on the track.

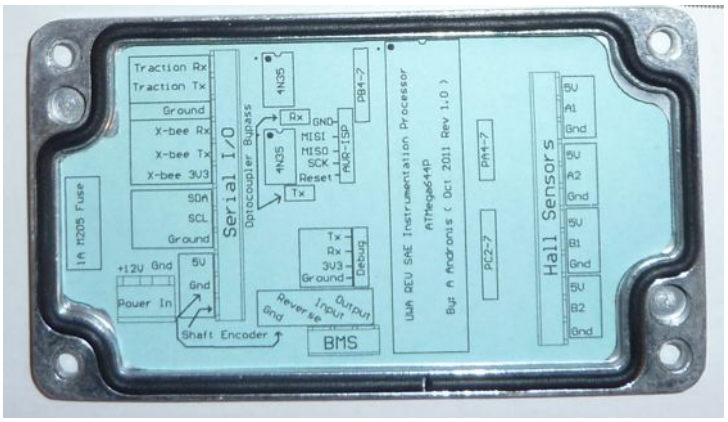


Figure 13 Diecast Box Hermetic Rubber Seal

It is also a rule for the SAE-A competition that the electrical systems of the vehicle are not affected by rain. External wiring enters the diecast box via two sealable rubber glands as illustrated by Figure 13. The wiring loom must be fitted with a plastic sleeve, like heat sing tube, to complete the water tight seal. If an overall electrical shield is added to the wiring loom then the shield must be electrical connected to the diecast box through the rubber gland.

Figure 14 Water Tight Rubber Gland

The metal casing shields the external sensors I/O devices from any EMI generated by the PCB as well as protecting the device from EMS. The PCB was designed to fit the Aluminum case perfectly and line up with the screw mounting points. In order to allow for connections from external components such as power source, sensors and I/O devices holes were drilled into each side of the case and fitted with watertight cable glands. This allows for cables to be fed into the box, while still remaining watertight. A hole was also drilled for the M205 fuse holder as well as a hole for a blue status LED.

4. Hardware Design

4.1 AVR Microprocessor

An AVR ATMega644P was selected as the most suitable microcontroller for the system. AVR was chosen as the preferred brand of microcontroller because they have an extensive development knowledge base, they can be programmed in C and are cheap to replace.

Another option that the author looked into was to use an Auduino Evaluation Board, as this is what was used in the 2010 UWA Electric SAE car. Ultimately the author decided against this option. A major reason against using Auduino's is that they are too expensive to replace, a new Auduino retails at $60 while an AVR only costs $10-$15, this makes the Auduino less financially viable. This is particularly important because part of the criteria's for judging the SAE-A competition is cost efficiency of a team's vehicle. Finally the Audruino is seen as a hobbyist's board as it comes in an evaluation kit, AVR's are a more professional solution and are more robust in terms of functionality. There is a large range of AVR's that have different features and are available to suit different requirements there is only a limited range of different Auduino evaluation kits.

A third alternative was to use the Eyebot M6 developed in-house by the UWA EE robotics department and Professor Thomas Bräunl. This integrated system contains an ARM9 processor running Linux and a built in touch screen display, which could be potentially useful as a driver side display [9]. Other features include dual color camera sensors, ADC inputs and built in motor controllers (PWM outputs)[9]. Most of these features would be wasted, as this project does not involve any motor control or image processing. Use of a system with a full-scale operating system was also deemed unnecessary. It was decided that the touch screen itself was not required and would pose as too much of a distraction to the driver of a race car, It wouldn't even be essential for the driver to know their current speed as there are no real speed restrictions on a race track.

Thus an AVR powered onboard instrumentation system was chosen. The next decision was to find the most suitable AVR. This was a rigorous process, the AVR was chosen from a list of potential microcontrollers which was downloaded from the

AVR website along with a list of features that was compared against the selection criteria.

Initially the author researched Automotive AVR microcontrollers with a "Control Area Network (CAN) Bus", the industry standard communication protocol for automotive instrumentation. CAN is a protocol that operates by using a single communication line to transmit sensor data simultaneously between multiple microcontrollers i.e. anything that is transmitted by any sensor can be read by any microcontroller connected to the bus. After further research it was decided that this feature was not required because there are only two microcontrollers that need to communicate detailed information between each other. Implementation of this feature would be very difficult and time consuming, a simple serial communication protocol using the AVR's USART communication lines was chosen as an alternative to CAN because the system only required point to point communication.

Another important design choice was to choose a microcontroller with two or more USART interfaces. One is required for communication with the traction control microcontroller and the other is required for communication with the X-Bee wireless transmitter, to transmit data to an on-site computer. This narrowed down the list of potential microcontrollers considerably, as it is not very common for AVR's to have more than one USART connection.

The number of Analog to Digital converter inputs was also taken into consideration. AVR microprocessors generally only have 8 analog inputs; the initial design of the system required 12 ADC inputs. Therefore there were two potential options, 1) select a microcontroller with more analog inputs 2) extend the number of ADC inputs using an external ADC 8-input module connected via SPI/ I^2C (Digital IO Device). The latter option was selected due to the lack of availability of devices with > 8 ADC inputs that also met other aspects of the selection criteria. The final design only needed 5 inputs (with 3 spare) but the option is always available to extend the number of ADC inputs if required for future revisions.

To make development easier/faster it was preferred that a microcontroller that supports PDIP type packaging was chosen. Compared to chips with TQFP/QFN packaging. PDIP does not require any soldering or a breakout board to access all pins. This is a packaging with external pins, which will directly plug into a breadboard/

prototyping board, therefore it can be swapped very easily in the case of the chip failing or needing to upgrade to a different type of microcontroller (with compatible packaging).

A final design consideration that was not as essential but still needed to be taken account for was the amount of flash memory the AVR contained. It was found that a 16kB chip would not have enough memory to hold the instrumentation processing software, so a 64kB chip was chosen instead as a precaution to ensure that the AVR doesn't run out of storage.

Atmega644P should not be confused with Atmega644; they are both very similar devices however the Atmega644 lacks the additional USART ports. Atmega644A is another alternative and essentially identical to Atmega644P without certain unused features, however it is not unavailable at local suppliers such as Element14.

4.2 Pedal Box

Sensors needed to be designed to track the position of the Accelerator and Brake pedals. The outputs of these sensors are an essential input to the traction control system. The position of the accelerator pedal is used to determine how much power should be supplied to the motor based on the driver's desired speed. For safety reasons the vehicle is required to have mechanical brakes i.e. the system cannot rely on entirely electric braking, however it is still very useful to measure the position of the brake pedal for purposes such as regenerative braking.

The main idea behind a pedal sensor is that the further down the pedal is pushed the higher the sensor output voltage is. The output of this sensor must be approximately 0V when the pedal is not being pressed and 3.3V when it has been pressed all the way down. It is required that this relationship is as linear as possible in order to correlate absolute position with voltage. Linear hall sensors are utilized to measure pedal position. They operate by outputting a voltage, which is proportional to an applied magnetic field [10]. Hence they are sensitive to magnetic field strength and in theory should be able to detect the presence of a nearby magnet, and output a voltage proportional to absolute distance.

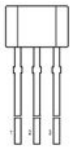


Figure 15 Linear Hall Sensor [10]

The diagram illustrated in Figure 15 shows the pin out of the linear hall sensors used by the pedal position sensors. It is an analogue sensor with a supply voltage of 5V and 3 pins corresponding to 1:$V_{cc,}$ 2:GND and 3:V_{out}. As the output voltage is higher than the 3.3V operating voltage of the ATMega644P, a voltage divider is used to attenuate the voltage to a compatible voltage of 3.3V as described in Section 2 of this report.

Two different tests were conducted to determine the linearity of the hall sensor. In the first test, the hall sensor was connected to a 5V power source and a magnet, which was initially positioned 2cm away and is slowly moved directly towards the sensor until they are touching. This is the way that the pedal position hall sensors operated in the 2010 REV SAE vehicle's pedal box and the results are illustrated in Figure 16, evaluating the performance of this design.

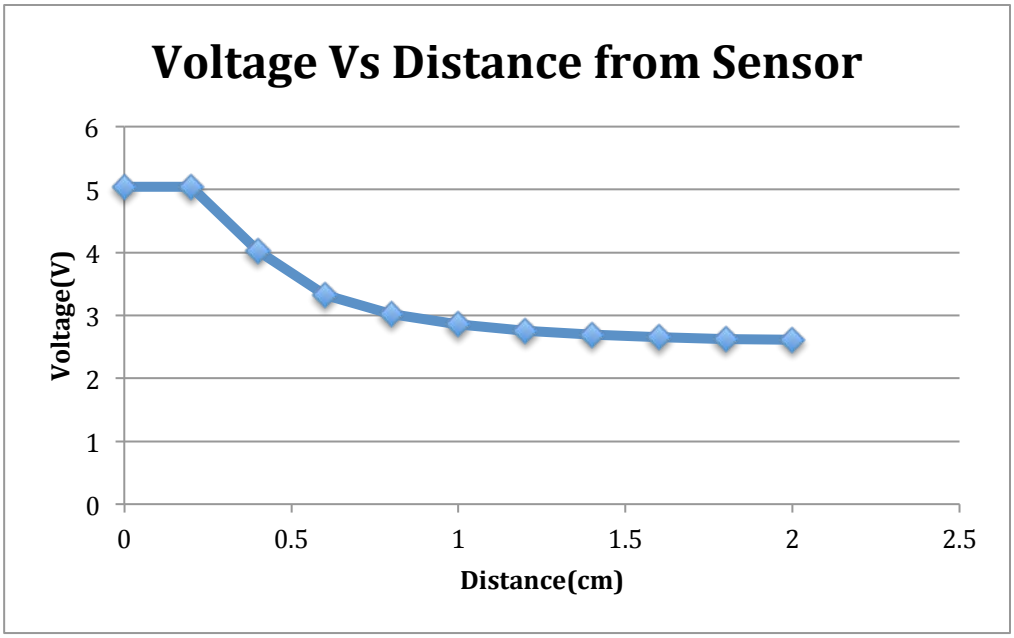


Figure 16 Hall Sensor Linearity Test – Case 1

This is clearly not a very linear relationship between voltage and distance from the sensor. At distances > 0.6cm away the signal does not vary significantly enough to give an adequate reading as voltage only varies by 0.72V over the entire 1.5cm range and increases at a very slow rate which isn't quite linear. Thus in this region the resolution is not high enough. At distances between 0.5cm and 0.2cm from the sensor the relationship is more linear, however this is only for a very short distance of 0.3cm as the sensor saturates at a maximum value of 5V even before the magnet has made contact.

In the second case that was tested, the sensor was again connected to a 5V supply but this time, sliding the magnet past the sensor. The test started with the centre of the magnet 1cm to the left of the centre of the sensor and ended with the centre of the magnet 1cm to the right of centre of the sensor. The total distance travelled by the magnet was 2cm.The results of this test are illustrated in Figure 17.

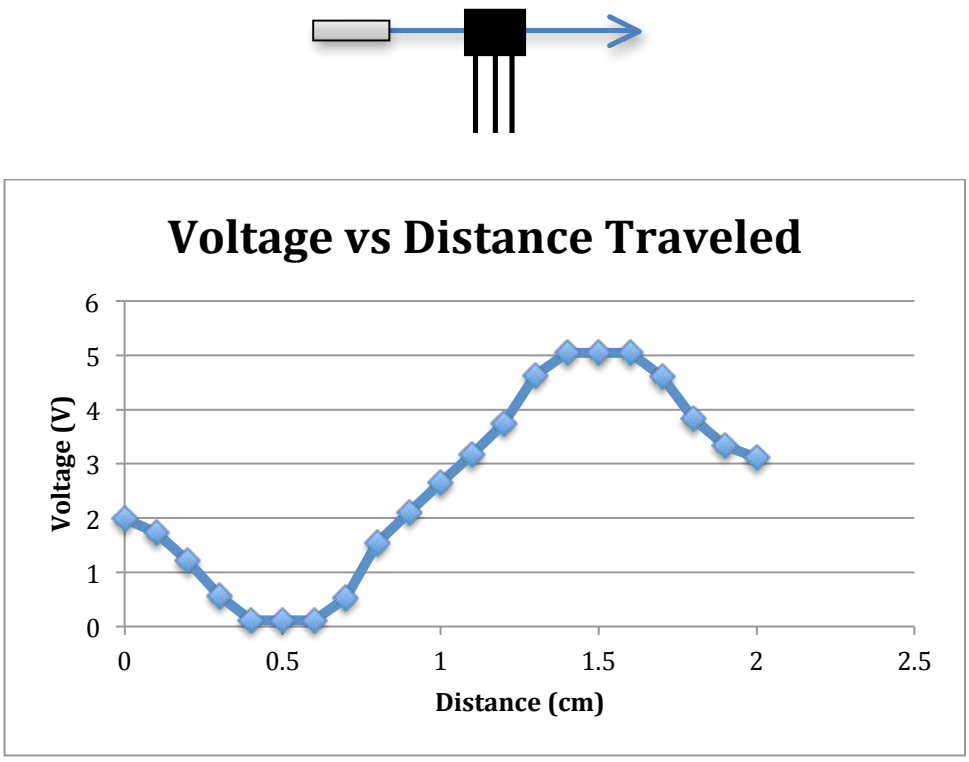


Figure 17 Hall Sensor Linearity Test – Case 2

This appears to be a much more linear relationship than that of case 1. There is a linear relationship mapping a voltage of 0 – 5V to a variation in distance between 0.5cm to 1.5cm covering a range of approximately 1cm. This corresponds to the region where the magnet is moving directly past the sensor. There is a small dead-

band on either side of this 1cm region but this is so small that its effect on the sensor would be negligible. The final design of the pedal position sensors is based on this configuration rather than the configuration used in the 2010 REV SAE vehicle, as it provides a more linear signal, which is accurate over a larger range.

A requirement of the Electric Formula SAE-A competition is that each pedal must have a backup measurement sensor as a fail-safe [2]. This redundant sensor must measure exactly the same quantity as the primary sensor so it can be used to determine if the pedal position sensor is working properly. If one of the two sensors on either pedal stops working then the instrumentation system should shut down the entire system to prevent any unpredictable behavior, such as motors running at incorrect speeds or becoming unstable due to corrupted measurements from the sensors.

The final implementation of the pedal position sensors is shown in Figure 18 and Figure 19. Hall sensors are mounted to part of the frame of the vehicle on either side of each pedal. Permanent magnets are bolted to each side of the pedal, in such a way that pushing the pedal will cause the magnets on either side of the pedal to slide past their corresponding hall sensors. These sensors were installed in a modified version of the 2010 REV SAE vehicle's pedal box for testing purposes. The 2011-12 vehicle will use a completely redesigned pedal box, which will better align the magnets and their hall sensors.

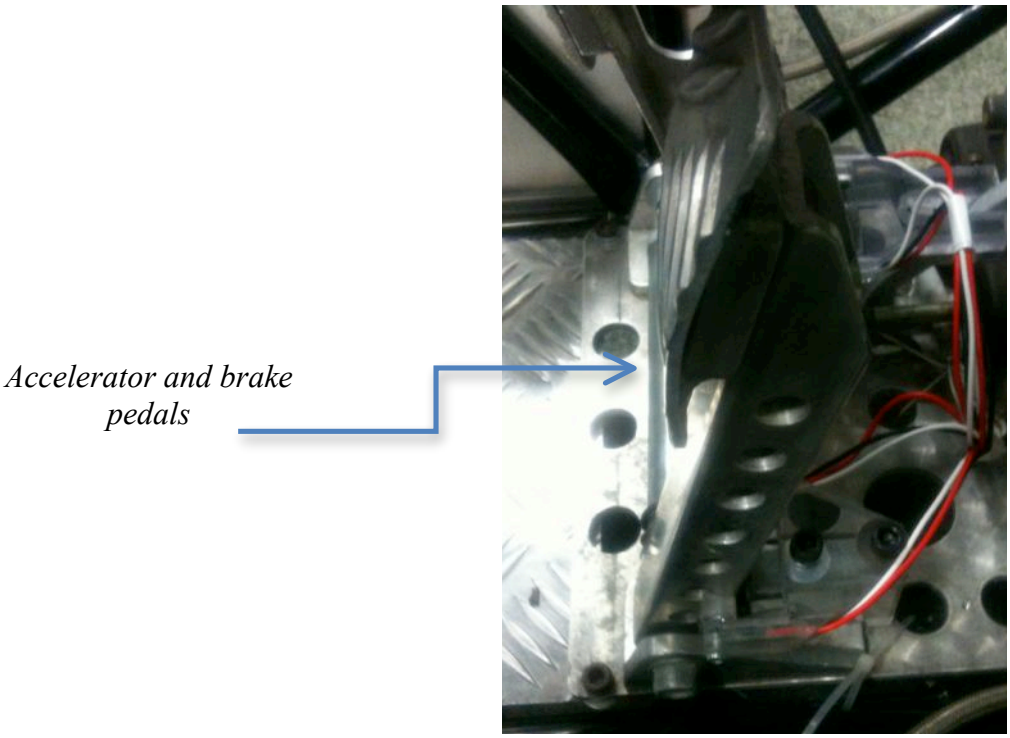


Figure 18 Hall Sensor Pedal Mounting

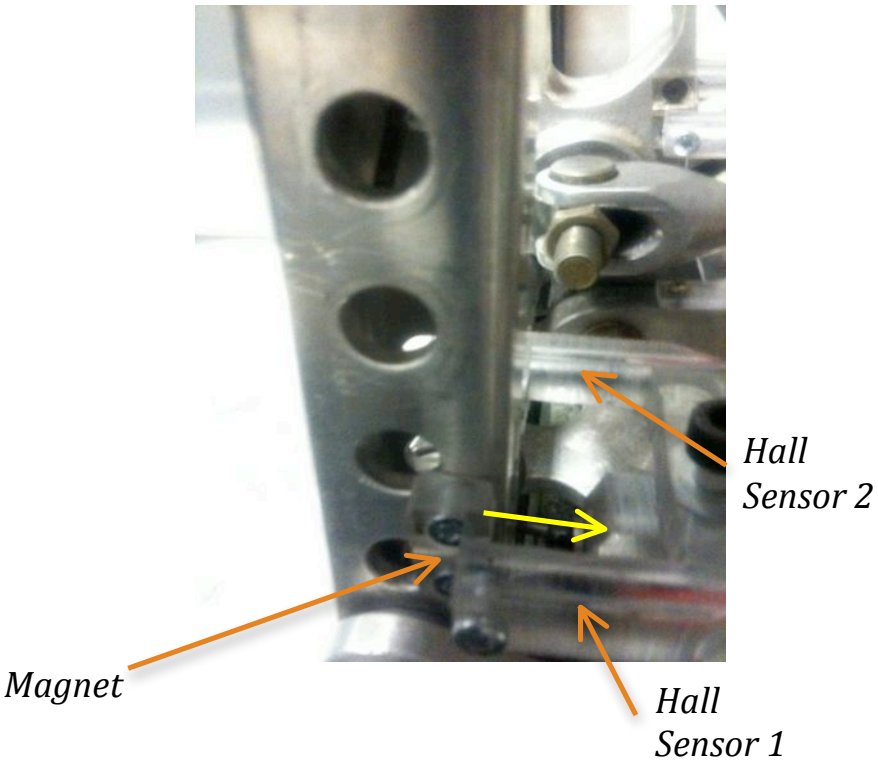


Figure 19 Magnet and Dual Hall Sensor Mounting

Pedal position sensors were tested by connecting both sensors of each pedal to an oscilloscope, it was shown that the output signals of each sensor did vary significantly as there was an offset between the voltages of the two sensors at the same mechanical pedal positions. The reason for this was that the hall sensors were not aligned properly, which also resulted in hall sensors not always operating within their linear region. The maximum voltage output occurred before the pedal had been completely pressed down therefore pushing the pedal further would cause the output voltage saturate and then start to decrease.

These negative characteristics can be fixed by properly aligning the hall sensors. This requires a redesigned pedal box, as the hall sensors need to be installed lower down on the pedal in order to better utilize the hall sensor's linear range.

4.3 Digital Accelerometer/Gyroscope Board

An accelerometer is used to measure the vehicle's acceleration in the x, y and z directions and a gyroscope is used to measure pitch roll and yaw rates, i.e. changes in angular position around the x, y and z rotational planes. Acceleration in the x, y and z direction is measured in g's ($1g = 9.8m/s^2$) and pitch, roll and yaw is measured in degrees/second. An external device known as a 6-axis inertial measurement unit (IMU), containing both an accelerometer and a gyroscope in an integrated circuit, can be connected to a microcontroller to measure these quantities. This device comes in both digital and analog variants and part of the design process was to choose a specific device that would be suitable for the instrumentation system.

The author's initial choice was to use an analog IMU, as this would be easier to interface with the microcontroller than a digital device. An analog sensor left over from the incomplete instrumentation system of the 2010 REV SAE vehicle was the first device tested as the vehicle's IMU. This device was a 6 Degree of Freedom (6DOF) Analog Combo Board Razor (SEN-10010), manufactured by SparkFun Electronics [11]. The device ran on a 3.3V input and featured an accelerometer with 300mV/g sensitivity over a range of $\pm$ 3g and a gyroscope with a $0.83mV/°/s$ or $3.33mV/°/s$ (4x amplified) sensitivity over a range of $\pm$ $300°$[11].

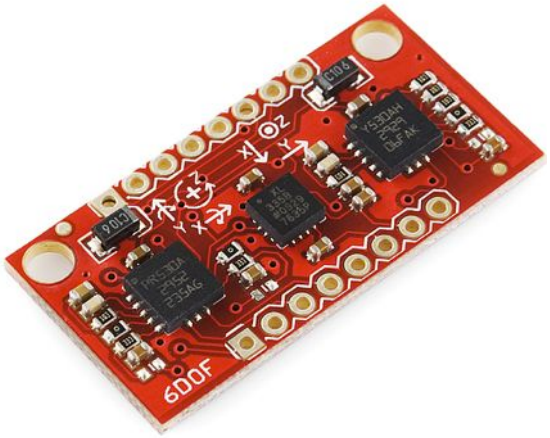


Figure 20 Analog Gyro/Accelerometer IMU

This IMU was tested by connecting its outputs to an oscilloscope and moving the device in different directions and rotational angles. It was concluded that this device was working as expected, however plugging it into an AVR and printing the output to a terminal via USART resulted in values, which did not correspond to the expected analog voltages, it was apparent that there was a DC offset. This was attributed to the output impedance of the IMU being much higher than that of the AVR microcontroller. This problem could be easily solved, by using a voltage follower (buffer) to lower the output impedance of the IMU.

For several reasons the author decided against using this IMU in the final system design. A major reason why this device was unsuitable is because it required 6 ADC inputs. Considering the ATMega664P only has 8 ADC inputs and 5 are reserved by other analog sensors, an additional external 8-input ADC device would be required for there to be a sufficient amount of ADC inputs for the system. Use of this external ADC device and voltage followers would result in a much more complex design while using a digital IMU would require a much simpler design. The particular analog IMU being used for testing had also been discontinued, meaning that SparkFun Electronics no longer supports or manufactures it. Therefore if this device was to be used in the final design, and it malfunctioned, the author would not be able to replace it and would need to find another alternative either way.

The IMU chosen for the final design of the system was a 6DOF Digital Combo Board (SEN-10121), which is also developed by SparkFun Electronics [12]. The main difference between the two boards is that the SEN-10121 is a digital sensor. This IMU

also has a 3.3V input voltage. The accelerometer used on this IMU is an ADXL345, manufactured by Analog Devices. It has a 10-bit output resolution with a sensitivity of 256 LSB/g over a range of ± 2g [13]. It can be set to operate over higher ranges but ± 2g was chosen, as the electric Formula SAE vehicle is not expected to exceed these maximum accelerations at any time. The Gyroscope used by this IMU is an ITG-320, manufactured by InvenSense. It has a 16-bit output resolution with a sensitivity of 14.375 LSB/°/s [14]. It is clear that the digital IMU has a much better sensitivity than the analog IMU, especially the gyroscope output, which has a 16-bit output resolution. This is much larger than the ATMega644P's 10-bit ADC resolution. Sending a command to the corresponding device can set its sample rate, by default the sample rate is set to 100Hz.

Rather than requiring 6 analog inputs, the digital IMU only connects to 2 pins on the AVR, the SDA (data line) and SCL (clock line), These pins are used by the I^2C digital communication protocol. Both the SDA and SCL lines require pull up resisters to operate, maximum clock frequency of the bus depends on the value of the pull up resisters (connected between the SDA or SCL pin and Vcc). For a standard SCL frequency of 100kHz, the minimum recommended resister value is 10kohms.

Figure 21 IMU 6DOF Digital Combo [12]

4.4 Steering Wheel Angle Sensor

The purpose of a steering wheel angle sensor is to measure the angular position of the vehicle's steering wheel, i.e. how far to the left or right it has been turned. For automotive applications, a linear shaft encoder is normally used to measure the steering wheel position. There are two different types of shaft encoders, incremental

and absolute: incremental encoders set a zero point based on the initial position of the shaft when the device first turned on. This means that output values for different mechanical positions can vary between uses. Absolute encoders have a unique (fixed) output value for each mechanical position, independent of initial shaft position. Therefore it is clear that for this system an absolute shaft encoder is more suitable, as an incremental sensor would need to be calibrated every time the vehicle is switched on.

Unfortunately, commercial heavy-duty shaft encoders cost $500+, which exceeds this project's budget. A cheaper alternative was to use an MA3 miniature absolute magnetic shaft encoder, manufactured by US Digital [15] illustrated in Figure 22. Its only shortcoming is it has a much smaller diameter than commercial shaft encoders. Its size and diameter may have an impact on performance, however this is a comprise that needed to be made to find a more economic option. This device runs off a 5V supply, so like the pedal position hall sensors, needs a voltage divider to attenuate the output voltage down to a range of 0-3.3V. The MA3 is an analog sensor with a maximum sampling frequency of 2.6kHz[15] (However due to the bandwidth limitation caused by the low pass filter, sampling will be at a much lower rate, see Section 2). The maximum analog output resolution is 10-bits, which is the same AVR's ADC resolution. The specification sheet recommends that the output impedance is greater than or equal to 4.7kOhms for a more linear operation [15].

Figure 22 US Digital MA3 Linear Shaft Encoder[15]

A test was conducted to determine the linearity of the sensor. The sensor was connected to a 5V supply and the output voltage was measured using a multimeter. To zero the sensor, the shaft was turned until the output voltage dropped to 0V. The shaft

was then rotated 10 degrees at a time (measured using a protractor), over a range of 0 to 400 degrees. The results are shown in Figure 23.

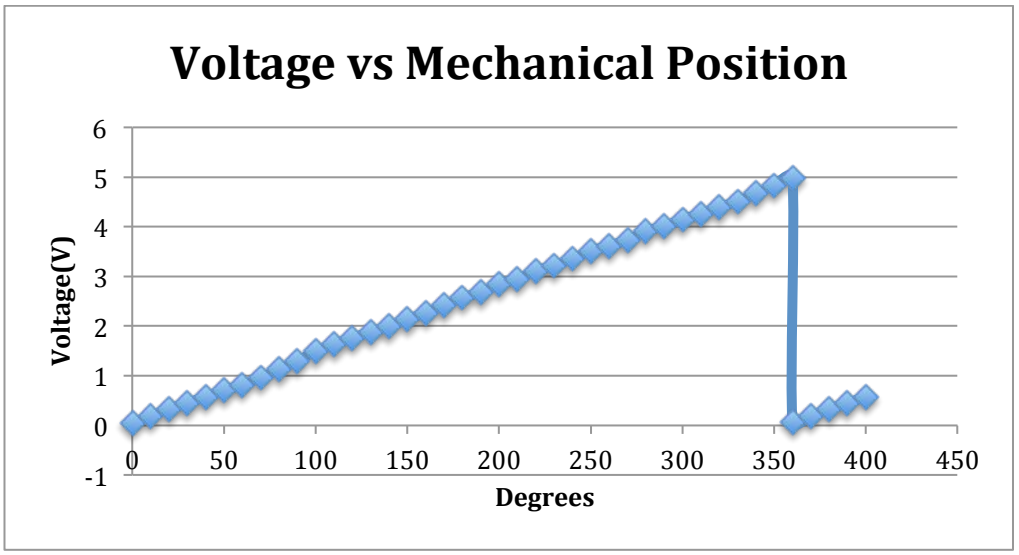


Figure 23 Shaft Encoder Angular Position vs Voltage

The results show that this sensor is extremely linear, therefore suitable for measuring the angular position of a steering shaft. There are however non-linearity's at the boundary points of 0 degrees and 360 degrees. As shown in the Figure 23 the voltage will jump between 0V and 5V when it's on either side of a boundary point because the shaft encoder only has a 360-degree range. If the shaft is turned beyond 360 degrees, it will reset to 0 degrees (0V). Therefore the system will become unstable, if the steering wheel is turned more than 180 degrees in either direction.

The sensor should be connected to the steering shaft in a way that when the steering wheel is straight the shaft encoder should be at 180 degrees, outputting a voltage of 2.5V (1.67V after the voltage divider). One method that could be used to prevent the sensor from crossing the boundary points is to limit its range of 0-360 degrees rotation to 10-350 degree, by physically restricting the steering shaft from turning more than 170 degrees in either direction. This is assuming that the steering wheel will not need to turn more than 170 degrees in a particular direction. This is a reasonable assumption for a race car which will be travelling at high speeds and not taking sharp turns. If 170 degrees each direction is not sufficient then a different gear ratio may be required with a larger gear connected to the shaft encoder so that larger turns of the steering wheel will result in a smaller change in shaft encoder position. The effectiveness of this method may be limited by the sensitivity of the shaft encoder.

The author initially planned on installing the sensor inside the steering column, as the 2010 SAE car had a hollow steering column; unfortunately the 2011 model has a solid steering column, so a different approach was required. The diameter of the steering column is much larger than that of the shaft encoder; therefore two gears of equal size would be needed to connect the steering column and shaft encoder and achieve a 1:1 gear ratio. As described above a different gear ratio could also be used to increase the range of the shaft encoder.

4.5 Opto-Isolated USART link

The design of this system is bound by the rules of the Formula SAE-A competition, which states, "The entire high voltage and low voltage systems must be galvanically isolated"[2]. This means the cabin electronics such as the microcontroller and sensors need to be isolated from high voltage components such as the batteries and motors. This is a safety precaution to prevent the driver from coming in direct contact with high voltages.

In order to satisfy this requirement, instrumentation and motor control functionality was separated into two separate sub-systems. This project focused on the instrumentation component of the vehicle while another concurrent student project focused on the traction control components. Both subsystems run on separate microcontrollers and communicate critical information over an optically isolated USART connection.

Opto-couplers are used on both Transmit (*Tx0*) and Receive (*Rx0*) communication lines as a method of galvanic isolation. The configuration used by the author to design the opto-coupler circuitry was based on the recommended configuration detailed in an opto-coupler Application Note [16] written by the manufactures. An opto-coupler consists of an LED and a phototransistor; these two components are not actually connected electrically. The transistor is configured with the collector connected to the voltage supply of the receiving device and the emitter is connected to the input Rx0 pin of the receiving device. The diode is connected between Tx0 and GND of the sending device. When Tx0 on the sending end is high, the diode will switch on, this will pull the phototransistor emitter voltage to high (in respect to the receiving end voltage). When the sending voltage is low, then the diode will be off, resulting in the transistor's emitter voltage being 0V.

Opto-couplers U3 on the receive line (*Rx0*) and U4 on the transmit line (*Tx0*) are installed in the instrumentation system as illustrated by Figure 24. This configuration is not entirely functional for several reasons; firstly Pin 2 of U3 should be connected to the GND of the traction control system rather than to the GND of the instrumentation system and Pin 5 of U4 should be also connected to the voltage supply of the traction control system rather than the instrumentation system's 3.3V supply. These changes were not made mainly due to space constraints on the PCB for additional headers. It is also a common practice for the opto-coupler to be installed on the receiving end of a communication line so technically U4 should be part of the traction control system, but was left in this circuit for completeness. W1 and W2 represent bypasses on the communication lines, in the case that an opto-coupler is not required

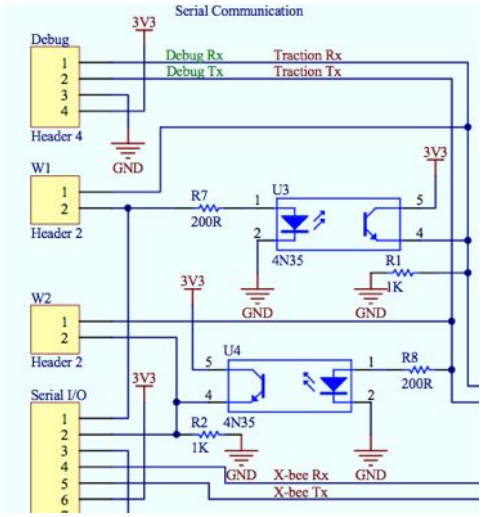


Figure 24 Traction Serial Link Opto Isolators

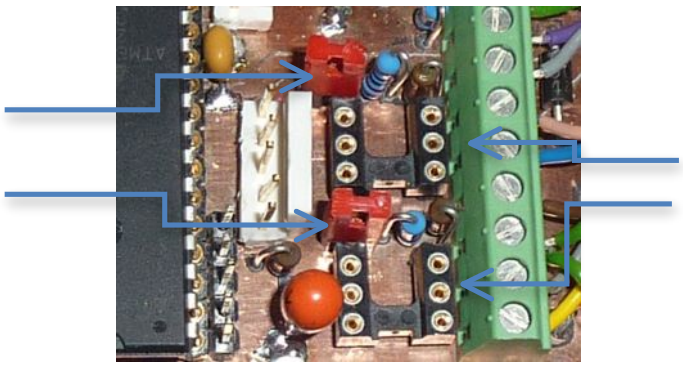


Figure 25 Opto-Couplers Bypass

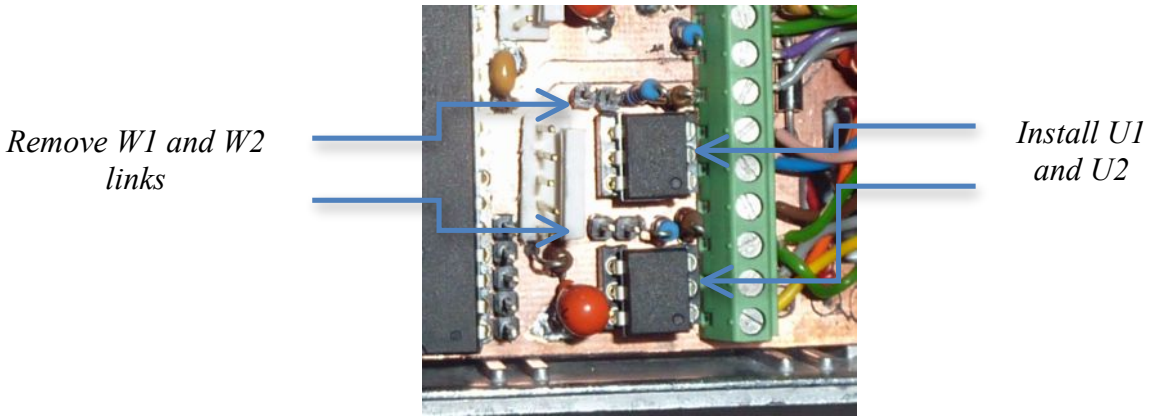


Figure 26 Opto Couplers Installed

The system also provides the facility to observe data being across the USART link using a terminal program (such as HyperTerminal) or send debug messages from the AVR to the terminal using the *sendDebug(char* msg)* command. This can be done by connecting a USB to TTL cable to the debug header pins 1-4 shown in Figure 27. If the USB to TTL cable has a 3.3V supply pin, then this can be used to power the circuit.

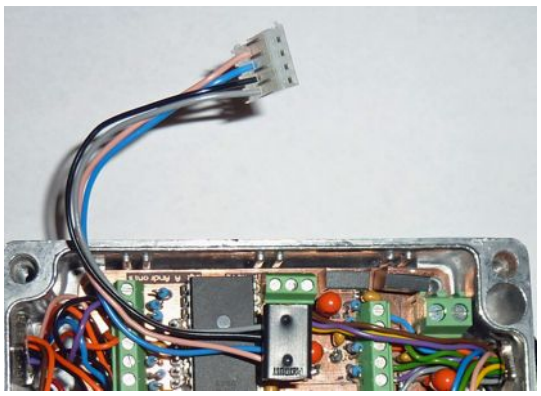


Figure 27 Debug Port

4.6 Battery Monitoring System (BMS) Interface

Although the BMS interface is largely un-implemented the author had extensive discussions with the student in charge of designing this system, regarding how the instrumentation system should communicate with the BMS. Due to time constraints this system has not been completed, however three potential communication protocols were devised and three electrical BMS terminal block has been provided. The BMS system itself consists two AVR microcontrollers, which measure the individual voltage of each battery cell using ADC inputs. REV team member Valentin Falkenhahn was in charge of developing this system for the duration of Semester 2 of

2011. The three options for communication between the instrumentation system and the BMS that were discussed are:

1) Use two digital pins as input and output pins between the two systems. One representing an "ignition" signal as an input to the BMS and the other representing a status signal (batteries flat/faulty) as an input to the instrumentation system.
2) Use of a USART serial communication protocol to transmit detailed battery data and status information to the instrumentation AVR and start and stop commands to the BMS
3) Use of a digital I/O protocol such as I^2C or SPI to transmit information between the two devices

Due to the limited amount of USART connections on the instrumentation AVR, option 2) was be ruled out. At this stage in the development process it was too late to select a different microcontroller for this system, which had an extra USART. This may still be a suitable option for future work, as the author has already implemented an extensive serial communication protocol that could be easily utilised. It is desirable for detailed battery voltage information to be sent from the BMS to the instrumentation system, this could potentially be done as suggested in option 3). That being said, the BMS system was not completely built until near the end of semester therefore not enough time was left to develop and test a full-scale digital communication protocol between the two systems by the end of the semester. For this reason option 1) was chosen as such a protocol is the bare minimum requirement for the SAE vehicle to be functional.

The ignition signal is connected as an output pin to PD6 and the status signal connected as an input pin to PD5 on the instrumentation microcontroller. The basic operation principles of this protocol are that when the vehicle is switched on, a 12V supply is connected to the instrumentation system. Once the instrumentation system is fully initialised, the AVR will set the ignition pin to '*high*', which will be read by the BMS instructing it to initialise. The BMS system will reply by setting the status pin to '*high*'. If at any stage the BMS fails or the batteries are critically low, the status pin will change to a 'low' signal and the instrumentation AVR will shut off the system.

The BMS input/output pins can be accessed through the BMS header pins on the PCB board. Although not actually implemented as part of the PCB design, opto-couplers will be required on both pins, in compliance to SAE rules because the BMS AVR is connected directly to the batteries which are a high voltage system which need to be gavanically isolated from the cabin electronics [2].

4.7 X-Bee Pro Wireless Transmitter

One of the main ideas of the instrumentation system was for it to be able to send information about the on-board sensors to an on-site computer. This is useful for performance analysis, especially during test drives of the vehicle. To add this functionality to the system, a wireless transmitter is required. There are several different wireless transmission standards that are commonly; these include WiFi, Bluetooth and Infrared. The main issues with these types of wireless transmitters are they have a short transmission range, which would not be suitable for use at an outdoor racetrack and in some cases very power demanding. Wi-Fi has the better range out of the three alternatives, but uses the largest amount of power and is quite expensive, therefore unsuitable for this project.

A X-Bee transmitter/receiver is a low cost, low power device capable of wirelessly transmitting data over long distances [17]. It is commonly used for telemetry in industries, including the resources and automotive (racing) industries. The instrumentation system uses two X-Bee Pro “series 2” transmitter/receiver devices, one is connected to the instrumentation system on the vehicle and the other connected to a PC via a USB adaptor. The device itself has a power output of 10mW, a maximum RF data rate of 250Kbps and a maximum wireless transmission range of 1.6km (line of sight) [17]. The potential range of RF transmissions decrease with obstacles such as walls between the two devices. On a racetrack this is not a major problem as there are very few obstructions and the vehicle will never be more than a kilometer away from the on-site computer. Its operating frequency is 2.4 GHz [17], which is a certified operating frequency in Australia, however there is the possibility of interference with other devices such as wireless routers as Wi-Fi uses the same frequency. The device supports both point-to-point and point-to-multipoint communication [17], however in this system it is only used for point-to-point.

Figure 28 X-Bee Pro [17]

The X-Bee Pro Series 2 runs on 3.3V, it is a 20-pin device, which is illustrated in Figure 28. The only pins used by the instrumentation processor are Vcc (Pin 1), GND (Pin 10), DOUT (Pin 2) and DIN (Pin 3) as specified in the product documentation [17], the remaining pins are analog inputs and left disconnected. DIN and DOUT are USART communication lines, which are connected to the AVR microcontroller. DIN is connected to Tx0 and DOUT is connected to Rx0 on the instrumentation microcontroller. Message packets of a specific form, described in the product documentation must be sent from the microcontroller to the X-Bee transmitter via a USART link, the X-Bee then transmits this message as an RF packet which is received by the X-Bee receiver. If the receiver is connected to a computer via the USB adaptor, the received message can be viewed through HyperTerminal.

4.8 Reverse Switch

The reverse switch is a control accessible to the driver that puts the motors into reverse operation. The switch has 4 pins, Vcc, GND, output 1 and output 2. When the switch is in the off position Vcc is connected to output 1, which is not connected to the circuit. Switching from output 1 to output 2 will connect Vcc to output 2, which is connected to PD4 on the AVR. Since PD4 is an input then it can either receive a '1' (5V signal) meaning that motors are operating in reverse or a '0' (no input) meaning that the motors are operating in forward mode (normal operation). The position of the reverse switch can be read by the microcontroller using the command:

```
reverse =  PIND >> PD4;
```

5 Software Design

5.1 Overall Program Flow

Figure 34 illustrates the flow of the instrumentation AVR program, operating under normal conditions. Once the AVR has been switched on, AVR parameters are initialized this includes setting pin directions, enabling the ADC converter, enabling the communication protocols and enabling global interrupts. Following initialized an output signal is sent to the battery management system as an ignition signal for the vehicle. After preset delay, the main system loop is entered and the status of the BMS is checked through the digital input line. If the BMS system is not working then something has gone wrong and the system generate a error LED code and shut itself off. Otherwise if the BMS is running correctly, the system should continue its executing process and sample the sensors (at their correct sample rate). Data from the sensors is then processed and packaged into the form of a message. The message is then transmitted to the traction control AVR through the first USART link and to the X-Bee Transmitter through the second USART link; if everything is successful the program returns to the start of the loop and repeats the above process.

The software framework developed for this system consists of the following source files (See Appendix B):

REV_SAE.c – The main program, bridging all the system's components together. This initializes the system, sets up the main polling loop and controls the overall flow of the program shown in Figure 29.

REV_SAE2.c – The Receiving end program used to receive and decode messages sent from the instrumentation AVR to the Traction Control AVR

USART.c – A library used to communicate over USART0 connection to the traction control AVR. Contains both sending and receiving functions as well as debug functions

TWI.c – A third party open source library used to communicate with devices over I^2C. Full credit goes to Ryan Owens for developing this library [18].

digSensor.c – A library used for reading/writing to the Accelerometer or Gyroscope. Encapsulates lower level functions from `TWI.c` into a more user friendly API for communicating with the IMU.

xBee.c – A library used to transmit RF packets using the X-Bee transmitter.

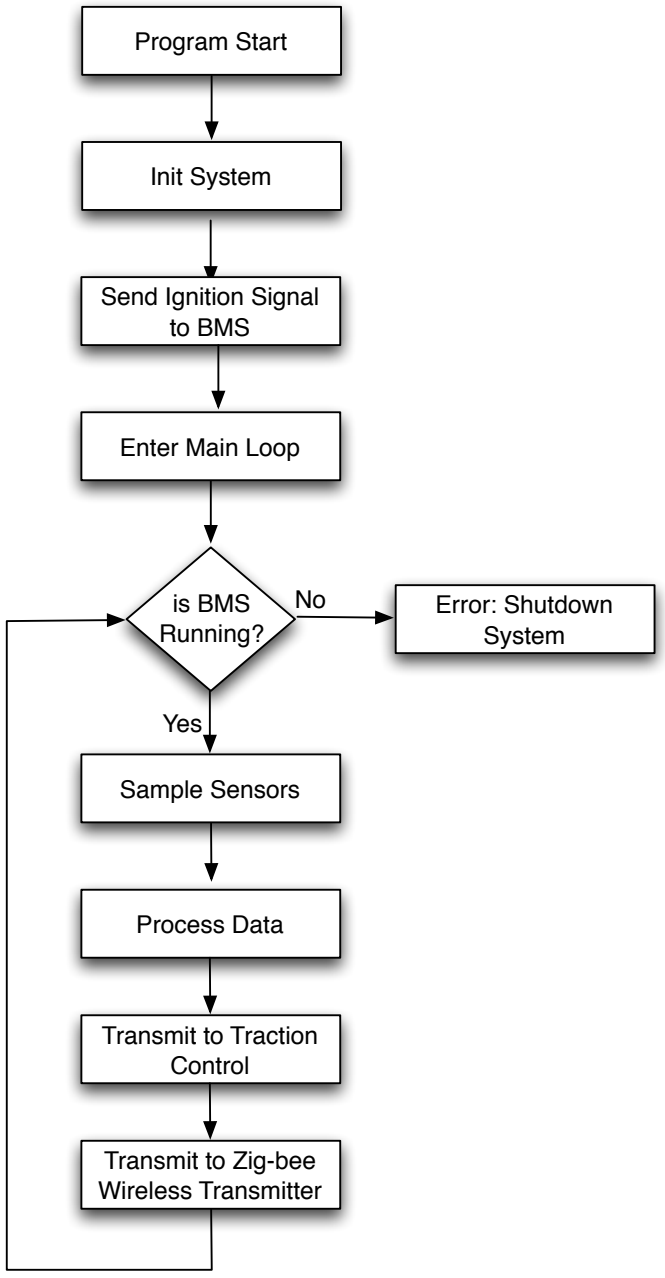


Figure 29 Software Program Flow Chart

5.2 Initialization

I/O Direction

The first step of the initialization process is a call to the function `SetupPorts()` which sets the input/output direction of each pin on the AVR. The AVR is split into four ports A-D each consisting of 8 pins. The direction of each of these pins can be set to a 1 for output or 0 for input. By default all pins are set as inputs. To set the direction of a pin to 1 the following command is used:

```
DDRn |=  (1<<PnX);
```

To clear the output direction of a pin, the command:

```
DDRn &= ~(1<<PnX);
```

is used, where *PnX* is the port number (A-D) and X is the pin number (0-7). In this system, the only pins that need to be set as outputs are the BMS output pin "*PD6*" and the LED status pin "*PD7*" Note that "*Tx0*" and "*Tx1*" are set as outputs by default.

ADC Initialization

The analog to digital converter is initialized by setting the enable bit *ADEN*, in the Analog to Digital Control and Status Register *A* (*ADCSRA*) to a value of 1 and also the clock prescaler to divide 32 by setting the pins ADPS 2:0 in *ADCSRA* to a value of 101. Since the clock is running at 8MHz, the ADC clock speed will be 250kHz. A higher prescaler value was chosen because Analog to Digital converters operate with a higher accuracy at lower clock speeds, furthermore 250kHz is much higher than the sample rate of the analog sensors used in this system, so a lower ADC clock speed will not be the bottleneck in reading from the sensors.

Interrupt Initialization

A call to `sei()` is made to enable all global interrupts, this is essential to all timer interrupts and *RX0* receive interrupts . Interrupts are important because they essentially allow the microcontroller to run certain tasks in the background. For example, instead of having to constantly poll the receive buffer to check if a byte has been received, an interrupt will automatically be triggered every time the receive buffer is full. This temporarily alters the normal flow of the program by jumping to a sub routine to handle the interrupt. Once this interrupt has been handled the program will resume normal execution. Specific interrupts still need to be enabled separately,

the global interrupt enable function `sei()` only turns on the AVR's interrupt handling functions.

UART Initialization

The USART transmitter and receiver modules of the AVR also need to be turned on for both interface 0: connected to the traction control AVR and interface 1: connected to the X-Bee transmitter. Serial interfaces USART0 and USART1 are enabled by calling the functions:

```
void USART_Init(unsigned int ubrr);
void xBee_Init(unsigned int ubrr);
```

The variable, UBRR is dependent on the clock speed of the source microcontroller and desired baud rate. On this microcontroller the clock speed is 8MHz and the baud rate is set to 11920bps. It can be calculated using the following equation:

```
UBBR = (Freq Clock/(Baud rate * 16)) - 1;
```

This value then stored in the USART registers `UBRRnH` and `UBRRnL` splitting the 16-bit `UBBR` value into 2 8-bit values. Setting `RXENn` and `TXENn` to 1 in the `UCSRnB` register turns on the transmitter and receiver. The frame format is set to 8 data bits and 1 stop bit by setting the `UCSZn2, UCSZn1` and `UCSZn0` to 011 (using command:

```
UCSRnC |= (3<<UCSZn0));
```

Finally the receive interrupt needs to be enabled by setting `RXCIEn` to 1 in the `UCSR0B` register. '*n*' represents the two different interfaces where '*0*' is the traction control USART connection and '*1*' is the X-Bee USART connection.

I^2C Initialization

The I^2C bus, used for communication with digital sensors is initialized using the function `twiInit(unsigned long scl_freq)`,where `scl_freq` is the frequency of the `scl` clock line used to synchronize the digital IO device with the AVR. For this system `scl_freq` is set to 100kHz and this is used to set the Two Wire Bit Rate (TWBR) register.

LED Flash Error Codes

A panel mounted LED provides visual feedback that the software is operating correctly. During normal operation the main polling loop drives the blue LED at 1Hz.

If an initialization error occurs the LED is latched on. If a run time error occurs a flash code provides diagnostic feed back as described by Table 2.

Figure 30 Blue LED Error Flash Codes

Error Code	Flashing Rate
No Power	Off
Normal (Heart Beat)	1 Second
X-Bee error	2 Second
Accelerometer/Gyro error	4 Second
Fatal Error	On

Table 2 Blue LED Error Flash Codes

5.2 Battery Management System I/O

The BMS can be turned on by setting the ignition pin to 1 (high) this outputs a 5V signal to the BMS connecting the batteries to the motors using a relay:

```
PORTD |= 1 << PD5;
```

The system should then wait an appropriate amount of time to allow for the BMS to initialize. The status line should be checked at the start of the main loop to ensure the BMS hasn't failed. The status pin can be checked using the following command:

```
status =  PIND >> PD6;
```

If status is equal to '1' then the status of the batteries and BMS system is ok, if status is equal to '0' then either the BMS system has failed or the batteries contain

insufficient charge and the BMS has requested to turn off the system. A message should then be sent to the traction control system using the command:

```
sendMessage(SHUTDOWN, SYSTEM);
```

and after allowing sufficient time for the traction control AVR to shutdown, the ignition pin should be set to 0 (low) using the command:

```
PORTD &= ~(1<<PD5);
```

The final stage is to put the instrumentation AVR into an endless loop. This endless loop will not exit until the system has been restarted.

5.3 Sampling Analog Sensors

Analog to Digital inputs can be read by calling the function:

```
unsigned int read_adc(unsigned char adc_input);
```

The function requires an input parameter of `adc_input` to select the correct analog device connected to Ports PA0-PA7. Shown in Table 3 is a list of analog devices and their corresponding values of `adc_input`:

ADC Input	Analog Device
0x00	Accelerator Pedal Hall Sensor 1
0x01	Accelerator Pedal Hall Sensor 2
0x02	Brake Pedal Hall Sensor 1
0x03	Brake Pedal Hall Sensor 2
0x04	Steering Wheel Rotary Encoder
0x05	(Unused)
0x06	(Unused)
0x07	(Unused)

Table 3 ADC Input Devices

The function outputs an unsigned integer between 0 – 1024. This is a digital equivalent of the analog input signal generated by the sensors. The AVR's built-in analog to digital converter has a resolution of 10-bits and a ±2 LSB accuracy.

The process to read an analog input using the ADC is as follows: The selected `adc_input` is placed into the ADMUX register using the command:

```
ADMUX=adc_input | (ADC_VREF_TYPE & 0xff);
```

Calling the function `_delay_us(10)` puts the processor to sleep for 10us, leaving sufficient time for the ADC input voltage to stabilize. Setting the ADSC bit to "*1*" in

the *ADCSRA* status register starts the analog to digital conversion. The function then uses a while loop to wait for the conversion to complete terminating only when the *ADIF* (AD interrupt flag) is set to 1, indicating that the conversion is complete. The resulting digital output is stored in the *ADCW* register and returned by the function.

It was observed that the output of the ADC was slightly affected by noise, as the output signal's LSB oscillated between ±2. Several methods were used to reduce the effect of noise on the signal. The first attempt was to oversample the signal by taking the average of 4 or 8 readings. This method had very little effect on reducing noise because the noise was random and averaging random noise still results in random noise in the output signal. Another noise reduction method was to use the AVR's built-in ADC noise isolation function [8]. Theoretically, the noise isolation function should put the AVR CPU to sleep before taking an ADC reading, reducing the amount of internal noise generated by the AVR. This unfortunately did not have any effect on decreasing the output signal noise. Therefore this noise was attributed to the limitations in resolution of the analog to digital converter and it was decided that it would have very little effect on the final system and its effects should be fully tested when the vehicle is running.

5.4 Sampling Digital Sensors

Digital sensors used by this system communicate over the I^2C protocol. This is a two-wire system with a data line (*SDA*) and a clock synchronization line (*SCL*). The primary device that communicates over I^2C is the Accelerometer/Gyroscope IMU. Each device on this board has a different identifier, the accelerometer is addressed by sending a message to *device_id* 0xA6 (0xA7 for read-only) and the gyroscope is addressed by sending a message to *device_id* 0xD0 (0xD1 for read-only). The process for reading/writing to a device is shown in the Figure 31.

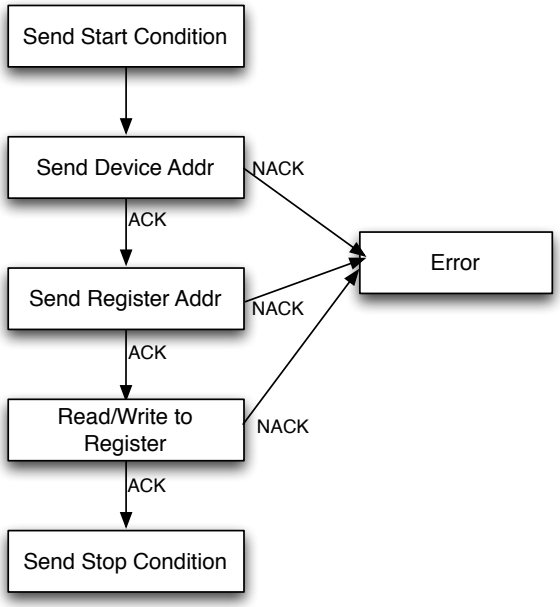


Figure 31 I²C Communication

The functions used for reading and writing to a particular register on the selected device can be found in the `Twi.c` external library [18]:

```
char Read(char device_addr char, register_addr char *value);
char Write(char device_addr char, register_addr char value);
```

Following the flow chart above, when a read or write command is initiated, a start condition is sent opening the I²C lines for communication. The device address is sent and if the device receives the message it sends back an acknowledgement (ACK) if there is no response a negative acknowledgment (NACK) is received. Following this the register address is sent, if the register on the specified device exists then another ACK is sent otherwise a negative NACK is returned. A request is made to either read or write to this register, if this operation is successful an ACK is sent otherwise a NACK is returned. Upon completion a stop condition is sent down the I²C bus to terminate communication. If at any stage a NACK has been received, the function terminates returning an error message, is outputted by the program, otherwise a *"0"* is outputted if the operation was successful.

Table 4 describes the different registers, which are used to store data by the Accelerometer/Gyroscope board. Each quantity being measured is stored in a High and Low bytes because each register can only store 8-bits while data from the accelerometer is 10-bits and gyroscope is 16-bits. Precompiler definitions for these registers can be found in the header file `digSensor.h`.

Register	Value
Accelerometer	
ACCEL_XOUT_L	0x32
ACCEL_XOUT_H	0x33
ACCEL_YOUT_L	0x34
ACCEL_YOUT_H	0x35
ACCEL_ZOUT_L	0x36
ACCEL_ZOUT_H	0x37
Gyroscope	
GYRO_XOUT_L	0x1D
GYRO_XOUT_H	0x1E
GYRO_YOUT_L	0x1F
GYRO_YOUT_H	0x20
GYRO_ZOUT_L	0x21
GYRO_ZOUT_H	0x22

Table 4 Gyroscope/Accelerometer Device ID

High and low bytes can be joined in the following way:

```
unsigned int value =  (int) high << 8 + (int) low;
```

Functions to read and write to a specific device are encapsulated by higher-level functions in the `digSensor.c` library (see Appendix B). These functions make calls to lower level functions from the external third party library `twi.c`. These four primary I/O functions are shown below:

```
char read_Accel(char register_addr, char * value);
char write_Accel(char register_addr, char value);
char read_Gyro(char register_addr, char * value);
char write_Gyro(char register_addr, char value);
```

5.4 Processing of Data

Any processing of raw sensor data is done on the sending side (instrumentation AVR) before it is transmitted over the USART link to the traction control system. The only real processing that is required is by the system is for the pedal box hall sensors. This is because each pedal has two sensors, one being a redundant sensor; only one value for each pedal position needs to be sent across the USART link and checks need to be performed to ensure that the pedal position sensors are operating properly to avoid sending corrupted sensor data. If one of the hall sensors has failed, then the instrumentation microcontroller is responsible for shutting down the vehicle.

Given that there are two sensors on each pedal, let A1 be sensor 1 and A2 be sensor 2. For correct operation of the pedal position sensors, it is required that the difference between these two sensor readings is as small as possible. Considering that there is a ±2 LSB inaccuracy when reading from the ADC due to noise, the two sensor values on each pedal will not always match exactly. Therefore a tolerance needs to be defined for the minimum and maximum difference between sensor values. In theory this only needs to be a tolerance of about ±3, however in practice a much larger tolerance needs to be set to compensate for DC offsets between sensor readings. The following function can be used to compare the two different sensor values *A1* and *A2*, such a function is used in place of the `math.h fabs` (absolute value) function as this is a more efficient method of comparison on an AVR.

Example of an algorithm used to compare sensor values

```
int checkHallSensors(unsigned int A1, unsigned int A2){

      if( A1 > A2){
            if( A1-A2 <= THRESHOLD)
                  return 1;
            }
      else{
            if( A2-A1 <= THRESHOLD)
                  return 1;
      }

      return 0;

}
```

If the function returns a '1' then the pedal position sensor is working correctly and an average of the two hall sensor values should be stored, as the final result. A logical shift right by two bits should be used instead of a divide because dividing is not a natural operation for an AVR and consumes larger amounts of CPU time:

```
hallValue = (A1 + A2) >> 2;
```

If the function returns a '0' then one or both of the hall sensors on the pedal have failed and could be outputting random values, therefore to prevent unexpected behavior by the vehicle, the instrumentation microcontroller must shutdown the entire system including turning off the ignition signal to the BMS and sending a system shutdown message over USART to the traction control microcontroller using the command:

```
sendMessage(SHUTDOWN, SYSTEM);
```

5.6 Transmitting data to the traction control AVR

For transmitting and receiving data through USART interface 0 to the traction control AVR, a serial protocol was developed. Instead of using external libraries to implement the serial protocol, a simple protocol was written by the author. The reasoning behind this is that the complexity of pre-existing protocols was not required for the purpose of this system and it would be better to implement a simple protocol tailored to the requirements of this particular project. Commonly used protocols such as a "*sliding window protocol*" or "*selective repeat protocol*" were not necessary due to the fact that messages are sent so frequently that it doesn't matter if a particular message is corrupted, it will be discarded and the receiver will wait for the next message to arrive.

The USART interface can only send a single 8-bit frame at a time; therefore the protocol needs to package frames into a message. The message structure went through several iterations until the most efficient format was determined. To be understood by the serial protocol messages must be encoded in the following format:

```
<Start><Msg Type><PAYLOAD><CRC><End1><End2>
```

Start – The start delimiter, used to indicate the start of a new message. The value assigned to start during testing is *'$'*.

Msg Type – The type of message being sent, this is used to describe the type sensor that the message relates to. Each type of message has a certain expected payload length, which can be found in the `USART.h` header file. This relation is shown in Table 5

Msg Type	Value(char)	Expected Payload Length	Value(int)
HALL	'H'	HALL_LEN	5
ROTARY	'R'	ROTARY_LEN	2
ACCEL_GYRO	'M'	ACCEL_GYRO_LEN	4
SYSTEM	'S'	SYSTEM_LEN	1

Table 5 Message Type Protocol

The lengths above will be constant for any message of a particular type, and can be used for error checking at the receiving end. Note that these lengths are measured as the number of bytes of the payload.

Payload – The payload is the actual data being transmitted for a given sensor. The payload has a different structure for each message type shown in the table above; hence different types of messages have different lengths. Shown below is the required payload structure for each message type.

- **HALL**:
  - `<Direction (1 byte)> <Accelerator Position (2 bytes)> <Brake Position (2 bytes)>`
    - Where Direction is either FORWARD (`'F'`) or REVERSE (`'R'`) signifying the direction that the motors should rotate.
- **ROTARY:**
  - `<Steering Wheel Position (2 bytes)>`
- **ACCEL_GYRO:**
  - `<Accel x (2bytes)> <Accel y (2 bytes)> <Accel z (2 bytes)> <Gyro Pitch (2 bytes)> <Gyro Roll (2 bytes)> <Gyro Yaw (2 bytes)>`
- **SYSTEM:**
  - `<System Message Type (1 byte)>`
    - Where the System Message Type can either be ACK (0x01) or SHUTDOWN (0xFF). Even though system messages aren't used by the current protocol they were included for completeness and future development.

CRC – The Cyclic redundancy check, is used for error detection. A CRC is generated at the sending end, which summarises the message excluding delimiter symbols.

End1, End2 – The end delimiter, consisting of two unique bytes of data used to indicate the end of the message. During testing End1 was set to `'\r'` and End2 was set to `'\n'`. This is equivalent to a new line when the message is read as an ASCII string.

The message can be sent across the serial link using the function:

```
void sendMessage(char *message, char type);
```

Where `message` is the payload being sent and `type` is the message type. Note that each message type has a corresponding expected length, this relation is shown in Table 5.

Data read from the ADC is in unsigned integer form, in order to send this over the USART link, the following function must be called to convert the 16-bit number into to 8-bit symbols:

```
char* intToChars(unsigned int num);
```

The formation of the complete message is handled by the sendMessage function; this includes the generation of the CRC for the string `<msg type> <Payload>`.

Example of transmitting Hall Sensor Data to the traction control AVR across USART0: *NOTE: Assume accelData and brakeData are integer values that have been read from the analog to digital converter at an earlier stage in the program:*

```
char * accelMsg = intToChars(accelData); //2 bytes
char * brakeMsg = intToChars(brakeData); //2 bytes
char * msg = calloc(5,sizeof(char));
strcat(msg,accelMsg); // append accel to message
strcat(msg,brakeMsg); // append brake to message
free(accelMsg);
free(brakeMsg);
sendMessage(msg,HALL);// format and send message
free(msg);
```

Sending the message in this format does have its downsides, as the message uses a direct conversion of integers into two char values, this does not directly correspond to an ASCII string that represents that integer number e.g. the integer 12 is **not** converted to ASCII characters '1' '2', therefore it is not human readable in the case that the user wants to connect the USART to a computer terminal to read the output using HyperTerminal (or equivalent serial communication programs). This particular method has been chosen because it is a more efficient way of transmitting data than using the ASCII representation of integer numbers as a 16-bit number represented in ASCII corresponds to four 8-bit ASCII characters. If the actual values need to be outputted in a readable form, the helper function

```
sendDebug(char *msg);
```

which sends a normal unencoded string can be used along with the function :

```
char* intToHex(int num int len);
```

to convert an integer into a readable hex ASCII string. The value of len should be set to 3 for a 10-bit number (from the ADC) or 4 for a 16 bit number

5.7 Receiving sensor data at the traction control microcontroller

The receiving end is responsible for decoding the received message and performs error detection to ensure that the received message is valid and has not been corrupted during transmission. The message is read into a receive buffer using a state machine to detect the start and end of a message and detect any errors due to incorrect length. A simplified flow chart of the process used by the receiver to detect the current state is shown below:

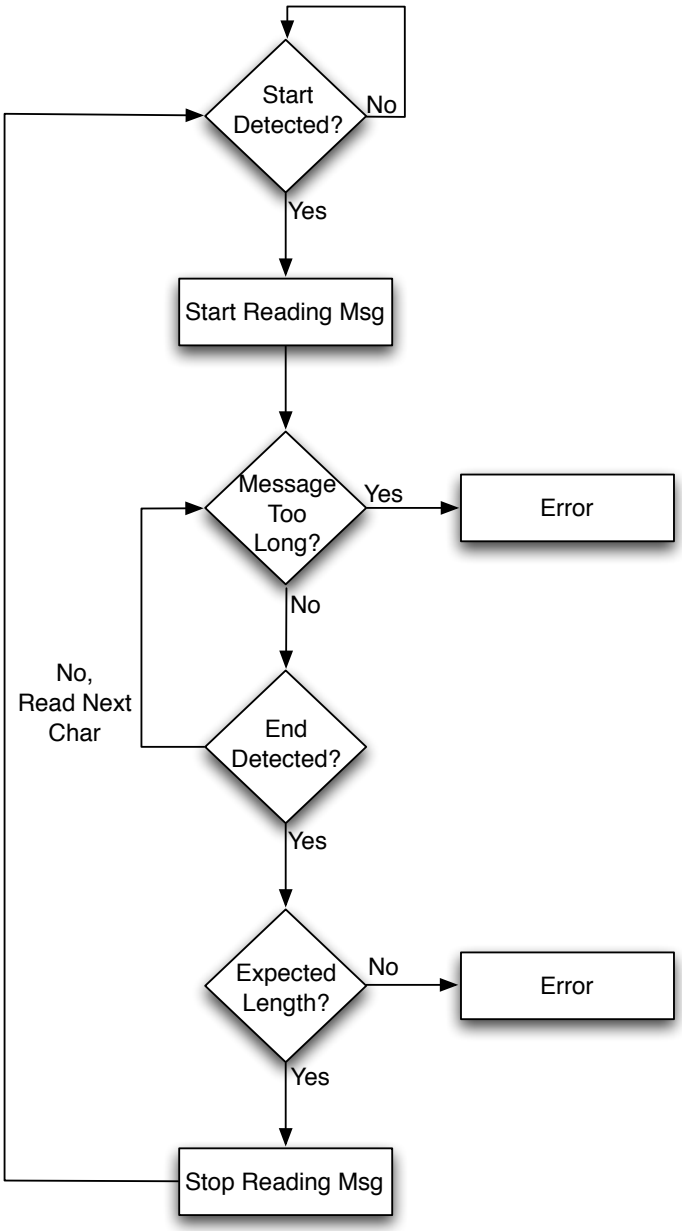


Figure 32 Receive Sensor Data Flow Chard

The state machine takes an input of two variables, current frame and previous frame and outputs the current state, which can take a value of `STARTED`, `ENDED`, and `ERROR`. The `STARTED` state is entered when the current frame is equal to Start (Start delimiter byte) and the previous frame is equal to `End2` (end delimiter byte 2). A global variable `idx` keeps track of the current length of the message. To prevent buffer overflow, `idx` is checked against `BUFLEN`, the maximum buffer length (a precompiler `#define` in `USART.h` that can be set by the user. If at any time `idx` exceeds `BUFLEN` then an error state is entered. The `ENDED` state is entered when the current frame is equal to `End2` and the previous frame is equal to `End1` (end delimiters) and the

message is the expected length. If an end sequence is detected and the message is of incorrect length (too short or too long) an ERROR state will be entered instead. Expected length can be calculated using the following expression:

```
Expected Length(bytes) = len(payload) + len(delimiters) +
                         len(type) + len(crc);
```

Where `len(payload)` can be determined from the table above, `len(delimiters)` = 3 bytes (1 start two end bytes), type = 1 byte and CRC = 2 bytes.

Receiving messages should be interrupt based, with an interrupt sub routine triggered whenever a byte is received. This can be handled by the function:

```
char recieveChar(char* buf);
```

Where `buf` is the receive buffer which should be created with a length of BUFLEN, as defined in `USART.h`. The `receiveChar` function will handle the buffer ensuring that when an ENDED state is detected, the string is terminated with a '\0' or when an ERROR state is detected the buffer will be zeroed. One byte is stored into the buffer every time the function is called and the current state is outputted.

Example of Receiving Hall Sensor Data across USART0:

```
char * receiveBuf = calloc(BUFLEN,sizeof(char));

ISR(USART0_RX_vect){  //interrupt service routine

      char state = recieveChar(receiveBuf);
      if (state == ENDED){

            if(receiveBuf[1] == HALL){

                  decodeHall();
            }

      }
}
```

Example of Decoding Hall Sensor Data: *assuming variables:* ***char direction****,* ***unsigned int accel*** *and* ***unsigned int brake*** *have been previously defined.*

```
void decodeHall(){
      char *msg = calloc(7,sizeof(char));
      char *crc = calloc(3,sizeof(char));
      for(int i = 1; i < 7; i++){

            msg[i-1] = receiveBuf[i];
      }

      for(int i = 7; i < 9; i++){

            crc[i - 7] = receiveBuf[i];
      }

      unsigned int check = charsToInt(crc);
      free(crc);
      unsigned int actual = calculateCRC(msg);
      if(check != actual)sendDebug("CRC ERROR");
      else{

            direction = msg[1]; // motor direction
            char *c = calloc(3,sizeof(char));
            c[0] = msg[2];
            c[1] = msg[3];
            accel = charsToInt(msg); //Accel Position
            c[0] = msg[4];
            c[1] = msg[5];
            brake = charsToInt(msg); //Brake Position
            free(c);

      }
      free(msg);

}
```

One of the ideas behind this instrumentation system is it should act as a central computer for the vehicle. Therefore it should be able to detect if the traction control AVR stops responding. If communication with this microcontroller is lost then the vehicle may start acting unpredictably. As this microcontroller is connected directly to the motors this could become a serious hazards to the driver. Ideally if contact is lost with the traction control system, the instrumentation system should shut off the entire system to prevent any hazards. In its current state, the serial communication protocol does not take account of this, it is currently only used as a one way transmission line to send data to the traction control system and does not use its receive communication line to receive confirmation messages. It would be useful if the traction control end replied to each message with an acknowledgement, that way the instrumentation AVR would always know if the traction control is running. A timer could be used to measure the time since the last ACK was received and if that timer passes a certain threshold, the system would be shut off. A similar feature should be implemented on the receiving end; by using a timer to measure the amount of time elapsed since sensor data was received. Due to time constraints this feature was never implemented but most of the infrastructure has been put in place to make this a simple task.

5.8 X-Bee Transmitter

Before using the two X-Bee devices, they must be configured using "X-CTU" configuration and test utility [19], which is the software supplied with the X-Bee devices. Each X-Bee must be configured individually by plugging them into a computer using the supplied USB adaptor. Once connected to a computer the X-Bee will be recognized as a COM device and running X-CTU will allow the user to change certain parameters. The most important parameters, which need to be taken into consideration, are baud rate and source address. By default, the baud rate of each device is set to 9600bps; this must be set to 11920bps by altering the "BD-Baud Rate" parameter[19]. For each device the "*MY- Source Address*" parameter [19] must be set to a unique 16-bit identifier. As this is only a point-to-point system it should be sufficient to set the transmitter's address to 0x0001 and the receiver's address to 0x0002.

As specified in the X-Bee Pro "Series 2" product documentation [17], the following packet structure is required when sending a message to the X-Bee via the USART interface:

```
<Start Delimiter><Length><API Specific Structure><Checksum>
```

Start Delimiter – Start delimiter is always 0x7E, which signifies the start of a new message (1 byte)

Length – a 16-bit number (consisting of two 8-bit chars: MSB|LSB) representing the number of bytes of the "API Specific Structure" (2 bytes)

API Specific Structure consists of:

```
<API Identifier><Frame Id><Destination Address><Options><RF Data>
```

API Identifier – Command ID, set to 0x01 for a Tx request using 16-bit addresses (1 byte)

Frame Id – A randomly generated sequence number for the packet, used for acknowledgements sent back by the receiver, setting this byte to 0x00 disables acknowledgments. (1 byte)

Destination Address – The 16-bit (MSB|LSB) address of the receiving X-Bee device. In this case set to 0x00 0x02. Alternatively set to 0xFF 0xFF to broadcast to any X-Bee device within range (2 bytes).

Options – Can be set to 0x01 to disable Acknowledgements from the receiver, otherwise set to 0x00 (1 byte)

RF Data – The actual message being sent (Maximum 100 bytes per packet)

Checksum – An alternative to the CRC-16 algorithm. All message bits are added together and subtracted and the last 8-bits of the resulting number are subtracted from 0xFF (1 byte)

The function `sendXbee(char * message)` can be used to transmit an RF packet to receiver device 0x0002. The full implementation of this function can be found in Appendix B under `xbee.c`. Below is an example of sending hall sensor data as an RF packet:

Example of transmitting Hall Sensor Data as an RF packet using interface USART1: *NOTE: Assume accel1, accel2, brake1 and brake2 are integer values that have been read from the analog to digital converter at an earlier stage in the program:*

```
/* Desired msg in the form "Hall Sensor Data: A1=xxx,
A2=xxx, B1=xxx, B2=xxx\r\n" */

char * msg = calloc(51,sizeof(char));
strcat(msg,"Hall Sensor Data: A1=");
char* A1 = intToHex(accel1,3);
strcat(msg,A1);
free(A1);
strcat(msg,", A2=");
char* A2 = intToHex(accel2,3);
strcat(msg,A2);
free(A2);
strcat(msg,", B1=");
char* B1 = intToHex(brake2,3);
strcat(msg,B1);
free(B1);
strcat(msg,", B2=");
char* B2 = intToHex(brake2,3);
strcat(msg,B2);
free(B2);
strcat(msg,"\r\n");

sendXbee(msg); // TRANSMIT THE MESSAGE
free(msg);
```

5. Conclusion

6.1 Outcomes

The primary requirements for this project were to provide serial telemetry for the following sensors:

- Dual redundant pedal position.
- Steering wheel angular position.
- 6 degree of freedom Accelerometer Gyroscope.

If time permitted the following optional functions were to be implemented:

- X-Bee wireless telemetry.

Late requirement were added to provide an interface for the following future system:

- Battery Management System interface.
- Reverse switch.

All the above requirements have been met.

The final outcome of this project is a fully programmable instrumentation system, which can be configured to interface with pedal position sensors, a steering wheel angle sensor and a digital accelerometer/gyroscope board. The system is also able to interface with other AVR's and an X-Bee wireless transmitter over USART links as well as having two I/O pins dedicated to digital communication with a BMS, which is to be implemented in the 2012 vehicle.

A PCB prototype board has been built for future testing of the system and is designed to be EMC compliant for use within Australia (although not yet certified by an approved measurement laboratory).

The serial protocol for communicating between two microcontrollers over a USART connection was fully tested by connecting two ATMega644P's together using their Tx0 and Rx0 pins. The author was successfully able to use this protocol to transmit test data over the USART0 interface. The CRC check very rarely failed, this is because the cable used to connect the AVR's was relatively short. In practice a longer

cable will be used, which may result in a higher error rate due to increased cable capacitance.

The X-Bee Pro device was tested by connecting one device to the AVR and the other to a PC outputting serial data to HyperTerminal. Test messages were transmitted from the AVR using the function `XbeeSend(msg)` implemented in `xbee.c` (see Appendix B). Messages were received by the PC and outputted to HyperTerminal with a very high success rate. This was only tested over a 1 meter distance. Maximum communication range tests should be run when the vehicle is fully built to determine the efficiency of the X-Bee Pro transmitter.

Below is a list of the Author's biggest achievements:

- Software
  - AVR8 polling loop and interrupt service loop framework.
  - AVR8 I/O initialization.
  - USART Serial Protocol.
  - Hall and shaft encoder sensor ADC.
  - I^2C IMU communication protocol.
  - X-Bee communication protocol.
- Hardware
  - Electronic component selection.
  - Sensor selection/design.
  - Circuit design.
  - Bread board Prototype
  - Double Sided PCB Prototype
  - Double Sided Through hold PCB design.

6.2 Limitations

Due to time constraints, the 2011 REV Formula SAE vehicle was not completed in time for the 2011 SAE-A competition. The REV SAE team created a Gantt chart to determine if the vehicle could be finished by the end of the year and based on this chart it was concluded that the amount of work required was too much given the amount of time available. All production was halted at the beginning of October. Therefore in its final state only the chassis of the vehicle was built, making it very

difficult to test the instrumentation system in the vehicle. Essential components such as the pedal box and steering shaft were not built and relevant sensors could not be installed. The vehicle will be entered in the 2012 SAE-A competition instead, giving an additional year for all sub-systems to be completely implemented.

The entire system was never tested as a whole however individual sensors were tested extensively. For this reason the program `REV_SAE.c` does not reflect a fully functional system, as it was very difficult to implement certain features while the vehicle had not yet been entirely built. Critical electrical systems such as the traction control system and BMS (developed by other students) were also not completely implemented in time to be integrated and connected to the instrumentation system.

The author had particular problems with the Accelerometer/Gyroscope IMU. The AVR failed to communicate with this external device. When trying to send a message to the IMU device over I^2C, the AVR would receive a negative acknowledgement, which indicated that the destination device was not responding/not found on the I^2C bus. The author spent several weeks trying to debug this problem with no progress. It was concluded that either the IMU device was faulty (most likely due to accidental exposure to ESD) or the external I^2C library being used was not suitable for ATMega644P and may need to be modified (unlikely).

The PCB built is only a prototype; it is not of professional quality as it is prone to short circuits or open circuits caused by over etching or under etching as illustrate in Figure 33. For that reason it is not suitable for use in a vehicle while on the track as vibrations may induce a fault. It is recommended when the final system is implemented in the vehicle, the PCB should be manufactured professionally as illustrated by the Figure 39 and Figure 40 3D models. Note: that the track width, track clearance be decreased when manufacturing the PCB professionally, as the equipment used is much more precise than manufacturing the board by hand.

Figure 33 Home Made PCB Prone to Short/Open Circuit, Under/Over Etching

6. Future Work

Although this system satisfied the main requirements of vehicle instrumentation, certain features were left unimplemented due to time constraints. There are many ways that this instrumentation system can be improved or extended to in future. This would make an excellent final year project for a student keen to work on vehicle instrumentation. Some of these changes are essential for the instrumentation to be fully functional and integrated into the 2012 Formula SAE Vehicle:

- Properly install and integrate existing sensors into the vehicle including: Pedal position sensors, the steering wheel angle sensor and accelerometer/gyroscope IMU
- Debug and determine the cause of digital communication with the IMU device failing
- Improve the Serial protocol to account for System Messages and Acknowledgements from the receiver.
- Develop a new sensor to measure the wheel rotation speed, it is suggested that the PWM output of a motor can be used in conjunction with the ATMega644P's 16-bit timer to determine a wheels revolutions per second.
- Modify the PCB design to properly utilize opto-couplers for proper galvanic isolation (see Section 4.6).
- Professionally manufacture the PCB. (This will also require alterations to the PCB design).
- Integrate the system's X-Bee telemetry with "Crystal Ball" Telemetry software developed by UWA student Frank Tan.

References

[1] SAE International. *Formula SAE Rules*. Available: http://students.sae.org/competitions/formulaseries/rules/2011fsaerules.pdf, (2011).

[2] FSE Germany. *Formula Student Electric Rules*. Available: http://www.formulastudentelectric.de/uploads/media/FSE_Rules_2011_v1.1.0.pdf, (2011).

[3] SAE-Australasia. *Formula SAE-A Rules Addendum*. Available: http://www.saea.com.au/wp-content/uploads/2010/11/FORMULA-SAE-A-Addendum-2011-Final.pdf, (2011).

[4] C. Trois, "EMC Design Guide for ST Microcontrollers," STMicroelectronics, 2003.

[5] STMicroelectronics ,"Designing with Micrcontrollers in Noisy Environments," 1998.

[6] C. Banyai and D. Gerk, "EMI Design Techniques for Microcontrollers in Automotive Applications," I. Corporation, 1995.

[7] STMicroelectronics, “EMC General Information Guide," 2000.

[8] Atmel Corporation. *ATMega644P Data-sheet*. Available: http://www.atmel.com/dyn/resources/prod_documents/doc8011.pdf, (2010).

[9] Professor T. Braunl.. *EyeBot M6 Documentation*. Available: http://robotics.ee.uwa.edu.au/eyeM6/, (2009).

[10] Allegro Microsystems. *A1301 Continuous-Time Ratiometric Linear Hall Effect Sensor ICs Data-Sheet*. Available: http://www.allegromicro.com/en/Products/Part_Numbers/1301/1301.pdf, (2010).

[11] SparkFun Electronics. *Analog Combo Board Razor - 6DOF Ultra-Thin IMU Specification*. Available: http://www.sparkfun.com/products/10010, (2010).

[12] SparkFun Electronics. *IMU Digital Combo Board - 6 Degrees of Freedom ITG3200/ADXL345 Specification*. Available: http://www.sparkfun.com/products/10121, (2010).

[13] Analog Devices. *ADXL345 Data-Sheet*. Available: http://www.sparkfun.com/datasheets/Sensors/Accelerometer/ADXL345.pdf, (2009).

[14] InvenSense. *ITG-3200 Data-Sheet*. Available: http://www.sparkfun.com/datasheets/Sensors/Gyro/PS-ITG-3200-00-01.4.pdf, (2010).

[15] US Digital. *MA3 Miniature Absolute Magnetic Shaft Encoder Data-Sheet*. Available: http://www.usdigital.com/assets/general/101_ma3_datasheet_1.pdf, (2010).

[16] Vishay Semiconductors, ""Faster Switching" from "Standard Couplers",” 2008.

[17] Digi International. *X-bee/X-bee Pro Product manual*. Available: http://ftp1.digi.com/support/documentation/90000982_E.pdf, (2011).

[18] R. Owens. *Two wire Interface Software Framework (twi.h/twi.c)*. Available: https://github.com/a1ronzo/6DOF-Digital, (2010).

[19] Digi International. *X-CTU: Configuration & Test Utility Software User Guide*. Available: http://ftp1.digi.com/support/documentation/90001003_A.pdf, (2008).

Appendix A Electrical Design Data

A.1 Instrumentation System Data Sheet

REV SAE Instrumentation System					
Data Sheet					
Parameter	**Condition**	**Min**	**Typ**	**Max**	**Unit**
External Interface					
Hall Sensors	Model Number: ALLEGRO A1301				
Input Impedance			30		KΩ
Voltage		0		5	V
Low Pass Filter			16		Hz
ADC					
Sample rate			32		Hz
Resolution				10	Bits
Accuracy			+/-2		LSB
Inputs					
A1	Accelerator # 1				
A2	Accelerator # 2				
B1	Brake # 1				
B2	Brake # 2				
Rotary Sensors	Model Number: US Digital MA3				
Input Impedance			30		KΩ
Voltage		0		5	V
Low Pass Filter			16		Hz
Sample rate			32		Hz
ADC					
Sample rate			32		Hz
Resolution				10	Bits
Accuracy			+/-2		LSB
BMS	Battery Managment System				
Inglition	TTL Logic Input	0		3.3	V
On			3.3		V
Off			0		V
Status	TTL Logic Output	0		3.3	V
Ok			3.3		V
Error			0		V
Reverse	TTL Logic Input				
Reversing			3.3		V
Forward			0		V
I²C	Accelerometer and Gyroscope Interface				
SDA	Serial Data (I/O)	0		3.3	V
SCL	Serial Clock	0		3.3	V
Baud Rate	Baud rate function of R3 and R22		100,000	400,000	Hz
Sampling rate			100		Hz
Accelerometer	Model Number: SEN-10121				
Resolution				10	Bits
Sensitivity			256		LSB/g
Gyroscope	Model Number: SEN-10121				
Resolution				16	Bits
Sensitivity			14.375		LSB/°/S
Traction	TTL RS232				
Rx0	Receiver	0		3.3	V
Opto-Isolator	Optional (Link selectable)	0	3.3	5	V
Tx0	Transmit	0		3.3	V
Opto-Isolator	Optional (Link selectable)	0	3.3	5	V
Baud Rate	Max baud rate limited by external cable capacitance	240	19200	250000	Baud
Debug	TTL RS232				
Rx0	Receiver	0		3.3	V
Tx0	Transmit	0		3.3	V
Baud Rate	Max baud rate limited by external cable capacitance	240	19200	250000	Baud
X-Bee	Model Number: X-Bee Pro Series 2				
Rx1	Receiver	0		3.3	V
Tx1	Transmit	0		3.3	V
Baud Rate	Max baud rate limited by external cable capacitance	240	19200	250000	Baud
Status Indicator	External Blue LED				
Flashing Codes					
OK			1		Hz
Error	On 100 %		0		Hz
No Power	No Light				
Processor					
Model	AVR ATMega644P	2.7	3.3	6	V
ALU	Integer			8	Bits
I/O				32	Pins
RAM				4.0	K Bytes
Flash				64.0	K Bytes
Internal Oscilator		1.0	8.0	8.0	MHz
Power Supply		2.7	3.3	5.5	V
Electrical					
Power Supply					
Voltage		8	12	24	V
Current		25	100	500	mA
Power		0.2	1.2	12.0	W
Protection					
Reverse voltage	Diode protected				
Over current	External M205 fuse		1.00		A
EMI/EMC	Metal box sheilding				
Environmental and Mechanical					
Wedth			65		mm
Length		115		154	mm
Depth			32		mm
Operating Temperature Range		0		50	°C
Mechanical Shock				30	G
Vibration				5	G
Weight		0.23		0.25	kg
RoHS	ROH and Lead-free components and assembly processes except for the Rev 1 and 2 prototypes.				

Table 6 Instrumentation System Data Sheet

A.2 Circuit Diagram

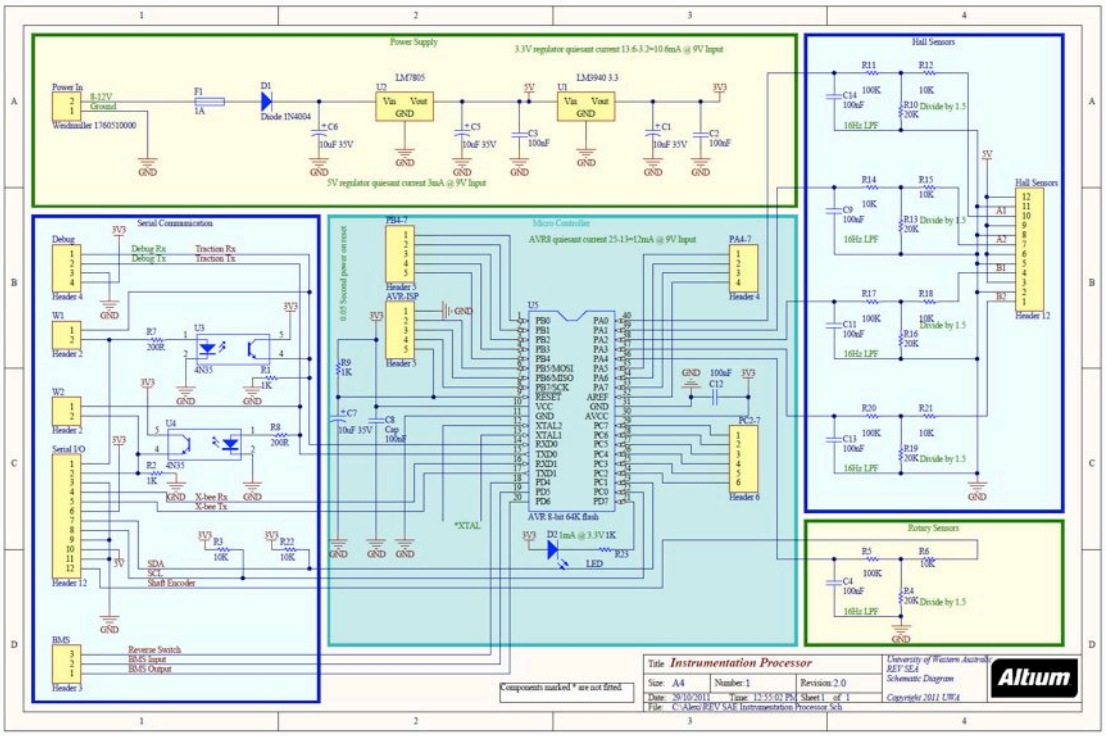


Figure 34 PCB Design (Schematic)

A.3 Bill of Materials

REV SAE Instrumentation System			
Bill of Materials			
Designator	Description	Quantity	Value
AVR-ISP, PB4-7	Header, 5-Pin	2	0.1" Pitch 5 pin Header
BMS	Header, 3-Pin	1	0.1" Pitch 3 pin Header
C1, C5, C6, C7	Polarized Capacitor (Radial)	4	10uF 35V Tanatlum 0.2" Capacitor
C2, C3, C4, C8, C9, C11, C12, C13, C14	Capacitor	9	100nF _.0" Through hole ceramic capacitor
D1	1 Amp General Purpose Rectifier	1	IN4001 Diode
D2	LED	1	Blue pannel mount LED
Debug, PA4-7	Header, 4-Pin	2	0.1" Pitch 4 pin Header
F1	Fuse	1	M205 Pannel mount fue holder
Hall Sensors, Serial I/ O	Header, 12-Pin	2	0.15" pitch 12 way PCB screw terminal block
PC2-7	Header, 6-Pin	1	0.1" Pitch 6 pin Header
Power In	Header, 2-Pin	1	0.2" pitch 2 way PCB screw terminal block
R1, R2, R23	1/ 4W radial lead resister	2	1K
R3, R6, R12, R15, R18, R21, R22	1/ 4W radial lead resister	7	10K
R4, R10, R13, R16, R19	1/ 4W radial lead resister	5	20K
R5, R11, R14, R17, R20	1/ 4W radial lead resister	5	100K
R7, R8	1/ 4W radial lead resister	2	200R
U1	5V 1A Voltage Regulator	1	LM7805 TO220
U2	3.3V !A Voltage Regulator	1	LM3940 TO220
U3, U4	Opto coupler	2	4N35
U5	AVR 8-bit 64K flash	1	ATMega644P
W1, W2	Header, 2-Pin	2	0.1" Pitch 2 pin Header
	Gromets	2	Pannel mount gromets
	Diecast box	1	120mm x 65mm x30mm dicast box
	Printed circuit board	1	REV SAE IP Rev 2

Table 7 PCB Bill of Materials

A.4 External Wiring

Figure 35 and Table 8 illustrate the colour coding of the external wiring. There are several 5V and 3.3V lines providing power to external sensors. These power lines are not short circuit protected and will either blow the fuse of shut down the linear regulators if overloaded. The 5V and 3.3V wire bare ends have been cut off and taped over to prevent short circuits.

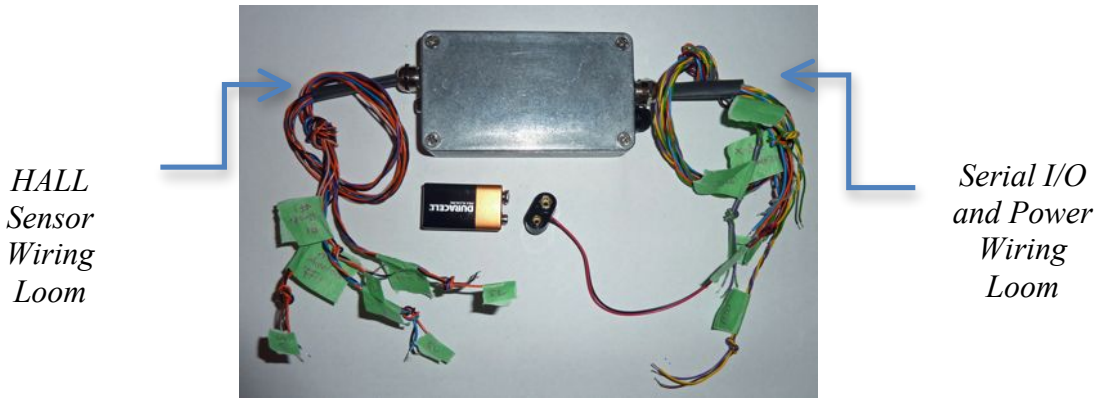


Figure 35 External Wiring (Colour Coding)

REV SAE Instrumentation System					
External Wiring					
Port	**Group**	**Signal**	**Connector**	**Pin**	**Wire Colour Code**
Left	External Power	Power	Power In	2	Red
		Ground		1	Black
	Traction	Rx	Serial I/O	1	Yellow
		Tx		2	Gray
		Ground		3	Green
	X-Bee	Ground		3	Green
		Rx		4	Brown
		Tx		5	Blue
		3V3		6	Pink
	I^2C	SDA		7	Gray
		SCL		8	Purple
		GND		9	Green
	Shaft Encoder	5V		10	Orange
		Ground		11	Green
		Shaft Encoder		12	Blue
	BMS	Output	BMS	1	Orange
		Input		2	Green
		Reverse		3	Blue
Right	A1	5V	Hall Sensor	1	Orrage
		Ground		2	Black
		A1		3	Brown
	A2	5V		4	Orrage
		Ground		5	Black
		A1		6	Blue
	B1	5V		7	Orrage
		Ground		8	Black
		A1		9	Pink
	B2	5V		10	Orrage
		Ground		11	Black
		A1		12	Purple

Table 8 External Wiring

A.5 Internal Wiring

Figure 36 and Table 9 illustrate two cable adaptors that convert the PCB AVR-ISP and Debug header plugs to sockets which are required for the ISP programmer and the USB TTL USART.

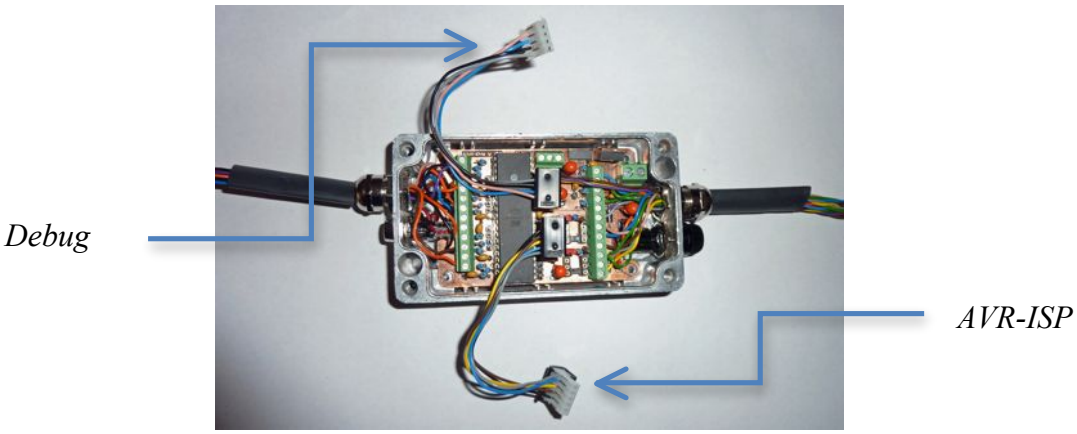


Figure 36 Internal Wiring (Program and Debug Adaptors)

Instrumention System			
Internal Wiring			
Connector	Signal	Pin	Wire Colour Code
AVR-ISP	Ground	1	Grey
	MOSI	2	Blue
	MISO	3	Yellow
	SCK	4	Brown
	Reset	5	Black
Debug	Rx	1	Grey
	Tx	2	Blue
	Ground	3	Yellow
	3V3	4	Brown

Table 9 Internal Wiring

A.6 PCB Top Layer

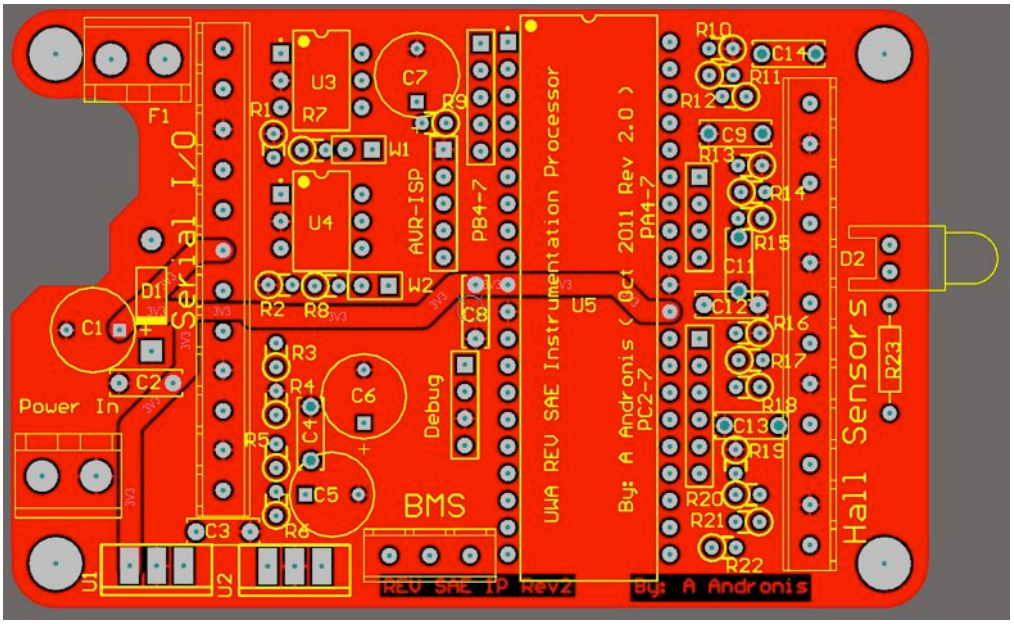


Figure 37 PCB Design (Top Overlay and Top Layer)

A.7 PCB Bottom Layer

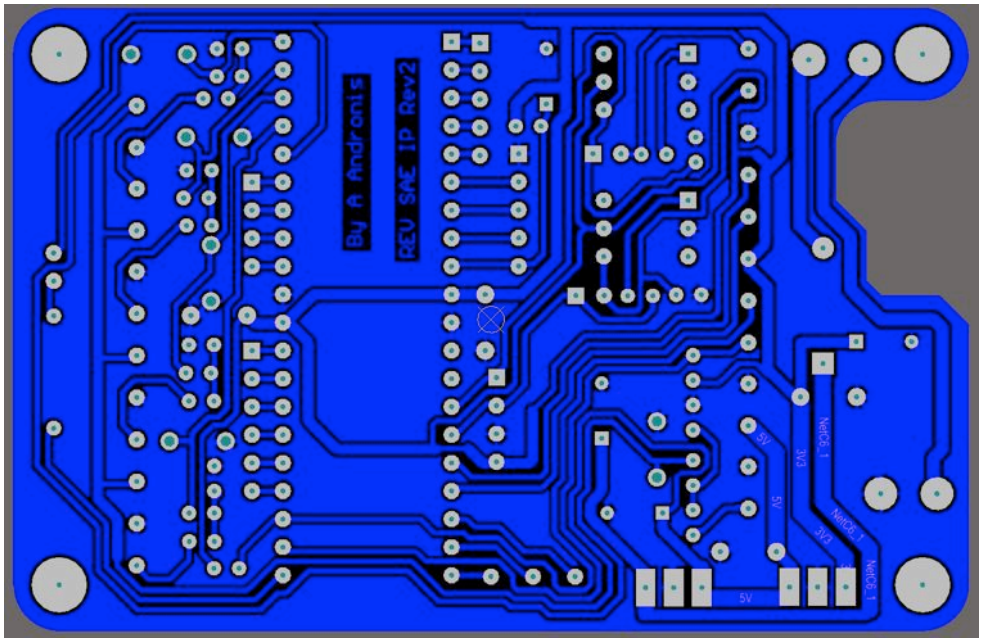


Figure 38 PCB Design (Bottom Layer)

A.8 PCB 3D Model (Top Layer)

Note: Figure 39 and Figure 40 illustrate the 3D model of what a professionally manufactured PCB will look like except the holes were deliberately made small to aid hand drilling. The final PCB requires that all the holes sizes be set to the correct component diameters.

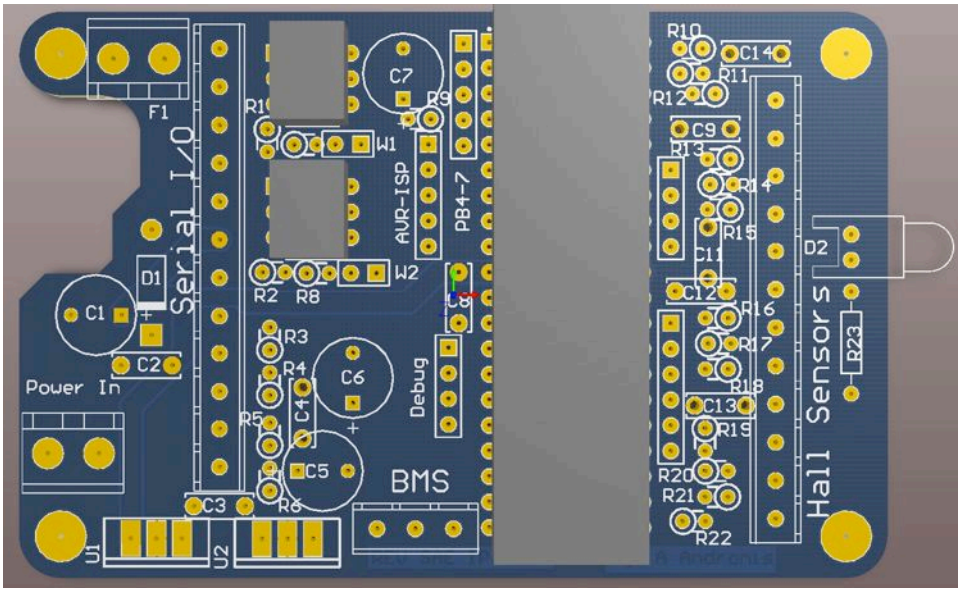


Figure 39 PCB 3D Model (Top Layer)

A.9 PCB 3D Model (Bottom Layer)

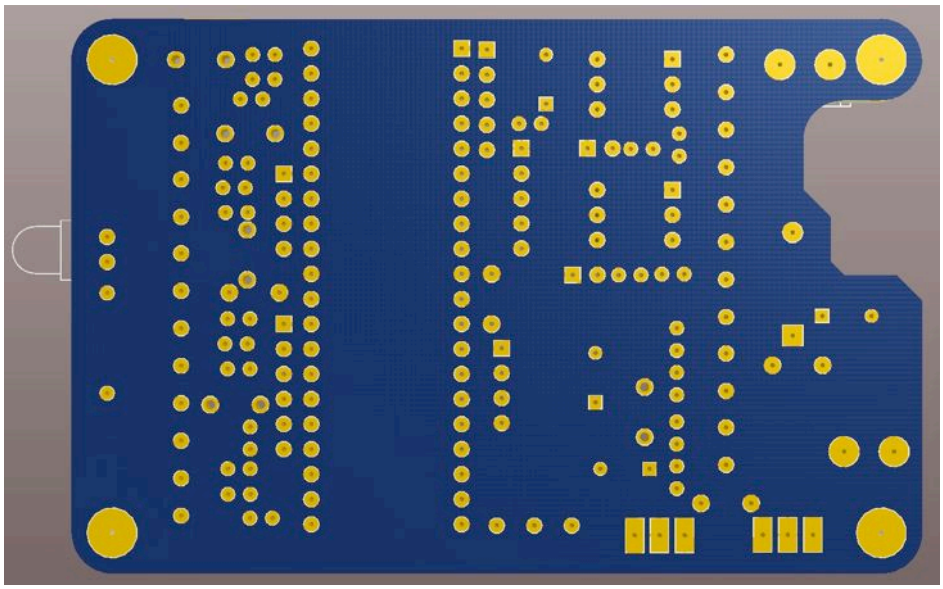


Figure 40 PCB 3D Model (Bottom Layer)

A.10 PCB Cardboard Prototyping

Figure 41 illustrates the PCB cardboard prototyping which was used to determine the housing size and the PCB geometry. The cardboard prototype was also useful in identifying electronic component mechanical clearance issues.

Figure 41 PCB Cardboard Prototyping

A.11 PCB Double Sided Manufacture

Due to the lack of time and resources a homemade PCB was constructed. Figure 42 illustrates the manufacturing steps require for a hobbyist double sided PCB with no through hole plating. The double sided PCB was used to validate the circuit design. Although the double sided PCB provided a functional Instrumentation system the PCB manufacturing and soldering is not of sufficient quality to provide reliable operation on a vibrating vehicle. It is highly recommended that the final PCB be professional manufactured and assembled by a technician with good soldering skills. Several attempts were required before a PCB of usable quality was obtained. The double sided PCB was assembled using leaded components and solder which makes assemble easy for the unskilled but it is not ROHs compliant. When the professionally manufactured PCB is assembled it is highly recommended that lead free solder be used with the appropriate lead free equipment.

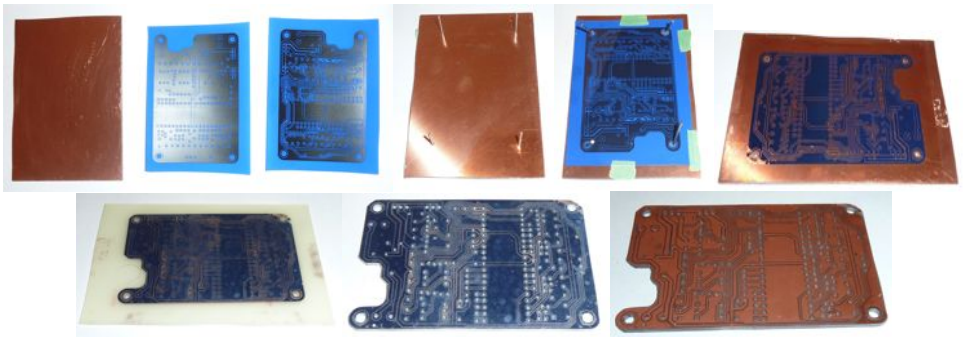


Figure 42 PCB Double Sided Manufacture

Appendix B Software

B.1 Rev_SAE.c - Main Program Source file

```
// Clear CLKDIV/8 Fuse for 8MHz Clock
// Need to disable JTAGEN fuse for normal Port C functionality

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/eeprom.h>
#include <avr/pgmspace.h>
#include <util/delay.h>
#include <avr/wdt.h>
#include <util/crc16.h>
#include "USART.h"
#include "digSensor.h"
#include "xbee.h"


#define ADC_VREF_TYPE 0x40
#define FOSC 8000000// Clock Speed
#define BAUD 19200 // Baudrate
#define MYUBRR (((FOSC / (BAUD * 16UL))) - 1)
#define THRESHOLD 10 // Needs to be set properly once final
pedal box is tested


/* reads analog input PA0-PA7 (0x00-0x07) and
returns the result as a value between 0-1024 */
unsigned int read_adc(unsigned char adc_input)
{

      ADMUX=adc_input | (ADC_VREF_TYPE & 0xff);
      _delay_us(10); // Delay needed for the stabilization of
//the ADC input voltage
      ADCSRA|=0x40; // Start the AD conversion
      while ((ADCSRA & 0x10)==0); //Wait for the AD conversion
                                  //to complete
      ADCSRA|=0x10;
      return ADCW;
}

/* Initialises the Pin directions on the AVR as well as
enabling the ADC and global interrupts */
void SetupPorts()
{
      DDRA = 0x00;
      DDRB |= 1<<PD6| 1<<PD7; // LED and BMS OUT
      DDRC = 0x00;
      DDRD = 0x00;
```

```
	ADCSRA = 0b10000101; // ADC clock prescaler is /32 with
                         // 101 as final bits

	sei(); // Enables global interrupts
}

/* Checks if the diff between two sensor values
are within a user defined THRESHOLD region */
int checkHallSensors(unsigned int A1,unsigned int A2){
	if( A1 > A2){
		if( A1-A2 <= THRESHOLD)
			return 1;
		}
	else{
		if( A2-A1 <= THRESHOLD)
			return 1;
	}
	return 0;
}



/* NOTE: ACCEL/GYRO communication has been removed because it
causes the program to crash. Problem must be investigated in
the future */
int main(void)
{

	SetupPorts();
	unsigned int accel1,accel2,brake1,brake2,rotary,reverse;

	// Initialise USART interfaces
	USART_Init(MYUBRR);
	xBee_Init(MYUBRR);

	/* ACCEL/GYRO communication functions cause program to
	crash
	initAccel();
	initGyro();
	*/

	// Main Loop
	while (1)
	{

	//TODO:
	//		LED CODES
	//		BMS communication
	//		Fix Accel/Gyro communication

		//Read analog inputs
		 accel1 = read_adc(0x00);
		 accel2 = read_adc(0x01);
		 brake1 = read_adc(0x02);
		 brake2 = read_adc(0x03);
		 rotary = read_adc(0x04);
```

```
       reverse =  PIND >> PD4;

/* Example of reading Digital Inputs (NOT WORKING)
      char* Xhigh = 0;
      char* Xlow = 0;
      readAccel(ACCEL_XOUT_H,&Xhigh);
      readAccel(ACCEL_XOUT_L,&Xlow);
      int x = (int) (Xhigh << 8) + Xlow;
*/


      //
      // Proccess and Transmit Hall Sensor Data
      //
      unsigned int accel,brake;
      if(checkHallSensors(accel1,accel2)){
            accel = (accel1 + accel2) >> 2;
      }
      else{
            //TODO handle sensor error
            break;
      }

      if(checkHallSensors(brake1,brake2)){
            brake = (brake1 + brake2) >> 2;
      }
      else{
            //TODO handle sensor error
            break;
      }

      char *hallMsg = calloc(6,sizeof(char));
      if(reverse){
            hallMsg[0] = 'R';
      }
      else{
            hallMsg[0] = 'F';
      }

      char * A = intToChars(accel);
      char * B = intToChars(brake);
      strcat(hallMsg,A);
      strcat(hallMsg,B);
      sendMessage(hallMsg,HALL);
      free(A);
      free(B);
      free(hallMsg);


      //
      // Transmit Rotory Sensor Data
      //

      char * R = intToChars(rotary);
      sendMessage(R,ROTARY);
```

```
            free(R);


      //INSERT X-BEE TRANSMISSION CODE
      // sendXbee(HallMessage);
      // sendXbee(RotaryMessage);
      // sendXbee(IMU_Message);

      _delay_ms(32); // Analog Sensor sample rate 32Hz = 1
                     //sample every 32ms
      //In future this should be replaced by a timer system
      //because Accelerometer/Gyroscope IMU is sampled at a
      //different rate

      }

// HANDLE FATAL ERROR

}
```

B.2 Rev_SAE2.c – Receiving End Program Source File

```
// Clear CLKDIV/8 Fuse for 8MHz Clock
// Need to disable JTAGEN fuse for normal Port C functionality

#include <stdlib.h>
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/eeprom.h>
#include <avr/pgmspace.h>
#include <util/delay.h>
#include <avr/wdt.h>
#include <string.h>
#include "USART.h"

#define ADC_VREF_TYPE 0x40
#define FOSC 8000000// Clock Speed
#define BAUD 19200
#define MYUBRR FOSC/16/BAUD-1

char receiveBuf[BUFLEN];
unsigned int hall;



/* Sample algorithm developed to decode hall
sensor data at the receiving end */
void decodeHall(){

                char *msg = calloc(HALL_LEN+1,sizeof(char));
                char *crc = calloc(3,sizeof(char));
                for(int i = 1; i < HALL_LEN+1; i++){
                     msg[i-1] = receiveBuf[i];
                }
                for(int i = HALL_LEN+1; i < HALL_LEN+3; i++){
```

```
                    crc[i - HALL_LEN-1] = receiveBuf[i];
                }
                 unsigned int check = charsToInt(crc);
                 free(crc);
                 unsigned int actual = calculateCRC(msg);

                 if(check != actual)
                    sendDebug("CRC ERROR");
                else{
                    char *c = calloc(3,sizeof(char));
                    c[0] = receiveBuf[0];
                    c[1] = receiveBuf[1];
                    hall = charsToInt(c);
                    free(c);
                }
                free(msg);
}

/* Interrupt for whenever a byte is received on RX0 */
ISR(USART0_RX_vect){

    char state = recieveChar(receiveBuf);

    if (state == ENDED){
        if(receiveBuf[1] == HALL)
            decodeHall();
        if(receiveBuf[1] == ROTARY){
        // Handle Rotary Message
        }
        if(receiveBuf[1] == ACCEL_GYRO){
        // Handle IMU Message
        }
        if(receiveBuf[1] == SYSTEM){
        // Handle SYSTEM Message
        }
    }

}



int main(void)
{


    sei(); // Enable Global Interrupts

    USART_Init(MYUBRR);
    while (1) // Main loop
    {
    //Wait for receive interrupt
    }
}
```

B.3 USART.h – Serial Communication Header File

```
#include <stdlib.h>
#include <avr/io.h>
#include <util/crc16.h>
#include <string.h>

// Receive State Codes

#define STARTED 0x01
#define ENDED   0x02
#define ERROR   0x03

#define BUFLEN  30

// Delimiters

#define START   '$'
#define END1    '\r'
#define END2    '\n'

// Message Types

#define HALL          'H'
#define ACCEL_GYRO    'M'
#define ROTARY        'R'
#define SYSTEM        'S'

// Message Lengths

#define HALL_LEN           5
#define ACCEL_GYRO_LEN     4
#define ROTARY_LEN         2
#define SYSTEM_LEN         1


void USART_Transmit(char data );
unsigned int calculateCRC(char *str);
char currentState(char frame);


void USART_Init(unsigned int ubrr);


char* intToHex(unsigned int num,int len);

char* intToChars(unsigned int num);
unsigned int charsToInt(char* c);

void sendDebug(char *msg);

void sendMessage(char *buf, char type);
char recieveChar(char *buf);
```

B.4 USART.c– Serial Communication Source File

```
#include "USART.h"

int idx = 0;
int MsgLen = 3;
char state = ENDED;
char prev = '\n';

/* Initializes the USART 0 interface
for the Instrumentation -> Traction Control Link*/
void USART_Init(unsigned int ubrr){

      /* Set baud rate */
      UBRR0H = (unsigned char)(ubrr>>8);
      UBRR0L = (unsigned char)ubrr;
      /* Enable receiver and transmitter */
      UCSR0B = (1<<RXEN0)|(1<<TXEN0);
      /* Set frame format: 8data, 1stop bit */
      UCSR0C |= (3<<UCSZ00);
      /* Enable receive interrupt */
      UCSR0B |= (1 << RXCIE0);

}

/* Transmits a single byte */
void USART_Transmit(char data ){

      /* Wait for empty transmit buffer */
      while ( ( UCSR0A & (1<<UDRE0)) == 0 ){};
            UDR0 = data;
}

/* Transmits a Debug Message (uncoded) that
can be read in HyperTerminal */
void sendDebug(char *msg){
      int i = 0;
      while( msg[i] != '\0'){
            USART_Transmit(msg[i]);
            i++;
      }
      USART_Transmit('\r');
      USART_Transmit('\n');
}


/* Sends A formatted message acrross the USART Link
accepts raw message data and the type of message */
void sendMessage(char *message, char type){

      char *msg = calloc(strlen(message) + 4,sizeof(char));
      msg[0] = type;
      strcat(msg,message);
      char *crc16;
      unsigned int crc = 0;
      crc = calculateCRC(msg);
```

```
	crc16 = intToChars(crc);

	strcat(msg,crc16);
	free(crc16);

	USART_Transmit(START);

	int i = 0;

	while( msg[i] != '\0'){
		USART_Transmit(msg[i]);
		i++;
	}
	free(msg);

	USART_Transmit(END1);
	USART_Transmit(END2);
}

/* Helper function to turn an integer into its ASCII
representation accepts parameter num - the number being
converted and len - length of the ASCII string = 3 for a 10bit
number or 4 for a 16bit number */
char * intToHex(unsigned int num,int len){

	char * str = calloc(len+1,sizeof(char));

	for(int i = 0; i<len; i++){
		str[i] = (num >> (len-i-1)*4)& 0x0f;

		if(str[i] <= 0x09)
			str[i] += 0x30;
		else if( str[i] > 0x09)
			str[i] += 0x37;
	}
	return str;
}

/* Helper function to turn an integer into 2 chars (non
readable)*/
char* intToChars(unsigned int num){
	char *c = calloc(3,sizeof(char));
	c[0] = (char) (num >> 8);
	c[1] = (char) num;
	return c;
}

/* Helper function to turn an 2 chars into a 16-bit integer*/
unsigned int charsToInt(char* c){
	unsigned int i = 0;
	i += (int) (c[0]<< 8);
	i += (int) c[1];
	return i;
}
```

```
/* Function to calculate the CRC of a given string
Outputs a 16 bit integer */
unsigned int calculateCRC(char *str){

	int i = 0;
	unsigned int crc = 0;
	while (str[i] != '\0'){
		crc = _crc16_update(crc,(uint8_t) str[i]);
		i++;
	}
	return crc;
}


/* Function to receive a single char from Rx0
Outputs the current state of the buffer, accepts the receive
buffer being used as an input received chars are appended to
the end of this buffer*/
char recieveChar(char* buf){


	char byte = UDR0;

	state = currentState(byte);
	if(state == STARTED){

		if(idx == 1){
			MsgLen = 5; // 5 = 1 Start bit 2 CRC Bits + 2
					// End bits
			char c = buf[idx];
			switch (c) {
				case HALL:
					MsgLen += HALL_LEN;
				case ACCEL_GYRO:
					MsgLen += ACCEL_GYRO_LEN;
				case ROTARY:
					MsgLen += ROTARY_LEN;
				case SYSTEM:
					MsgLen += SYSTEM_LEN;
				default:
					state = ERROR;
			}
		}

		buf[idx] = byte;
		idx++;

	}
	else if (state == ERROR){

		idx = 0;
		buf[0] = '\0';
	}

	else if( state == ENDED){
```

```
            buf[idx] = '\0';
            idx = 0;

      }

      prev = byte;

      return state;
}


/* Function used by ReceiveChar to determine the current
buffer state. Takes the current frame as an input and outputs
the current state */
char currentState(char frame){


      if (frame == START && prev == END2){
            state = STARTED;
            }
      else if(frame == '\n' && prev == '\r' && idx != (MsgLen-
1)){
            state = ERROR;
      }
      else if (idx >= BUFLEN-1){
            state = ERROR;
      }
      else if( idx > MsgLen){
            state = ERROR;
      }

      else if(frame == END2 && prev == END1)
            state = ENDED;



      return state;
}
```

B.5 digSensor.h– IMU communication header file

```
#include "twi.h"

#define ADXL_ADDR_READ      0xA7
#define ADXL_ADDR_WRITE     0xA6

#define ITG_ADDR_READ 0xD1
#define ITG_ADDR_WRITE      0xD0

// Accel Registers
#define ACCEL_XOUT_L  0x32
#define ACCEL_XOUT_H  0x33
#define ACCEL_YOUT_L  0x34
#define ACCEL_YOUT_H  0x35
```

```
#define ACCEL_ZOUT_L  0x36
#define ACCEL_ZOUT_H  0x37

// Init Accel
#define POWER_CTL     0x2D
#define DATA_FORMAT   0x31
#define MEASURE       (1<<3)
#define RANGE_0       (1<<0)

// Gyro Registers
#define GYRO_XOUT_H        0x1D
#define GYRO_XOUT_L        0x1E
#define GYRO_YOUT_H        0x1F
#define GYRO_YOUT_L        0x20
#define GYRO_ZOUT_H        0x21
#define GYRO_ZOUT_L        0x22


// Init Gyro
#define SMPLRT_DIV    0x15
#define DLPF_FS       0x16
#define DLPF_CFG_0    (1<<0)
#define DLPF_CFG_1    (1<<1)
#define DLPF_CFG_2    (1<<2)
#define DLPF_FS_SEL_0 (1<<3)
#define DLPF_FS_SEL_1 (1<<4)
#define PWR_MGM       0x3E
#define PWR_MGM_CLK_SEL_0   (1<<0)

void initAccel();
void initGyro();

char read_Accel(char register_addr, char * value);
char write_Accel(char register_addr, char value);
char read_Gyro(char register_addr, char * value);
char write_Gyro(char register_addr, char value);
```

B.6 digSensor.h– IMU communication source file

```
#include "digSensor.h"
#include "USART.h"

/* initialises the Accelerometer Device */
void initAccel(){
      twiInit(100000); // Init SCL at 100kHz
      write_Accel(POWER_CTL, MEASURE); // Set Accel to
                                       // measurement mode
      write_Accel(DATA_FORMAT, RANGE_0); // set range to +/- 2g
}

/* initialises the Gyroscope Device */
void initGyro(){
      twiInit(100000); // Init SCL at 100kHz
```

```
      write_Gyro(DLPF_FS,
DLPF_FS_SEL_0|DLPF_FS_SEL_1|DLPF_CFG_0);
      write_Gyro(SMPLRT_DIV, 9);
      write(PWR_MGM, PWR_MGM_CLK_SEL_0);

}

/* Reads the contents of register_adddr on the accelerometer
and stores it into value.Outputs 0 if successful or an error
code otherwise */
char read_Accel(char register_addr, char * value){
      twiReset();
      return twiReceive(ADXL_ADDR_READ, register_addr, value);
}

/* Writes they byte "value" to register_adddr on the
accelerometer
Outputs 0 if successful or an error code otherwise */
char write_Accel(char register_addr, char value){
      twiReset();
      return twiTransmit(ADXL_ADDR_WRITE, register_addr,
value);
}

/* Reads the contents of register_adddr on the gyroscope
and stores it into value.Outputs 0 if successful or an error
code otherwise */
char read_Gyro(char register_addr, char * value){
      twiReset();
      return twiReceive(ITG_ADDR_READ, register_addr, value);
}

/* Writes they byte "value" to register_adddr on the gyroscope
Outputs 0 if successful or an error code otherwise */
char write_Gyro(char register_addr, char value){
      twiReset();
      return twiTransmit(ITG_ADDR_WRITE, register_addr, value);
}
```

B.7 xbee.h– X-Bee communication header file

```
#include <stdlib.h>
#include <avr/io.h>
#include <string.h>

void xBee_Init(unsigned int ubrr);
void sendXbee(char *message);
void Xbee_Transmit(char data);
```

B.8 xbee.c– X-Bee communication source file

```
#include "xbee.h"

/*Initializes xbee transmitter device */
void xBee_Init(unsigned int ubrr){

	/* Set baud rate */
	UBRR1H = (unsigned char)(ubrr>>8);
	UBRR1L = (unsigned char) ubrr;
	/* Enable receiver and transmitter */
	UCSR1B = (1<<RXEN1)|(1<<TXEN1);
	/* Set frame format: 8data, 1stop bit */
	UCSR1C |= (3<<UCSZ01);
}
/* Transmit a single byte to the X-bee transmitter */
void Xbee_Transmit(char data ){

	/* Wait for empty transmit buffer */
	while ( ( UCSR1A & (1<<UDRE1)) == 0 ){};
		UDR1 = data;
}


/* Convert an ASCII message to an RF Packet
to be transmitted wirelessly */
void sendXbee(char *message){
	Xbee_Transmit(0x7E);
	int len = strlen(message);
	char c = (char) (len >> 8);
	Xbee_Transmit(c);	// Length MSB
	c = (char) len;
	Xbee_Transmit(len);  // Length LSB
	Xbee_Transmit(0x01); // API Identifier
	Xbee_Transmit(0x00); // Frame Id (no ACKs)
	Xbee_Transmit(0x00); // Destination Address MSB
	Xbee_Transmit(0x02); // Destination Address LSB
	Xbee_Transmit(0x01); // Options: Disable ACKs

	int sum = 0;
	for(int i = 0; i< len; i++){
		Xbee_Transmit(message[i]); // RF Data
		sum += message[i];
	}
	char checksum = 0xFF - ((char)(sum >> 8));
	Xbee_Transmit(checksum); // Checksum

}
```