EEL 4915 – GROUP 5

TUV-ADDS

Tactical Up-Armored Vehicle – Automatic Distress Detection System

Julien Mansier – Jason Skopek – Eric Nachtigal – Alyssa Almanza

7/18/2011

TABLE OF CONTENTS

1 EXECUTIVE SUMMARY

The concept of the Tactical Up-armored Vehicle – Automatic Distress Detection System (TUV-ADDS) is a system that is intended to be able to detect if a vehicle has been in a specific distress event resulting in the need for assistance. This system, which was designed for the Iraqi and Afghanistan theaters, will automatically communicate a distress signal to a designated command center. This project has a two-phase design, phase one being the recognition of certain characteristics of a distress-causing event through specific sensor values that quantify the distress. The second phase is that the system automatically and accurately communicate that there is a vehicle in need and the position of where that vehicle is to command center.

The scope of the system is meant to be relative to up-armored, but not heavy armored (an example of an up-armored vehicle can be seen in Figure 1), wheeled tactical vehicles, and two specific distress instances: rollover and significant Improvised Explosive Device (IED) hits. After speaking with some members of the armed forces who served in the recent Iraq and Afghanistan theaters, there were several accounts where vehicles had become nonoperational and help was needed to continue a mission or save a life. The TUV-ADDS system is intended to aid occupants of specific vehicles in these critical instances.

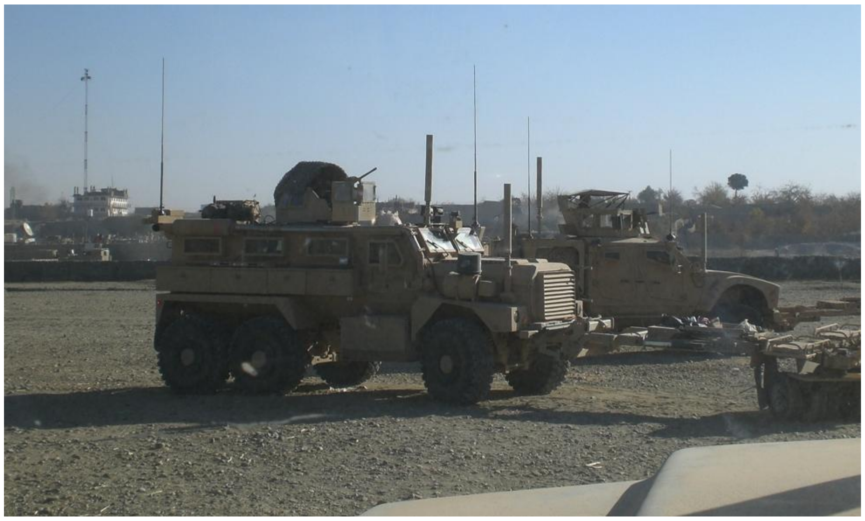


Figure 1: U.S. Marines MRAP; example of an up-armored wheeled tactical vehicle

(Permission from Sgt. Irizarry, Eduardo)

The way TUV-ADDS recognizes if a vehicle is in distress is through a sophisticated network of sensors. The system senses and monitors certain percentage increases such as vertical displacement, flash, and heat to indicate an IED blast. For rollover indication, an accelerometer is used to sense if the vehicle is overturned. These sensors collect the data, which then the system analyzes and concludes whether the information combined is an indication of a distressed vehicle.

After recognizing the vehicle has been involved in a distress situation based on the specified criteria, TUV-ADDS wirelessly communicates that an incident has occurred to the command center. This is achieved by applying the use of a Global Positioning System (GPS) device and wireless communication capability to the design. This feature allows for occupants of the distressed vehicle to focus on other tasks rather than radioing in for support themselves.

TUV-ADDS is a system that consists of many key components to detect a vehicle in distress along with software to analyze the data. The software portion of this system is able to identify if there are more than one characteristic of a distressed vehicle from the information taken in from the sensors, and with that, accurately relay to the command center that there is a vehicle in need. In the end, this feature is what makes this system useful, it will allow for timelier responses to vehicles in need along with allowing the vehicle occupants to focus on other immediate tasks that they are responsible for.

2 PROJECT DISCRIPTION

2.1 MOTIVATION

There is a lot of motivation for the TUV-ADDS project to be pursued. First of all, general interest was a main motivation. Many of the group members had interest in pursuing a defense related project in hope that it will be beneficial in pursuing a defense or military related career. After recognizing that this type of project was what the team wanted to pursue, it then opened the door to a variety of project ideas. A drive to design a system that would be of assistance to the men and women in uniform was a significant motivating factor for the team. News of the fighting in the ongoing war in the Middle East far too often reference harm resulting from IED strikes. With this method of attack being so predominant, the team decided to pursue a project that would assist in these specific incidents.

As important as our future careers are to the team, there was also an amount of personal motivated by many of the teammates having ties by either family members or close friends that are in the military and felt that this is a beneficial way to apply the engineering skills we have acquired. The safety of our fellow Americans fighting in the Middle East, and any theater of war for that matter, was another motivation for our team to pursue this project idea.

The TUV-ADDS group is made of entirely electrical engineering students. There was motivation for the group to pursue a project with a large amount of controls design. The amount of sensor interfacing and the lack of need for high level programming experience in this project was an appealing idea, while taking into consideration the group's specialties.

2.2 GOALS AND OBJECTIVES

The objective of TUV-ADDS was to design a system that is not only successful in detecting that specific events have occurred to a vehicle and recognizing that they are important, but also having the system notify the command center that a timely response for assistance is needed and provide the distressed vehicles position. Our goal was to make TUV-ADDS user-friendly, accurate at detecting specific events, accurate in relaying the position of the vehicle in need, and have it embedded into the vehicles previous structure so that it is not a burden to the user. The objective is to allow the user to focus on what is wrong with their vehicle, engage an enemy, or even save time when trying to escape a vehicle while feeling confident that there is help on the way without having to radio for it themselves.

Sometimes, a vehicle's occupants may be in a situation where they could better use the abilities of their skills right away rather than having to call for assistance. There is also the possibility that the occupants are in a condition that they are unable to call for assistance. The objective of TUV-ADDS is intended to help in these very situations. The system was designed with the hope that an automatic detection of distress and communication of these events could possibly be an effective aid in saving time, money, and even in some cases lives.

Accuracy was a very important goal for TUV-ADDS. The objective is to avoid false alarms if at all possible. The intent was for this system to be beneficial, not a nuisance, so it is imperative that it is accurate. The system is supposed to automatically communicate if the vehicle is in need of assistance, we do not want the system to alert for help if none is needed. This would cause a loss of time and resources that could be used somewhere else in a wartime situation, not to mention put others in danger for no reason by causing them to venture out into areas that are very dangerous. That is why TUV-ADDS will only communicate a need for assistance if there are more than one distress characteristic indicated. The group hoped this would allow for the vehicle to do its purpose without setting off any false need for assistance requests. The group understands that it is necessary for up-armored vehicles to go through rough terrain and that fact has been incorporated into the design. For the system to be considered beneficial, it is also very important that the GPS location is accurate. If a vehicle is truly in distress the system needs to relay the exact position to the command center, allowing them to aid them. Without an accurate location, this system would be very little assistance.

Another goal for this system was for it to be practically invisible to the user. It is ideal that TUV-ADDS is out of the way and only in use when needed. The idea is that the system should not be an inconvenience, therefore making it unfavorable to use. If the system is a burden, it is little help to the intended user because they could just as easily perform the current procedures that are implemented instead.

2.3 SPECIFICATIONS AND REQUIREMENTS

There are many technical specifications that this system needed to meet to achieve the goals that the group had set for this system. In the following requirements list, the specifications have been broken down the system into five (5) modules to make the requirements easier to follow. The hardware flow diagram can be seen below in Figure 2.

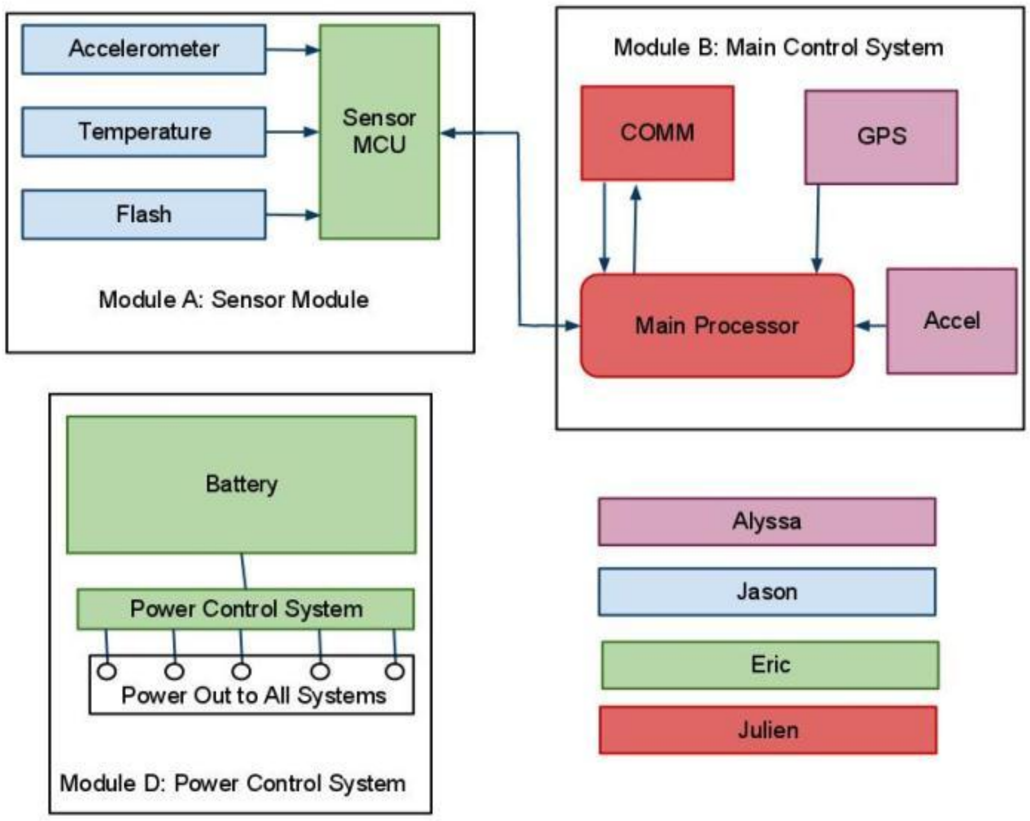


Figure 2: System Hardware Overview

Module A describes the specification requirements for the sensors that were used in the system. This module will be further broken down into four (4) main components; the Sensor MCU, Accelerometer, Temperature Sensor, and a Photodiode. The Sensor MCU collects and interprets data from the various external sensors and transmits the data to the main processor if it detects a distress event. The Accelerometer quantitatively measures the change in motion of the system and sends that data to the Sensor MCU. The sensitivity of the accelerometer was chosen because it is a low G accelerometer and was suffice for the testing the system endured. It was decided to require three (3) axes because the software is able to detect significant motion in the X, Y, and Z directions. Digital interface was required for this component allowing it to connect directly to the Sensor MCU. The photodiode measures the luminance applied to the system, and sends the data that it measures to the Sensor MCU. The required wavelength (see Table 1 below) of the photodiode is due to the fact that we tested the system with visible light, meaning the photodiode would need to be measuring within the visible light spectrum. The Temperature sensor collects temperature data and relays this to the Sensor MCU. The digital output was a requirement so it could communicate with the sensor MCU. High operational frequency of the sensor was also required so it can transfer data as quick as the sensor MCU could interpret the data.

Module A-Sensor MCU	
Need for External interrupts?	Yes
Processing Speed:	> 8 MHz
Supported Digital interfaces protocols:	SPI and I2C
Operating Voltage:	6 Volts or less
Number of Analog inputs needed	At least 3

Module A-Accelerometer	
Supply Current Consumed (normal mode):	250uA
Ideal Voltage supply:	1.8V
Supported Digital interfaces protocols:	SPI or I2C
Sensitivity:	± 2g to ± 16g
Axes:	X, Y, and Z
Bandwidth:	1kHz to 32 kHz
Operating Temperature:	-40°C + 85°C

Module A-Temperature Sensor	
Typical max current consumed:	150uA
Voltage supply range:	2.7V to 5.5V
Output:	Digital
Supported digital interface protocol:	I2C
Accuracy:	±2.0°C from −25°C to +85°C (max)
Speed of Data transfer:	High
Operating temperature:	−55°C to +125°C

Module A-Photodiode	
Supply Current (max):	1.7mA
Supply voltage:	2.7V – 5.5V
Output:	Analog or digital, light intensity to voltage
Operating Free-air Temp:	0°C to 70°C (recommended), -25°C to 85°C (max)
Wavelength:	320nm to 1050nm

Table 1: Module A Specifications

Module B contains the Main Control System. This module goes into detail about the communication, GPS, Accelerometer, and Main processor. The Main Control system is the liaison interface between the raw data that is collected from the sensors and the command center GUI. The specifications maximize efficiency of the system. The requirement for at least three (3) USART lines in the main processor was to allow the processor to communicate with other components that utilized USART protocol. To prevent interference with other wireless devices the network parameters had to be configured. The Accelerometer is the same as the accelerometer in module A, so we are not repeating the specifications in this table. Accuracy of the GPS was an important requirement for the system design because one of the main objectives of TUV-ADDS is reliable position information. Other general requirements of the all the devices specification can be seen below in Table 2.

Module B-Main Processor	
Operational Frequency:	16 MHz
Need for interrupt services?	Extensive
Need for data lines:	Ability to run multiple likes simultaneously
Minimum flash ram size:	512K
Need for WatchDog Timer?	Yes
Supply voltage:	3.3V – 5V
Amount of UART lines:	At least 3

Module B-Communication	
Minimum range:	100ft
Ability to send packets?	Yes
IEEE 802.14.5 Protocol:	Yes
Need for Interference avoidance capabilities?	Yes
Supply voltage:	3.3V – 5V
Network Parameter Controllability:	Assignable PAN ID

Module B-GPS	
Accuracy:	Within 5 meters
Update rate:	Frequent
Speed accuracy:	> 60 mph
Cold start up time:	< 35 Seconds
Supply Voltage:	< 5V

Table 2: Module B Specifications

Module C, the CAN-bus functions as a connection for all the nodes of the design for the system, such as the sensor controller. It also allows for easy integration of new hardware without the worry of bus collisions. The need for the interrupt output pin is to save clock cycles by allowing the microcontroller to directly check whether a message is waiting rather than it having to communicate with the CAN Controller. The USART communication is necessary to the system design because the microcontroller needs to send and receive messages to the CAN controller. The BUS speed of 500 Kbits/sec (see Table 3) was chosen as a requirement because it will be easy to interface with and be more than sufficient to meet or system design needs.

Module C-Can Controller	
Number of interrupt output pins:	1
SPI or I2C USART serial ports:	1
Need to implement CAN 2.0 A/B?	Yes
Bus Speed:	500 Kbits/sec

Table 3: Module C Specifications

Module D is the Power System with general specifications that can be seen below in Table 4. The power system draws power from a 9.6V/1600mAH battery and then uses voltage regulators to step the voltage down to the appropriate levels needed for each component. Therefore, the requirement for several different regulators was very important because each device has a different voltage requirement.

Module D- Voltage Regulator	
Need for several different Voltages:	Yes; 5V, 3V, and 1.8V
Current:	Consistent
Stability:	Must be stable
Need for fail-safe?	Yes
Need for Low Dropout?	Yes

Table 4: Module D Specifications

Module E goes into detail about the main processor and command module software. The software takes the data collected and decides whether or not a distress event has occurred based on a series of characteristics. Programmable interrupts are required to control the flow of data throughout the system.

Software needed to send data as packets for the system to be able to operate at a high efficiency level. The choice of programming language facilitated the graphical user interface as well as interfacing with serial ports. Other general software requirements can be seen below in Table 5.

Module E-Main Processor	
Need for interrupt services?	yes
Ability to send data in packets?	yes
Clock speed:	High for efficiency
Wake-up time:	Fast, for efficient data control

Module E-Command Center	
Platform:	Intuitive GUI Stable Platform
Decoding needs:	Ability to decode data packets from serial (Wi-Fi)
Control requirements:	User must have control over previous vital data
Coding languages:	C++ or Java

Table 5: Module E Specifications

3 PROJECT RESEARCH

3.1 SIMILAR PROJECTS

In a way, TUV-ADDS is similar to the popular system called OnStar©, which is a system that is integrated to many civilian vehicles and includes a feature that will contact the OnStar© representatives if the system identifies that the vehicle has been involved in an accident. The system designed in this project has similar objectives, but the system design is geared to detect events very specific to a handful of military vehicle types and two events that occur frequently in the current war in the Middle East.

3.2 GOVERNMENT RELEVENCE

While TUV-ADDS is not a military grade system, research of military vehicles and the difficulties that the military has with some of their vehicles was performed and the system design is intended to be relevant to the vehicles and users that would benefit from a system of similar characteristics. Great measures were taken to make TUV-ADDS a system that soldiers, marines, sailors, and other military personnel alike, could relate to. While our team has neither access to genuine government equipment nor the budget to create a system that would be to military grade standards, the group has geared our project towards emulating a solution for the Department of Defense.

After talking with several DOD personnel and being aware of the situation in the Afghanistan and Iraq theaters, it is quite obvious that IED hits are very prominent and the cause of many American injuries and deaths. It is also evident that the United States has been taking great measures in combating this type of warfare. Our group decided that an automatic vehicle distress alert system could be a helpful aspect to the warfighter. Our team took on the challenge to research characteristics of military vehicles being hit by an IED. This process included questioning members of the armed forces that had personally been encountered with IED hits in their deployments as well as members of the PEO STRI IEDES team, to gain insight to these characteristics on different vehicles, and characteristics of IEDs themselves. After accumulating this research it was realized that most vehicles acted differently. Since the project was on a tight budget, stringent schedule, and had limited testing abilities, it was decided to narrow the scope of the project to a small amount of vehicles. The group chose up-armored, not heavy armored tactical vehicles, because we felt we would be able to simulate characteristics of these vehicles in an IED blast most efficiently with our team knowledge, project schedule, testing abilities, and monetary assets.

There are a few key characteristics that these specific vehicles generally endure in a significant IED blast that the project will focus on. The idea is, when all of these characteristics occur together, that would define a “distress situation.” A sudden drastic increase in heat is one characteristic we have chosen to measure, because IEDs generally emit a large about of heat when they blow up. The system will also measure any greatly significant change in any one direction, but mostly in the vertical direction, since when IEDs explode they generally produce a momentum upward. A significant increase in illumination will be another characteristic that will be measured. When the systems sense all of these activities happening in a close interval of time, the system will recognize this as a “distress situation.”

Our group decided to add the detection of a roll-over incident (shown in Figure 3) after research indicated that certain up-armored vehicles have a tendency to do so. After talking with SGM Patrick Ogden, it was made clear that a roll-over can be a traumatic experience, and can be very dangerous which is evident from the pictures below. Our group looked into the ways we could detect a rollover and decided to include this into our system design.

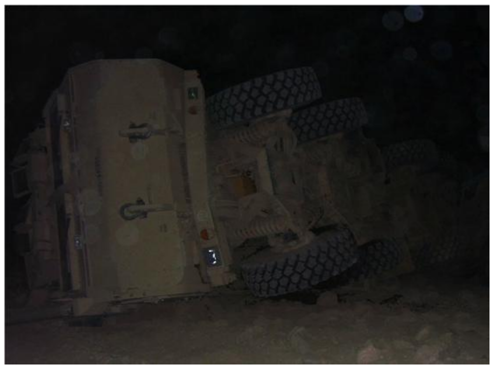
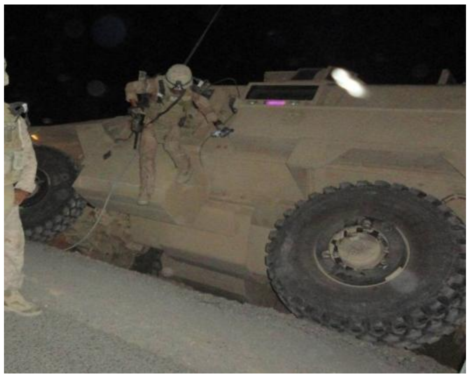



Figure 3: U.S. Marines LVSR; examples of a vehicle rollover
(Permission from Sgt. Irizarry, Eduardo)

An accelerometer will be used to detect if the vehicle is in a vulnerable position for a specified period of time. Those are the characteristics that are needed to alert the TUV-ADDS system that the vehicle is in distress.

3.3 PROJECTED USER RESEARCH

As usual, all systems start with one thought: the user. The TUV-ADDS group has done extensive research into the projected user of this system. Interviews were conducted with members of the armed forces and their information was incorporated to make our system more geared to a vehicle’s occupants. As stated before it is not expect that this system will be military standard, but the focus of this project was with a military relevance. It was our goal to do our best to create a system that men and women of the armed forces could relate to and could see as beneficial.
Brittney Johnson, who served in the U.S. Army from September 2001 to December 2008, was able to inform the TUV-ADDS team about different vehicles

she was familiar with from her time in the Army. She also gave insight she gained when she was deployed to Iraq from 2003 to 2005 for Operation Iraqi Freedom (OIF) 1 and OIF 2, on what the projected user would find useful and very honestly what an average solider would not care about or not use. She explained if a system is too inconvenient to operate, then it will most likely be tossed aside by the average solider. Also, she stated that currently there is a way to communicate if you are in distress, but there is nothing that could communicate automatically, and that could be very helpful to a soldier in certain circumstances.

While in a state of distress, the vehicle occupants would usually have a radio to request assistance; it was the objective of the project that a system such as TUV-ADDS might be helpful to the user by automatically sending requests for help to command. The idea was that this would save time if the vehicle occupants need to react quickly to a vehicle problem or if there are insurgents to be engaged. The system would be useful in much more urgent situations as well, such as, requesting help when there are injuries or circumstances that impede the vehicle occupants to be able to request help for themselves.

During research, the team interviewed Sgt. Irizarry, Eduardo, who served in the U.S. Marine Corps from June 2004 – May 2011 and just recently returned from Afghanistan. With the knowledge he gained from being involved in dozens of IED strikes, both directly and as a first responder, he expressed that with the implementation of a system of this scope to military operations could allow for more efficient tactical and strategical employment of assets, personnel, equipment, vehicles, and weaponry. He told the team that his experience also taught him that the first action taken after an IED strike occurs is communication, without communication, there will be no response for assistance. Therefore, if the communication is automatic, this would cut down on the number of procedures required to follow before the occupants can attend to the vehicle, fellow occupants if needed, or possible ambush. As a point vehicle commander, navigator, and scout element, he explained that a system such as this would increase situational awareness of the battlefield by communicating the status and location of vehicles involved in distress situations and amplify the effectiveness of first responders.

4 DESIGN RESEARCH

4.1 RESEARCH METHODS

To achieve the TUV-ADDS design that met our specification requirements, there were several different methods of research that the TUV-ADDS group relied on. Web research, data sheet analysis, price comparisons, previous projects research, and personal interviews were the main methods of research that were used. Incorporating these different methods allows us to get the best components for our design and design a system that was relevant to the projected user and scope of the project.

In the early stages of the project personal interviews were used to define the scope of the project and to ensure relevance to the government and projected user. Several sources knowledgeable on related subjects were sought out and questioned about the way several different vehicles reacted in certain events, and what characteristics of the different vehicle in these events we could quantify to recognize that these events have occurred. As more information was collected about these characteristics, the clearer the scope of the project would be based on available testing abilities, budget, and team specialties. These personal interviews also were helpful in understanding accurate characteristics of IEDs so accuracy of the system could be maintained.

Once the project was defined and the specification requirements had been laid out, web research was widely used by the group to research different components to use in the design. This allowed the team to better define the general design of the system and make sure that the correct components would be used to work as a system. The team also used web research to find testing equipment which will be able to test the system in the future, as well as see reviews online about different brands of the same component. When researching different designs for the system, the team took advantage of the plethora of information that could be gathered from what students of previous years had done in their projects.

After the initial design was brainstormed, research of different variants of the same components was done through analysis of multiple data sheets. This was needed to make sure the variants of the component would align with the specifications and requirements. Comparing data sheets allowed the group to understand how the part would perform and allowed for the best selection for the system performance to be made.

Since TUV-ADDS is a student sponsored project, there is a budget that had to be kept in mind when research on parts was performed. If a part is outside of the price that the budget allows then it was not a possible option for incorporating into the system. Significant research was been done in various companies sample programs to stay in the budget for this project. This project calls for the best parts that are within the range of prices that are budget approved.

4.2 MODULE A – SENSOR MODULE

For this design, an array of sensors was needed to satisfy the system objective of being able to detect several select parameters, which would be characteristic to an IED strike or rollover event. Several sensors where considered to include:

- Accelerometer
- Ambient Pressure sensor
- Thermo sensor
- Photodiodes

The ambient pressure sensor was considered in the early stages of the system design, as one of the most distinct characteristic experienced when in close proximity to an explosion. Unfortunately, this sensor was quickly cut from the design. The rational came from the fact that there was no practical way to create a drastic change in pressure to test the system with. The cost of the sensor was also proving to be outside the reach of the planned budget.

It has been noted that the expectation of this project is not to produce a military grade system, but to build a system at a smaller scale that could be related to the needs of the military. The specification requirements of the various sensors will only be required to meet the test requirements of this project, and not the intensity of an actual IED strike or rollover incident. With that said, the design of this system has incorporated sensors which will be sufficient to the events that are laid out in the test plan. The sensors incorporated in the design at this point in time are:

- Accelerometer
- Thermo sensor
- Photodiode

For simplicity of physically mounting and testing, the entire sensor will be mounted to a single circuit board. Mounting the sensors on one PCB should help with the team’s concern of lack of space available on the Power Wheels vehicle.

The objective of this design decision to mount all the sensors on one PCB also helps keep TUV-ADDS true to its objective of a system that does not bring burden to its users. Compacting all the sensors to one board will also simplify the testing of this module because it will eliminate the struggle to keep the same applied events to separate boards when testing specific characteristics.
It is likely that the system will require two or more sensors to reach a predetermine threshold before a distress event would be recognized by the system. Therefore it is crucial that the sensors that are selected can monitor a range of values for the characteristic they measure. For example, to detect an IED strike, the sensors would have to detect a predetermined change in motion, rise in temperature, and percent increase in luminance before an event could be categorized as a “distress event” by the system. This is the failsafe the team has built into the system to help minimize and isolate any false positive signals, which is another goal of the system design so the sensors must have a wide range of values that they are able to read to meet.

4.2.1 SENSOR MCU

The initial design of the sensor MCU board contains 4 basic components; these are communications, sensor input, data logging, and the micro-controller. The sensor unit must be fast enough to store, process, and report data from several sensors to a master controller as well as respond and make requests to both the master controller and other sensor units. The communication between the sensor MCU and other networked controllers will take place using the MilCAN communications protocol.

The communication portion of the design can be broken down into two further design components: the software implementation, and the hardware implementation. The first consideration for software implementation is whether or not to have the CAN capability integrated into the sensor micro-controller, or to use a separate chip to handle all of the CAN communication. The issues with using a single MCU to handle both CAN protocols and sensor data is that the controller itself would have to be more powerful and costly than otherwise necessary in order to handle both tasks. It would also not allow much flexibility if a major design revision was required for the CAN or MCU. Due to these drawbacks it will be assumed that the design will use a separate microcontroller and CAN controller, though further research may find a suitable single chip for these purposes. The hardware portion of the CAN protocol must be handled by a CAN transceiver in order to ensure that all ISO 11898 protocol standards are met. The major considerations for this device are size, cost, and operating voltage.

During normal operation of the sensor microcontroller the unit must take readings from several external sensors. It must then compare this data with preprogramed nominal values and if it detects a value outside of normal range it will flag the current timestamp as a possible event and gather data for the next 3 seconds. While collecting the data during this time period the controller will write it to a removable external memory device with a timestamp. Once the 3 seconds have elapsed the controller with observe data on the other sensors around the time of the flagged event. If it determines that the data is showing abnormal data it will flag the captured chunk of data to be transferred via the CAN-bus to the main processor for further interpolation and necessary action. The microcontroller selected must be able to handle all of these functions with little to no loss of critical data while still being simple enough to keep the cost down as well as reduce the risk of a malfunction caused by interfering controller features that are not being utilized.

Firstly, the external flash memory must be able to capture at least 1 minute worth of data while having a high enough transfer speed as to prevent unnecessary system slowdown during a write cycle. Optimally it will communicate with the sensor microcontroller via the SPI USART protocol as it is already a requirement for this communication to be implemented on the microcontroller. The memory must also be easily removable from the system and be able to simply interface with a PC through commercially available adaptors. The device however does not require hot swap capabilities as the optimal placement of the sensor device will be inaccessible during normal usage. Durability and read/write lifetime will also be a factor in the selection of this device.

Figure 4 shows the basic design of the sensor control unit. The main component of the device is the microcontroller, which communicates with the removable memory device and the CAN controller via the USART SPI protocol. During a normal cycle the microcontroller will read data from the sensors, analyze them for any events that would need to be sent to the main controller for further processing and write that data to a removable memory device. If the controller determines that a sensor input is outside of user defined nominal readings it will forward the previous 10 seconds worth of data from the Flash memory to the main processor via the CAN controller communicating with the MilCAN protocol. The CAN controller is required to implement the software level of CAN communication, while the CAN transceiver is required to ensure the physical CAN layer is within the ISO 11898-2 standard [1]

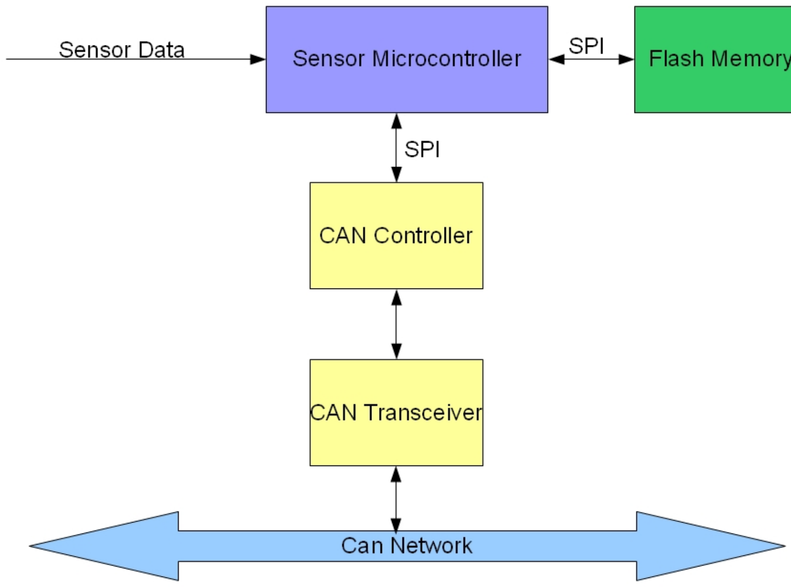


Figure 4: Sensor MCU and CAN bus

Micro-Controller Requirements:

- External interrupts
- 8 MHz processing Speed
- SPI & I2C USART Capabilities
- 6V or less operating voltage
- 3 or more Analog inputs
- Low Cost

The first controller design choice is whether to use an FPGA or a microcontroller to handle the sensor processing applications. Basic research in both cost and the development cycle for an FPGA shows that the high startup cost for learning to program an FPGA board as well as the group's lack of experience with FPGA development would be major obstacles to overcome and could bottleneck development to a point where other features would have to get cut. The benefits of an FPGA system would be parallel processing as well as very high processing speeds.[2] The development team however already has some experience coding for microcontrollers and also own very basic development boards. Keeping in mind the relatively low cost (compared to FPGA's) for basic microcontrollers and it becomes clear that while the FPGA's advantages would be and enhancement to the project the hindrances far outweigh any possible gains. With this knowledge further research will be given solely to microcontrollers as the sensor units embedded processor.

The lower end microcontrollers will be the first to be examined in the research phase as they will most likely be powerful enough to handle the requirements of the sensor MCU while remaining low cost in both individual chip cost as well as the development tools. The two processors in this category that stood out were the TI Msp430g2231 and the Atmel Atmega328P. The largest selling point of these chips is a wide array of community developed tools and available support in the event of issues in the development cycle. The Atmega328P has the added feature of a more user friendly programming language that is based largely off of the C / C++ languages[3]. This will allow the group to move more or less directly to the prototyping stage of the project rather than have to spend several weeks learning how code on the target board. While the Msp430 is a very functional board and has its own large user base and support community it is nowhere near as user friendly or well documented as the Atmega328P [4]. Another major detractor from the MSP430 is the very small program space. At only 2k it is very possible that there would not be enough room on the chip for the necessary code. The different features offered by each chip are shown below in Table 6. [5] [6]

Microcontroller	MSP430g2231	Atmega328P
Manufacturer	TI	Atmel
Cost	$2.17	$4.98
Clock Speed	16MHz	20Mhz
I/O Pins	10	23
Analog pins	8	6
Operating Voltage	1.8 – 3.6 Volts	1.8 – 5.5 Volts
Package	14 Pin PDIP	28 Pin PDIP
Communication	1 USART(SPI/IC2)	1 USART (SPI, I2C)
On-Board Memory	2k Flash	32k Flash
External Interrupts	2	2
Architecture	16 Bit RISC	8 Bit RISC
Programming (Hard.)	ICSP	ICSP
Programming (Soft.)	C, C++	Processing
CAN Compatibility	None	None

Table 6: Main MCU Comparison Chart

Next mid-end microcontrollers were also considered for this design. The mid end controllers tend not to have more processing power than the lower end models though they do tend to have more I/O pins as well as larger program memory. Once again the amount of community support was the deciding factor as to why these two boards were selected for review. The TI Msp430f6638 corrects many of the g2231's faults. It has a larger program memory, more I/0 pins, and a faster clock speed. It also includes a hardware multiplier which is a useful feature that the g2231 lacked. On the other hand the Atmega2560 uses the same code and programming environment as the Atmega328P and also has the same excellent documentation and community support. For an added bonus both mid-end chips have multiple USART ports which would prevent the need for a multiplexer for communication with sensors and the CAN controller. While the mid-end chips would serve our purpose well they are overkill for our project. Also they come in a less durable package than the PDIP offered by the low end controllers. Almost 90% of the pins on either of these controllers would be unused and many features would be unused. This adds to the cost while adding no appreciable performance gain. The Atmega328P remains the best controller for the job reviewed so far. Table 7 shows the comparison between the mid end micro-controllers, we can see that both chip-sets are very similar in features though the Atmega2560 is double in price it is offset by the community support available. [7] [8]

Microcontroller	Msp430f6638	Atmega2560
Manufacturer	TI	Atmel
Cost	$9.24	$17.97
Clock Speed	20Mhz	16Mhz
I/O Pins	74	86
Analog pins	12	16
Operating Voltage	1.8-3.6 Volts	4.5-5.5 Volts
Package	100 PZ	100-lead TQFP
Communication	2 USART	4 USART (SPI,I2C)
On-Board Memory	256K Flash	256k Flash
External Interrupts	4	8
Architecture	16 Bit RISC	8 Bit RISC
Programming (Hard.)	ICSP	ICSP
Programming (Soft.)	C / C++	Processing
CAN Compatibility	None	None

Table 7: Sensor MCU Comparison Chart

Finally, microcontrollers with on board CAN2.0 support were considered. The benefit of on board CAN is that the sensor MCU board can be much smaller, interfacing two different controllers (the sensor MCU and the CAN MCU) is no longer a problem and in general it will allow the CAN functions to run in parallel with other functions. Major downsides to utilizing this type of chip is that there is very little available documentation for controllers of this type because they are more specialized and thus have a smaller installed user base and therefore less community support. The first chip of this type considered is the Atmel AT90CAN32 the architecture of this chip is very similar to the Atmega line and the code written for the Atmegas can with time be migrated over to the AT90CAN32. This would still consume valuable development time and would prevent any progress on sensor interface as well as the CAN-bus until a reliable method could be developed for migrating the code. This chip however fits the basic requirements for the sensor unit very well with multiple external interrupts and multiple USART communications as well as several analog input pins. If later in the development cycle it is decided to migrate to an integrated CAN MCU for the sensor unit this would be an excellent choice. The AT91SAM9263 is also considered for this project. This controller would fall into the high end microcontroller category with 2 USB 2.0 ports, Ethernet capabilities, based around an ARM core, and a high price tag. While this controller would definitely achieve the requirements of the sensor MCU it would be overkill for the project. Table 8 shows a comparison between the Atmel AT90CAN32 and the AT91SAM9263 while the AT90CAN32 shares very similar specs to all of the previously considered controllers it is clear that the AT91SAM9262 has far more features and power than this design requires. [9] [10]

Microcontroller	AT90CAN32	AT91SAM9263
Manufacturer	Atmel	Atmel
Cost	$6.97	$29.28
Clock Speed	16 Mhz	240 Mhz
I/O Pins	54	160
Analog pins	7	4
Operating Voltage	2.7-5.5 Volts	1.08-1.32 Volts
Package	64-lead TQFP	324-ball LFBGA
Communication	2 USART (SPI)	2 USB 2.0, 4 USART
On-Board Memory	32k Flash, 1k EEPROM	96Kbytes SRAM
External Interrupts	8	160
Architecture	16 Bit RISC	ARM926
Programming (Hard.)	ICSP	ICSP
Programming (Soft.)	C , C++	C , C++
CAN Compatibility	2.0A / 2.0B	2.0A / 2.0B

Table 8: CAN Comparison Chart

Having reviewed many controller options the Atmel ATMega328P was selected for the relative ease of coding, available examples, and excellent community support. There are several available examples of similar projects for the

ATMega328P which will allow for a more rapid design cycle as well being able to quickly prototype the sensor board. Along with all of these helpful features the ATMega also has two open source header files that simplify the interface with the microchip MCP2515 CAN controller (CAN.h and SPI.h). If it is determined that this chip can serve the necessary CAN functions for the system it will significantly speed up prototyping as well as allow more time to be used for optimization of the packet contents and troubleshooting data interface issues between the sensor units and the main processor. If the CAN interface required cannot be achieved with the Atmega328P, the AT90CAN32 will be used as the main microcontroller.

CAN Controller Requirements

- 1 interrupt output pin
- 1 SPI or I2C USART serial communications
- Implements CAN 2.0 A/B

The main factors to be considered for CAN controller selection will be USART capabilities (SPI preferred), simple to no programming necessary for the CAN controller, full CAN2.0 support to the ISO 11898-2 standards. Lesser factors will include package, cost, sampling, manufacturer documentation, available application examples. The first chip looked at is the MCP2515; this CAN chip is preprogrammed and is entirely controlled via SPI register modification. Another major bonus is open source header files available for the ATMega328P which combined with the SPI header files makes interfacing the two chips much more simple. The second CAN controller researched is the Phillips SJA1000. This chip functions very similar to the MCP2515, except rather than USART communication this controller requires 15 I/O pins to select and control the various registers used to transmit CAN messages. While this would not be a major setback, the 15 pins could be called from only a few sensor MCU pins and a shift register, without existing header files for the Atmega328P this chip becomes a less than ideal solution for the CAN controller though it will remain an option in the event the MCP2515 has unforeseen interfacing issues.
Table 9 below shows the differences between the two chips that were researched above. [11] [12]

Can Controller	MCP2515	SJA1000
Manufacturer	Microchip	Phillips
Cost	$1.98	$3.27
Clock Speed	16 Mhz	24 Mhz
Operating Voltage	2.7-5.5 Volts	4.5-5.5 Volts
Package	18 Pin PDIP	28 Pin PDIP
Communication	1 USART (SPI)	15 Bit proprietary I/O
Programming (Hard.)	None	None
Programming (Soft.)	None	None
CAN Compatibility	2.0A/2.0B	2.0A/2.0B

Table 9: CAN Controller Comparison Chart

The MCP2515 Can controller was selected due to the existence of the open source software library that allows it to be more easily interfaced with the Atmega328P [13], its relatively low cost, and it meets all of the necessary criteria to adhere to both the MilCAN standards as well as the project requirements. Application notes from Microchip indicate that an optimal CAN transceiver to use in conjunction with the MCP2515 is the MCP2551. With this in mind the MCP2551 will be used in this design [14].

There are several ways to program the Atmel ATmega, of which two of the most common ways will be utilized for this project. The first, for initial prototyping will be done via the Arduino bootloader [15]. This code allows the microcontroller to be programed with only serial input with a computer and can be done with a USB to Serial converter or on the proprietary Arduino board. The second method that will be utilized is a JTAG programmer. Pins on the final printed circuit board will be made available to program the chip without needing either the preinstalled bootloader (saving valuable memory-space) and eliminating the need for an external serial connection on the final PCB.

The external data logging will be done via an external MicroSD flash card formatted to the FAT16 file system. This is done because of the existence of a C++ library utilizing the SPI library to communicate with the MicroSD flash card [16]. This supports short 8.3 filenames and will greatly simplify organizing the memory space, interfacing with a computer, clearing the data and overall project organization and simplicity. Due to the simple nature of the data being collected, 1gb, (the smallest size commonly available), will be more than adequate to store large quantities of sensor information. It also is available in an extremely small package as seen below in Figure 5.

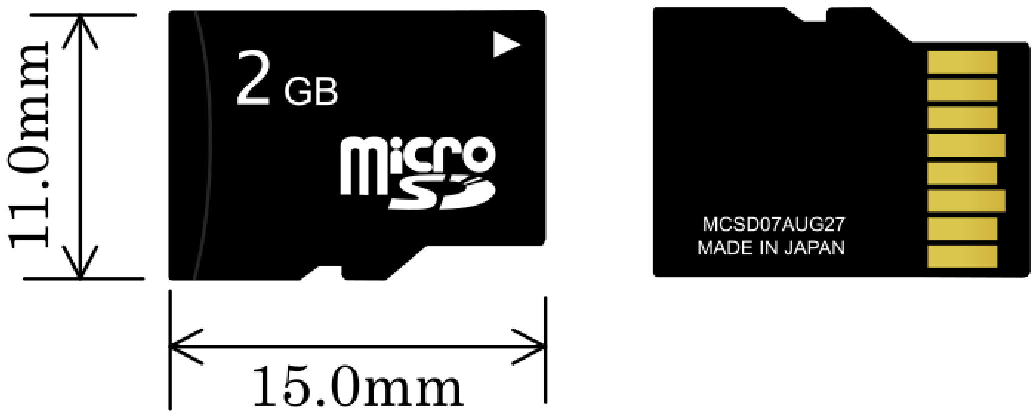


Figure 5: MicroSD Card with dimensions
(Wikimedia Commons)

With multiple devices communicating over the SPI protocol, chip select will need to be taken into account to avoid collision errors on the ATMega328P's single USART port. Due to the relatively small number of devices being used for this purpose and the excess of I/O pins the ATMega will handle chip select with no external interface needed. Though, if later in the project more SPI devices are added to the sensor unit a decoder can be used to free up I/O pins. Figure 6 shows an example of this USART connection.

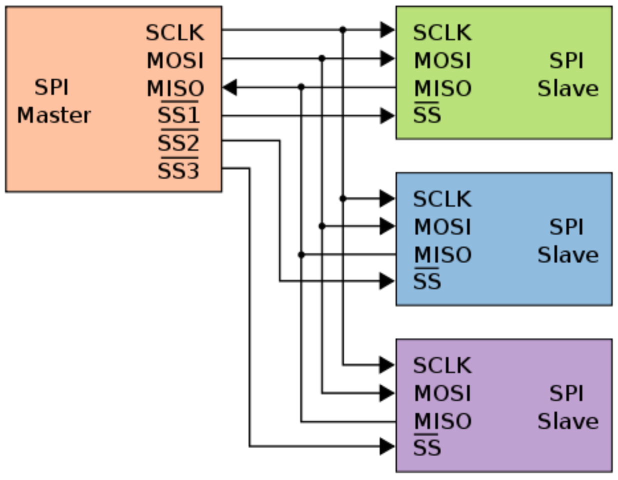


Figure 6: Chip Select SPI Connection
(Wikimedia Commons)

4.2.2 ACCELEROMETER

An accelerometer was chosen to detect any significant jarring, or change in motion, which would make this component well suited for aiding in recognizing an explosion such as an IED strike. The team understands from conducted research, when an IED strike occurs there is significant propulsion of the vehicle in a specific direction in a very short span of time. Axes, accuracy, range, bandwidth, power consumption, dimensions, and output type are the specifications of a part that will be considered when selecting the best accelerometer for TUV-ADDS.

A three axis: X-Y-Z, accelerometer would be most ideally suited because the direction of the change in motion will typically be unknown. The direction of propulsion of the vehicle is an unknown factor because there would be no way of knowing at what direction the IED was directed and at what angle the vehicle would approach the IED. The ability to measure values along all three axes would ensure good qualitative measurements regardless of the direction of an explosion.

The sensitivity of the accelerometer is not the highest priority when it comes to the characteristics of the device. The IED strike would produce an explosion that would place significant force on the vehicle, which would make an accelerometer with very precise accuracy would be unnecessary for the scope of this project. Understanding the effects that would be placed on the vehicle, our limitations for resources, and the extent of force the vehicle will undergo in testing, a low G accelerometer is sufficient. A typical low sensitivity sensor appears to be ± 2 to ±4g's, which is usually controlled by the user. This typically can be accomplished by the pin lay out per the manufactures instructions.

Bandwidth is a crucial requirement pertaining to the accelerometer. The unpredictable nature of the events that the system is monitoring for makes the need for a high bandwidth accelerometer sensor. If the bandwidth is too low, there is a risk of missing an event, which would cause the system to not meet the basic goals it was designed for. Having looked at the accelerometers readily available for purchasing, a bandwidth of greater than one or two hundred Hertz would be sufficient.

Power consumption of the accelerometer, as well as the other sensors, would ideally be as low as possible. The system will be operating off a fixed 6-volt battery, and since the system design will consist of an array of sensors and micro-controllers, preservation of the operational lifespan of the system before recharging is necessary. Keeping the power consumption of the components low is an effort to make our design as efficient as possible.

All of the sensors will be placed on a circuit board together since all the sensors will connect directly to the sensor MUC that will control and monitor all the sensors using a digital interface. Research has resulted in multiple types of common digital interface approach methods such as SPI and I2C. In some instances the accelerometer can incorporate either method, by the user's choice. The data sheets for individual components indicate which particular pins are used for each method of interfaced used. Once again, in this design we'll need to make sure the sensors and sensor MCU have compatible interfaces.

The dimensions of the average accelerometer are quite small for instance, Bosch manufacturers the BMA220 which had the dimensions of 2mm x 2mm x 0.98mm. Understanding this, the team is currently looking into way to make their lack of size more manageable for our prototype. One option is an accelerometer that is pre-mounded on a breakout board. Then this break out board could be incorporated into the board that is planned to contain the remaining sensors. However, this will drive up the cost significantly which may lead the team to opt to deal with the small nature of the accelerometer and adapt traditional surface mounting techniques to accomplish this task.

We plan on connecting the accelerometer to a MCU, which means a digital output would be most desired. However, should we find an analog output accelerometer that meets our need and cost requirements we do have the capability to use an AD converter to the sensor usable to the MCU.

4.2.3 TEMPERATURE

A temperature sensor also seemed an effective way to detect an IED strike. It's understood that an explosion from an IED, would release a significant amount of energy. This means heat would be a sensible way to gauge that an explosion took place near the vehicle. There are an abundance of types of temperature sensors and the team researched many such as: thermocouples, thermistor, IC CMOS and purely just for insight non-contact infrared styles. While the non-contact IR types were the most interesting, we failed to foresee its successful implementation into our design. Of the remaining choices, the IC type proved most appealing to the team. Through research of data sheets, IC type temperature sensors seem to have the most linear voltage output through the whole range of specified temperatures. With a thermistor we would have to assure a constant unvarying current source, as the voltage output would be a function of the resistance of the thermistor. So, at this point in the design an IC type of temperature sensor will be focused on to reduce the risk of a false trigger of an event if for some reason there is a chance in the current.

As with the accelerometer, the dimensions tend to be quite small. The dimensions are less that 3mm x 3mm, so use of a breakout board may be considered, depending upon cost and availability. These dimensions are again proving to be beneficial to the team because the sensors will be mounted on the same PCB.

The prototype design will be tested with something more along the lines of a heat gun or hair dryer. After some research it was concluded by the team that heat at about 170°F was produced from an 1800W hair dryer. Through these findings, it was concluded that the sensitivity of the temperature sensor is not critically important. This is because the system will be looking for an extreme change in the temperature not necessarily an extremely high temperature that would be outside any normal operating conditions. Likewise our design, as implemented for this class, will only call for a temperature sensor to operate near 70 °C maximum.

Many of the sensors that were researched by the team, such as National Semiconductor's LM94021, operate in the analogue domain. Published data regarding the time that it would take the sensor to report the change in temperature could not be found. This would suggest the change in output voltage was continuous, with respect to the change in temperature, in the IC for the analog configuration. The digital temperature sensors such as Texas Instruments TMP100 seem to have a conversion rate of 75ms (max) for a 9-bit resolution. The time for the conversations appeared to be the most intensive and time consuming part of the cycle as the all other parameters could be measured in nanoseconds.

As stated before, we plan to feed the data from all the sensors to the sensor MCU. Therefore, in an effort to reduce complexity we would like to choose the digital output temperature sensor. This will avoid the use of having to include an additional A/D converter. Also, stated earlier, It is idealistic that the power consumption of the temperature sensor be as low as possible to preserve the operational lifespan of the system before recharging is necessary to keep the power low in an effort to make our design as efficient as possible.

4.2.4FLASH/PHOTODIODES

A photodiode has also been selected to encompass our design, as we could consider a luminance a characteristic output of IED explosions. Most often with any explosion there is an admittance of light as one of the forms of energy released in the exothermic reaction. In an effort to attempt to measure this luminance we will encompass a photodiode into our design to detect and light or drastic change in lighting conditions.

The realm of sensor we are interested in will have a linear output with respect to the irradiance sensitivity. The particular wavelength is not necessarily a concern since there is likely a whole range of wavelengths emitted during an explosion. However, for the scope of this project and the corresponding test procedure; we will implement a sensor that can detect the visible portion of the electromagnetic spectrum. This would allow something as simple as a flashlight to trigger the desired test condition.

The photodiodes that are being considered for this design are analog type since they are the predominate type. The output of the analog type photodiodes can be one of two types; current output or voltage output. The team decided that through research, the voltage output type photodiode would be easier to incorporate into the system design. The greater the intensity of the light source applied to the photodiode, the greater the output voltage.

This sensor idealistically will be place on a circuit board with the other sensors incorporated in our design so they are all located close to the sensor MCU since they will all be connected to it. It is also ideal that the power consumption of the temperature sensor be as low as possible to make our design as efficient as possible.

4.3 MODULE B – MAIN CONTROL UNIT

4.3.1 MAIN PROCESSOR

The importance of a suitable processor is evident in knowing the tasks it will be responsible for. Some of these include: GPS, wireless RF (radio frequency), Sensor data, and probably most important, decision making. These tasks require a processor that can handle several tasks simultaneously without fault and with efficiency. With the given requirements, our research has been able to narrow down to three possible solutions: Texas Instruments' (TI) Stellaris ARM processor, Microchip's PIC24 processor, and Atmel's ATmega processor. The processor will have interface with many subsystems using several protocols including UART and CAN. Figure 7 below displays how the processor will interact with all of the other TUV-ADDS systems.

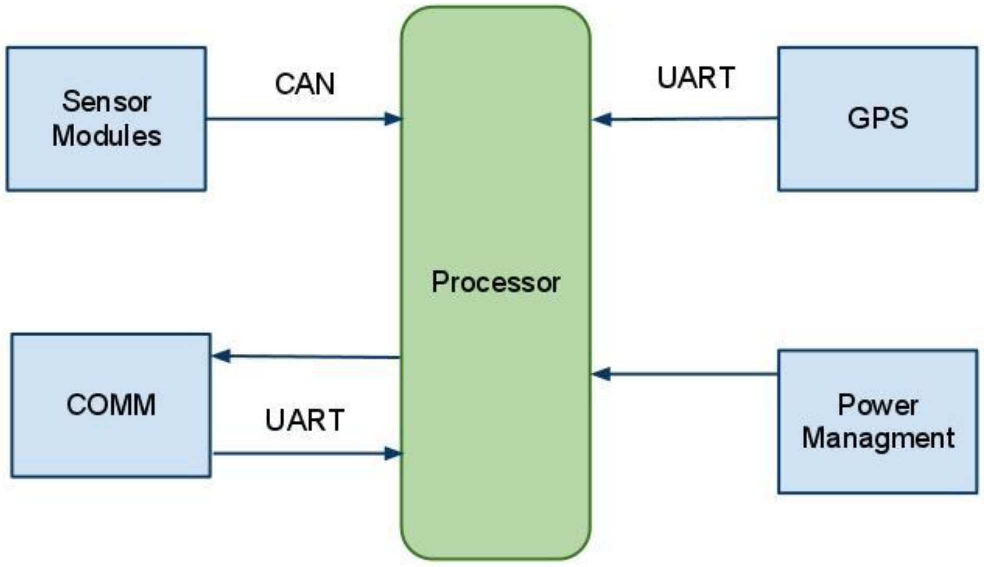


Figure 7: Main Controller Functional Diagram

The magnitude of the performance required of the processor means that strict requirements will be set for selecting the appropriate product. A list below (Table 10) compares the vital requirements between the three semiconductor products. After a discussion reviewing the specifications available, the TUV-ADDS Main Control System will utilize an Atmel ATmega2560 processor. This processor is used in the Arduino Mega (Figure 8) development board. This decision was based on two factors; One, the processor meets all specification requirements; Two, prototype development will be simplified with the Arduino coding enviorment and physical development platform. Several group members have experience with developing complex systems using the Arduino development board.

Specification	TI Stellaris ARM	Microchips PIC24	Atmel Atmega
Frequency	50MHz	10MHz	16MHz
Flash Memory	256Kbytes	32Kbytes	256Kbytes
RAM	64Kbytes	2Kbytes	8Kbytes
ROM	None	None	4Kbytes
Software Interrupts	Yes	Yes	Yes
Hardware Interrupts	Yes	Yes	Yes
Total Interrupts	28	5	8
UART/USART	2 UART	2 UART	4 UART
CAN Parts	3	2	None
Supply	3.3V	3.6V	5.5V
WatchDog Timer	Yes	Yes	Yes
I/O Pins	46	35	54
Language	Ansi-C	Assembly	C derivative
Development Software	CodeComposer	MPLab	Arduino IDE

Table 10: MCU Comparison Chart

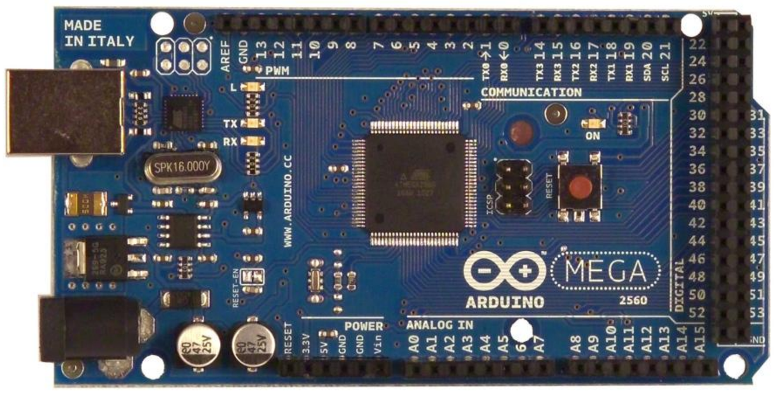


Figure 8: Arduino Development Board
(Print with Permission from Arduino)

The CPU frequency and available Flash memory play a vital role in the efficiency of the processor. The processor must be able to run through a given complex task, such as parsing data, as quickly as possible. The ATmega2560 series has an operational CPU 16MHz [17]. This frequency will provide sufficient processing power for the functions to be implemented on the device. Also a large amount of Flash memory is required so that data from many sources can be temporarily stored. The ATmega series provides 256KBytes flash memory [17]. The sources include the senor modules, GPS, and the COM system. Data from the sensor modules must then be evaluated to check for user designated thresholds. The GPS and COMM data will be bundled into large data structures, which must then be parsed. The parsed data will then be temporarily stored then be evaluated to user defined requirements.

The way that data is transmitted to and from the processor is also an issue that requires some detailed research. Appropriate protocols must be followed so that the data can be sent safely and without failure or corruption. The COM and GPS unit will be embedded on the same PCB as the processor, so basic UART/USART (Universal Synchronous/Asynchronous Receiver/Transmitter) protocols will be used. The ATmega contains four fully programmable [17]. For the components (i.e. the Sensor Module and the Drive Unit) that are not physically connected to the processor's PCB, a CAN (Controller Area Network) bus will be used. The Atmega does not directly support a CAN bus. Research and Development is being done to construct an independent CAN line. The CAN capability is vital knowing the environment that the TUV-ADDS system will be operating in. According to National Instruments "Wires that have to pass through the door of a vehicle are low-speed/fault-tolerant in light of the stress that is inherent to opening and closing a door. Also, in situations where an advanced level of security is desired, such as with brake lights, low-speed/fault-tolerant CAN offers a solution."[18]. Additional information on the research and development of both the independent CAN bus as well as USART interaction can be seen later in this report.

Once the processor receives a signal from either the CAN or UART line, a hardware interrupt service will need to be activated. This will allow the data to be dealt with in the most efficient process. The ATmega provides eight hardware interrupts as well as software interrupts [17]. The incoming data could also trigger a software interrupt. If data received needs to be parsed, then an Interrupt Service Routine (ISR) will be run to do so. Once this ISR is complete then the system can either run the next routine in the ISR stack, or return control to the main system. The ATmega will be able to dynamically reprioritize an interrupt meaning, if desirable, the system can change the order of the stack to allow a high-priority routine to run next in the chain.

The project may require the use of other simple devices to interact with the processor. An example of this would be a simple Light-Emitting Diode (LED) to blink when the GPS is initializing triangulation. For this example, the LED would need to be attached to a General Purpose Input/Output (GPIO) port. The 54 GPIO ports available are 5volt tolerant for both input and output [18].

4.3.2COMMUNICATION

The wireless components utilized with TUV-ADDS is vitally important to support the overall purpose of the entire system. The major aspect of the system is that it can wirelessly transmit data from the vehicle to the command center. The basic design of this system is rather simple since two Radio Frequency (RF) devices can generally communicate with each other without much additional design required. The basic design that the wireless system will incorporate is displayed below with Figure 9.

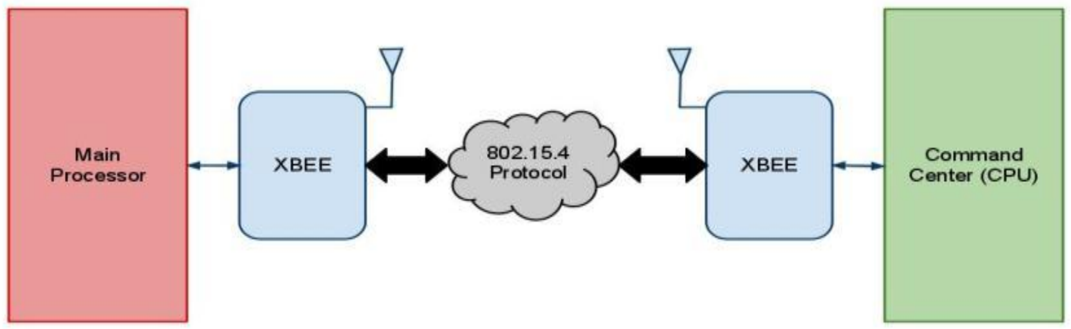


Figure 9: General TUV-ADDS Wireless System

There is an unimaginable number of different wireless systems available for commercial use on the internet. Some are radio based, some are RFID, and others are infrared. Because of the large distance we want the on-board system to be from the Command Center, the only appropriate option would to go with a Radio Frequency (RF) transceiver. After referencing a few of the Do-It-Yourself (DIY) sites for electronic hobbyist, such as Adafruit Industries (ref), the XBee was considered a top product for projects, such as the TUV-ADDS system.

The XBee is a ZigBee Alliance Z-Stack© derivate developed by Digi, Inc. The XBee modules meet IEEE 802.15.4 standards [19]. Both the XBee and the XBee-PRO module consume a minimal amount of power compared to other RF devices. The PRO module does consume more power but it has a much further range compared to the basic module. Our project will utilize the extended range of the Pro module. Table 11 below compares the two XBee models.

Parameter	XBee	XBee - PRO
Meets IEEE 802.15.4	Yes	Yes
Operation Frequency	2.4 GHz	2.4 GHz
Indoor Range	100 feet	300 feet
Outdoor Range	300 feet	1 mile
Transmit Power	1 mW	63 mW
Receiver Sensitivity	-92 dBm	-100 bDm
RF Data Rate	250Kbps	250Kbps
TX Peak Current	45 mA	250 mA
RX Peak Current	50 mA	55 mA
Power Down Current	< 10 mA	< 10 mA
Serial Interface Baud	1200bps – 250Kbps	1200bps – 250Kbps
Supply Voltage	2.8 – 3.4 V	2.8 – 3.4 V
Operation Temperature	-40 - 85° C	-40 - 85° C
Number of Channels	16 Direct	12 Direct

Table 11: Wireless Device Comparison Chart

There are several different antenna options for the PRO module. This includes Integrated Whip (Figure 10), Chip (Figure 11), or U.FL Connector. The U.FL requires additional components so it will not be considered because of the complexity of the antennae needed, which also drives up cost. The Whip is a short wire that protrudes upward from the XBee module. This may be problematic because of the nature of the environment the system will be in. The chip is the only option left. This is a small square IC chip that is soldered directly to the module. This chip is capable of the same characteristics as the whip antennae in a smaller package. Some users on various online forums claim that the whip model has a greater range, but the manufactures have not released any data proving this claim.

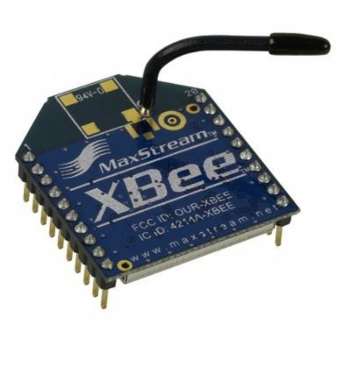


Figure 10: XBee with Integrated Whip
(Permission from Digikey)

Figure 11: XBee – PRO with Chip
(Permission from Digikey)

There are many different types of XBee adapters which aid in the interface of the module with other devices. Each adapter focuses on certain parameters such as power management, data transformation (such as UART to USB). Adafruit Industries makes a simple adapter which steps down a standard five volts to the XBee supply voltage of 3.3V (Figure 12). The adapter also has the data in/ data out lines conveniently located. Adafruit Industries has given us permission to embed the schematic of this adapter directly into our system design. The modules must also be configured to have matching network parameters; if these parameters do not match then the units will not communicate. One way to easily upload the Personal Area Network Identification (PAN ID) number is by using the XCTU (see Figure 13 below) software available for free from DigiKey.

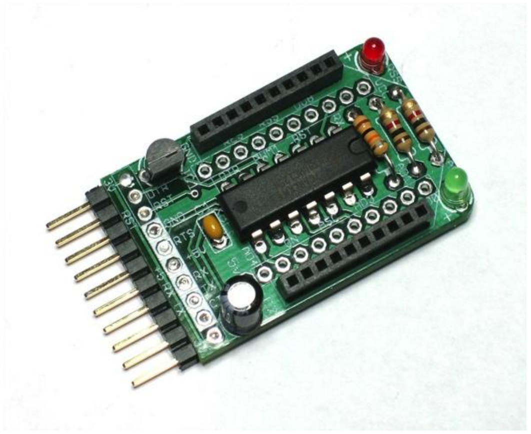
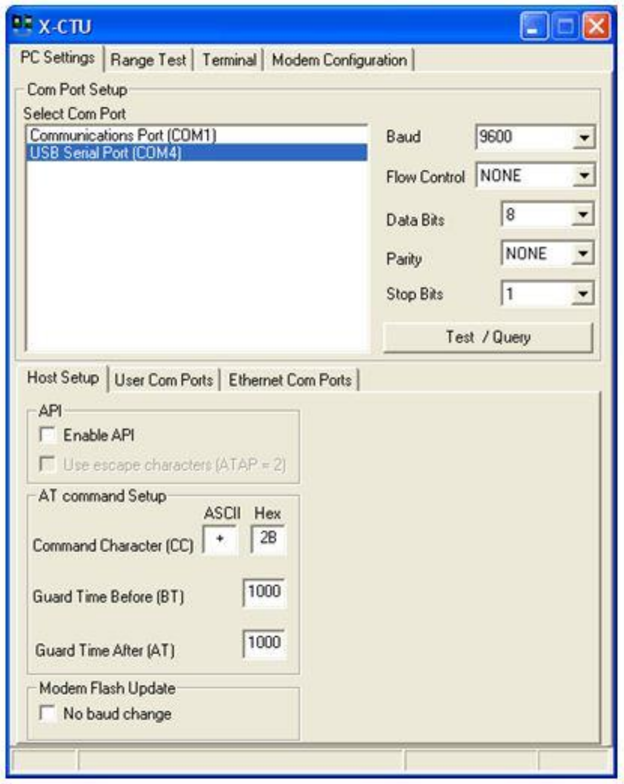


Figure 12: XBee Adapter
(Printed with Permission from Adafruit Industries)



Com Port Setup

Select Com Port

Communications Port (COM1)

USB Serial Port (COM4)

Baud	9600
Flow Control	NONE
Data Bits	8
Parity	NONE
Stop Bits	1

API

☐ Enable API

☐ Use escape characters (ATAP = 2)

AT command Setup

	ASCII	Hex
Command Character (CC)	+	2B
Guard Time Before (BT)		1000
Guard Time After (AT)		1000

Modem Flash Update

☐ No baud change

Figure 13: Picture of the XCTU software
(from Digikey)

4.3.3 GPS

The Global Positioning System (GPS) component of the TUV-ADDS system is arguably one of the most vital. The idea of the TUV-ADDS is that the Command Center will receive and accurate position of the vehicle. To do this, the system must have a high-grade GPS unit. There are many similarities between the TUV-ADDS and a previous UCF Senior Design project, named ARMORD [20]. The ARMORD team used a GPS unit from SparkFun. This GPS unit that team used is no longer available from that vendor. However, the D2523T Helical GPS Receiver (see Figure 14) is comparable unit.

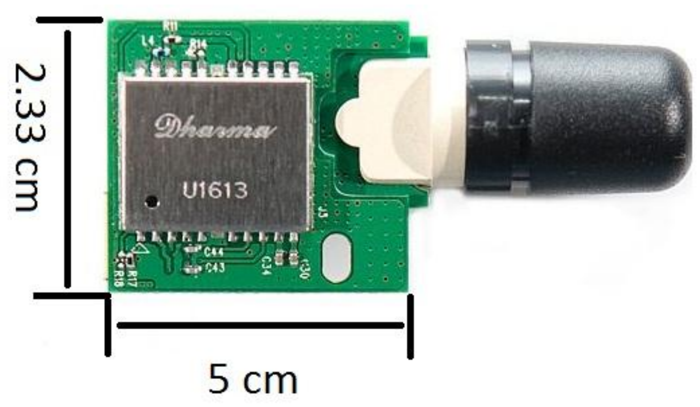


Figure 14: Global Positioning System (GPS) Component
(Printed with Permission from SparkFun)

The GPS will be required to be accurate and get readings as fast as possible. The Helical system is a 50 channel with standard civilian L1, Clear/Acquisition (C/A) code based on Galileo [21]. The C/A for this unit runs at a 1.023MHz chip rate. The unit allows a horizontal Position Accuracy of near 2.0m using Satellite-based Augmentation System (SBAS). The SBAS uses additional satellites in a wide-range area to recalculate the triangulation. This also allows a max velocity of 515 m/s (1151 mi/hr) [21] which is greatly above what they system will require.

The Helical unit will communicate with the main processor (an ARM processor) through UART (TTL). Both system have UART capabilities so attaching the GPS unit to the processor will be as easy as connecting the output of the GPS to a GPIO port on the processor.

4.3.4 GYROSCOPE

One of the TUV-ADDS requirements is that the system will detect vehicle rollover. This was originally thought to be easily detected using a gyroscope (gyro). If the gyro is offset by nearly 180° of a set threshold for a long duration of time (30 seconds or more), then it can be assumed that the vehicle has rolled over. Because it will not be possible to detect which direction the vehicle will roll over using only a single axis gyroscope, meaning the system will have to implement a 3-axis gyroscope (seen below in Figure 15). It was later decided to just use the accelerometer (Figure15) for the detection of a rollover.

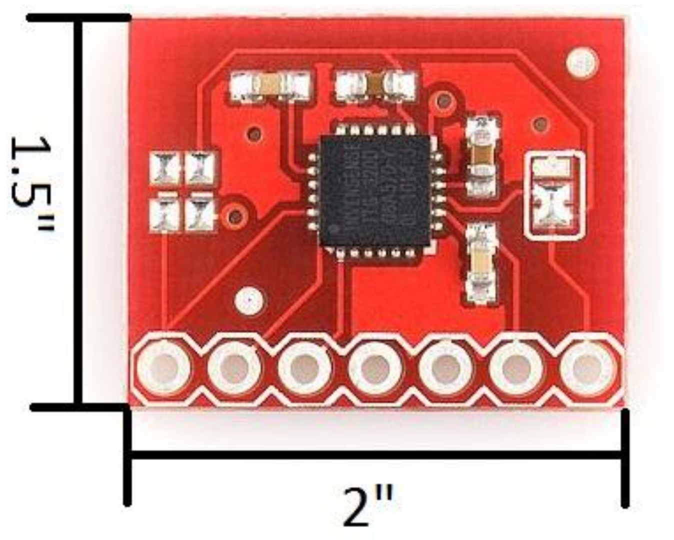


Figure 15: Accelerometer Component
(Printed with permission from SparkFun)

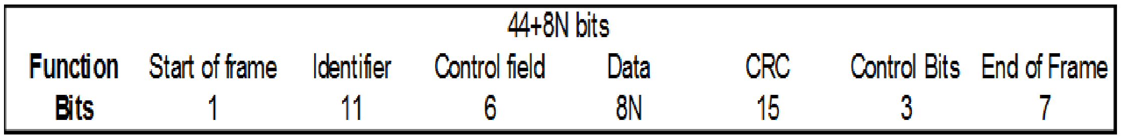
4.4 MODULE C – CAN

TUV-ADDS will be utilizing the MilCAN communication standard as laid out by the MilCAN Working Group. This deterministic protocol is designed to be implemented on the ISO 11898-2 CAN-bus standard physical architecture also known as CAN2.0. Milcan is an asynchronous standard that can transmit at speeds of up to 1Mbit per second. The protocol uses bit stuffing to allow such high data transfer without use of a common clock. Due to the deterministic nature of MilCAN (not CAN-bus in general) the protocol allows for the user to know precisely the amount of time it will take for a message to reach its destination regardless of interfering factors. We will be implementing Milcan A which is based on the ISO11898-2 standard of the general communications protocol Can2.0 [22].
CAN-bus 2.0 is based on the broadcast communication method. This means that the standard describes message transmission and contents rather than the physical transmitter itself. There are many devices and mediums capable of implementing the CAN protocols in general though they must communicate using the same message format in order to be considered CAN. Firstly, every message has its own unique identifier rather than the node. This allows the importance of the message determine the outcome of a message priority check, furthermore referred to as message arbitration. Message priority is determined by dominant bits in the ID, in the CAN protocol a dominant bit is logic zero, this means that the lowest message ID will be the first transmitted over the bus. Message transmission begins thus, there is a start of frame (SOF) Dominant bit sent when

the transmitter detects zero bus traffic, the message then begins to transmit its unique identifier, 11 bits for CAN2.0A and 29 Bits for CAN2.0B, and It also monitors the bus at this time. If at any time during the ID transmission the transmitter sends a recessive bit (logic 1) but detects a dominant bit (logic 0) on the bus it determines another unit with a higher message priority is attempting to transmit and end its send attempt and waits for the bus to bus to become free again to transmit. This is known as Carrier sense multiple access with collision detection or CSMA/CD [23].

Data for CAN 2.0 is arranged in what referred to as frames. There are four basic types of frames as described by ISO 11898-2 standards. The first frame and most often called by the user, is a Data Frame. This is used to transfer normal operating data between nodes on the network and is the frame the system designer has the most control over and flexibility in using. The second frame that is called by a user under normal operating conditions is the Remote Transmit Request Frame. This is used to send a request to any node to send some specified information. The remaining frames are the Overflow Frame and the Error Frame. These are not called by the system designer but rather are sent when an error is automatically detected by the can nodes and are used to implement some of the error checking features of the protocol.

The Data Frame begins with the SOF dominant bit. This is followed by the 11 or 29 bit identifier for 2.0 A and 2.0 B respectively. This is followed by the RTR bit which is set dominant. These are followed by two more dominant bits and then the 4 bit data length code. This is used to announce how many data bytes will be carried by the data frame with a minimum of 1 up to a max of 8. Immediately following is the data field which contains the number of bytes specified in the data length code. The next 16 bits is a cyclic redundancy field which allows other nodes to perform error checking on the received message. There are then 3 reserved recessive bits followed by 7 recessive end of frame bits. The Remote Transmit Request Frame is set up the same as a Data Frame, the only difference is the RTR bit is set dominant rather than recessive. In Figure 16 we see a blank data frame for the CAN 2.0A standard (11 Arbitration Field bits) with all of the reserved values displayed and all of the system designer controlled variables left blank.



44+8N bits							
Function	Start of frame	Identifier	Control field	Data	CRC	Control Bits	End of Frame
Bits	1	11	6	8N	15	3	7

Figure 16: Blank Data Frame

There are several types of errors recognized by CAN 2.0. These are bit error, stuff error, form error, acknowledge error, and a CRC error. The bit error occurs when the transmitter sends a dominant bit and a recessive bit is detected on the bus, when this occurs and error frame is generated and the original message is resent after a delay. The stuff error occurs when the system detects 6 bits of the same polarity in a row. This violates the CAN 2.0 bit stuffing where after 5 bits of the same polarity are sent a 6th bit of the opposite polarity is inserted, this is due to the asynchronous nature of ISO 11898-2 and it allows clock data to be recovered from the message. If this error occurs than an error frame is sent and the message is repeated from the transmitting node. A form error occurs when a node detects a dominant bit in the inter-frame space, acknowledge delimiter, CRC delimiter, or the end of frame. When this is detected a form error message is created and the message is resent. An acknowledge error occurs when a transmitting node detects recessive, bits in the acknowledge error field. When the transmitter sends these as recessive a node receiving the message is supposed to fill this slot with dominant bits. If no dominant bits are detected than no nodes received the message an error frame is created and the message is resent after a delay. A CRC error occurs when the CRC (Cyclic Redundancy Check) number that is generated by the transmitting node doesn’t match the value calculated by a receiving node. The node (or nodes) that detect the error generates an error frame and the transmitting node has to resend the message after a delay.

A CAN node has 3 modes it can be in, which are, error active, error passive, or bus off. The node will determine which state it is in based on two registers the transmit error counter (TEC) and the receive error counter (REC). These registers store a value from 0 to 255. The transmit error counter is incremented every time the node detects an error when it is transmitting a message, the receive error counter increments whenever an erroneous message is received. The state of the system is determined by the current TEC and REC. Error active is the normal operating state of a CAN-bus node, it can send and receive messages as well as generate error frames with no limitations. If however the TEC or REC are greater than 128 the node switches to error passive mode. In this mode error flags generated are done with recessive bits instead of dominant bits, and when there is an error with a message it transmits it must delay an 8 additional bit times before it will attempt to resend the message. If the TEC counter is greater than 255 the node switches to bus off mode where it is unable to communicate over the bus in any capacity, this mode can never be activated by the REC register, preventing one faulty node from shutting down the entire system. The device must be reset, clearing the TEC and REC registers in order for the node to return to the error active mode. In Figure 17 below a state diagram shows the progression of node states with respect to the TEC and REC.

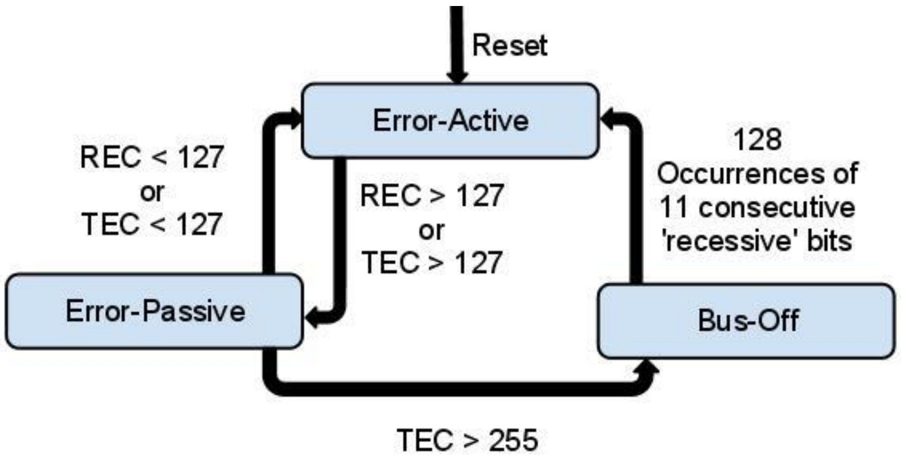


Figure 17: Error Modes of the MCP2515

4.5 MODULE D – POWER CONTROL SYSTEM

The power control systems requirements are directly related to the devices selected for this project. However the 6v max output of the vehicles battery limited the maximum operating voltage of the devices chosen. As the last section researched the voltage and current output requirements were determined easily and previous experience allowed the required components to be quickly selected.

Power System Requirements:

- 5v 1A max regulated output for each sensor unit
- 3.3v 1A max regulated output for main processor sensors
- 1.8v 100mA max regulated output for each accelerometer

Due to the 5v component supply requirement being within 1v of the 6v battery supply a low dropout voltage regulator is required to provide power to the ATmega boards as well as some of the sensors. The 1A max requirement led to the selection of two 5v regulators to be considered. These are National

Semiconductors LM2940CT-5 in the TO-220 package, and the STMicroelectronics L4941 also in the TO-220. Below in Table 12 is the side by side comparison between the two. Due to the similarities between the two [24][25] devices and the ease of ordering samples from National Semiconductors the LM2940CS-5 was chosen for this project. For the 3.3 and 1.8 volt voltage regulator, past experience with the LM1117 family from National Semiconductors made it the clear choice for this purpose as it meets the specification requirements with 800mA max supply current and 0.2% output voltage tolerances. The LM1117T-1.8 will be used for 1.8 voltage regulation and the LM1117T-3.3 will be used for the 3.3 voltage regulation. These models are in the TP-220 Package [26].

Device	**LM2940**	**L4941**
Dropout	.45-.7V	.5-.8V
Input voltage (Max)	16V	26V
Output voltage	4.8-5.2V	4.85-5.15V
Package	TO-220	TO-220

Table 12: 5v Low Dropout Regulators

4.6 MODULE E – SOFTWARE

Proper software is the only true compliment to a well designed hardware system. The TUV-ADDS system cannot function up to required specifications without some sophisticated software development. This development will be handled in two theaters: first, the sensor MCU and interfacing it with the main processor through a CAN bus; second, the main processor and the wireless interface with the ‘virtual’ command center. Figure 18 below displays this connection between the systems.

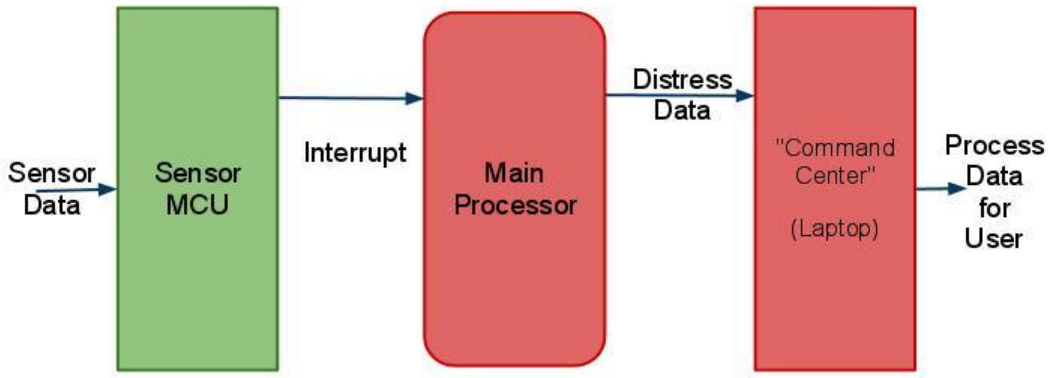


Figure 18: System Data Flow Diagram

The main processor as well as the sensor MCU will be prototyped on an Arduino development board. The group members involved with the sub-systems utilizing the Atmel MCU both have experience with developing with the Arduino, which is the basis for the decision. The Arduino uses a user-friendly Integrated Development Environment (IDE) which is based on Processing. The language developed for the Arduino IDE is a C language derivative. Many of the control syntax (for loops, if/else statements) are exactly the same as the C language [27]. The obvious advantage of the Arduino comes with port mapping. The development board will easily control I/O from both dedicated analog and digital pins. A simple line of code will set a digital output pin (pin 13 in this example) to 1 or HIGH:

```
digitalWrite(13, HIGH);
```

The other source of software development is stemmed from the user's 'virtual' "Command Center." This Graphical User Interface will be the only interaction with the user. The GUI will display the vital information sent by the TUV-ADDS in the event of distress. All GUI require an Object-Oriented Programming (OOP) Language, such as Java or C++. The team has experience programming Java since the OOP class offered at UCF is focused on Java. This proves to be beneficial since Java's Virtual Machine (JVM) would allow the Command Center software to be run on any computer platform (Windows, Mac, Linux, etc.) just as long as the computer has the most up to date Java software. Java also has well documented serial communication projects and there is a Java-API for the XBee protocol available.

Good programming practice can help maintain time management and reduce the chances of errors. The general convention for this type of process of programming is known as the Software Design Process, which can be seen below with Figure 19.

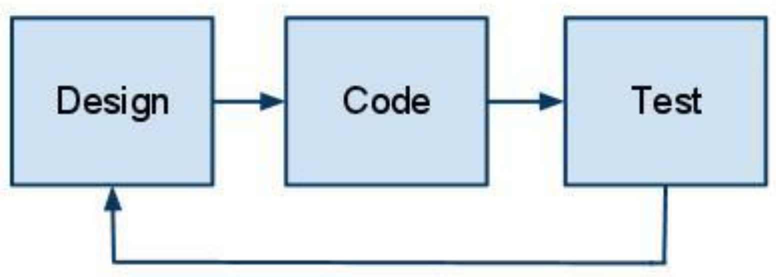


Figure 19: Software Development Cycle

4.6.1 MAIN PROCESSOR

The main processor will be responsible for many functions including I/O using several different communication protocols like CAN bus and UART. All processes required will be run as an interrupt. The processor will have to differentiate between both hardware and software interrupts and place a certain priority on the interrupt request. It will then place the interrupt request on a priority stack (if need be). All of this will increase the efficiency of the system while helping prevent any failure in major processes. In the case of a failure, there will be fail safes in place to counteract the failure. An example of this would be to store all vital data in a separate flash memory unit so that it would not be destroyed. This will also allow the system to reset and reboot automatically, then read the stored data from the flash memory.

4.6.2 XBee – PRO

The XBee – PRO is a Zigbee derivative with a plethora of resources available. Every major carrier of the devices have a forum, and every forum has solutions available for any imaginable problem. This is a great resource to have. The Atmel can communicate to the RF unit using any of the four UART lines available. The programming language and development IDE has several XBee library which further simplifies the programming. One of these libraries is from Mikal Hart of Arduiniana [28]. This library allows several individual bidirectional point-to-point connections. It will also simplify the packet and parsing needed to send large amount of data over a short timeframe. The components of the library are initiated in a way which is similar to Java or any other class driven language. An example of the NewSoftSerial can be seen below.

```
#include <NewSoftSerial.h>

    // Sets up a new serial communication
    NewSoftSerial mySerial = NewSoftSerial(2,3);

    void setup() {
        // Starts the comm. At 9600 baud rate
        Serial.begin(9600);
        mySerial.begin(9600);
    }

    void loop() {
        //If there is a budder, read it in
        if (mySerial.available()){
            Serial.print((char)mySerial.read());
    }
```

4.6.2.1 GPS and Gyroscope

These two components will both have to connect to the processor through two of the four UART connections. The software for these devices will be nearly exactly the same except for which pins on the processor they are connected to. The ATmega can easily read in a buffer of data from the UART lines. The software will have to convert the raw data into something useful such as an integer, character, string, or array. All of these data types will allow the data to be stored, transferred, and used in a multitude of fashions. A sample of how the gyroscope code may be programmed using the Atmel development IDE can be seen below:

```
int pinX = 24;
    int pinY = 25;
    int pinZ = 26;

    void setup(){
    //Nothing to set-up
    }

    void loop(){
```

```
//Read in each of the values
int xValue = digitalRead(pinX);
int yValue = digitalRead(pinY);
int zValue = digitalRead(pinZ);

delay(10) //Slow it down
}
```

4.6.3 COMMAND CENTER

The command center will be written with Java, which is an Object Oriented language. The system will only utilize the basics of the Graphical User Interface (GUI) capabilities that Java has. This decision was made based on the fact that the focus should really be on the sending of vital data to some sort 'command center.' All that the project requires is that the information can be seen by the user on a laptop (or desktop). If time permits, it could be possible to scale up the complexity and user ability of the command center. Several members have experience with Java.

Java makes it easy to communicate via serial ports; it has built in libraries for serial communication. Java can also dynamically select from any of the available ports which will reduce the chance of error. The first part of the GUI will require the user to select from a list of available ports. After this the software will make an initial wireless check then go into standby. In standby the software will do nothing except wait to grab data from the selected serial port. If a distress signal is caught, the software will then display all vital data including the event that took place, where (GPS), what time, and so forth.

4.6.4COMMAND CENTER MODEM

The command center modem will connect to the command center (computer) using a TTL-232R USB to TTL Serial cable. This cable has a USB to TTL converter embedded directly into the cord. This will allow to XBee module to "talk" directly to the Java software passing all data through an Atmel processor. The processor will wait for an available message from the vehicle, organize the data, and finally send the data serially to the Java software. The XBee can communicate directly to the Java software through a serial port and the TTL cord, but the module does time-out the network to save power. The processor will 'wake-up" the XBee while it is getting the data ready to send.

5 DESIGN DETAILS

5.1 MODULE A – SENSOR MODULE

5.1.1 SENSOR MCU

The project utilizes an Atmel ATmega328P in a 28 PDIP package. The pinout for this device as well as the functions of these pins are seen below Table 13.

Pin	Name	Function
1	(PCINT14/RESET) PC6	Reset, Connected 5v w/ switch to ground
2	(PCINT16/RXD) PD0	Unconnected
3	(PCINT17/TXD) PD1	Unconnected
4	(PCINT18/INT0) PD2	Digital I/O, connected pin 10 BMA220
5	(PCINT19/OC2B/INT1) PD3	Digital I/O, connected pin 16 MCP2515
6	(PCINT20/XCK/T0) PD4	Digital I/O, connected chip select Microsd
7	Vcc	Connected to 5v
8	GND	Ground
9	(PCINT6/XTAL1/TOSC1) PB6	Connected to 16Mhz oscillator
10	(PCINT7/XTAL2/TOSC2) PB7	Connected to 16Mhz oscillator
11	(PCINT21/OC0B/T1) PD5	Digital I/O, connected pin 3 4043
12	(PCINT22/OC0A/AIN0) PD6	Digital I/O, connected pin 2 4043
13	(PCINT23/AIN1) PD7	Unconnected
14	(PCINT0/CLKO/ICP1) PB0	Unconnected
15	PB1 (OC1A/PCINT1)	Unconnected
16	PB2 (SS/OC1B/PCINT2)	Unconnected
17	PB3 (MOSI/OC2A/PCINT3)	SPI serial in, connected MOSI bus
18	PB4 (MISO/PCINT4)	SPI serial out, connected MISO bus
19	PB5 (SCK/PCINT5)	SPI clock, connected SCLK bus
20	AVCC	ADC voltage supply, connected to 5v
21	AREF	Analog reference voltage, Externally decoupled
22	AGND	ADC Ground, connected to ground
23	PC0 (ADC0/PCINT8)	Analog input, Connected to flash sensor
24	PC1 (ADC1/PCINT9)	Unconnected
25	PC2 (ADC2/PCINT10)	Unconnected
26	PC3 (ADC3/PCINT11)	Unconnected
27	PC4 (ADC4/SDA/PCINT12)	I2C Data line, connected to pin 6 TMP100
28	PC5 (ADC5/SCL/PCINT13)	I2C Clock line, connected to pin 1 TMP100

Table 13: Atmel ATmega328P Pinout

MicroSD cards communicate natively in SPI and the ATmega is most compatible with the FAT16 filesystem that can be implemented on the card. The pins for the SDcard are industry standard and listed in Table 14 below.

Pin	Function	Pin	Function
1	Chip Select	6	GND
2	MOSI	7	MISO
3	GND	8	No Connection
4	5V	9	No Connection
5	SCLK		

Table 14: MicroSD pinout

Below in schematic (Figure 20) the Atmel ATmega 328P is combined with the CAN module and the required connections are made available for the various sensors that will be pooled for the unit. The RS latch used to store an interrupt from the CAN controller is shown as IC2. The Atmega 5v input is recommended coupled to ground to prevent any possible instability in line voltage from affecting the operation of the device. Connections to the external sensors are shown in the bottom left along with the LM117T-1.8 voltage regulator used to supply 1.8 volts to the accelerometer. The CAN-bus connection is labeled as X1 and next to it is the MicroSD connector. Both controllers require a 16 Mhz oscillator to function as the CPU clocks.

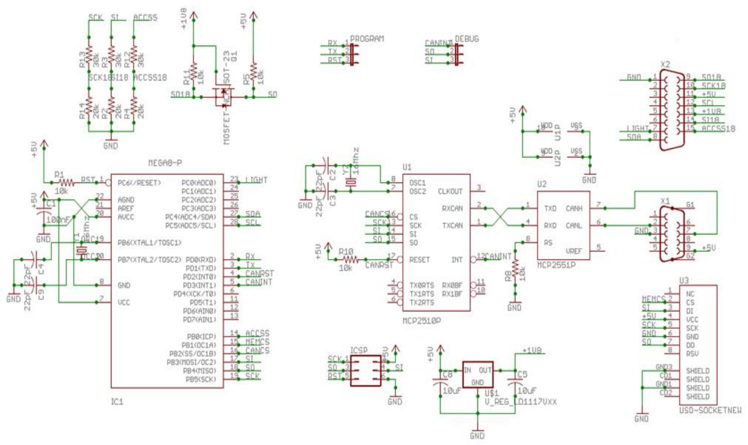


Figure 20: Sensor Unit

5.1.2 ACCELEROMETER

The accelerometer we have chosen is the BMA220, manufactured by BOSCH; this is a three-axis low g accelerometer. The cost is $3.09 per unit and we will be able to get these accelerometers form the online vender Digi-Key.

Referencing the specifications, from the data sheets, this accelerometer has the potential to meet our requirements. It is a low g accelerometer with a digital interface. The BMA220 can support either a SPI or I2C digital Interface as selected by the user. The acceptable I/O supply voltage is 1.6 to 3.6 volts, which is obtainable through the six-volt battery that is contained on the vehicle.

The user, as shown below in Table 15 below, can select the scale and resolution of the accelerometer. The testing that the system will endure will only call for low g forces to be applied to the system. It is likely the testing will only need to stay in the realm of ± 2g's. If the sensitivity at ± 2g is found to be under sensitive once the prototype is constructed, the component allows for the user to adjust the sensitivity with the use of software routines.

Scale and resolution				
range[1:0]	Scale	Sensitivity	Resolution	Topical use
00	± 2g	16 LSB/g	62.5mg/LSB	Orientation
01	± 4g	8 LSB/g	125mg/LSB	
10	± 8g	4 LSB/g	200mg/LSB	
11	± 16g	2 LSB/g	500mg/LSB	Shock &Vibration

Table 15: Accelerometer Scale and Resolution Table

The orientation of the pin numbers and function is shown in Figure 21 below. The figure is of the top view orientation of the accelerometer. This accelerometer will be required to be surface mounted on the PCB. The pin and pin function will be needed in the creation of a library file for the schematic of the circuit that will need to be generated.

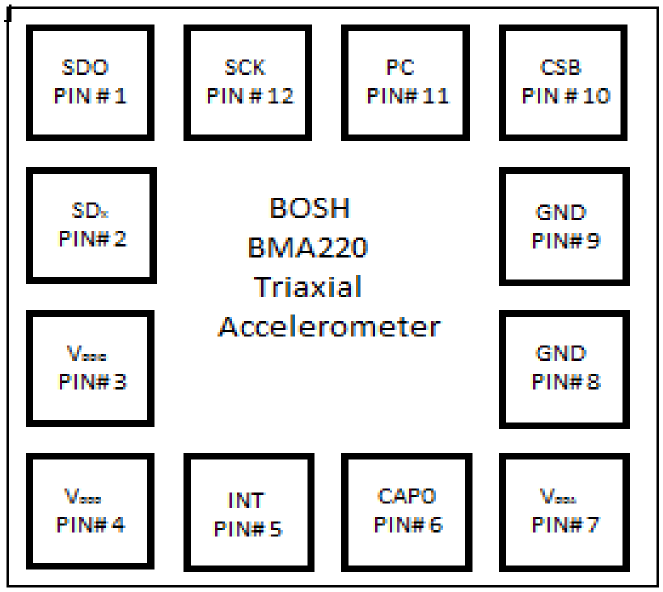


Figure 21: Accelerometer Pin-out

The accelerometer is capable of working with several types of interfaces: SPI and I2C. Table 16 below will aid in the identification and understanding the function of each pin on the accelerometer. This information will also be helpful when deciding which pins need to be connected to what source to achieve the selected interface method. We will need to connect pins# 1, 2, 10, 12 and 11 to ground. We will also need to create an EAGLE library for this sensor. Therefore, the pin numbers and names will need to be referenced in order to maintain consistency between the schematics and PCB board design layout. For this sensor we are going to use the SPI interface method to pass on data to the sensor MCU.

Pin Description (FOND on PAGE 47)						
Pin #	Name	Type	Description	Connect to (SPI 4w)	Connect to (SPI 3w)	Connect to (I2C)
1	SDO	Digital Out	SPI serial data output	SDO	NC	NC
2	SDx	Digital I/O	SDA for I2c serial data in/output SDI for serial data input (SPI 4wire) SDA serial data in/output (SPI 3wire)	SDI	SDA	SDA
3	VDDIO	Supply I	I/O supply voltage (1.62 to 3.6 volts)	VDDIO	VDDIO	VDDIO
4	VDDD	Supply I	Power supply for analog domain	VDDD	VDDD	VDDD
5	INT	Digital I/O	Interrupt Output	INT	INT	INT
6	CAP0	DNC	Do not connect	NC	NC	NC
7	VDDA	Supply I	Power supply for analog domain	VDDA	VDDA	VDAA
8	GND	Ground	Shared GND for digital, I/O, analog	GND	GND	GND
9	GND	Ground	Shared GND for digital, I/O, analog	GND	GND	GND
10	CSB	Digital In	Chip-select for SPI mode. Address-select for I2C mode, see chapter 8.3. Pint must not float	CSB	CSB	CSB
11	PS	Digital In	Protocol select pin (0=SPI, 1=I2C, float = µC-less); pin must not float unless dedicated mode is used, see chapter 6.1	GND	GND	VDDIO
12	SCK	Digital In	SCK for SPI serial clock SCL for I2C serial clock	SCK	SCK	SCL

Table 16: Pin Name with Respect to its Corresponding Interface Function

Table 17 below identifies the pin name with respect to its corresponding interface function. This will be helpful when mounting the accelerometer to the board and choosing which interface method will be preferred when connecting the sensors to the sensor MCU. This will also be needed as reference when designing the interface to connect the sensor to the MCU.

Interface Pin Name	
PIN Name	PIN Description
CSB	SPI serial enable bar
SCK	SPI serial clock
SCL	I2C serial clock
SDI	SPI serial data input
SDO	SPI serial data output 3wire
SDA	I2C serial data
SDO	SPI serial data output

Table 17: SPI and I2C Chart

This is one of the suggested power setups given, by BOSCH, which will show where to place a voltage sources and capacitors required operating the chip. As seen in Figure 22 there are three 100nF capacitors connected to the chip. These will be placed on the PCB that will contain all the sensors. The figure also shows the voltage that will need to be provided to the respective pins.

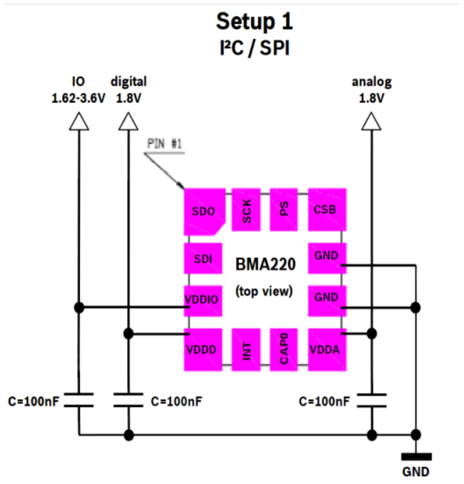


Figure 22: Minimum Circuit Requirements

Shown below (Figure 23) is the recommended dimension of the landing pattern with respect to the accelerometer. This information will need to be consulted while designing the sensor PCB. Existing EAGEL libraries for this sensor have not been found, therefore, we will have to reference the below diagram to create a library for this sensor so it can be included in the schematics board design. The figure below also includes the solder mask pattern and where the solder mask should stop. All measurements of this accelerometer are given in mm and not mil just to make a note, as to avoid confusion later on in the design. This sensor will be surface mounted on the sensor board and this figure will need to be consulted in the landing pattern and surface mounting design.

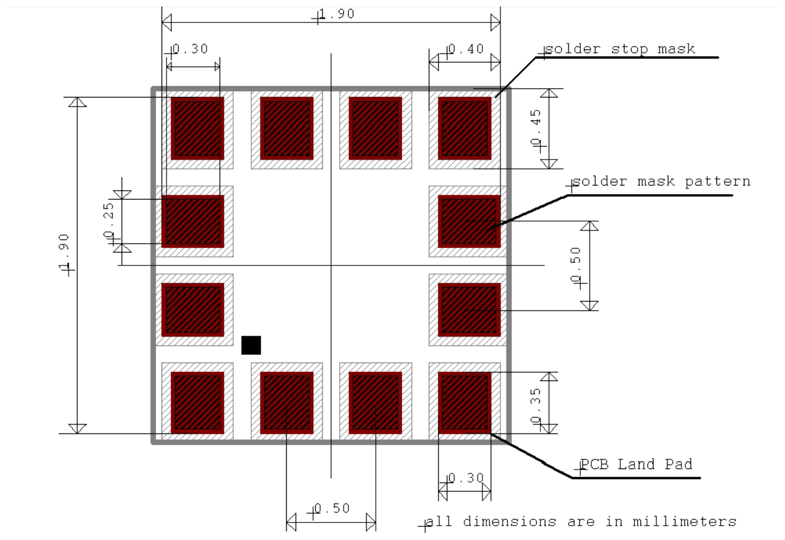


Figure 23: Accelerometer Pin Spacing

5.1.3 TEMPERATURE

The team via the TI sample program acquired five Texas Instruments temperature sensors. The model is the TMP100, Digital Temperature Sensor with I2C Interface. This model meets our specifications and is no cost to the team, so it will be incorporated into the design of the system.

Looking at the specifications published by Texas Instruments the TMP100 appears to meet our expectations, in what we needed in a temp sensor. To be more precise, our testing procedure will call for a temperature sensor capable of measuring a temperature of near 70°C; this is well within the maximum operational temperature listed at 125°C. The IC temp will take the temperature at the surface of the board wherever it is mounted; being this will be mounted on a board with only other low power sensors any erroneous heat sources can be neglected.

Because of the nature of the events that this system would be detecting, it would be ideal if it had the ability to record the data, process it and have it transmitted before any permanent damage is done to the sensor. The TMP100 can operate in what TI calls high-speed mode, defined as frequencies above 400 kHz, and would have a better chance of transmitting data before it is no longer operational. It is noted that the "master device must issue an Hs-mode master code (00001XXX) as the first byte after a START condition to switch the bus to high-speed operation. The data rise and fall times for this particular temp IC can be as fast as about 160ns. This should be fast enough to accomplish the task quick as possible. The operational frequency can be as high as 3.4MHz in the high-speed mode.

Figure 24 below shows the pin configuration along with a block diagram of the temp sensor. This is a top view orientation of the temp IC chip and will serve useful when designing the landing pad on the PCB for the placement of this sensor.

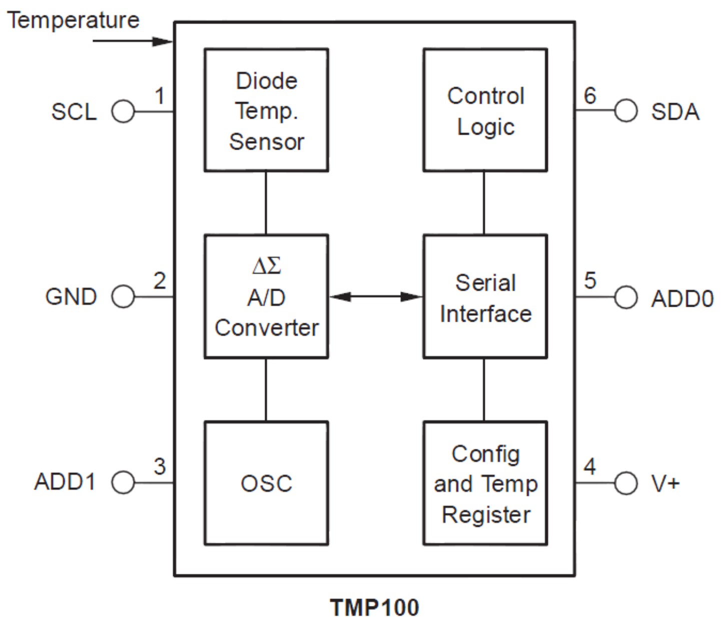


Figure 24: Temperature Sensor Pin and Function Diagram
Courtesy Texas Instruments

Shown in Figure 25 is further explanation of the setup with additional hardware called for such as the 0.1 µF capacitor and a pull-up resistors. The data sheets state the 0.1 µF capacitor is recommended but not required but the pull-up resistors are required though. These additional elements will need to be placed on the PCB board.

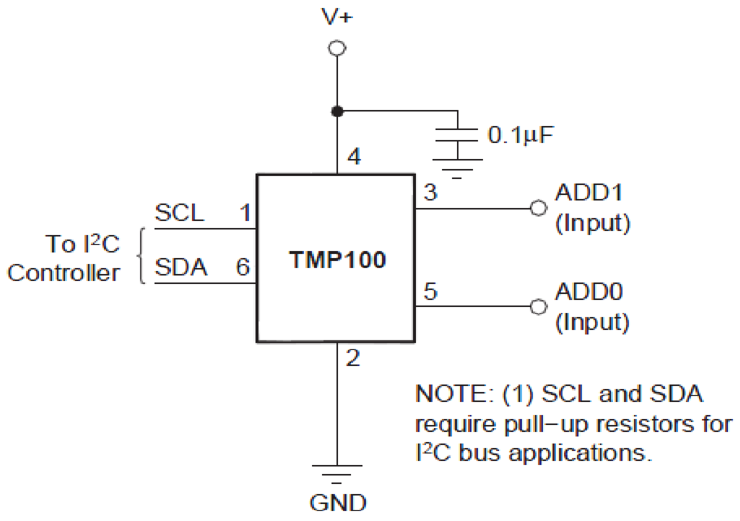

V+
0.1µF
4
3
ADD1
(Input)
SCL 1
To I2C
Controller
SDA 6
TMP100
5
ADD0
(Input)
2
NOTE: (1) SCL and SDA require pull–up resistors for I2C bus applications.
GND

Figure 25: Temperature Sensor Minimum Circuit Requirements
Courtesy Texas Instruments

Once again, we are more concerned with speed at which we will retrieve and process the data and not necessarily with the accuracy. The system is just looking to recognize that the temperature did increase in a short period of time. The table below (Table 18) will help judge the accuracy verses speed of the sensor relationship.

Resolution			
R1	R0	Resolution	Conversion Time
0	0	9 Bit (0.5°C)	40ms
0	1	10 Bit (0.25°C)	80ms
1	0	11 Bit (0.125°C)	160ms
1	1	12 Bit (0.0625°C)	320ms

Table 18: Temperature Sensor Resolution Chart
Courtesy Texas Instruments

Table 19 gives the temperature characteristics of the IC thermo chip. As seen in the table there is a tradeoff between the temp range and general accuracy of the device. Our test procedure will call for a heat source near 70°C. This should give our design a maximum error in temp of ± 2.0°C

Temperature Characteristic					
Parameter	Test Condition	Minimum	Typical	Maximum	Unit
Range	----------	-55	------	+ 125	°C
Accuracy	-25°C to +85°C	------	± 0.5	± 2.0	°C
Accuracy	-55°C to +125°C	------	± 1.0	± 3.0	°C
Resolution	Selectable	------	± 0.0625	------	°C

Table 19: Temperature Sensor Temperature Characteristics
Courtesy Texas Instruments

The input-output characteristics of the temp sensor are shown below in Table 20. The max and min logic levels are given as well as the resolution from 9 bits to 12 bits. Also, the conversion time is listed for each bit of resolution. Most likely a 9-bit resolution will do just fine and the test procedure will call for a large change in temp therefore making our ability to test or precession less important.

Digital I/O Characteristic					
Parameter	Test Cond.	Minimum	Typical	Maximum	Unit
Input Logic:					
V_{IH}	------	0.7(V+)	------	6.0	Volt
V_{IL}	------	-.05	------	0.3(V+)	V
I_{IN}	$0V \leq Vin \leq 6V$	------	------	1	μA
Output logic:					
V_{OL} SDA	I_{ol} = 3mA	0	0.15	0.4	V
V_{OL} ALERT	I_{ol} = 4mA	0	0.15	0.4	V
Resolution	Selectable	------	9 to 12	------	Bits
Conv. Time	9 - Bit	------	40	75	ms
	10 - Bit	------	80	150	ms
	11 - Bit	------	160	300	ms
	12 - Bit	------	320	600	ms
Conv. Rate	9 - Bit	------	25	------	s/s
	10 - Bit	------	12	------	s/s
	11 - Bit	------	6	------	s/s
	12 - Bit	------	3	------	s/s

Table 20: Input-Output Characteristics of the Temperature Sensor
Courtesy Texas Instruments

The power supply requirements for the TMP100 are listed below in Table 21. The voltage supplied will need to be in the range of 2.7 to 5.5 volts, the power will be supplied from the MCU. Because of the low current rating, it can be attached directly to a microcontroller.

Power Supply					
Parameter	Test Condition	Minimum	Typical	Maximum	Unit
Operating Range		2.7	------	5.5	V
Quiescent current: Iq	Serial Bus Inactive	-----	45	75	µA
	SBA, SCL freq=400kHz	-----	70	-----	µA
	SBA, SCL freq=3.4MHz	-----	150	-----	µA
Shutdown Current: Isd	Serial Bus Inactive	-----	0.1	1	µA
	SBA, SCL freq=400kHz	-----	20	-----	µA
	SBA, SCL freq=3.4MHz	-----	100	-----	µA

Table 21: Power Supply Requirements for the TMP100
Courtesy Texas Instruments

This thermo sensor has a temperature range as shown below in Table 22. The accuracy of the sensor is affected by the range of the temperature applied, the greater the range the less accurate the temperature sensor is capable of measuring. Due to the maximum operational temp of some of the other sensor, this design will be capable of staying in the tighter accuracy range.

Temperature Range						
Parameter	Accuracy	Test Condition	Min	Typical	Max	Unit
	±2.0°C		-25		+85	°C
Specified Range	±3.0°C	------	-55	------	+125	°C
Storage Range		------	-60	------	+150	°C
Thermal Resistance θ*JA*		SOT23-6 SM	------	200	------	°C/W

Table 22: Thermo Sensor Temperature Range
Courtesy Texas Instruments

The format of the digital output is represented in a 12 bit binary and is capable of logging data from -25°C to +128°C in binary as 1110 0111 0000 and 0111 1111 1111 respectively. The output is also given in hexadecimal format from 7FF to

E70 respectively as well. The various temperatures and their respective output are more detailed show below (Table 23).

Temperature Data Format		
Temperature (°C)	Digital Output (Binary)	HEX
128	0111 1111 1111	7FF
127.9375	0111 1111 1111	7FF
100	0110 0100 0000	640
80	0101 0000 0000	500
75	0100 1011 0000	4B0
50	0011 0010 0000	320
25	0001 1001 0000	190
0.25	0000 0000 0100	004
0.0	0000 0000 0000	000
-0.25	1111 1111 1100	FFC
-25	1110 0111 0000	E70
-55	1100 1001 0000	C90
-128	1000 0000 0000	800

Table 23: Various Temperatures and their Respective Output
Courtesy Texas Instruments

The next series of drawings (Figure 26) illustrate the physical characteristics of the TMP100 sensor. This sensor will ultimately be surface mounted to a board with three other sensors. This information will be necessary when designing the layout for the sensor PCB. This physical package information is also required to create a library file for this sensory, should one not currently exist.

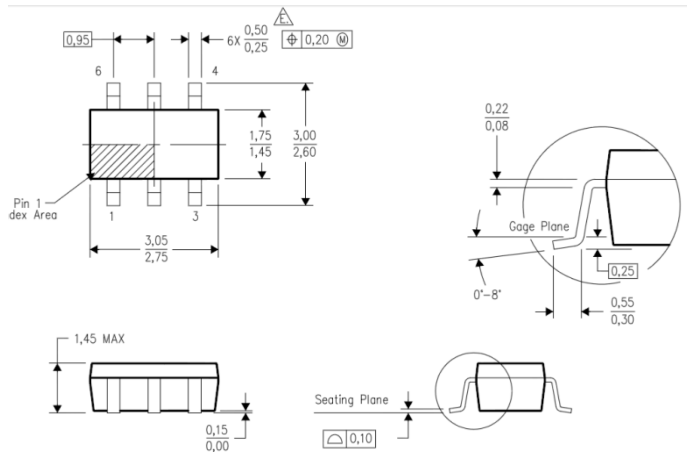


Figure 26: Physical Characteristics of the TMP100 Sensor
Courtesy Texas Instruments

5.1.4 FLASH/PHOTODIODES

The Light detection sensor chosen for this design is the TSL14S-LF light to voltage converter. This Photodiode sensor is manufactured by Texas Advance Optoelectronic Solutions. The team is currently attempting to acquire this diode from one of several venders, namely Net Semi and Mouser Electronics. Mouser Electronics have a nine-week lead-time for ordering at a cost of 1.26 per diode. In an effort to avoid this wait time I looked at Net Semi, they appear to have the sensor in stock without the wait time but they required a minimum order of $50.00. Being we are self funded, I have emailed requested to the company to authorize an order for an order of 6 to 10 diodes at $1.99 per sensor.

The manufacture has described the sensor as: “cost-optimized, highly integrated light-to-voltage optical sensors, each combining a photodiode and a transimpedance amplifier (feedback resistor = 80 MΩ, 20 MΩ, and 5 MΩ, respectively) on a single monolithic integrated circuit.”[29] This is an analog sensor that outputs a voltage with respect to the light intensity. The output voltage is linear with respect to the intensity of the light. This is Ideal for our design, as we will pass this sensor though an AD converter before connecting it to a MCU. The linear aspect will allow for consistent voltages reading for a multitude of light intensities. The wavelengths the photodiode are capable of responding to are in the range of 320nm to 1050nm. This should be more than wide enough to respond to and lighting conditions which will be chosen for testing.

Below is Table 24 which shows the Dynamic Characteristics of the photodiode. This table gives the various delay, rise and fall times for the photodiode. It is important to the system design to be able to detect a change in light and have it reported to the MCU very quickly. This will be useful so we can understand what sorts of time we can expect from the diode as it reports to the MCU.

Dynamic Characteristics			
Parameter	Test Conditions	Typical	Unit
Output pulse delay time for rising edge (0%-10%): t_{dr}	Min $V_{o\,=}$ 0 V; Peak V_o =2V	0.9	µs
	Min $V_{o\,=}$ 0.5 V; Peak V_o =2V	0.6	µs
Output pulse rise time (10% to 90%) :t_t	Min $V_{o\,=}$ 0 V; Peak V_o =2V	2.6	µs
	Min $V_{o\,=}$ 0.5 V; Peak V_o =2V	2.9	µs
Output pulse delay time foe falling edge (100% to 90%): t_{df}	Min $V_{o\,=}$ 0 V; Peak V_o =2V	0.8	µs
	Min $V_{o\,=}$ 0.5 V; Peak V_o =2V	0.7	µs
Output pulse fall time (90% to 10%): t_f	Min $V_{o\,=}$ 0 V; Peak V_o =2V	2.9	µs
	Min $V_{o\,=}$ 0.5 V; Peak V_o =2V	2.8	µs
*V_{DD} = 5V, T_A = 25°C, λ_P =640nm, R_L 10KΩ			

Table 24: Dynamic Characteristics of the Photodiode

The three terminals on the photodiode are as listed below in Table 25 along with the diagram (Figure 27). This information will be needed when mounting to the PCB to ensure the component is mounted correctly. The typical width of the terminals is 0.47mm by 0.74mm with a tolerance of $\underline{+}$ 0.25mm.

Terminal Functions			
Number	Name	Type	Description
1	GND	------	Power supply ground (substrate).
2	OUT	O	Output voltage.
3	V_{DD}	------	Supply voltage.

Table 25: Three Terminals on the Photodiode

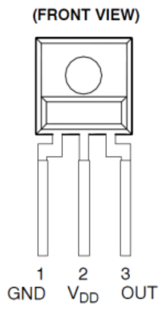


Figure 27: Three Terminals on the Photodiode

Below (Table 26) are the suggested operating ranges for the diode. This will need to be noted in the test procedure that the heat source cannot exceed 70°C. Our test procedure will likely call for a heat gun so the threshold for the temp will need to be set lower than 70°C. This will be more than adequate for our design to meet our test procedures and system goals.

Recommended Operating Conditions			
	MIN	MAX	UNIT
Supply Voltage, VDD	1.7	5.5	V
Operating free-air temperature, TA	0	70	°C

Table 26: Suggested Operating Ranges for the Diode

5.1.5 SENSOR PCB

The three sensors: accelerometer, photodiode and thermo sensor will be place on a PCB, which we will call the sensor module. It is anticipated that we will be able to place all three sensors on a two inch by two inch two layer PCB, this should comfortably be able to hold all three sensor and their additional components such as capacitors and pull up resistors.

We have chosen to use EAGLE software to create the schematics and board design layouts. For the sensors, in particular the accelerometer and the photodiode, we will need to create a library .lbr file to incorporate these into the design using the EAGLE software, as one does not publicly exist. In order to do this a package and symbol file must be created, then merged together to create a device. The data sheets are referenced in order to achieve the most accurate measurements of the footprint of the IC and relative location of the pads.

Below is schematic diagram of sensor module (Figure 28). This should demonstrate a basic schematic layout of the sensor board. This shows a complete lay out of the sensor and all of their respective hardware components, which will need to be included into the circuit design. This schematic was created using the EAGLE software and will also be used in the creation of the PCB during the prototype construction of this system.

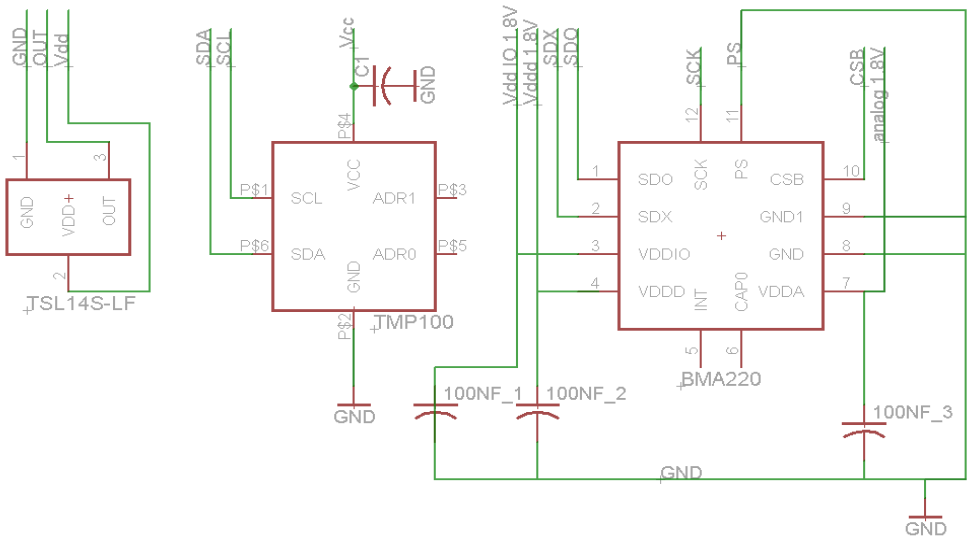


Figure 28: Schematic Layout of the Sensor Board

5.1.6 SENSOR MODULE DESIGN REVISIONS

In the initial design of the sensor module an accelerometer was called for; which was to be placed on the sensor board along with a temp sensor and photodiode. However, during the construction of the prototype the team had difficulties interfacing the BMA 220 accelerometer to the sensor microcontroller. This led to a discussion by the team in which it was realized we had to replaced our gyroscope with a Freescale accelerometer. It was then decided to have the TUV-ADDS run off of a single accelerometer located in a centralized location of the host vehicle. Further justification for this, if we were to implement this system (beyond the scope of our prototype) it would call for several of these sensor boards modules to be placed around the perimeter of the vehicle. For the temp IC and photodiode this still made sense; however, for the accelerometer this was not the case. To our understanding the vehicle is a ridged structure and the whole vehicle would experience an external force of an explosion with relative uniformity, therefore it seemed redundant and unnecessary to have multiple accelerometers. With this modification the TUV-ADDS system was capable of meeting our original design specifications.

With all this in consideration the team happened to have a second accelerometer which we implemented in the design of the TUV-ADDS. This accelerometer was produce by Freescale of and was the MMA7260QT. This unit was more costly to the group at $46.00 but successful implementation the component deemed it worth the cost. This accelerometer is a three axis accelerometer, as we still felt it was important to have this capability in the system. The MMA7260QT was an analog sensor and not a digital as compared to the BMA220. The Freescale accelerometer is a low g sensor, which is acceptable as our testing procedure will only call for low g forces to be applied to the host vehicle. This Freescale accelerometer is to be placed in the with the main control module which in turn was placed under the “front hood” of the power wheels vehicle.

5.2 MODULE B – MAIN CONTROL UNIT

5.2.1 MAIN PROCESSOR

Atmel lists several basic requirements that are needed to maintain stability with the ATmega. These include (but may not be limited to) decoupling on all voltage in and ground by using capacitors. Atmel also describes the use of an external 16MHz crystal buffered with two capacitors. The Main Control Unit must be flexible to allow any updates. The processor will be accessible through a JTAG line connect to an RS232, which links to a USB connector. The circuit seen in Figure 29 is the design linking a USB to the ATmega. It is also vitally important to maintain a stable voltage source (Figure 30). An entire subsection of the TUV-ADDS is dedicated completely to power supply. The processor will connect directly to the Power Supply System to draw a steady five volt supply. Figure 31 shows the port mapping of all of the devices that will communicate with the processor. This Figure also shows the requirements for both supply voltage decoupling as well as external crystal oscillator.

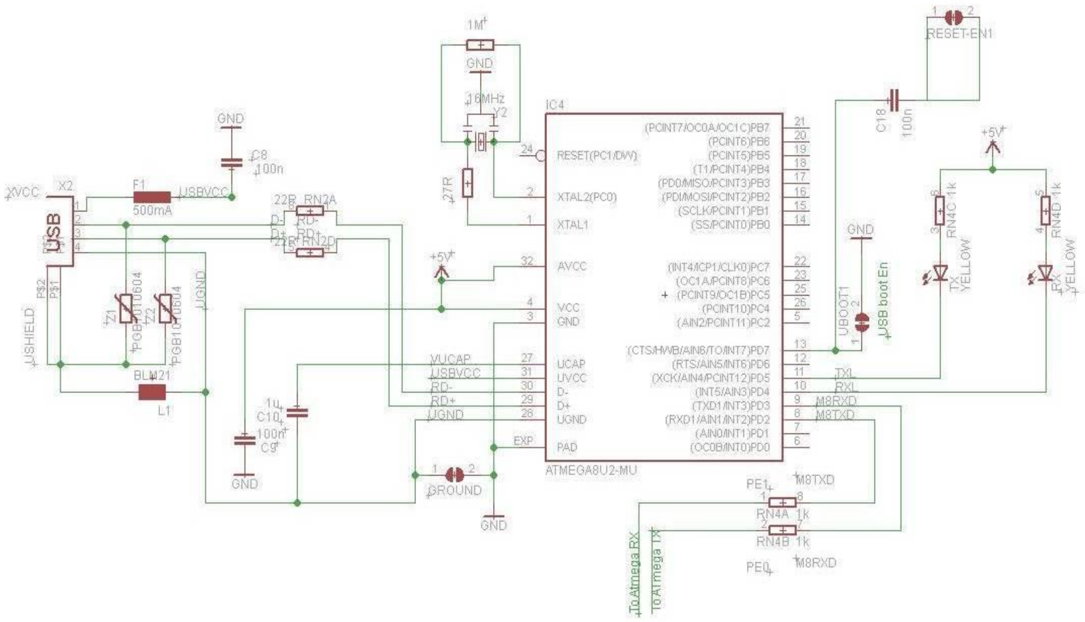


Figure 29: Design Linking a USB to the ATmega

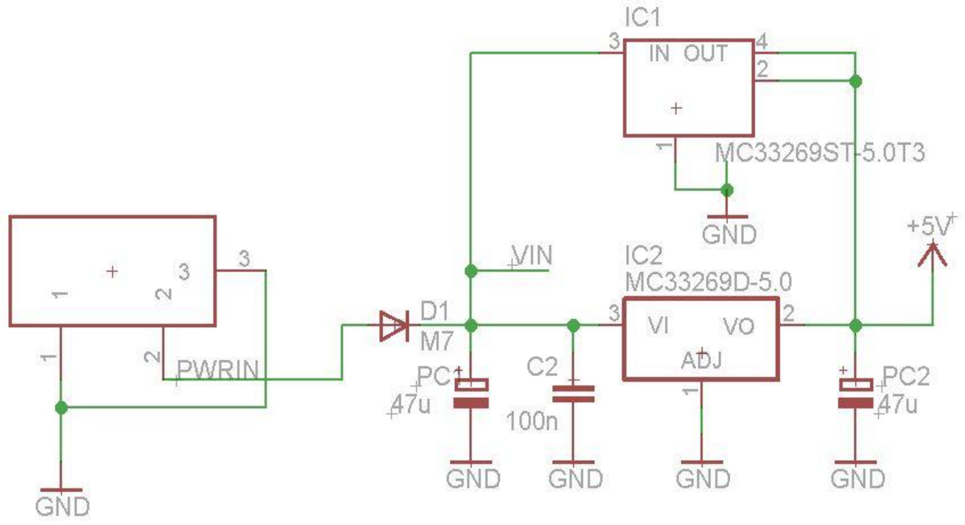


Figure 30: Stable Voltage Source

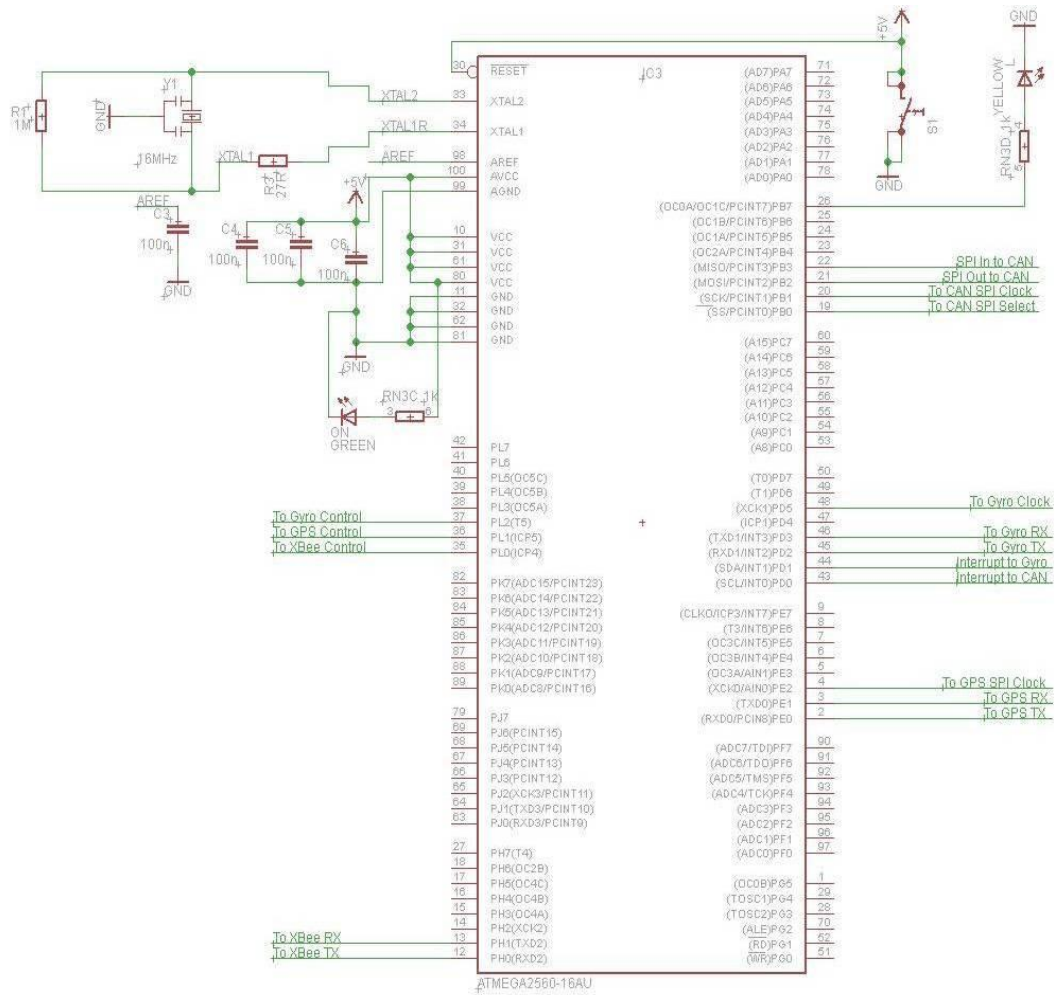


Figure 31: Port Mapping of all of the Devices that will Communicate with the Processor

5.2.2 COMMUNICATION

The XBee module has been proven to be an excellent product. There are a few draw backs in hardware designs that need to be addressed. A major issue is that the module is susceptible to burn out from a high (relatively speaking) signals. If there were to be a large spike in a signal that was slightly higher than five volts, the module could burn out. This situation can be easily avoided by using a five volt compliant buffer. The buffer will take a signal based around a five volt average and shift it down to 3.3 volts. This will prevent any spikes in signals to reach a dangerous level. The buffer is embedded in the schematic (Figure 32 seen below) of the XBee adapter as IC2(A-D). The output from the adapter will connect directly to one set of Transmit/Receive (Tx/Rx) ports that are designated through software.

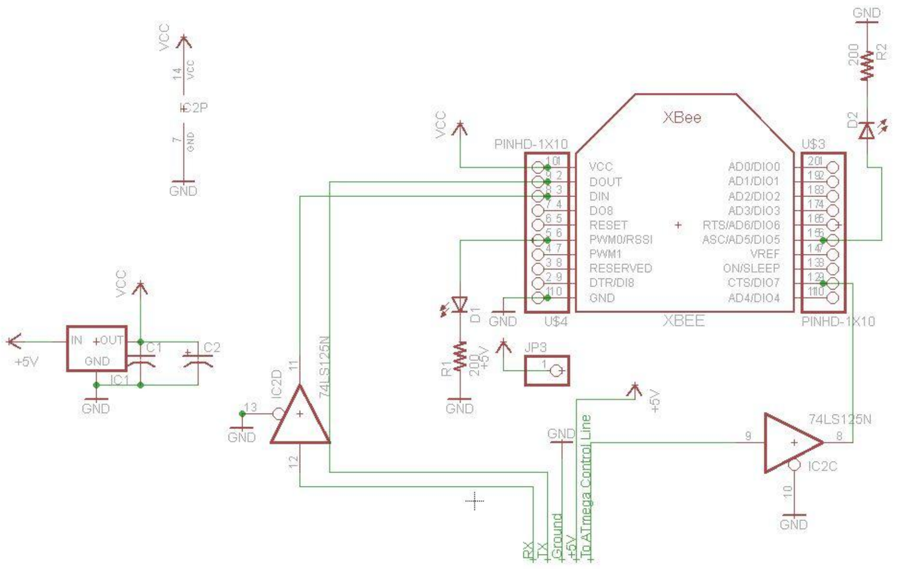


Figure 32: Buffer is Embedded in the Schematic of the XBee adapter as IC2(A-D)
(Printed with permission from Adafruit Industries, Inc.)

5.2.3 CAN BUS CONNECTION

It was decided early on to use a CAN bus between the Main Control Unit and the sensor modules. The Main Control Unit will implement a CAN 2.0 A/B system to communicate to the sensor modules. The CAN module will require an interrupt output pin and one SPI or I^{2}C USART serial line. A SPI line can be easily connected to the processor by following the pin set up seen below in figure 33.

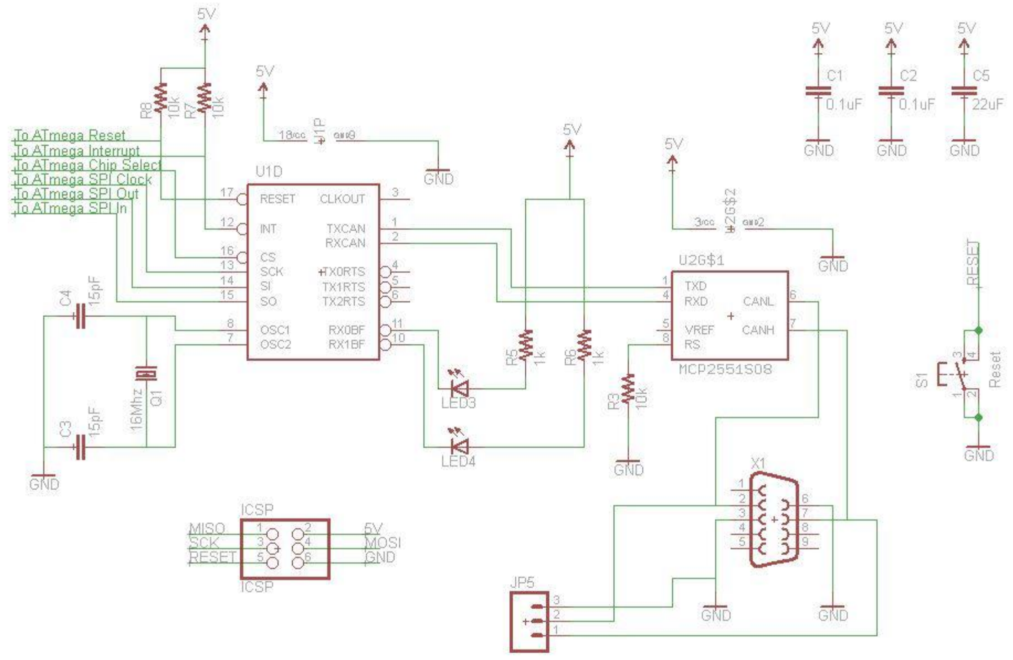


Figure 33: SPI line easily connected to the processor by this pin set up

5.2.4 GPS

Many of the GPS units that were researched had similar hardware configuration considerations. Some of these include a decoupled supply voltage as well as inductor near the RF antenna. With the particular GPS unit that the system will utilize it requires that there be a difference between the circuit common ground and the RF ground. All components connected and/or near the RF component will be connected to the RF ground (see Figure 34). Other than this RF requirement, the rest of the circuit is mostly devoted to stable communication interface.

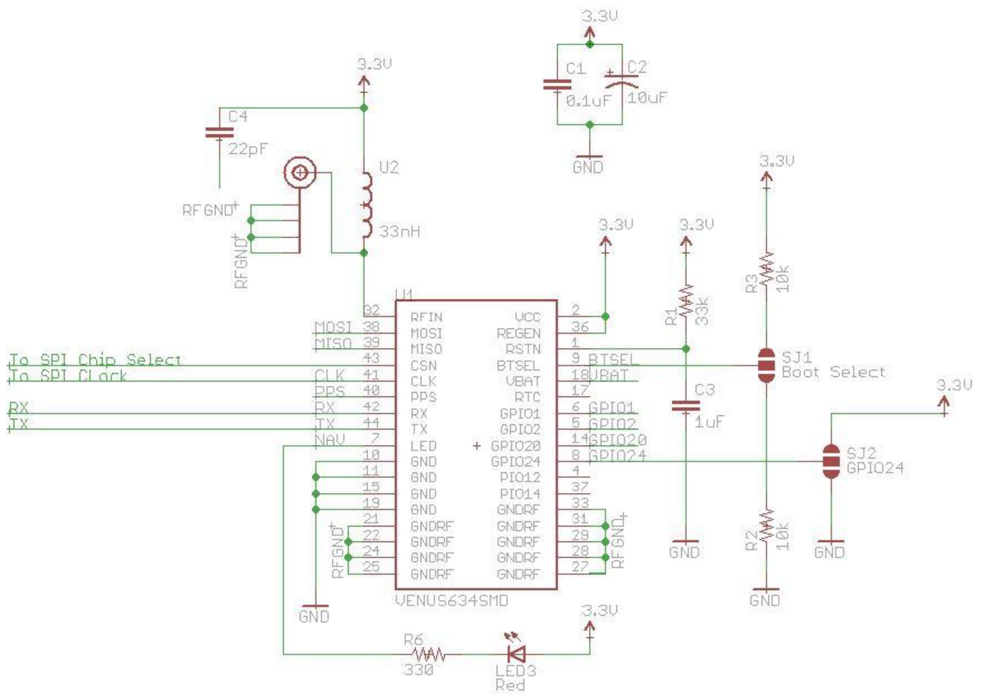


Figure 34: Components Connected to the RF Ground

5.2.5 GYROSCOPE

The gyroscope circuit requires a decoupled supply voltage. This is accomplished by putting a resistor and capacitor in parallel between the input voltage and the supply voltage of the unit. It is highly recommended that SPI communication lines have and attached voltage line as seen in Figure 35. This consideration facilitates the SPI communication protocol. The gyro circuit will also have both an interrupt input and interrupt output. The interrupt output will send a message to the microcontroller that the gyro is ready to send the data while the interrupt input will receive a message when the microcontroller is requesting data.

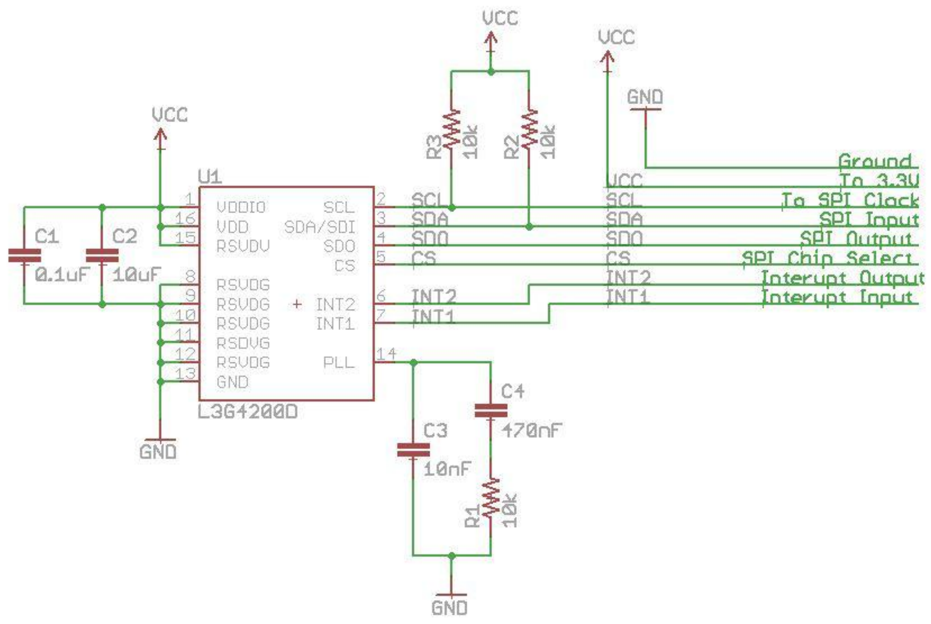


Figure 35: SPI communication lines have an attached voltage line

5.2.6 MCU DESIGN REVISIONS

The final TUV-ADDS system utilizes an Atmel ATmega 328P instead of the originally designed ATmega 2560. A mostly function Main Control Unit was realized on a PCB (see Figure 36 below) including the ATmega2560, which is a surface mounted device. After initial testing, we realized that we could not access the 2560 using our simple In-System Programmer (ISP). The access the JTAG lines on the chip, we needed to use a High-Voltage ISP to burn out the JTAG fuse which we did not have access to. To overcome this issue we decided to rebuild the entire Main Control Unit of a Perforated Board (PerfBoard) using the 328P. This device is a PDIP package which eases debugging and connection errors. A picture of the Main Control Unit built on the PerfBoard can be seen below in Figure 37. As will be explained in section 6.3, CAN communications was replaced with a falling edge triggered interrupt over pin 6 of the 9 pin D-SUB port.

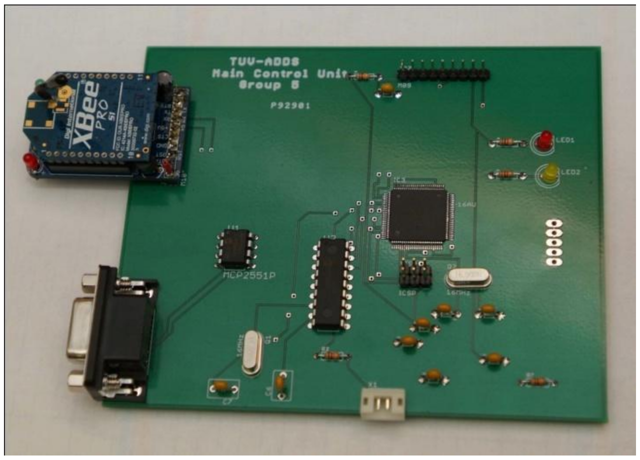


Figure 36: Main Control Unit PCB with ATmega2560

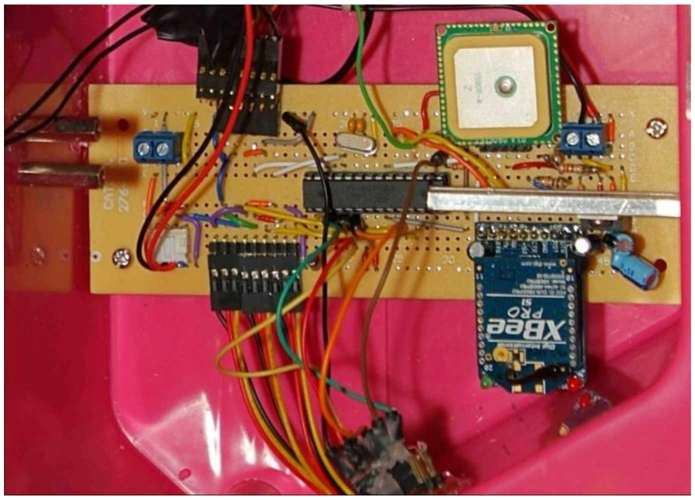


Figure 37: Final Main Control Unit with ATmega328P

Another design change to the Main Control Unit is the use of an accelerometer to detect Rollover instead of a gyroscope. During component level testing of the gyroscope (a picture of the gyro acquired can be seen below in Figure 38), we quickly realized that the gyroscope does not detect the desired change in angle relative to an origin. The gyro does so change in motion which could be potentially used to detect IED strikes but it most maintain its change relative to an origin to detect Rollover. Once we acquired the accelerometer (pictured below in

Figure 39) we were able to test the component using simple software to check that it could detect changes in motion as desired.

Figure 38: Original Gyroscope Acquired

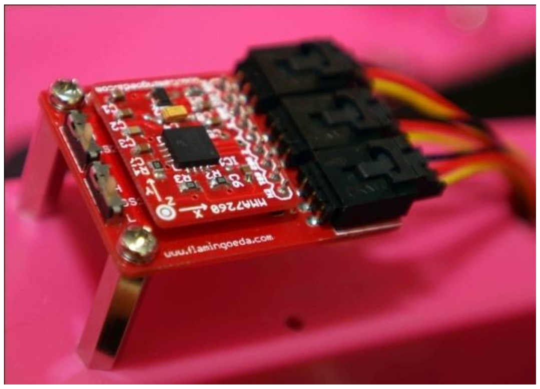


Figure 39: Accelerometer used in the Final System

The USB and Power circuit seen in Figures 29 and 30 do not exist in the final Main Control Unit design. For the USB interface, the final design utilizes a FTDI cable with an embedded RS232 chip. The RS232 chip interfaces directly to the

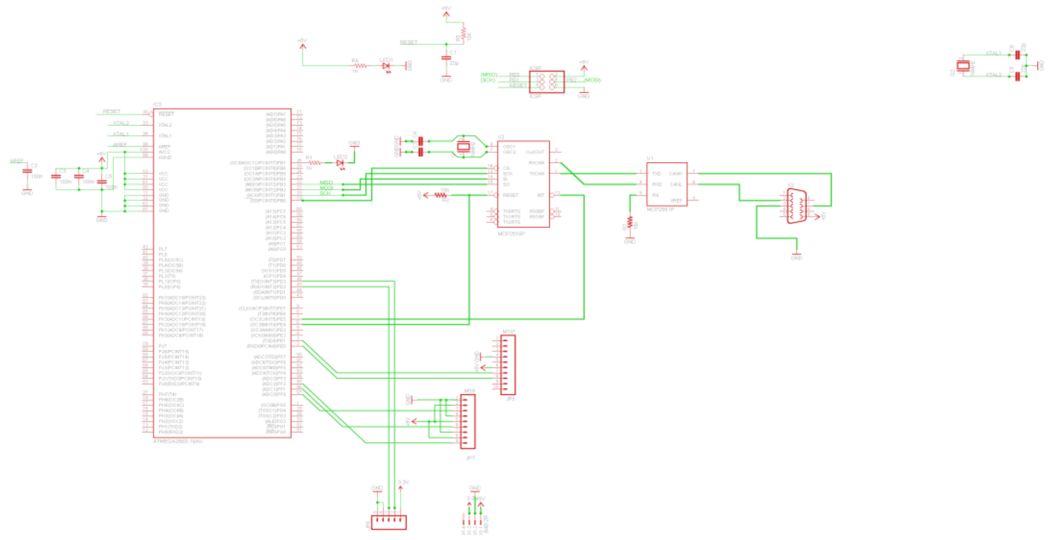


Figure 40: Final Schematic of the Main Control Unit

Rx/Tx lines of the Atmega chip. Above in Figure 40 is the final schematic of the Main Control Unit. This design utilizes break-out boards for the GPW and Wireless that connect to the PCB via headers.

5.3 MODULE C – CAN

The CAN controller is not programmed like a common microcontroller. It instead has a set run routine and its functionality is dictated by values written into various registers via SPI communications. Due to this, no software routines will be written for this chip specifically but rather it will be interfaced with the sensor microcontroller and functions will be written for the ATmega which will serve to operate the MCP2515 via SPI.

There are 18 pinouts of the PDIP package of the MCP2515. Table 27 contains the description of these pins as well as if they will be used in this project.

Pin	Name	Function
1	TXCAN	CAN-bus Transmit, connected pin 4 MCP2551
2	RXCAN	CAN-bus Receive, connected pin 4 MCP2551
3	CLKOT/SOF	Clock output, Unused
4	TX0RTS	Transmit buffer 0 request to send, Unused
5	TX1RTS	Transmit buffer 1 request to send, Unused
6	TX2RTS	Transmit buffer 2 request to send, Unused
7	OSC2	Oscillator Output, Connected 16Mhz Osc
8	OSC1	Oscillator Input, Connected 16 Mhz Osc
9	Vss	Ground
10	RX1BF	Receive buffer 1 interrupt pin, Unused
11	RX0BF	Receive buffer 2 interrupt pin, Unused
12	INT	Interrupt pin, connected pin 4 4043
13	SCK	SPI clock input, connected SCK Atmega
14	SI	SPI data input, connected MOSI Atmega
15	SO	SPI data output, connected MISO Atmega
16	CS	SPI select line, connected digital I/O Atmega
17	RESET	Connected high to prevent reset
18	Vdd	5v input

Table 27: MCP2515 Pinouts

The transmit flow based upon registers set via SPI will show the registers needed to be altered by the ATmega328P in order to send messages over the CAN-bus. The first function needed is to fill the desired transmit registers with the message ID, message data, whether or not to request receipt of message confirmation, or if the message is a remote transmit request. Once this has been done the sensor MCU sets the TXBnCTRL.TxREQ bit to 1. The MCP2515 than knows that the user has requested to send a message and it clears bits related to the previous transmit cycle. The CAN controller than checks the bus for a node currently transmitting if the bus is currently occupied it than waits for an opening and checks to see if the user cancels the message by either setting the CANCTRL.ABT bit to 1 or if the TxBnCTRL.TXREQ bit is set to zero. If the bus becomes available and the message has not been canceled it checks the TXBnCTRL.TXP field of the message id in buffers zero and one (if two messages have been loaded into the different buffers) and determines which one is the highest priority. It than attempts to transmit the message over the BUS engaging in arbitration with any other nodes currently attempting to transmit as described in section 4.4. If the message is sent successfully the device sets the TxBnCTRL.TXREQ bit to zero. For the purposes of this project there is no need for an interrupt once a message has been successfully transmitted so CANINTE.TXnIE will always be zero. Finally the device will set CANINTE.TXnIF and return to normal operation. If however when the message is transmitted there was an error due to lost arbitration the device will set TxBnCTRL.MLOA and attempt to retransmit the message so long as the TXBnCTRL.TXREQ bit is still set. If however the reason the device failed to transmit due to a message error the controller first sets TxBnCTRL.TXERR than determines if the CANINTE.MEERE is set. For this project an interrupt will not be generated when a message error occurs, therefore the CANINTE.MEERE will always be zero. Lastly the device will set CANTINF.MERRF and return to normal operations.

When receiving a message the controller goes through a very similar process as to when it transmits. When the receiver detects the start of a message it beings loading it into a buffer called the message assembly buffer or MAB. After the receiver detects the termination bits it ensures the message was valued. If the message is invalid it generates an error frame. If the message is instead valid it checks against a filters preloaded into the RXB0 register. If the message is passed by the filter the controller checks to see if the CANINTF.RX0IF is set. When zero, this indicates that the receive register is capable of receiving the message. If one the controller checks to see if it is allowed to store the message in the RXB1 register via the RXB0CTRL.BUKT. If this is not allowed then it generates a buffer overflow error and sets the EFLG.RX0OVR bit. The controller then checks to see if an interrupt should be generated by observing the status of the CANINTE.ERRIE bit. This project will not utilize this interrupt so the bit will be set to zero in the initialization of the CAN controller at the startup of the sensor MCU. If earlier the RXBO register was empty the message will be moved into the RXB0 register, CANINTF.RXOIF bit is set and RXBOCTRL.FILHIT is changed to reflect which filter passed the message. It will then check CANINTE.RX0IE bit. This bit controls weather an interrupt is sent when a message is successfully received this will be utilized for this project so it will always be 1. In this case an interrupt will be generated on the INT pin of the CAN controller; this will be stored on an external memory device to enable the microcontroller to quickly assess whether a message is waiting to be retrieved from the CAN controller. After this interrupt is generated it then sets the CANSTAT bits to determine which receive register the message is located in and then checks the BFPCTRL.B0BFM and BF1CTRL.BOBFE bits are set and if so it sets the RXBF0 bit low and if not it returns to start to await receiving a message. If in the beginning the RXBO filters rejected the message the controller checks to see if it meets one of the RXB1if not the controller returns to start to await another message. If however the message passed one of the RXB1 filters or was rolled over from RXB0 receive buffer the CAN controller checks if the register is full via the CANINT.RX1IF. If the bit is set meaning the register is full an overflow error is generated and it once again checks to see if an interrupt should be generated, which will not be done for this application. If the register is empty and the CANINT.RX1IF is set low then the message is moved into the RXB1 register and CANINTF.RX1IF is set high. The RXVOCTRL.FILHIT bits are set to inform the microcontroller which filter passed the message and it checks the CANINTE.RX1IE bit. This bit governs interrupts for successful receipt of messages in the RXB1 register this will also be used in this project and will utilize the same interrupt storage as the RXB0 register once it is generated by the controller on the INT pin. After the interrupt the CANTAT bits are set to inform the microcontroller that the RXB1 register received the message that generated the interrupt. The CAN controller then checks if BFPCTRL.B1BFM and BF1CTRL.B1BFE are set high and if so it sets RXBF1 low and returns to start. If the bits are zero than the CAN controller returns to start to await another message.

The MCP2515 is only able to generate CAN protocol messages, it however is not intended to be the physical layer link to the CAN-bus. For this purpose Microchip the makers of the MCP2515 recommend the MCP2551 CAN transceiver. This chip will take the formatted CAN compatible messages sent to it from the MCP2515 and allow it to safely connect to the CAN-bus. This will regulate the transmission and receive voltages and allow the system to operate within the ISO 11898-2 guidelines for both message protocols as well as the physical layer. Below, Table 28, show the pinouts and the usage of the MCP2551 for this project.

Pin	Name	Function
1	TXD	CAN transmit (from controller) Connected to pin 1 MCP2515
2	Vss	Connect to ground
3	Vdd	Supply voltage, 5v
4	RXD	CAN receive (to controller) Connected to pin 2 MCP2515
5	Vref	Reference CAN output voltage, unconnected
6	CANL	Low voltage CAN-bus Connection
7	CANH	High voltage CAN-bus Connection
8	Rs	Slope control resistor used to select transmit modes. 10K resistance for High speed

Table 28: Pin Definitions and Connections MCP2551

The Milcan A specifications require the MIL-C D38999/ffeA98zN Series 3 connector with shell size A and series 9 (3 pin). The cable for such a system is defined to be any ISO 11898-2 compatible line. For this S CB 626 from SAB North America is one of the few products for this purpose offered in the United States.

5.3.1 CAN REVISIONS

The CAN connection shown was designed and tested with the system; however the code required to run the module, delayed sensor sampling to an unacceptable level. It was determined that the CAN communications module could be substituted with a single interrupt line connecting both of the processors. In order to make CAN communications with this module a viable option a more powerful processor would have to be chosen. The interrupt is falling edge triggered and the signal is carried over pin 6 of the 9 pin D-sub connector.

5.4 MODULE D – POWER CONTROL SYSTEM

The power control system will consist of the LM2940CT-5, LM1117T-1.8 and LM1117T-3.3 low dropout voltage regulators for 5v, 1.8v and 3.3v supply voltages. Due to the existence of a 6v DC battery on the vehicle the connection to the voltage regulators can be direct. Figure 41 below shows the TO-220 package and pinout for the LM2940, and Figure 42 shows the package and pinout for the LM1117. Both images are front view.

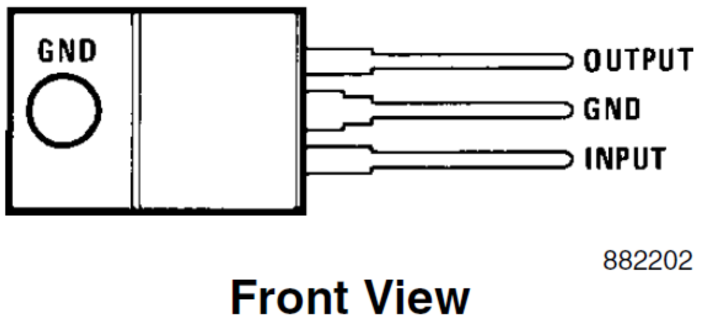


Figure 41: LM2940 Pinout
(Permission from National Semiconductor needed)

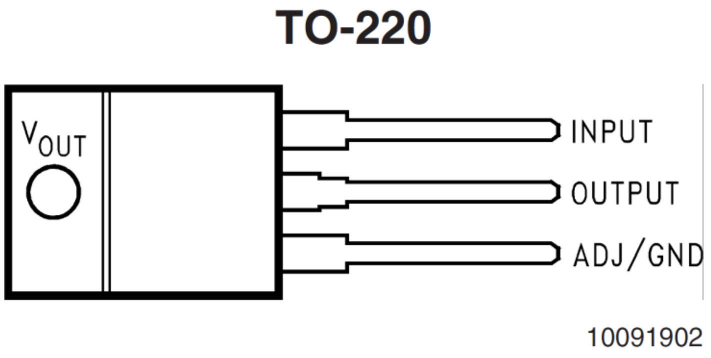


Figure 42: LM1117 Pinout
(Permission from National Semiconductor needed)

Schematic (Figure 43) shows the 5v and 3.3v regulators that will be used for to power the sensor units and main processor. While only one 5v connection to the sensor unit is shown for every additional sensor unit another 5v regulator must be added to power it.

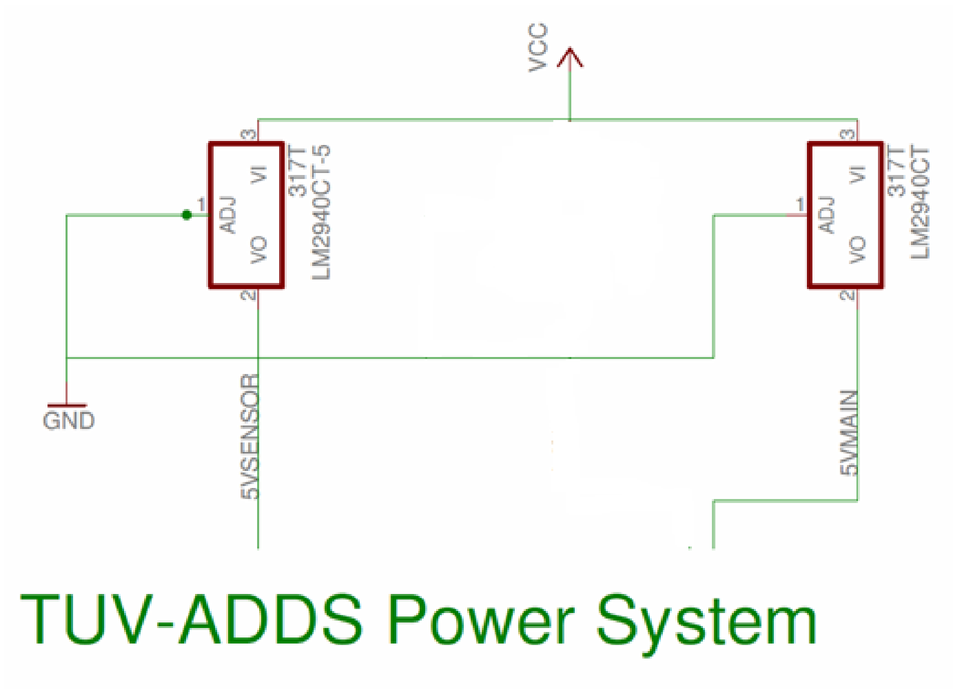


Figure 43: Power System Schematic

5.4.1 POWER CONTROL SYSTEM REVISIONS

The power system was revised to use a separate 9 volt 1.6 amp hour battery due to noise caused by the power-wheels motors on the on-board 6 volt battery. We also utilized only one 5v regulator and one 1.8 volt regulator as it was found that the X-bee module can provide the needed 3.3 volts. As can be seen in the revised sensor MCU schematic (Figure 40) the 1.8 volt regulator was moved to the sensor MCU board.

5.5 MODULE E – SOFTWARE

The TUV-ADDS system will exclusively utilize Atmel processors, all of which can be developed in the Arduino Integrated Development Environment. The Arduino language is a C language derivative blended with aspects of Object-Oriented Programming. The language it not designed to interact with classes per se, but it can work with many different types of generic objects, such as Strings. All of the control statements and data types match the equivalent statements from the C programming language. The software flow must follow a three step process which can be seen below in Figure 44.

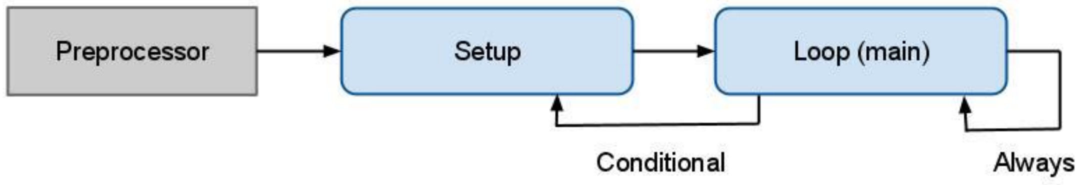


Figure 44: Three Step Process Software Flow

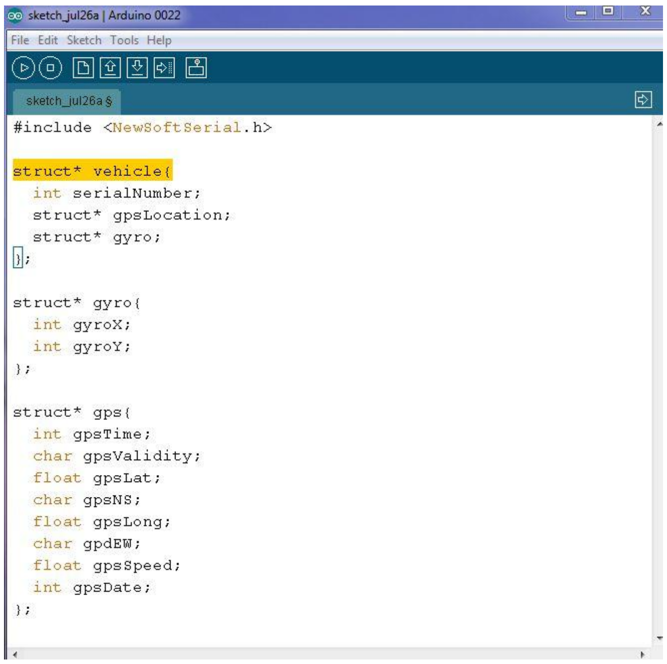
5.5.1 MAIN PROCESSOR

Each vehicle that contains the TUV-ADDS system will be assigned a unique serial number which will be embedded during construction. This serial number cannot be changed without a direct re-programming of the processor. This serial number will allow the command center to differentiate between all of the different vehicles and their specific properties. Originally the plan was to assign specific characteristics of a certain vehicle a pattern in the serial number. For example, if a vehicle would always be a lead in the convoy then the serial number would have a ‘01’ in the sequence. It was then decided to change this so that any enemy interference would not be able to decode and recognize any patterns. The serial number will be assigned to a non-volatile integer data type as a global variable definition. The serial number will also be stored in each of the sensor module so that in the case that a vehicle is mostly destroyed, there is a chance that one of the systems containing the serial number will survive.

The preprocessor of the main body of the software will contain a large set of globally defined variables (see Table 29) as well as the structures that are used in many of the methods. This setup of having all methods potentially able to access all variable fields and structures reduces the complexity of the methods. All of the structures will actually be pointers to structures; this will also greatly decrease complexity by only having to pass the pointer address instead of a series of variables or an array. This setup will also help with making vital decision on whether or not the system is detecting distress. Some of the sensors offer more than just the basics, which methods not associated with that component can utilize. Also defined in the preprocessor of the program are all of the libraries used as well as any constant definitions. Figure 45 below shows a basic preprocessor coded in the Arduino IDE.

Variable Definition	Purpose
int gpsTime	Stores the time of satellite fix.
char gpsValidity	Stores whether or not the position is valid, or there is a warning
float gpsLat	Stores latitude position as degrees and minutes
char gpsNS	Stores ‘N’ for North or ‘S’ for south
float gpsLong	Stores longitude position as degrees and minutes
char gpsEW	Stores ‘E’ for East or ‘W’ for West
float gpsSpeed	Stores speed over the ground
int gpsDate	Stores the date when signal was fixed
int gyro	Stores the gyroscope x-position
int gyroY	Stores the gyroscope y-position
int gyroTime;	Stores the time at which a reading was taken

Table 29: List of All Globally Defined Variables



```
#include <NewSoftSerial.h>

struct* vehicle{
  int serialNumber;
  struct* gpsLocation;
  struct* gyro;
};

struct* gyro{
  int gyroX;
  int gyroY;
};

struct* gps{
  int gpsTime;
  char gpsValidity;
  float gpsLat;
  char gpsNS;
  float gpsLong;
  char gpdEW;
  float gpsSpeed;
  int gpsDate;
};
```

Figure 45: Basic Preprocessor Coded in the Arduino IDE

The Arduino IDE is a copyright of Arduino. Permission under Fair Use.
The purpose of the next section of software is to start connections, initialize, and do any other tasks before going in to the main part of the code. The software will first start all serial connections to components including the wireless, CAN module, and the gyroscope. From there it will configure all components by sending a response request. Once the software receives the response, it will move to the next component. A simple diagram of this process can be seen below in Figure 46. This processes not only checks that all systems are running properly, but it will also fulfill another tasks required to be performed in the setup stage. This requirement is that all initial senor and position reading will be taken and stored as global variable. These values will be what is referenced later to check if a new value is caused by a distress characteristic. Once all of the initial values are stored, the setup section will then send to the command center the serial number and the current position on the vehicle.

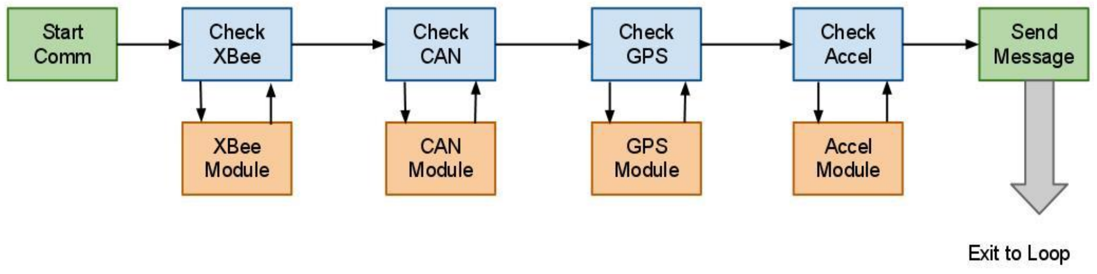


Figure 46: Configure all Components by Sending a Response Request

The main part of the software is the Loop section. This section will run non-stop unless told to do otherwise. It is possible to leave the loop and re-run the Setup section, and in fact our software will do so. In general the loop will check data sent from the various sensors and check them against previous values. Each section will be required to send current values then the software will compare those with previous values. If the data only changes by a small amount (below distress threshold or no change) then the software will update these as the new 'standard' to compare to. The other possible situation is that the incoming data is much higher than threshold, which may show characteristics of distress. This does not necessarily mean that distress is occurring. Only the situation where multiple data readings, such as both the accelerometer and the temp sensor, shows increases above threshold will the system go into distress mode. Distress mode will shut of all subsystem and send the distress signal to the command center. A detailed display of the grab and check process for the TUV-ADDS system can be seen below in Figure 47. Each of the squares represents a function that will be called from the main of the software.

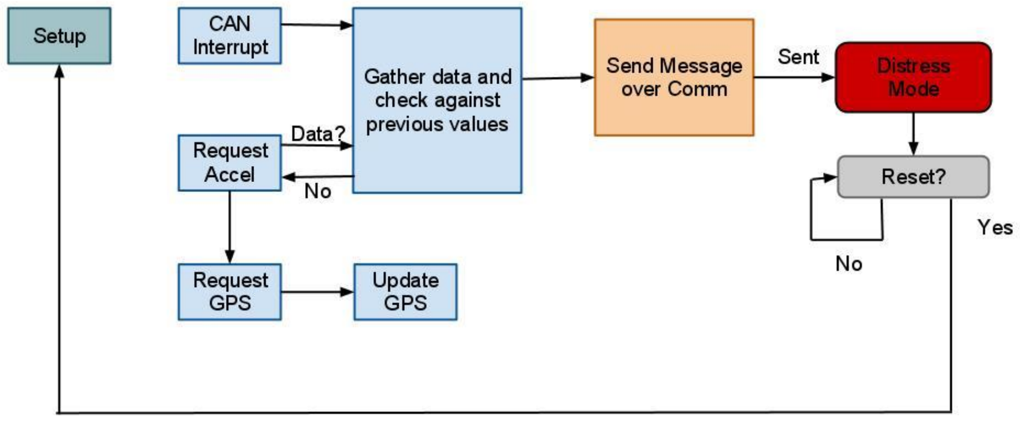


Figure 47: Display of the Grab and Check Process

All Method Prototypes:
int acquireGPS(struct gps* mainGPS);
void updateGyro(struct* gyro);
int commDistress(struct* vehicle, int distress);
int commGPS(struct* vehicle);
int compare(int oldValue, int newValue);
void distressMode(void);

5.5.1.1 WIRELESS

Wireless Method Prototype: int comm(struct* vehicle);

The basic design of the TUV-ADDS system will require a unidirectional communication service, meaning that the command center cannot send any information to the vehicle. If time permits, there are several features that could be added to the command center that would require a bidirectional communication system. The hardware design for the communication system is flexible enough to allow the expansion to include such features. From this point, the design of the software will only consider the unidirectional system.

During the setup stage of the software, the XBee device has to be calibrated and a network has to be defined. The device is calibrated by sending a request through the device line. After a few seconds the device will return a character, letting the software know that it is in command mode. Once in command mode, the software will set the Personal Area Network Identification (PAN ID) number. This is achieved by sending the XBee “ATID3330” where the 3330 is the PAN ID. This value can be anything between 0x0 and 0xFFFE in hexadecimal. Next step is to set the Destination High to ‘0’ which selescts 16-bit addressing mode. The Destination Low, which is the 16-bit address, can be anything between 0x0 and 0xFFFE in hexadecimal. Last the software will send a ‘ATCN’ to the XBee which will exit Command Mode. The device will return a character to signify that it has exited command mode.

```
void setup () {
            Serial.begin(9600);

            Serial.print("X"); // This clears buffer
delay(1100);

            Serial.print("+++");
            delay(1100);
```

```
   if (returnedOK() == 'T') {
                    // if an OK was received then continue
 }
 else {
                            setup(); // otherwise go back and try setup again
 }


Serial.print("ATID3330,");
Serial.print("DH0,");
Serial.print("DL1,");
                    Serial.println("CN");


                    if (returnedOK() == 'T') {
                          // if an OK was received then continue
                    }
 else {
                             setup(); // otherwise go back and try setup again
                    }
}
```

This method will require the pointer to the vehicle structure as an argument. This requirement allows the method access to all of the vital information without having to passes every individual variable in as an argument. Once the method is called it will create a temporary packet buffer (see Table 30) by dynamically allocating memory. It will then run through the vehicle structure and call a toString() function on the variable contents. This toString() function will take any generic data type and turn it into a string data type. After all of the elements of the buffer are converted to a string, then the buffer becomes the message packet. This packet can be sent using the Serial.write(buffer) command. Once the message has been sent, the method will free all of the dynamic memory allocation, and will return a ‘1’ indicating that the message has been sent, then terminate.

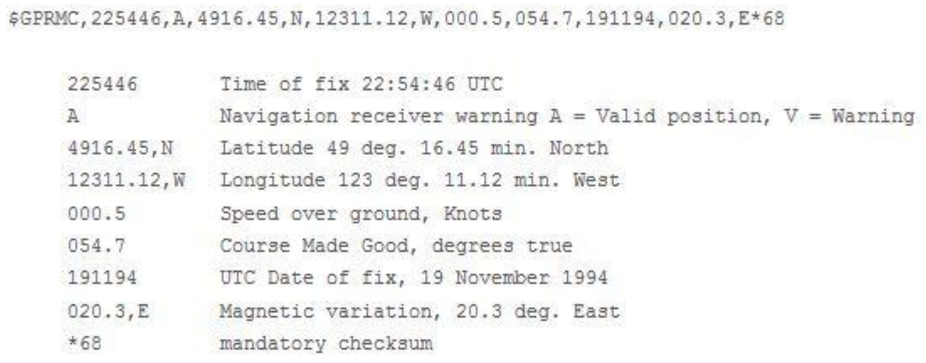
Variable Definition	Purpose
struct* vehicle messageBuffer	Allows data packaging for wireless transmission

Table 30: Wireless Communication Variables

5.5.1.2 <u>GPS</u>

GPS Method Prototype: int acquireGPS(struct gps* mainGPS);

The GPS module can send several different messages varying only by the protocol desired. The Helical GPS antennae by AGH Technology Co. Ltd. can send messages in all major protocols including $GPRMC (recommended minimum specific GNSS data). Included in the message is time, status, latitude, longitude, speed over ground, and date. All of these parameters are desired for use in the system. Figure 48 below is a sample of a message and how it can be broken down.



```
$GPRMC,225446,A,4916.45,N,12311.12,W,000.5,054.7,191194,020.3,E*68

     225446       Time of fix 22:54:46 UTC
     A            Navigation receiver warning A = Valid position, V = Warning
     4916.45,N    Latitude 49 deg. 16.45 min. North
     12311.12,W   Longitude 123 deg. 11.12 min. West
     000.5        Speed over ground, Knots
     054.7        Course Made Good, degrees true
     191194       UTC Date of fix, 19 November 1994
     020.3,E      Magnetic variation, 20.3 deg. East
     *68          mandatory checksum
```

Figure 48: Sample of a Message
(Printed with permission from Arduino)

The method is defined to have an integer return data type. This is simply to notify that the whether or not the GPS acquisition was successful. If a ‘1’ is returned then the acquisition was successful, while a return of ‘0’ means the processes was unsuccessful. The method will also require, as an argument, a pointer to a single global variable with a GPS structure data type. Once the method is called it will then request a message from the GPS unit. It will wait until the GPS unit sends a message back. At that point, it will check that the GPS reading is valid. If it is not, the method will exit returning an error. A valid message will be parsed and stored into the correct temporary variable (see Table 31). The method will then compare all temporary variables against the global GPS variables. This check will make sure of any extraneous errors. Once the check is complete the global variables will be updated, and then the method will return a ‘1’ meaning no error has occurred.

Variable Definition	Purpose
int tempGpsTime	Stores the time of satellite fix.
char tempGpsValidity	Stores whether or not the position is valid, or there is a warning.
float tempGpsLat	Stores latitude position as degrees and minutes
char tempGpsNS	Stores ‘N’ for North or ‘S’ for south
float tempGpsLong	Stores longitude position as degrees and minutes
char tempGpsEW	Stores ‘E’ for East or ‘W’ for West
float tempGpsSpeed	Stores speed over the ground
int tempGpsData	Stores the date when signal was fixed

Table 31: List of Variables Utilized in the GPS Method

5.5.1.3 GYROSCOPE

Gyroscope Method Prototype: void updateGyro(struct* gyro);

This method will send a request to the on-board gyroscope and wait for the response. A nice parameter that the gyroscope chip offers is that all position fields have separate analog lines. The ATmega has ten analog-to-digital (A/D) converts available for the set of analog pins (ten analog pins in total). The analog signal sends the position as raw data. As is, the information is useless. It must be converted to a readable angle then stored into the appropriate variable defined for the method (see Table 32). This is done by referencing the initial reading and comparing it to the offset, which is a constant for the device. This can all be encapsulated into a single equation as seen below. The other aspect of the method includes a timestamp of when the reading took place. This is simply done by storing the value pulled from the globally defined time field. This feature is vitally important for recognizing the characteristics of roll over.

gyroX = ((analogRead(gyroXpin) * gyroVoltage) / 1023) – gyroZeroVoltage) / gyroSensitivity;

Variable Definition	Purpose
int tempGyroX	Stores the gyroscope x-position
int tempGyroY	Stores the gyroscope y-position
int tempGyroTime;	Stores the time at which a reading was taken

Table 32: Gyroscope Method Variables

5.5.2 COMMAND CENTER

The command center software will be developed with Java so that any computer with the Java Virtual Machine can run it. This is not the only advantage of this decision. Java is also great for developing Graphical User Interfaces (GUI). Java is also flexible with communicating with ports. The Command Center software will have three separate classes, each of which will run an important function described above by utilizing several of Java's built in libraries (see Table 33 below).

Library	Description
java.io.*	Contains all GUI components
javax.swing.*	Help make GUI and adds flexibility to interfacing
java.io.*	Contains streaming parameters
java.util.*	Contains a large collection of random classes to facilitate all types of situations
gnu.io.*	Collection of serial and parallel comm under the GNU LGPL licenses agreement

Table 33: Java's Built in Libraries

The first class is will contain all of the structure involved with the software GUI. This class will use many different fields (as seen in Table 34) to display and control the vital data. The class will be sent all of the required data values from the main function. All of the fields except for the JButton are just display fields, so they have no control at all. The JButton will be programmed to disconnect the software from the serial port and close the entire GUI. Once this class is called from main with all of the vehicle data as arguments, the data will be checked against previous values and if decided to change, the new values will be displayed. A general GUI prototype can be seen below in Figure 49.

Field Name	Description
JField userName	Displays the User’s Name
JField panID	Displays the network Personal Area Network ID
JField date	Displays the date of GPS acquisition
JField time	Displays the time of GPS acquisition
Canvas warning	A small square which will change colors with an incoming distress signal
JField distress	Display which kind of distress is happening
JField vehSN	Displays the vehicles serial number
JField gpsLat	Displays the up to date vehicle Latitude position
JField gpsNS	Displays whether the Latitude position is ‘N’orth or ‘S’outh
JField gpsLong	Displays the up to date vehicle Longitude position
JField gpsEW	Displays whether the Longitude position is ‘E’ast or ‘W’est
JField vehSpeed	Displays the most current vehicle speed relative to ground
JButton exit	Will exit the TUV-ADDS Command Center program

Table 34: Structure Involved with the Software GUI

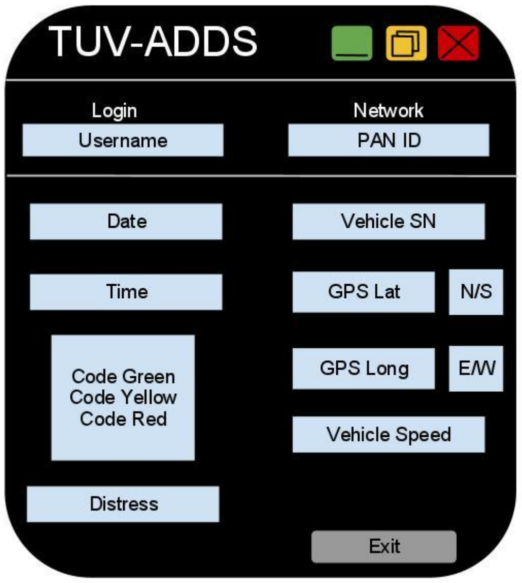
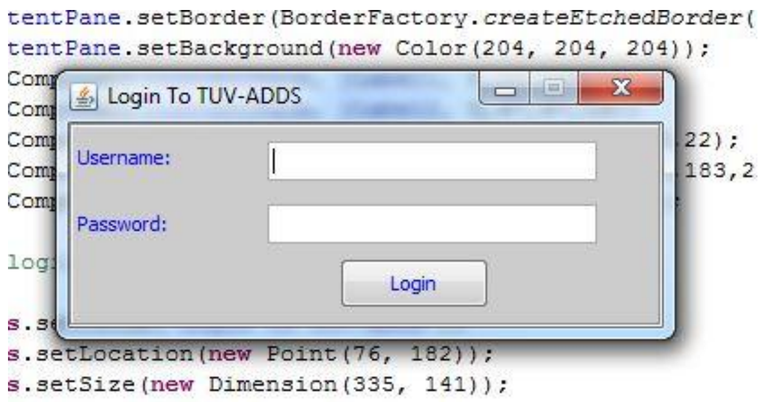


Figure 49: GUI Prototype

As small feature that will be added to the Command Center software is a basic username and password login prompt. This software was developed earlier as a side project in Java class. The login class will prompt a window with a text field available for both a username and a password (see Figure 50 below). If either of the two are incorrect or missing, an error message will be prompt and both fields will be erased (see Figure 51 below). Once both fields have the correct username and password, the login class will call up the class involved with the main Command Center.

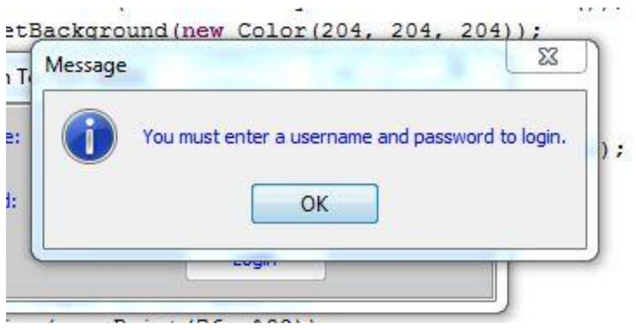


```
tentPane.setBorder(BorderFactory.createEtchedBorder(
tentPane.setBackground(new Color(204, 204, 204));
Com
Com
Com                                                  22);
Com                                                  183,2
Com

log

s.s
s.setLocation(new Point(76, 182));
s.setSize(new Dimension(335, 141));
```

Figure 50: Window for both a Username and a Password



```
etBackground(new Color(204, 204, 204));
```

You must enter a username and password to login.

Figure 51: Error Message

5.5.3 COMMAND CENTER MODEM

The command center modem is no more than a junction or gateway between the vehicle and a computer. The software for the gateway will wait until there is an incoming message from the vehicle. Once the message is received it will then redirect the message to the port of the processor that is connected to the USB adapter. The two serial set up utilize the New Soft Serial library for the Arduino IDE. This library allows any port (excluding the analog ports) of the Atmel to become a serial adapted port. The only requirement of the software created serial port is that it must have the exact baud rate as the hardware designated serial port. An example code below shows how a NewSoftSerial library code could be used. The software will not change or convert any of the messages in any way. It seems that the XBee wireless adapter could be directly connected to the USB adapter but the XBee goes into stand-by mode after a short period of time (this is to keep power consumption down). Once the software receives a message it can trigger the XBee to warm up then send the message.

```
#include <NewSoftwareSerial.h>

NewSoftSerial usb = NewSoftSerial(4,3); // RxTx connect to pins 4 and 3


void setup(){
        Serial.begin(9600); // Hardware designated serial ports
        usb.begin(9600); // Software designed serial ports
}

void loop(){
        if(Serial.available()){ //If there is an incoming message
                usb.write(Serial.read());
        }
}
```

5.5.4 SENSOR MCU

Below in Figure 52 the MCU program flow is shown. Upon startup the microcontroller must first initialize all of the variables to be used throughout the program and initialize SPI communications as well as define the transmit speeds for the various communications requirements. After all of the basic functions have been started the controller must then select to communicate with the CAN controller and program the bus speed into the controllers registers and send any required messages to the main processor to ensure it is ready to receive data. That ends the startup procedure. Now the program will arrive at the main loop where most of the sensor units' functionality lies. The first task of the controller will be to take sensor data and monitor it for abnormal values. If one is detected the program will then devote all of its resources for three seconds to logging data from all of the sensors to record the event as completely as possible. Once this is done it will read the stored data for each sensor and determine if the event was an obvious false alarm. If so it returns to the beginning of the loop. If the event is determined to be legitimate than the device reports all relevant data to the main processor and returns to the start of the main loop. If in the normal sensor read function no event is detected than the program proceeds to check the CAN controller for a received message. If none is detected it returns to the start of the main loop. If there is a message on the bus it will receive it from the CAN controller and decode what the message is asking for. The processor will then service this request and then return to the beginning of the main loop.

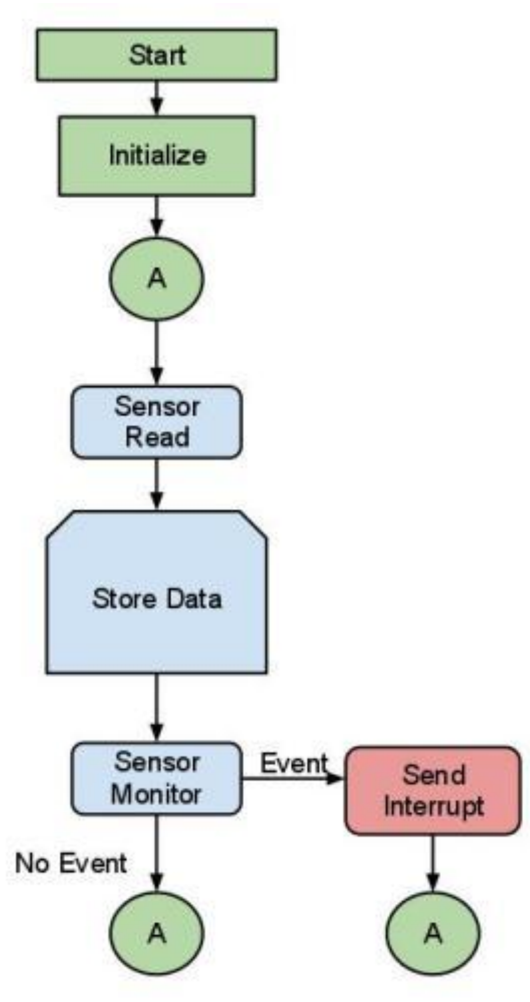


Figure 52: Sensor Unit Program Flow

A function is needed to send timestamp requests to the main controller in the event that the automated timer update message is missed. This function will take the form of CANTimeUpdate() and will generate a RTR Frame to be sent over the CAN-bus with a 1byte data field value of 00FF. This function will return the current system timestamp.

In the event of a general microcontroller error that causes either the loss of data logging or sensor input without triggering an event flag the controller needs a way to report this to the main processor and will do so with the function CANReportStatus(ErrorFlag). This function is input an error flag associated with the specific controller issue by calling an error monitor function. If the monitor function finds functionality returned it will call CANReportStatus(0000) which will tell the main controller the sensor MCU is once again functional via the CAN-bus.

If an event is flagged during operation or a request for stored data is recived from the CAN controller a function will be needed to generate the data frames necessary to send all of the required data to the main controller for processing. This function will take the form CANReportData(TimeX,TimeY,SensorFlag) where it will send the data logged for TimeX to TimeY for all sensors. If the special flag is set it will send the specific data requested be that the timestamp or the data for a specific sensor. This function will return void.

When an event is flagged, a way to prepare the main processor to receive data is needed. After an event is flagged and the data is logged the controller will use a function taking the form of CANReportEvent(EventFlag,SysTime). This will send a CAN data frame containing the time of the logged event so the main controller can prepare to handle the data. This function will return void.

The controller needs a function to handle the requests made by the main controller over the CAN-bus. This function will take the form CANRequest() and will be called via an interrupt generated by the CAN controller or during the main loop to monitor if any messages have arrived over the CAN-bus. This function will interface with the CAN controller to see what data or request was sent, be it an automatic system timer update or request for all of the flash sensor data for the last two seconds. It will decode the message and call the necessary functions based upon the nature of the CAN message and return void.

In the normal main loop a function to poll the sensor data, check to ensure that an event has not occurred, and log the data in the external memory must exist. This function will take the form of SensorReadNorm(). It will in turn pull data from all of the external sensors and compare it with the stored SensorValX value. If the sensor shows a higher than normal value the Event(EventFlag) function will

be called and it will indicate the sensor that caused the event in the EventFlag field. If nothing is detected it will finish logging the data and return void.
When the SensorReadNorm() function detects abnormal values the Event(EventFlag) function is called. This function will first call the SensReadEvt(Time) function. Then it will then compare the average logged flagged sensor values to SensorValX and if it still shows above normal values it will compare the remaining sensors in turn and set the appropriate flags for sensors with abnormal values. This function will then call the CANReportFlag() function. Then it will call the CANReportData() function with TimeX to TimeY being one second before the flagged event and three seconds after.

The SensReadEvt() function will be called in Event() it will record data to the external memory device from the sensors with no pre-processing. It will do this for three seconds to capture all relevant event data. When complete the function will return void.

A function will be needed to interface with the external memory in order to write sensor data to it. This function will be DataWrite(SensorValX,SensorXMem) and will be called in SensorReadNorm() and SensReadEvt().

The DataRead(SensorXMem) function will also interface with the external memory and read the data located at the location SensorXMem. This function will be called by Event() and CANReportData().

In order to ensure that no old or random data stored in the external memory a function will be needed to clear the data and prepare the memory space for normal operations. The ResetMemory() function will be called to clear all the old memory during startup and will perform these tasks.

Parameters:
CurrSensVal- A value used to temporarily store the initial data polled from the sensor interface before it is categorized.
SysTime- The current system clock regularly updated via the CAN-bus
TimeX- A value used to flag the starting time value when reading data from the external memory
TimeY- A value used to flag the ending time value when reading data from the external memory
ErrorFlag- A byte used to log various possessor or CAN-bus errors when transmitting this information for further use.
SensorFlag- A value used to determine which sensors will be used for a certain process such as when sensor value is being read or which sensor values to transmit over the CAN-bus

EventFlag- This byte is used to signal which sensors recorded a possible event
Sensor1Mem- This is the memory location being written to or read for sensor 1
Sensor2Mem- This is the memory location being written to or read for sensor 2
Sensor3Mem- This is the memory location being written to or read for sensor 3
Sensor4Mem- This is the memory location being written to or read for sensor 4
SensorVal1- This is the data being written to or read the memory space for sensor1
SensorVal2- This is the data being written to or read the memory space for sensor2
SensorVal3- This is the data being written to or read the memory space for sensor3
SensorVal4- This is the data being written to or read the memory space for sensor4
DeviceID- This byte will be unique to each node on the system and will be used for fielding requests from the main processor.

5.5.5 CAN CONTROLLER

The can controller cannot be traditionally programmed; therefore all of the functions dealing with CAN communications are located in the sensor microcontroller or the main processor. However, there will need to be several CAN message types in order to handle all of the functions required for this project.

The first type of message will be from the main processor to the sensor MCU to request the status of the sensor unit. It will be a request transmit return frame with only one byte of data. It will be a low priority message though if it is unsuccessful in transmit it will attempt to retransmit. For the purpose of this design this message will be referred to as StatusRequest().

A message that reports the status of the sensor unit to the main processor will be needed to not only reply to StatusRequest but also report if there is a possible event. This message will have varying priorities based on the status of the unit at the time. If an event is detected on more than one sensor or the device has determined there is a system error than the message will have very high priority. If an event is detected on only one sensor then the message will have medium priority, and if the sensor has detected no events and is operating normally than the message will have very low priority. The message will be a data frame with two bytes, the first byte will denote the type of error and the second will indicate the time of the error. This message will be referred to as

StatusReport(High,Med,Low). This will also prompt the main processor to be ready to receive data.

The next type of message will be referred to as DataSend(). It will be used to transmit sensor data as well as the timestamp for the data after the sensor MCU reports an event. It will be a data frame consisting of 5 bytes the first will be the timestamp for the data about to be sent, the second will contain the device ID. The next three bytes will be the values recorded for the three individual sensors. This message will have very high priority and will attempt to resend in the event of transmit failure. There also will need to be a way to inform the main processor that the data has finished sending so there will be a short message called DataTerminate() that will be a data frame one byte long and will also have a very high message priority. And will attempt to retransmit on message failure.

The main processor will also need a way to request various types of data from the sensor controller. A message referred to as DataRequest () will handle this function. It will be a data frame containing 4 bytes, the first will be the device ID it is requesting the data from, the second will be the type of data being requested. The third and fourth will contain the start and stop timestamps for the data needed. This message will have moderate priority and will attempt to resend if it fails to transmit.

Another message required will be one that allows all of the nodes to reset their clocks to ensure accurate data collection. This will be referred to as TimeUpdate(). Unlike the other messages this will be a series of messages between the main processor and the sensor unit. The first will be sent from the main processor telling the sensor units that it is ready to do a system wide time update. This will be a data frame containing 1 byte telling the receiving unit this is for time update, with low priority. It will then wait for all of the active sensor microcontrollers to send a message that is a data frame with two bytes, one being time update command byte and other is the device ID. Once all active sensors have acknowledged they are ready to update the time the main processor will then send out one final message that is one byte long containing only the current system time. If the main processor dose not receive a reply from all active nodes within a reasonable time of the first message it will instead cancel the update and retry in a set amount of time. All messages after the first will have moderate priorities.

The final message needed will be a command to reset the system memory. This will be a low priority message that will resend in the event of transmission failure. This message will be referred to as MemReset(). It will be a data frame two bytes long, the first will contain the device ID to be wiped and the second will contain the 8 bit reset command. This will have low priority.

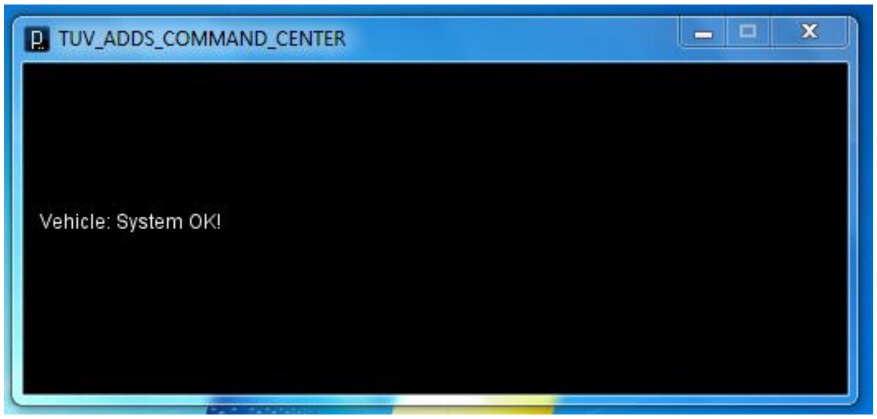
Table 35 below shows the priority of the various messages sent over the CAN-bus with respect to each other. This will determine the outcome of message arbitration and the selection of message ID’s.

Message:	**Priority:**
StatusReport(high)	1
StatusRequest()	2
DataSend()	3
DataTerminate()	4
StatusReport(med)	5
TimeUpdate(Updating)	6
TimeUpdate(Initial)	7
StatusReport(low)	8
MemoryReset()	9

Table 35: Message Priority Table

5.5.6 SOFTWARE DESIGN REVISIONS

The original plan was to program the Graphical User Interface (GUI) using Java in the Eclipse Integrated Development Environments (IDE). Some of the TUV-ADDS Command Center was developed using Java but when the entire system went through several design changes to simplify development, we switched to the Processing language. Processing is an Object-Oriented Programming language that the Arduino IDE was designed off of. Because Arduino is closely tied to Processing, it offers Serial functionality with minimal amount software development. The Command Center is simple in that it only displays (see Figure 53 below) the vehicle’s status and the distress signal with the GPS location and time. The software is available for Windows (32 and 64), Mac OsX, as well as Linux on the Project Website. For demonstration purposes, we ran the software in Windows on a laptop as seen in figure 54.


TUV_ADDS_COMMAND_CENTER

Vehicle: System OK!

Figure 53: Screenshot of the TUV-ADDS Command Center GUI

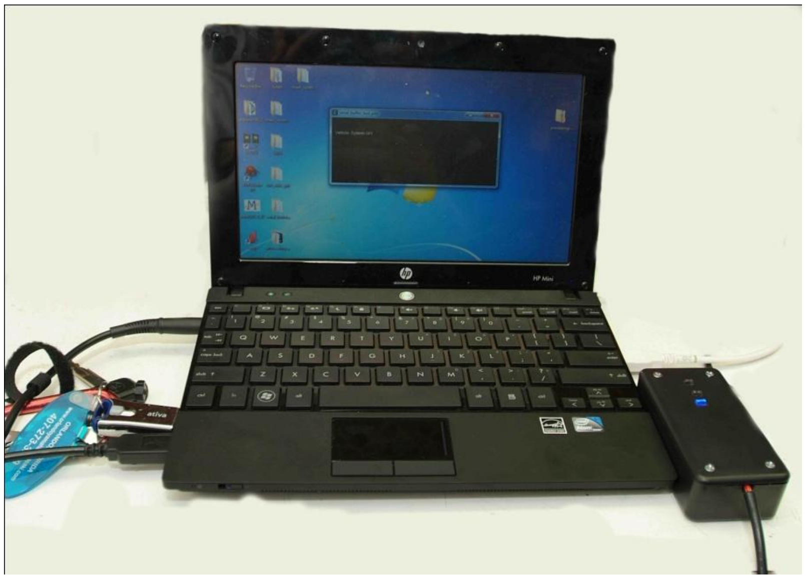


Figure 54: TUV-ADDS Command Center with Modem

The sensor MCU code underwent major revisions to optimize for an acceptable sensor sampling rate within the constraints of the Arduino boot-loader. The only major alteration was removal of the software support for CAN communications. It was with a simple digital output acting as an interrupt for the main processor. This increased the sample rate 4 fold and while the originally desired sample rate of 1ksps was not realized. Through further testing this sample rate was deemed unnecessary for the detection of a simulated explosion. The major remaining shortcoming of the system that prevented 1ksps from being realized is the overhead caused by programming in the non native Arduino environment rather than directly with C. If it was required for real life implementation the group is confident with the proper code optimization that 1ksps could be easily realized on the current hardware.

6 EXPLICIT DESIGN SUMMARY

Early in the development stage, the system was modulized in to separate sub-system (see Figure 55). This was done to ease with development and expedite the acquisition process. The schematics of the system can be seen with Figures 56 - 63 as well as 65. Figures 64 and 66 are software flow diagrams for the system and Table 36-40 go into details about the software.

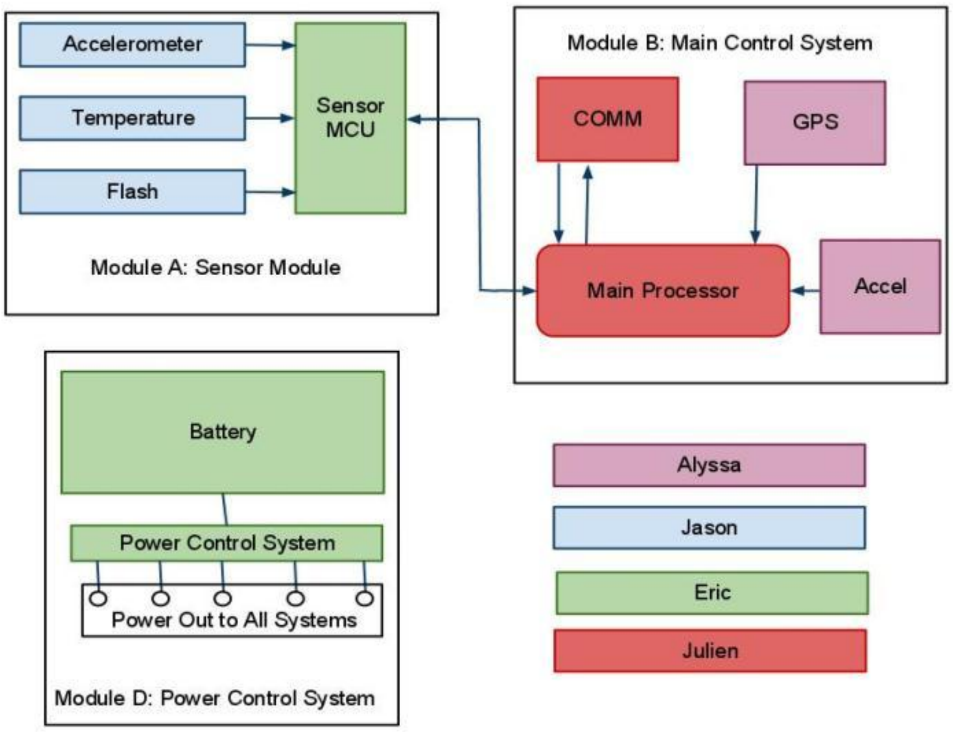


Figure 55: General Hardware Overview Diagram

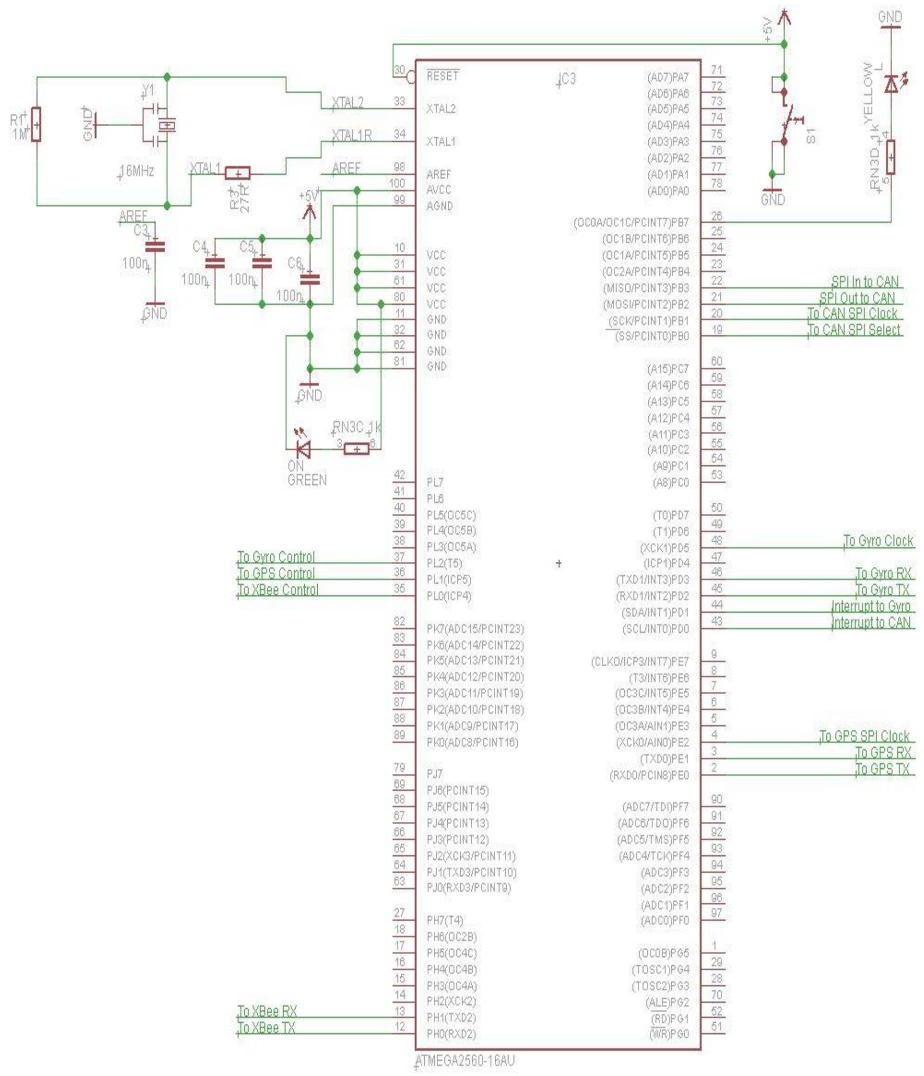


Figure 56: Pin Mapping for the Main Control Unit Processor

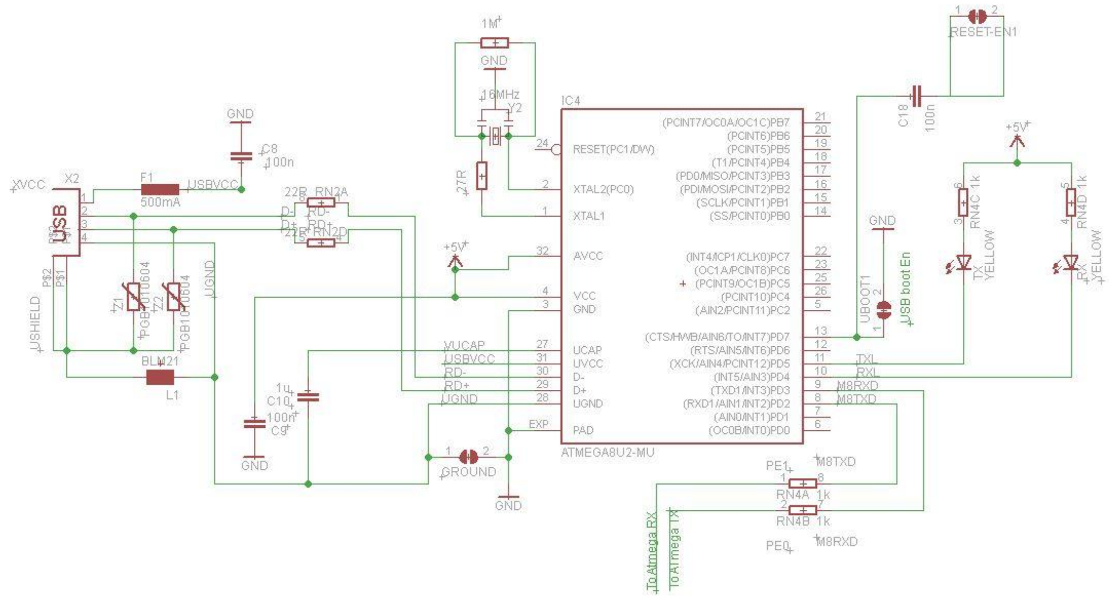


Figure 57: Schematic for the USB Interface on the Main Control Unit

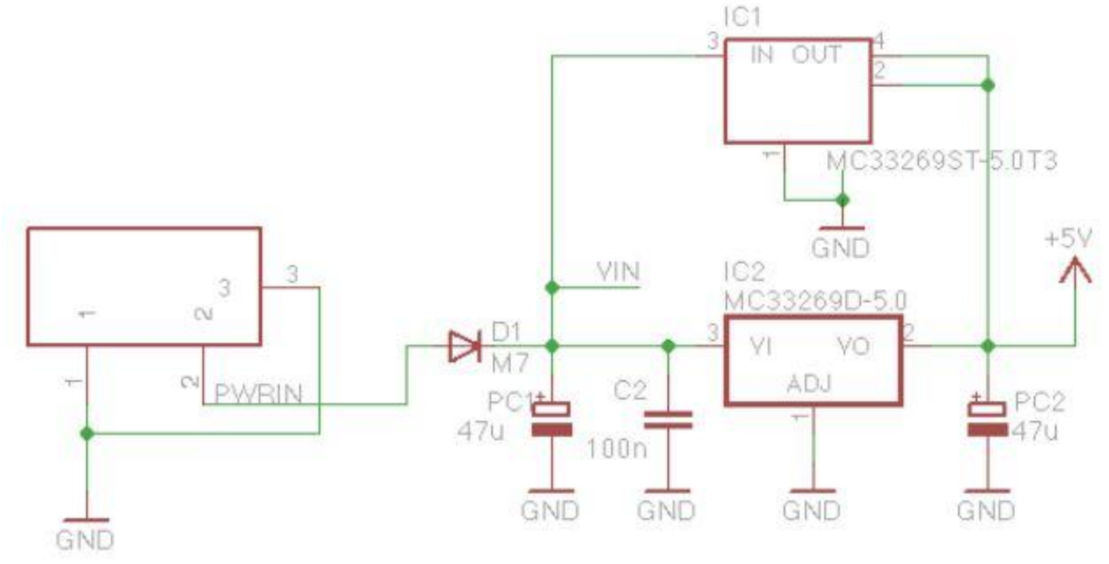


Figure 58: Power Stability Design for the ATmega on the Main Control Unit

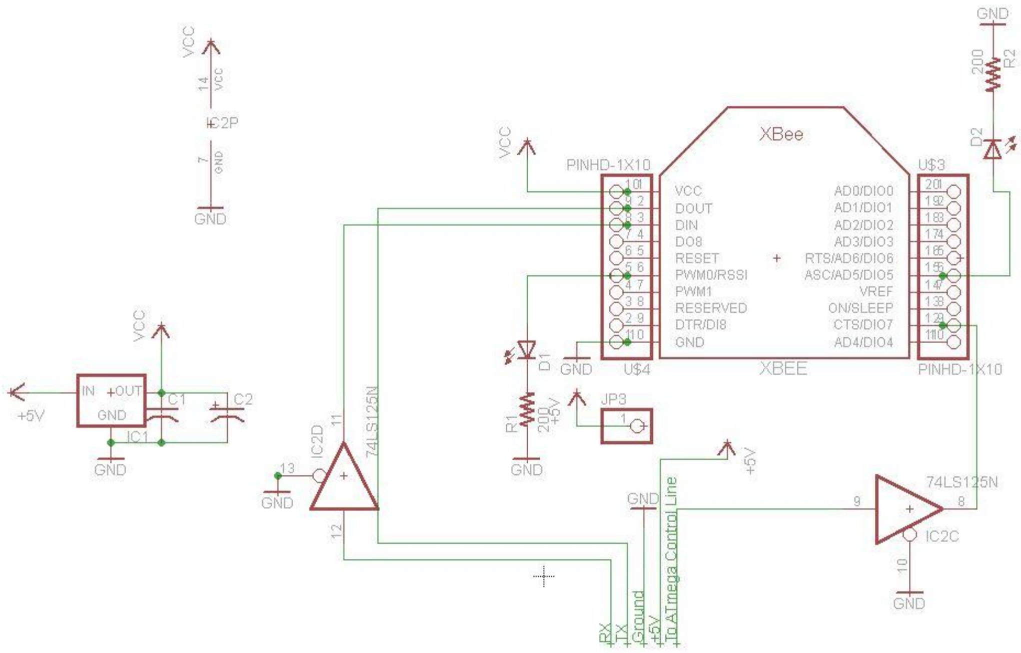


Figure 59: XBee Adapter on the Main Control Unit

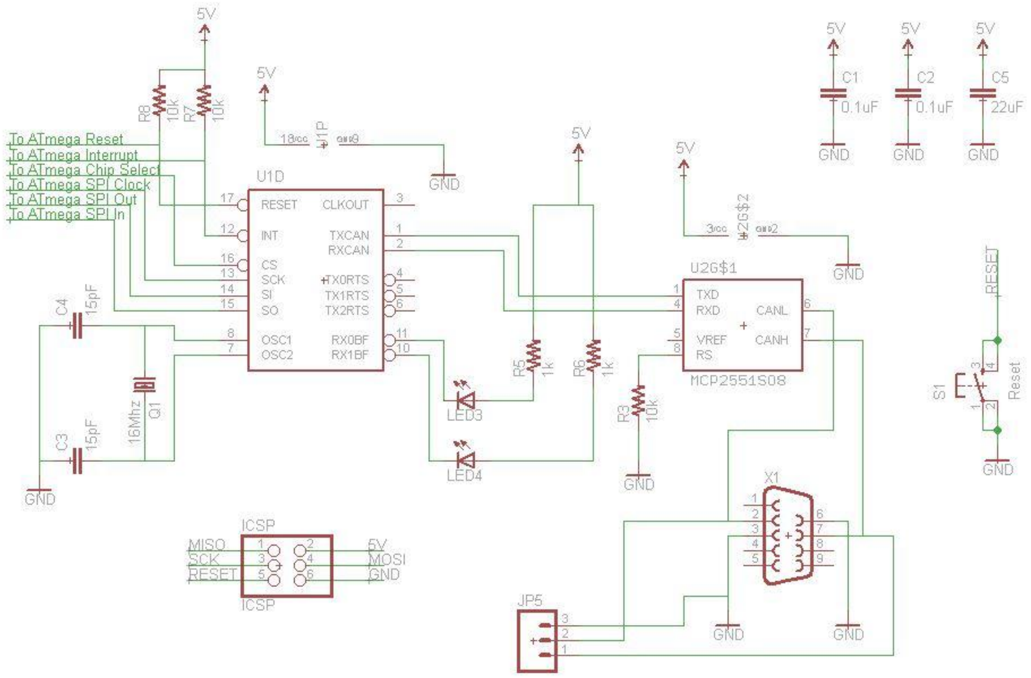


Figure 60: CAN interface on the Main Control Unit

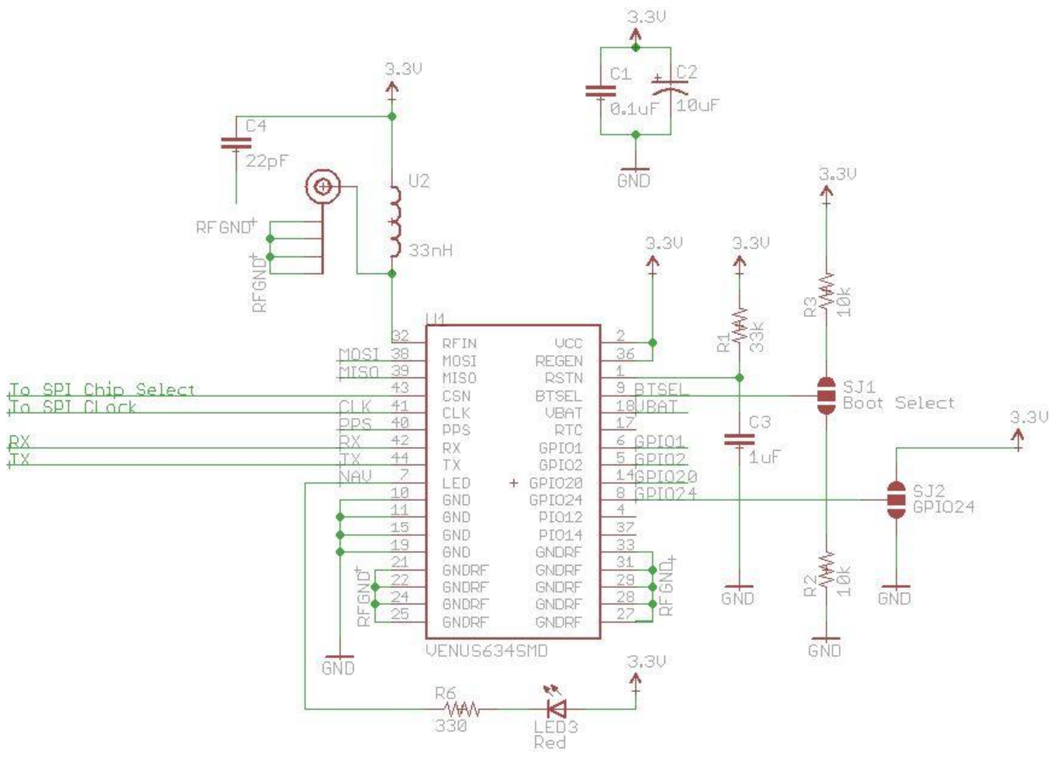


Figure 61: GPS Unit Embedded on the Main Control Unit

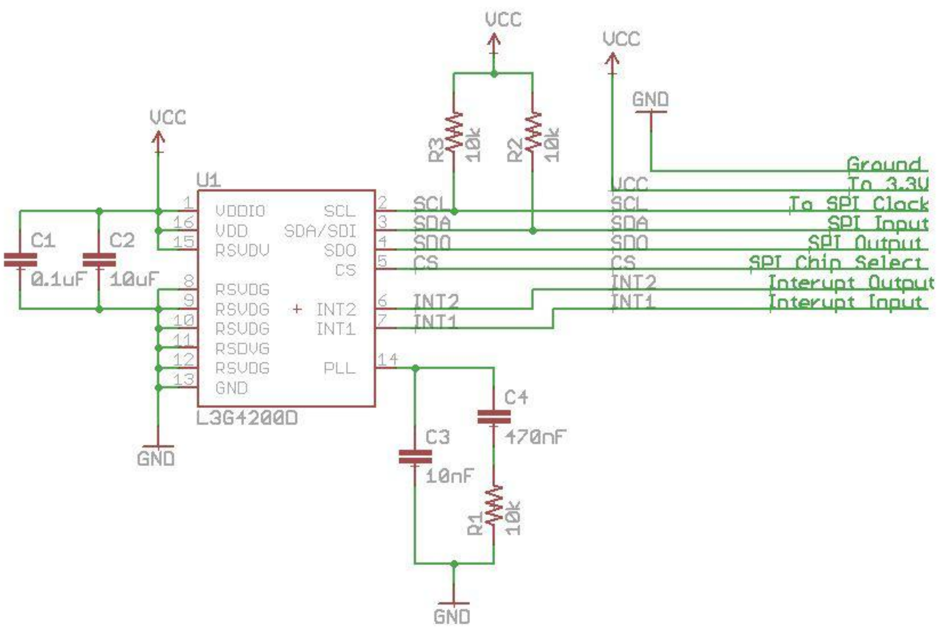


Figure 62: Gyroscope Circuit on the Main Control Unit

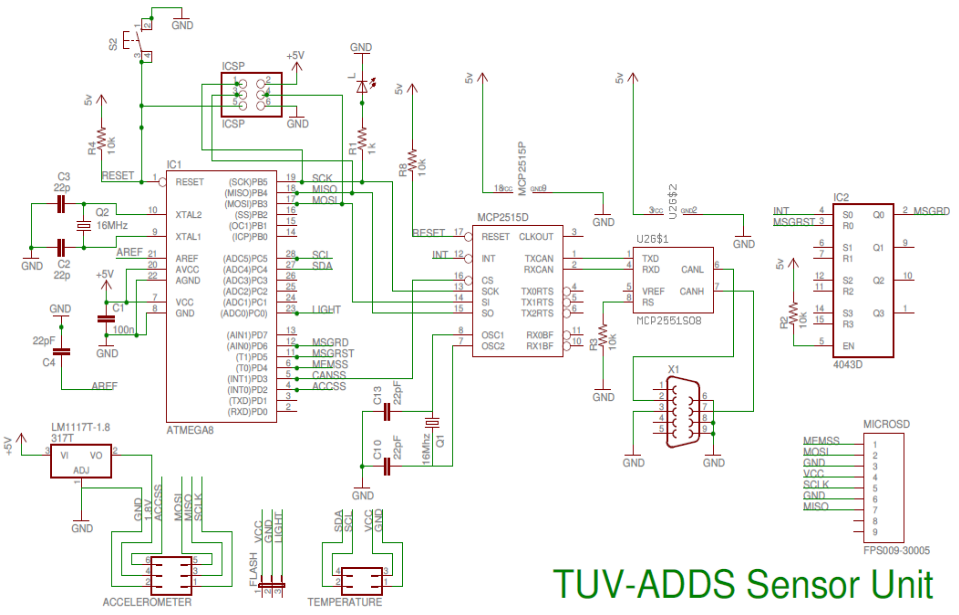


Figure 63: Sensor MCU with CAN Interface

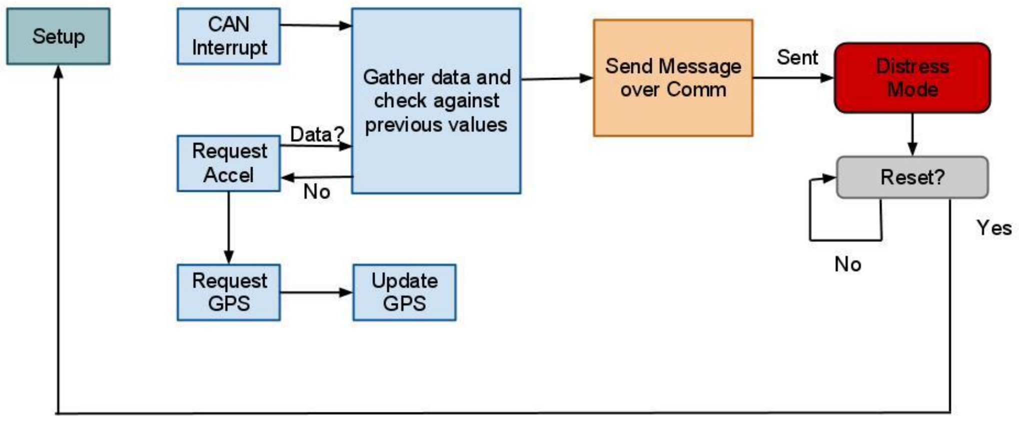


Figure 64: Main Processor Software Flow

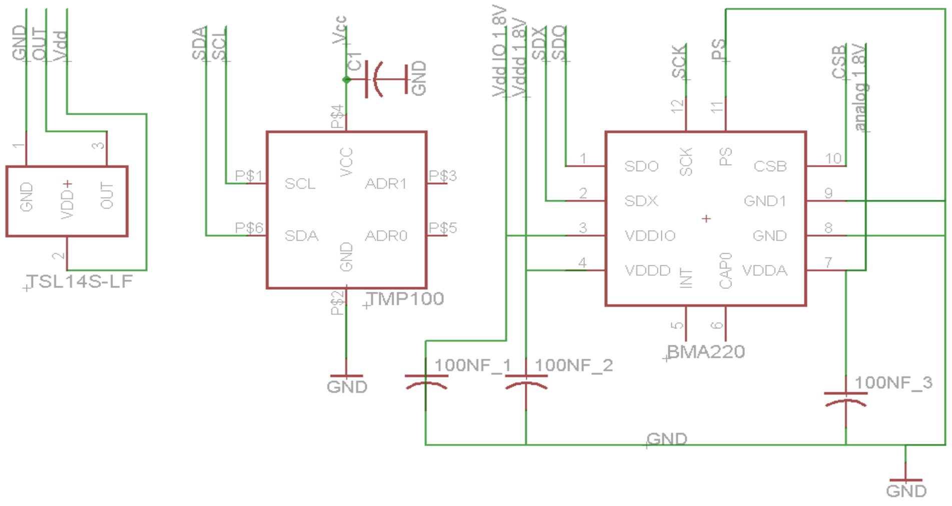


Figure 65: Sensor Module

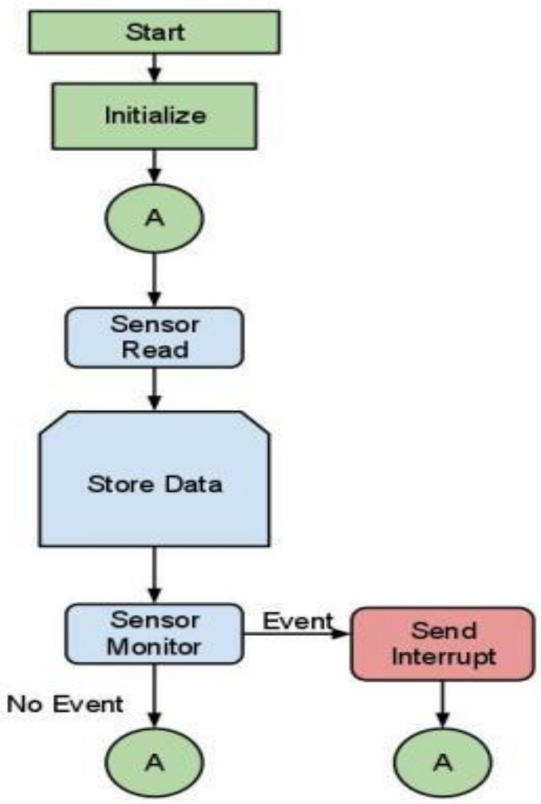


Figure 66: Sensor Software Flow

All Method Prototypes for the Main Processor software:

```
int acquireGPS(struct gps* mainGPS);
void updateGyro(struct* gyro);
int commDistress(struct* vehicle, int distress);
int commGPS(struct* vehicle);
int compare(int oldValue, int newValue);
void distressMode(void);
int comm(struct* vehicle);
int acquireGPS(struct gps* mainGPS);
void updateGyro(struct* gyro);
```

Variable Definition	Purpose
int gpsTime	Stores the time of satellite fix.
char gpsValidity	Stores whether or not the position is valid, or there is a warning
float gpsLat	Stores latitude position as degrees and minutes
char gpsNS	Stores ‘N’ for North or ‘S’ for south
float gpsLong	Stores longitude position as degrees and minutes
char gpsEW	Stores ‘E’ for East or ‘W’ for West
float gpsSpeed	Stores speed over the ground
int gpsDate	Stores the date when signal was fixed
int gyro	Stores the gyroscope x-position
int gyroY	Stores the gyroscope y-position
int gyroTime;	Stores the time at which a reading was taken

Table 36: List of all Globally Defined Variables for the Main Processor Software

Code Example for Configuring a XBee with Network Parameters:

```
void setup () {
                Serial.begin(9600);

                Serial.print("X"); // This clears buffer
delay(1100);

                Serial.print("+++");
                delay(1100);
```

```
 if (returnedOK() == 'T') {
                    // if an OK was received then continue
 }
 else {
                           setup(); // otherwise go back and try setup again
 }


Serial.print("ATID3330,");
Serial.print("DH0,");
Serial.print("DL1,");
                  Serial.println("CN");


                  if (returnedOK() == 'T') {
                        // if an OK was received then continue
                  }
 else {
                            setup(); // otherwise go back and try setup again
                  }
}
```

Variable Definition	Purpose
struct* vehicle messageBuffer	Allows data packaging for wireless transmission

Table 37: Variables used within Wireless Method

Variable Definition	Purpose
int tempGpsTime	Stores the time of satellite fix.
char tempGpsValidity	Stores whether or not the position is valid, or there is a warning.
float tempGpsLat	Stores latitude position as degrees and minutes
char tempGpsNS	Stores ‘N’ for North or ‘S’ for south
float tempGpsLong	Stores longitude position as degrees and minutes
char tempGpsEW	Stores ‘E’ for East or ‘W’ for West
float tempGpsSpeed	Stores speed over the ground
int tempGpsData	Stores the date when signal was fixed

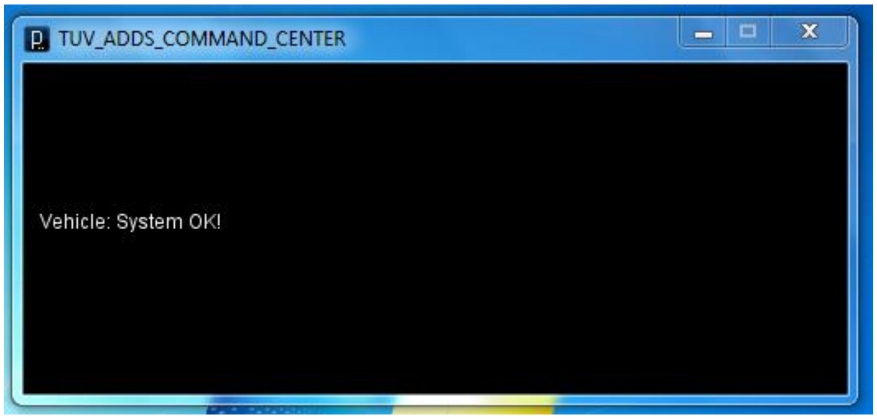
Table 38: List of Variables used in GPS Method

Variable Defintion	Purpose
int tempGyroX	Stores the gyroscope x-position
int tempGyroY	Stores the gyroscope y-position
int tempGyroTime;	Stores the time at which a reading was taken

Table 39: List of Variables used in Accelerometer Method

Field Name	Description
JField userName	Displays the User’s Name
JField panID	Displays the network Personal Area Network ID
JField date	Displays the date of GPS acquisition
JField time	Displays the time of GPS acquisition
Canvas warning	A small square which will change colors with an incoming distress signal
JField distress	Display which kind of distress is happening
JField vehSN	Displays the vehicles serial number
JField gpsLat	Displays the up to date vehicle Latitude position
JField gpsNS	Displays whether the Latitude position is ‘N’orth or ‘S’outh
JField gpsLong	Displays the up to date vehicle Longitude position
JField gpsEW	Displays whether the Longitude position is ‘E’ast or ‘W’est
JField vehSpeed	Displays the most current vehicle speed relative to ground
JButton exit	Will exit the TUV-ADDS Command Center program

Table 40: Components used in Command Center GUI



```
Vehicle: System OK!
```

Figure 67: Command Center GUI

Code Sample utilizing the New Soft Serial Library

```
#include <NewSoftwareSerial.h>

        NewSoftSerial usb = NewSoftSerial(4,3); // RxTx connected to pins 4 and
3
        void setup(){
                Serial.begin(9600); // Hardware designated serial ports
                usb.begin(9600); // Software designed serial ports
        }

        void loop(){
                if(Serial.available()){ //If there is an incoming message
                        usb.write(Serial.read()); //Read the message and send it to
USB
                }
        }
```

7 OPERATORS MANUAL

7.1 GENERAL OPERATION

TUV-ADDS is intended to be minimally invasive and operate unaided, once the system is turned on, for the end user. The system will setup wireless communication, acquire GPS signal and initialize the base line values for the sensor processing without any intervention by the user.

The TUV-ADDS system can be activated by a single toggle switch (shown in Figure 68) which provides power to the whole system. Three LED indicator lights have been place on the host vehicles, next to the main system power switch, are placed for the operator gratification to identify the system is working properly.

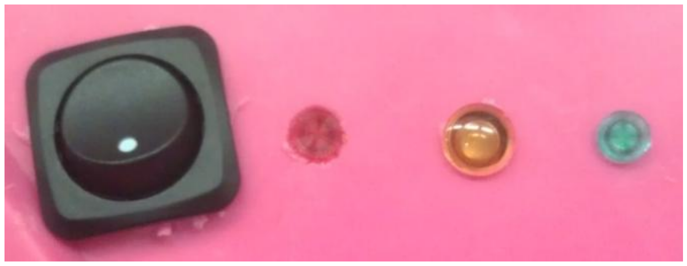


Figure 68: Power Switch and LEDs

The LED meanings are as follows:

- The red LED will indicate the overall system power. When the system is function properly this LED should remain constant and always lit.
- The Yellow LED indicates the main processor is active and is functioning. It is normal for the LED light to strobe a few times at start up, as elements of the system are be initialized; it will then remain constant and always lit.
- The Green LED indicates the GPS is successfully receiving a GPS signal; it's normal of this LED to strobe on and off while in operation. However, if it remains unlit then the GPS is unable to acquire signal successfully.

To reset the system the main power switch can be switched to the off position for a moment then returned to the on position. This will reset the system and allow it to acquire new baseline values.

7.2 GENERAL CONSIDERATIONS FOR SUCCESSFUL OPERATION

The system should be turned on turned on while on level ground. When the system is powered on it will sample the accelerometer using this value to initialize the system with a base line value; it will then compare all future values against this baseline evaluating the difference to meet a predetermined threshold.

The sensors are reset every thirty seconds; if the boards was recently exposed to a heat source a false signal could be set if the photodiode receives an increase of light. After 30 seconds the Temp sensor will acquired the temp of the board at reset this value as the base line. This process, of updating, will continue as long as the system is operational.

Once the system encounters a event which the system deems a distressing event the system will enter an infinite loop, which will cause the system lock on the warning message. This message and system can be reset using the main power switch, to turn the system off briefly then restoring power to become operational again..

As the system operates off an interrupt system; Electro-magnetic Interference could be a potential source of a false distress signal. In order to minimize this occurrence the default signal over the serial cable has been pulled high and will be pull low once an event has been registered by the sensor MCU.

The SD memory card on the sensor MCU will record the raw data from the sensor board. This is for external review at a later date if the user wished to do so. The memory card will store data for about 1.6 thousand hours and keep about $2.3x10^7$ individual measurements. The format of the data will be presented in the following format hour, minute, second, flash reading and temp reading. An example of the data stored to the SD card is shown in Table 41.

```
Hr: 0  Min: 0  Sec: 0  Flsh: 589  Temp: 352
Hr: 0  Min: 0  Sec: 0  Flsh: 601  Temp: 352
Hr: 0  Min: 0  Sec: 0  Flsh: 596  Temp: 352
Hr: 0  Min: 0  Sec: 0  Flsh: 568  Temp: 352
Hr: 0  Min: 0  Sec: 1  Flsh: 537  Temp: 352
Hr: 0  Min: 0  Sec: 1  Flsh: 414  Temp: 352
Hr: 0  Min: 0  Sec: 1  Flsh: 416  Temp: 352
Hr: 0  Min: 0  Sec: 1  Flsh: 416  Temp: 352
Hr: 0  Min: 0  Sec: 2  Flsh: 414  Temp: 352
Hr: 0  Min: 0  Sec: 2  Flsh: 407  Temp: 352
Hr: 0  Min: 0  Sec: 2  Flsh: 403  Temp: 352
Hr: 0  Min: 0  Sec: 2  Flsh: 399  Temp: 352
Hr: 0  Min: 0  Sec: 3  Flsh: 393  Temp: 352
Hr: 0  Min: 0  Sec: 3  Flsh: 396  Temp: 352
Hr: 0  Min: 0  Sec: 3  Flsh: 401  Temp: 352
Hr: 0  Min: 0  Sec: 3  Flsh: 398  Temp: 352
Hr: 0  Min: 0  Sec: 4  Flsh: 397  Temp: 352
Hr: 0  Min: 0  Sec: 4  Flsh: 396  Temp: 352
Hr: 0  Min: 0  Sec: 4  Flsh: 393  Temp: 352
Hr: 0  Min: 0  Sec: 4  Flsh: 402  Temp: 352
Hr: 0  Min: 0  Sec: 5  Flsh: 395  Temp: 352
Hr: 0  Min: 0  Sec: 5  Flsh: 394  Temp: 352
Hr: 0  Min: 0  Sec: 5  Flsh: 396  Temp: 352
Hr: 0  Min: 0  Sec: 5  Flsh: 396  Temp: 352
Hr: 0  Min: 0  Sec: 6  Flsh: 397  Temp: 352
Hr: 0  Min: 0  Sec: 6  Flsh: 395  Temp: 352
Hr: 0  Min: 0  Sec: 6  Flsh: 393  Temp: 352
Hr: 0  Min: 0  Sec: 6  Flsh: 369  Temp: 352
```

Table 41: Data Format

8 PROTOTYPE CONSTRUCTION

TUV-ADDS will need to make use of several Printed Circuit Boards (PCB). These will be modulated into four sections: The sensor MCU, the main control unit, the power control system, and three sensor modules. The sensor MCU will contain the Atmel micro controller and the CAN-bus. The main control unit will contain the ATmega, Gyroscope and XBee-PRO. The power control unit will contain the voltage regulators, and the three sensor modules will contain the accelerometer, photodiode and temperature sensor.

One consideration was to design the layout of all the modules on one large board, as in a pallet or array form. Then when the board from the manufacture arrived, separate the board into its respective modules. The thought process behind this comes about by the industry standard for pricing of boards. There tends to be a base price per board then the cost go's up relative to the square inches and layers used. ExspressPCB quoted a board with an area of eight by ten inches, by their formula online: \$55 + (\$0.65 * # Of Boards * Board Area in Inches^2) + (\$1.00 * # Of Boards) [30], which came out to about \$118.00. Another PCB manufacturing company, Advance Circuits, had a special program which they called "\$33 Each", but this had some stipulations to it. One, there had to be an order minimum of 4 boards placed, and two, a board could not go over 60 inches squared. This would put the cost at (\$33.00 * 4) + \$10(shipping) = \$142.00. [31]

Due to the industry standards it is likely the team will have to use a two layer PCB design. While it is possible to get a one layer PCB the standard minimum appears to be a 2 layer board, this will be more than sufficient to meet our design needs. As far as grounding the PCB, it is suggested that we use an entire layer as ground, which would be called a ground plane. "If you use a multilayer printed circuit board with surface mount components, place control ground plane on an inner layer so that it acts as a shield between the power and control circuits." [32] However, since a two-layer board will be required, the use of a ground rail that would run around the board should be sufficient. By doing this it, it should reduce any noise that could develop in the circuit and boards. "Keep power and ground tracks running in close proximity to each other if possible, don't send them in opposite directions around the board. This lowers the loop inductance of your power system, and allows for effective bypassing." [33]

The use of large traces to lower the impedance is also suggested. "Use copper, and lots of it. The more copper you have in your ground path, the lower the impedance. This is highly desirable for many electrical reasons. Use polygon fills and planes where possible." [33] It is advantageous to take these considerations into account as most of the signals used by the MUC's and sensors are of extremely low power. It is the objective of the team to attempt to maintain the most truthful system and eliminate all possibilities for false positive signal being relayed to the user.

It seems that having the PCB constructed by ExspressPCB will be the most cost efficient for the team. This company tends to offer the most active support for the cost. They offer the most cost effective solution between any of the PCB manufactures we looked into. ExspressPCB also appears to have a reputable turnaround time from submission to delivery of the final completed PCB's.

The software that will be implemented in the design of the PCB will be EAGLE CAD. This software package includes a schematic editor that is used for the design and layout of the board. This software also has the ability to produce Gerber files which is typically what the industry will use to manufacture the. This software also seems to be advantageous due to the fact that it is tremendously popular; there are a plethora of tutorials of how to use the software and typical design layout for common PCB's. EAGLE will also allow us to create schematic diagrams of the circuits we will be construction during the design of TUV-ADDS.

While designing the PCB several aspects will need to be considered. Table 42 below gives general guidelines for the width of the tracks on a PCB. The design should only call for low power sensors and shouldn't require anything wider that the rating for one to two amps.

Track Width Table			
Amps	Width (1 oz.)	Width (2 Oz)	Milli Ω/Inch
1	10	5	52
2	30	15	17.2
3	50	25	10.3
4	80	40	6.4

Table 42: Guidelines for the Width of the Tracks on a PCB
[33]

It was also noted that the group would need to pay attention to the grounding since our circuit is in a mobile vehicle and will have no form of external grounding. We want to preserve the integrity of the signals therefore; we will want to be conservative as possible in the layout of the PCB to avoid any unnecessary noise in the signals and communication process.

As far as populating the PCBs, we are considering one of two options. The first one is to mount the parts ourselves with the aid of the Armature Radio Club, as they made available their services in class to help and train students on use of common surface mount equipment and techniques. The second option is to outsource the task for populating the board to an independent company or even the Amateur Radio Club in place of a donation. As it stands now, the team will attempt to populate the PCB ourselves. Most of the parts are likely going to require the use of surface mount techniques. Most likely a reflow solder oven will be used, assuming there is a working one available on the university which we will have access to.

For our prototype we will be using a Power Wheels vehicle as a platform to host our system. Upon inspection of the Power Wheels we noted there was quite a bit of room in the "engine compartment." This area would be the best place to securely mount the hardware required for operation of the system. However, to relate this to the original intention of the design we would ideally like to place the unit in the passenger compartment of the vehicle. The physical volume of space required for the boards is minimal, so it shouldn't interfere with any day-to-day operations of the vehicle. It is also desirable that placement of the main processor would be in a reinforced housing box to protect the processors form any damage from either normal day to day operations or distress event, to allow the system to remain operable and send out the necessary distress signals.
Implementing a graphical user interface system in this prototype is another consideration of the group. This would be able to show the command center the status of the system and the vehicles and the determination made by the software. This would make the system very easy for the user to understand, regardless of their background with respect to the system.

9 TEST PLAN

After the research was conducted, components were selected and ordering of parts for the system began. The test plan consists of testing each component individually to make sure they work correctly. After all the components of at module were tested, the process of combining them into a working module began. Testing all of the modules then took place to ensure that the module works before integrating the modules into a complete system. After all the modules were tested individually, they were all be combined into a system and system verification testing then took place.

Module E, software, was tested differently than the other modules. The software was tested and debugged as it was written. Then when the system was integrated together, the system verification tests verified that the software correctly worked with the hardware. Module D was only tested at a component level because the module is simply several of the same components operating separately.

When testing TUV-ADDS as a system, it was attached to a vehicle that will suite its needs and it was put through simulated IED hits and a rollover event to see if the system works correctly by communicating to the command center that it had been in a distressful even. The system was also tested to make sure it was accurate by having it go through events that simulate non-distress events that should not set the system to alert for help such as rough terrain, as well as an event simulating an unusually hot day. This gave the team proper assurance that the system was working to the goals and objections desired.

9.1 TEST MATERIALS

To test the system as an entity, the team decided that mounting the system to a Hummer Power Wheels vehicle (see Figure 69) would be the most accurate test. This allowed for the team to test and demonstrate the system in a scaled down but still realistic event. Through some research the team found a way to make the Power Wheels drive by remote control. The team used a Wii Nunchuck breakout adapter (see Figure 70) that connected directly to the microcontroller. The microcontroller attached to the switches in the Power Wheels. This allowed the team to drive the Power Wheels with the Wii Nunchuck. This allowed for more accurate testing because a member was not needed to steer and accelerate the Power Wheels vehicle manually. It was very simple to install and was a helpful tool when it came to testing.

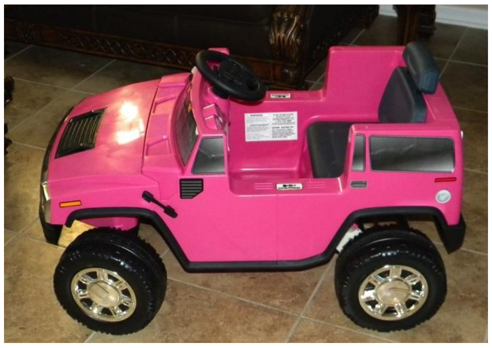


Figure 69: Hummer Power Wheels Vehicle

Figure 70: Wii Nunchuck Breakout Adapter
(Printed with Permission from Adafruit Industries)

For testing the system of the event of an IED blast, we quickly accelerated the vehicle in the upward position while simulating the heat and the flash with a lighter to see if the combination of these characteristics triggered a call to the system for assistance as it should have. When testing the rollover aspect of the system, we will simply manually cause the vehicle to turn completely on its side in any direction and this should alert for assistance after the specified time amount has been reached.

9.2 TESTING EVENTS PLAN

Component Level Testing Plan (Table 43) was broken down by component and listed in order the procedures and expected results of each procedure of the test. These were the events that the TUV-ADDS team tested to ensure that each component was accurately working and if all the expected results were met, then that would mean the system met sufficient criteria to move to module testing.

Step	Procedure	Expected Results	Actual Result
Component: Sensor MCU			
1	Program chip	The USBtinyISP AVR Programmer busy light should turn off.	Met/Not Met
2	Plug the chip into the Arduino	Board should turn on.	Met/Not Met

		Should be able to open up a serial terminal.	Met/Not Met
		And the serial output should be observed.	Met/Not Met
Component: Accelerometer Sensor			
1	Connect sensor to the microcontroller and a power source	The sensor information should show up on serial terminal.	Met/Not Met
2	Cause significant motion to the sensor	The MCU should receive data that reflects the motion applied.	Met/Not Met
Component: Photodiode Sensor			
1	Set the sensor up on a breadboard and connect to power and digital multimeter.	There should be some register of voltage.	Met/Not Met
2	Cause an event of great radiance to the sensor by shining a flashlight on the sensor.	The multimeter should reflect results that are a significant jump in voltage.	Met/Not Met
Component: Temperature Sensor			
1	Connect sensor to the microcontroller and a power source	The sensor information should show up on serial terminal.	Met/Not Met
2	Cause an event of significant temperature increase to the sensor by applying heat from a hair dryer.	The MCU should receive data the reflects the temperature increase applied	Met/Not Met
Component: GPS			
1	Connect GPS to microcontroller		

2	Program microcontroller	Should see consistent values.	Met/Not Met
3	Collect GPS locations to test components against		
4	Go to the locations that data was collected from	Should result in the microcontroller reading the same GPS locations that were collected in research.	Met/Not Met
Component: XBee			
1	Connect XBee to microcontroller	Should see a green blinking LED on the XBee	Met/Not Met
2	Repeat for a second and separate XBee component	Should see a green blinking LED on the XBee	Met/Not Met
3	Configure PAN ID for both		
4	Program both microcontrollers		
5	Connect microcontrollers to different computers		
6	User types message in one computer	Should result in seeing the message on the other computer.	Met/Not Met
7	Repeat process in other direction	Should result in seeing the message on the other computer.	Met/Not Met
Component: CAN Controller			
1	Program the ATMega with software that will alter the appropriate registers on the CAN controller to toggle various external pins	The USBtinyISP AVR Programmer busy light should turn off.	Met/Not Met
2	Property connect CAN controller and ATMega to power and each other		

3	Measure Voltage on toggle pins with a digital multimeter	Observe high and low voltage levels on toggle pins.	Met/Not Met
Component: Voltage Regulator			
1	Connect Regulator to voltage source		
2	Measure the voltage out of the regulator with an oscilloscope	Should result in rated voltage output.	Met/Not Met

Table 43: Component Level Testing Plan

The Module Testing Plan (Table 44) was divided by module tested, which occurred after all the components of the module had successfully completed component level testing. The plan lays out the procedures that will be taken step by step in order to verify that the components are integrated correctly as a module. The expected results were the exit criteria that must be met for the module to be able to be integrated into the system with confidence.

Step	Procedure	Expected Results	Actual result
Module A: Sensor Module			
1	Connect the Sensor PCB to the Sensor MCU PCB by a 10-pin jumper.	The Sensor MCU should power the sensor PCB.	Met/Not Met
2	Have a custom test program programmed onto the chip and monitor the sensors.		
3	Open a serial terminal	Should see data from the sensors.	Met/Not Met
4	Cause a significant motion event to the sensor	The MCU should receive data that reflects the motion applied	Met/Not Met

5	Cause an event of great radiance to the sensor by shining a flashlight on the sensor.	The multimeter should reflect results that are a significant jump in voltage.	Met/Not Met
6	Cause an event of significant temperature increase to the sensor by applying heat from a hair dryer.	The MCU should receive data the reflects the temperature increase applied	Met/Not Met
Module B: Main Control System Module			
1	Interface the Main Control PCB with the computer via USB	Computer should recognize a USB compatible device	Met/Not Met
2	Open a serial monitor on the computer	Should result in the wireless network being configured	Met/Not Met
		Should be reading in data from the GPS and Gyroscope components	Met/Not Met
3	Change locations	Should read different GPS Values	Met/Not Met
4	Tilt PCB	Should read different Gyroscope values	Met/Not Met
Module C: CAN Bus			
1	Program the ATMega for Loopback CAN Bus messages	The USBtinyISP AVR Programmer busy light should turn off.	Met/Not Met

2	Connect the ATMega to the CAN Bus components		
3	Open a serial terminal	Should output the transmitted data from ATMega	Met/Not Met

Table 44: Module Testing Plan

The System Testing Plan (Table 45) lays out the specific tests that were performed on the system that verified whether the system was working accurately and correctly. System Testing was completed once all the modules complete module level testing and were integrated together as a system. This included applying the developed software to the hardware. The plan then tested to make sure the system did what it is supposed to do, but there were also tests included in the plan to make sure system does not perform incorrect tasks. The way the test plan had been laid out, it clearly states the procedures to run the test while also stating what is expected to result at each point. After the system successfully met all the expected results listed, the team then had confidence that the system worked properly.

Step	Procedures	Expected Results	Actual Results
Event 1: IED Reaction Verification			
1	Drive the vehicle in the forward direction.		
2	Create an event that will cause the vehicle to increase its vertical position by the appropriate value while simultaneously increasing the heat by the appropriate percentage increase and creating a flash of appropriate luminance.	This event should cause an alert for help to another TUV-ADDS system within 30 seconds of the event happening.	Met /Not Met

		The TUV-ADDS system should relay the correct GPS location of the distressed vehicle to the system that was contacted for assistance.	Met /Not Met
Event 2: False Alarm Test (Vertical Position Increase)			
1	Drive the vehicle in the forward direction.		
2	Create an event that will cause the vehicle to increase its vertical position by the appropriate value but do not increase the heat or flash.	This event should not cause an alert for help to another TUV-ADDS system.	Met /Not Met
Event 3: False Alarm Test (Flash Increase)			
1	Create an event of significant luminance by the appropriate value but do not increase the heat or position.	This event should not cause an alert for help to another TUV-ADDS system.	Met /Not Met
Event 4: False Alarm Test (Temperature Increase)			
1	Apply significant heat increase to the system	This event should not cause an alert for help to another TUV-ADDS system.	Met /Not Met
Event 5: False Alarm Test (Multiple Characteristics Occur)			
1	Drive the vehicle in the forward direction.		
2	Create an event of significant luminance and vertical position by the appropriate value but do not increase the heat.	This event should not cause an alert for help to another TUV-ADDS system.	Met /Not Met
Event 6: Rollover Reaction Verification			

1	Drive the vehicle in the forward direction.		
2	Create an event that will cause the vehicle to flip on its side.	This event should cause an alert for help to another TUV-ADDS system within 30 seconds of the event happening.	Met /Not Met
		The TUV-ADDS system should relay the correct GPS location of the distressed vehicle to the system that was contacted for assistance.	Met /Not Met
3	Return the vehicle to the upright position		
4	Drive the vehicle in the forward direction.		
5	Create an event that will cause the vehicle to flip completely upside down.	This event should cause an alert for help to another TUV-ADDS system within 30 seconds of the event happening.	Met /Not Met
		The TUV-ADDS system should relay the correct GPS location of the distressed vehicle to the system that was contacted for assistance.	Met /Not Met
Event 7: False Alarm Test (Incline increase)			
1	Drive the vehicle in the forward direction.		
2	Drive the vehicle into a ramp that elevates two of the four wheels at a non-alarming	This event should not cause an alert for help to another TUV-ADDS system.	Met /Not Met

	degree, for an appropriate amount of time.		

Table 45: System Testing Plan

9.3 TEST EXPECTATIONS

It was the expectation of the team that TUV-ADDS perform to the specified goals that had been laid out for the system. This means, that when the appropriate characteristics of a rollover or IED hit are applied to the system, the result were that the system relay to the command center that there was a vehicle in distress and its accurate GPS location. These results ensured the team that the system was performing its tasks correctly and the modules were working well together.

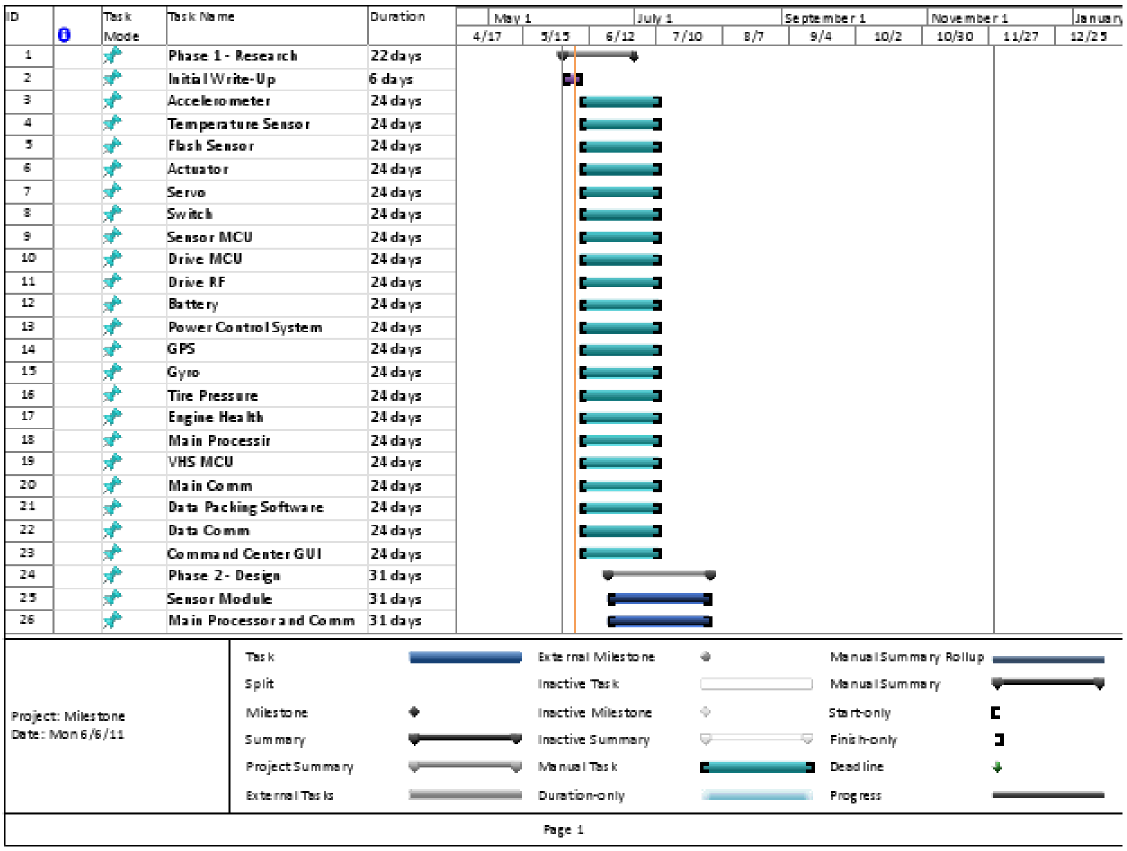
We also expected that the occupants of the vehicle in distress would not need to interact with the system at all for it to relay these messages, it is automatic. One of the goals of the system was to be as user friendly as possible, and ideally require no interaction at all by the distressed vehicles occupants. The objective was to allow the occupants to be free to take care of other important tasks if need be, or operate automatically incase of situations where the occupants are unable to interact with the system. This ensures that the system is user friendly and will be out of the way in the event of a rollover or IED hit.

The false alarm testing events resulted in no communication from the system at all. The system did not call for assistance when the vehicle was not placed through all of the required characteristics for one of the distress events. This guaranteed that the system was up to the accuracy expectations the team had set to make sure that the system was not putting more people in danger by sending people out for no reason.

10 SCHEDULING AND BUDGETING

10.1 PROJECT MILESTONES

The group made a milestone chart (shown in Figure 71) to keep a schedule for the TUV-ADDS project. This was made in Microsoft Project and was advised throughout the year as a guide to how far along we should be to finish the project on time.

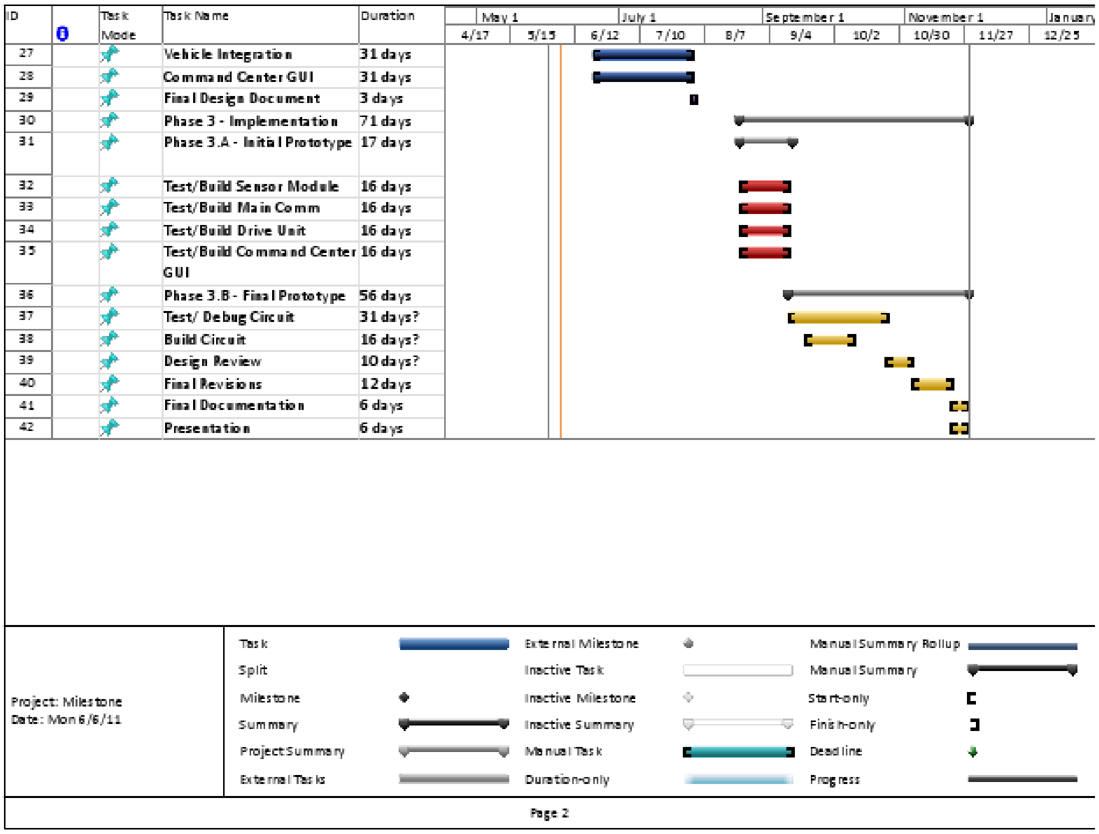


ID	ℹ	Task Mode	Task Name	Duration
1		📌	Phase 1 - Research	22 days
2		📌	Initial Write-Up	6 days
3		📌	Accelerometer	24 days
4		📌	Temperature Sensor	24 days
5		📌	Flash Sensor	24 days
6		📌	Actuator	24 days
7		📌	Servo	24 days
8		📌	Switch	24 days
9		📌	Sensor MCU	24 days
10		📌	Drive MCU	24 days
11		📌	Drive RF	24 days
12		📌	Battery	24 days
13		📌	Power Control System	24 days
14		📌	GPS	24 days
15		📌	Gyro	24 days
16		📌	Tire Pressure	24 days
17		📌	Engine Health	24 days
18		📌	Main Processir	24 days
19		📌	VHS MCU	24 days
20		📌	Main Comm	24 days
21		📌	Data Packing Software	24 days
22		📌	Data Comm	24 days
23		📌	Command Center GUI	24 days
24		📌	Phase 2 - Design	31 days
25		📌	Sensor Module	31 days
26		📌	Main Processor and Comm	31 days

Project: Milestone
Date: Mon 6/6/11

Task		External Milestone	◆	Manual Summary Rollup	
Split		Inactive Task		Manual Summary	
Milestone	◆	Inactive Milestone	◇	Start-only	[
Summary		Inactive Summary		Finish-only	]
Project Summary		Manual Task		Deadline	⬇
External Tasks		Duration-only		Progress	

Page 1



ID	Task Mode	Task Name	Duration
27		Vehicle Integration	31 days
28		Command Center GUI	31 days
29		Final Design Document	3 days
30		Phase 3 - Implementation	71 days
31		Phase 3.A - Initial Prototype	17 days
32		Test/Build Sensor Module	16 days
33		Test/Build Main Comm	16 days
34		Test/Build Drive Unit	16 days
35		Test/Build Command Center GUI	16 days
36		Phase 3.B - Final Prototype	56 days
37		Test/ Debug Circuit	31 days?
38		Build Circuit	16 days?
39		Design Review	10 days?
40		Final Revisions	12 days
41		Final Documentation	6 days
42		Presentation	6 days

Project: Milestone
Date: Mon 6/6/11

Figure 71: Milestone Chart

10.2 PROJECT BUDGET

TUV-ADDS was a student sponsored project, meaning the team had no corporate or faculty sponsors. Therefore the budget for this project was limited to what the team was able to afford. Through research of the parts needed to build a prototype of the system, we perceived the budget being between 400 and 700 dollars with a potential overrun to approximately 1500 dollars as shown in Table 46. We were able to keep the expense of the project to just over 500 dollars total as shown in Table 47.

When analyzing the itemized budget shown below, it is evident that the project cost could have easily fallen to the lower end of the projected costs if enough time is allowed to work with the components and not resulting in having to purchase the more expensive but easier to use components. To cut down on cost, the team attempted to source as much of the project from sample programs offered by manufacturers. Funding of the project was achieved by splitting the cost of the system between the four teammates.

Budget	**Low**	**High**	**Average**	**Low Part**	**High Part**
Vehicle	30	350	180	Metal Frame	Powerwheels
Power Control Systems	10	25	15	SparkFun	Jameco
CPU	17	50	25	Atmel ATMega	TI Stellaris
Sensor MCU	4	6	5	Atmel ATMega 328P-PU	Atmel ATMega
Photodiode	1.26	3.08	2	TAOS TSL145	TAOS TSL145
Accelerometer	3.09	53.16	9	Bosch Sensortec BMA220	ADIS16240ABCZ
Temperature Sensor	0	3.5	3.5	TMP03FSZ Analog Devices	TMP03FSZ Analog Devices
GPS	50	90	55	Venus w/ SMA connect	GS407 Helical
Window Motor	8	70	10	Ford 150 Window Motor	BMW 325XI Right window Motor
Engine Switch	0.99	17	3	Panasonic Corp	Tyco
Main Communication (one end)	60	70	65	XBee	XBee Pro
Gyroscope	7.53	25	17	L3G4200DTR	Atmel ATMega 328P-PU
PCB Fabrication	112	340	150	ExpressPCB - Student Program	ExpressPCB
33% Misc. Expenses	100.27	363.9042	178.035		
Total	404.14	1466.644	717.535		

Table 46: Estimated Budget Table

CURRENT EXPENDITURES	
ITEM (quantity)	**PRICE**
Power Wheels	$50
Xbee Pro (2)	$75
Temp Sensors (10)	FREE
Accelerometers (1)	$46
Photodiodes (8)	$10
FTDI Cable	$20
Relays (3)	$6
Atmel328 (3)	$13
16MHz Crystals (8)	$8
Flash Memory	$10
GPS	$45
CAN Module	FREE
Components	$80
PCB Fabrication	$150
TOTAL:	$513

Table 47: Actual Budget

11 CONCLUSION

11.1 POSSIBLE FUTURE IMPROVEMENTS

If time permits there are several system ideas that could be added to the TUV-ADDS to improve it. These ideas were created during the initial brainstorming session at the beginning of the semester. After looking deeper into what we wanted to be the fundamental system characteristics, the team decided to put off these features until more time permitted. To this day these ideas come up in conversation, and if we have the time then we will create these features.

The first of these ideas is to have a 'soldier presence' feature that would allow the command center to know how many soldiers are still in the vehicle. The original design of this feature included an RFID system local to a vehicle that could detect the number of people within a given area. This could then connect to the TUV-ADDS system, which would communicate this data to the command center. Along with the RFID, the soldier would wear wireless heart-rate monitors. In the case of distress, it would be good to let the higher authority to know the heart-rates of the present soldiers. In the tragic case of a heart rate reading a zero pulse, it could be assumed that the soldier is dead. This could save more lives by allowing proper allocation of aid.

The team also decided to possibly add feature to help make other manufactures interface their equipment to the TUV-ADDS. This would be accomplished by adding two hardware configurations. First, an unused CAN line could be added to the Main Control Unit to allow any sort of device to communicate with the main processor. The other feature would allow the user to program the main processor in the case of adding features or updating software. This could be done by connecting to the on-board USB line that interfaces with the main processor. A power configuration will prevent power to the main control unit from both the USB line and the power control unit.

11.2 FEATURES LEFT OUT

The design originally had two features that were scrapped after a final initial design review. There features will promote soldier safety so they did fit under the general system characteristics. It was decided, however, that the features were either too complex to fit in our time schedule, or not necessary. The first of these two features was a Vehicle Health System (VHS). This system would connect directly to the vehicle's on-board computer and gather appropriate data. The main processor would then decide whether or not the vehicle was safe (or was even able) to drive. After an initial review, it was decided that this feature would be too difficult to accomplish in our time frame with available supply. The other feature was a tire pressure system. This system would check the pressure, both static and dynamic, and relay this information to the main processor. The processor would then decide whether or not the tire was capable of continued driving and send this information to the command center. After further research into IED blasts, it was discovered that vehicles hit would in almost all cases loose one of the tires. This fact would make the tire pressure system useless.

11.3 REFLECTIONS

TUV-ADDS was designed with the idea of helping the men and women of the armed forces perform their duties with a convenient safety feature. With the creation of this system, the need for reliable distress communication for these men and women that face the obvious danger in the Middle East everyday was kept in the minds of the team. The system design is a network of sensors and the supporting hardware and software that will be integrated to meet this need.

This paper is a documentation of what the TUV-ADDS team has accomplished as we researched, designed, and produced the prototype of the system. Everything from defining the project to researching specifications and components to our final design and prototype has been completed by the team and has been included into this paper. After completing extensive research on the project idea and components for the system design, the TUV-ADDS team feels that our preparation allowed us to prepare the prototype more efficiently and with fewer errors.

When conducting research for the system design, there were many changes and design discussions about what parts and design would be the best for the system. The team has did their best to finalize the design with every team member's input and considering every angle of a design or component decision. Even with this preparation, the team is aware that some design changes would occur.

Throughout this semester, significant knowledge has been gained about design and component choices as well as the military and Government relevance to our project. The experience of gathering data about different parts and comparing data sheets as well as collaborating with each other has been very beneficial to the team as we prepare to enter the professional world of engineering.

In the upcoming semester the team will turn its attention to the building of the TUV-ADDS prototype. This will take significant effort on the part of the team to complete and we have prepared plans to assist the team as we move into this next stage of the project. The team looks forward to utilizing the knowledge we have all gained thus far to produce a system that will perform to the goals we have set as well as gaining more professional experience through completing this project.

During our final presentation to the review board we were able to demonstrate the success of the system. The demonstration went through system start-up, testing against false-alarms, as well as showing the system experiencing both an IED hit as well as Rollover. All features worked as described in the Test Plan.

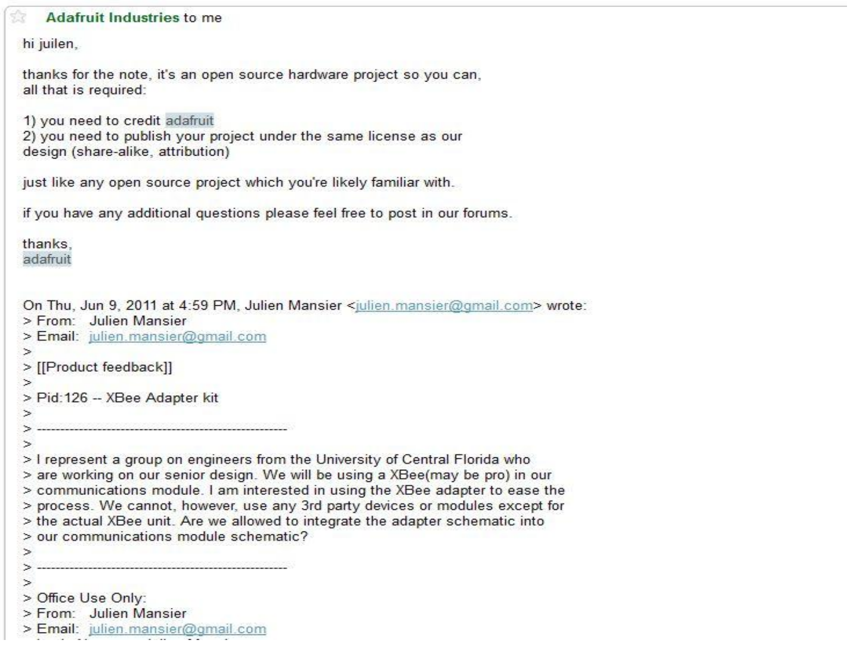
12 SOURCES/ COPYRIGHT PERMISSIONS

[1] Controller Area Network (CAN) Basics Microchip
[2] Xilinx.com
[3] http://arduino.cc/en/Guide/Environment
[4] http://focus.ti.com.cn/cn/lit/ug/slau144h/slau144h.pdf
[5] ATMega48PA/88PA/168PA/328P Atmel Datasheet
[6] MSP430G2x31/2x21 Texas Instruments Datasheet
[7] ATmega640V/1280V/1281V/2560V/2561V Atmel Datasheet
[8] MSP430F663x Texas Instruments Datasheet
[9] AT90CAN32/64/128 Atmel Datasheet
[10] ARM926EJ-S Technical Reference Manual Atmel
[11] MCP2515 Microchip Datasheet
[12] SJA1000 Phillips Datasheet
[13] http://code.google.com/p/canduino/
[14] MCP2551 Microchip Datasheet
[15] http://arduino.cc/en/Hacking/Bootloader?from=Main.Bootloader
[16] http://www.arduino.cc/en/Reference/SPI
[17] http://www.atmel.com/dyn/resources/prod_documents/doc2549.pdf
[18] http://zone.ni.com/devzone/cda/tut/p/id/2732
[19] http://ftp1.digi.com/support/documentation/90000982_B.pdf
[20] http://eecs.ucf.edu/seniordesign/fa2009sp2010/g09/docs/Gp9FinalDocumentation.pdf
[21] http://www.sparkfun.com/datasheets/GPS/Modules/D2523T%20V1.pdf
[22] MilCAN A Specification MWG-MILA-001Revision 3 Milcan Working Group
[23] ISO/CD 11898-2
[24] L4941 STMicroelectronics
[25] LM2940/LM2940C 1A Low Dropout Regulator National Semiconductors
[26] LM1117/LM1117I 800mA Low-Dropout Linear Regulator National Semiconductors
[27] http://arduino.cc/en/Reference/HomePage
[28] http://arduiniana.org/libraries/newsoftserial/
[29] TAOS DATA SHEETS for TSL14S-LF
[30] http://www.expresspcb.com/ExpressPCBHtm/SpecsStandard.htm
[31] https://www.my4pcb.com/net35/promotion/orderFileUpload.aspx
[32] http://www.smps.us/layout.html
[33] http://alternatezone.com/electronics/files/PCBDesignTutorialRevA.pdf

TABLE OF FIGURES

TABLE OF TABLES

Adafruit Industries



Adafruit Industries to me

hi juilen,

thanks for the note, it's an open source hardware project so you can,
all that is required:

1) you need to credit adafruit
2) you need to publish your project under the same license as our
design (share-alike, attribution)

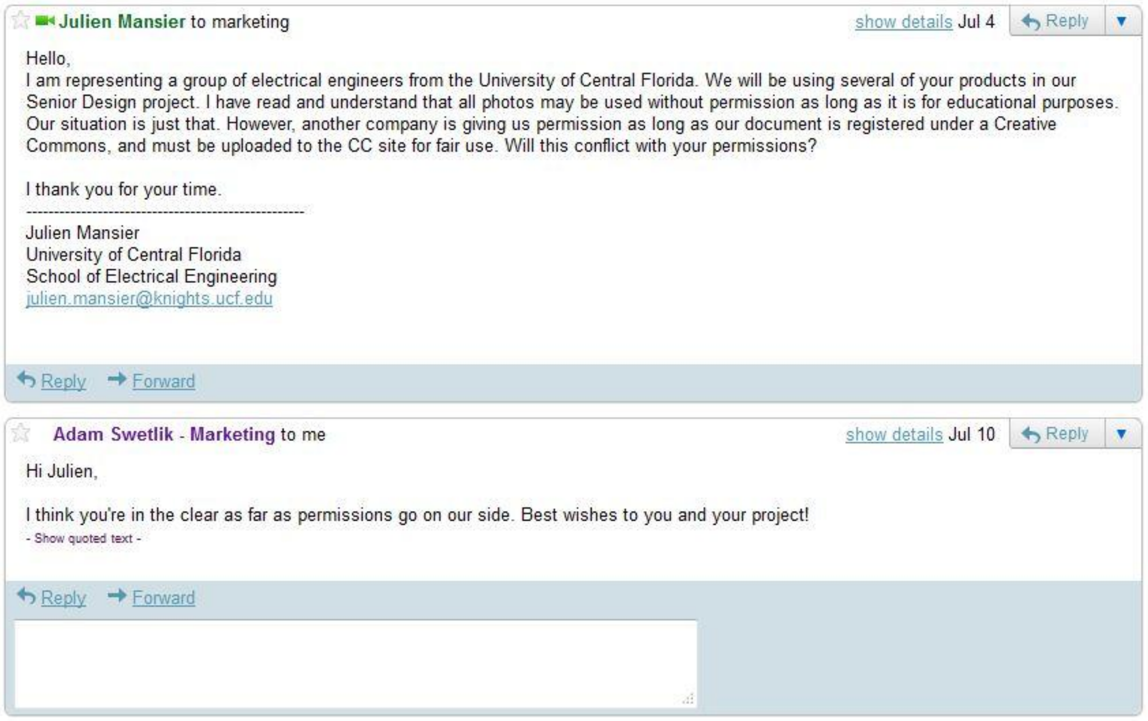
just like any open source project which you're likely familiar with.

if you have any additional questions please feel free to post in our forums.

thanks,
adafruit

On Thu, Jun 9, 2011 at 4:59 PM, Julien Mansier <julien.mansier@gmail.com> wrote:
> From: Julien Mansier
> Email: julien.mansier@gmail.com
>
> [[Product feedback]]
>
> Pid:126 -- XBee Adapter kit
>
> --
>
> I represent a group on engineers from the University of Central Florida who
> are working on our senior design. We will be using a XBee(may be pro) in our
> communications module. I am interested in using the XBee adapter to ease the
> process. We cannot, however, use any 3rd party devices or modules except for
> the actual XBee unit. Are we allowed to integrate the adapter schematic into
> our communications module schematic?
>
> --
>
> Office Use Only:
> From: Julien Mansier
> Email: julien.mansier@gmail.com

SparkFun



Julien Mansier to marketing — show details Jul 4 — Reply

Hello,
I am representing a group of electrical engineers from the University of Central Florida. We will be using several of your products in our Senior Design project. I have read and understand that all photos may be used without permission as long as it is for educational purposes. Our situation is just that. However, another company is giving us permission as long as our document is registered under a Creative Commons, and must be uploaded to the CC site for fair use. Will this conflict with your permissions?

I thank you for your time.

Julien Mansier
University of Central Florida
School of Electrical Engineering
julien.mansier@knights.ucf.edu

Reply — Forward

Adam Swetlik - Marketing to me — show details Jul 10 — Reply

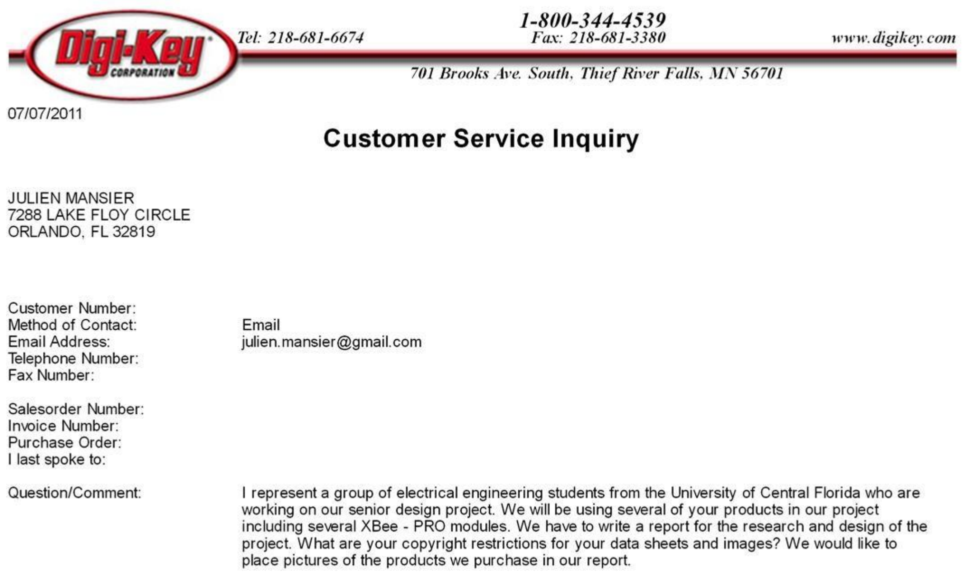
Hi Julien,

I think you're in the clear as far as permissions go on our side. Best wishes to you and your project!
- Show quoted text -

Reply — Forward

DigiKey, INC

Digi-Key Corporation - Customer Service Inquiry https://ordering.digikey.com/Forms/DKC/Print/WWW/CustomerServiceIn..



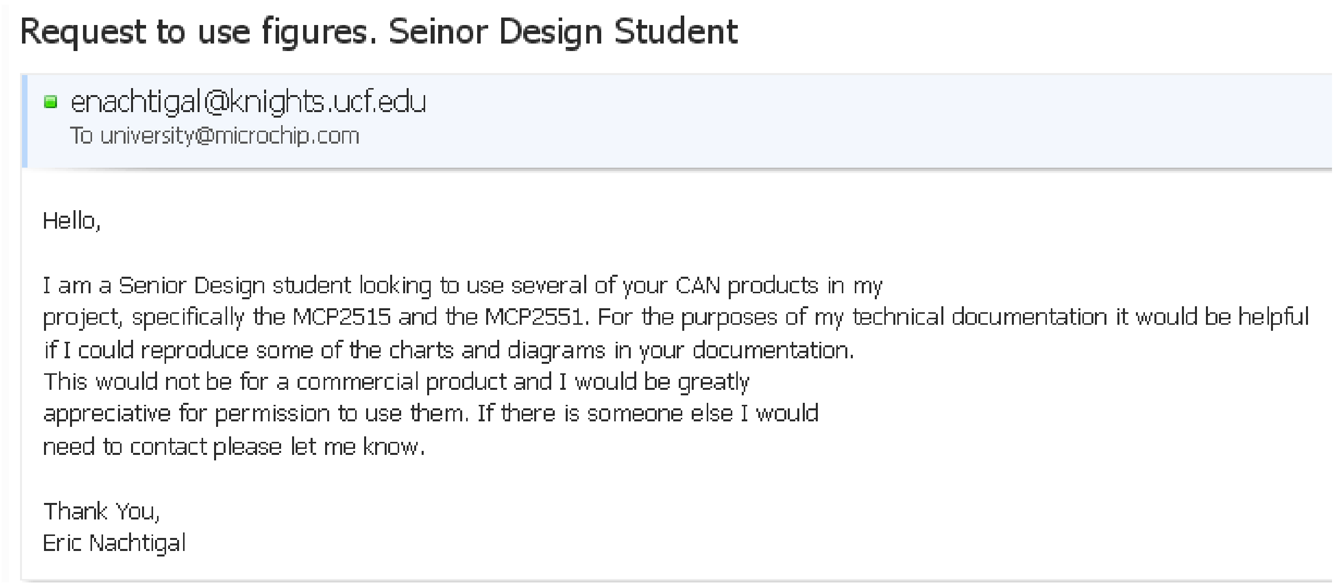
Tel: 218-681-6674 1-800-344-4539 Fax: 218-681-3380 www.digikey.com
701 Brooks Ave. South, Thief River Falls, MN 56701

07/07/2011

Customer Service Inquiry

JULIEN MANSIER
7288 LAKE FLOY CIRCLE
ORLANDO, FL 32819

Customer Number:	
Method of Contact:	Email
Email Address:	julien.mansier@gmail.com
Telephone Number:	
Fax Number:	
Salesorder Number:	
Invoice Number:	
Purchase Order:	
I last spoke to:	
Question/Comment:	I represent a group of electrical engineering students from the University of Central Florida who are working on our senior design project. We will be using several of your products in our project including several XBee - PRO modules. We have to write a report for the research and design of the project. What are your copyright restrictions for your data sheets and images? We would like to place pictures of the products we purchase in our report.



Request to use figures. Seinor Design Student

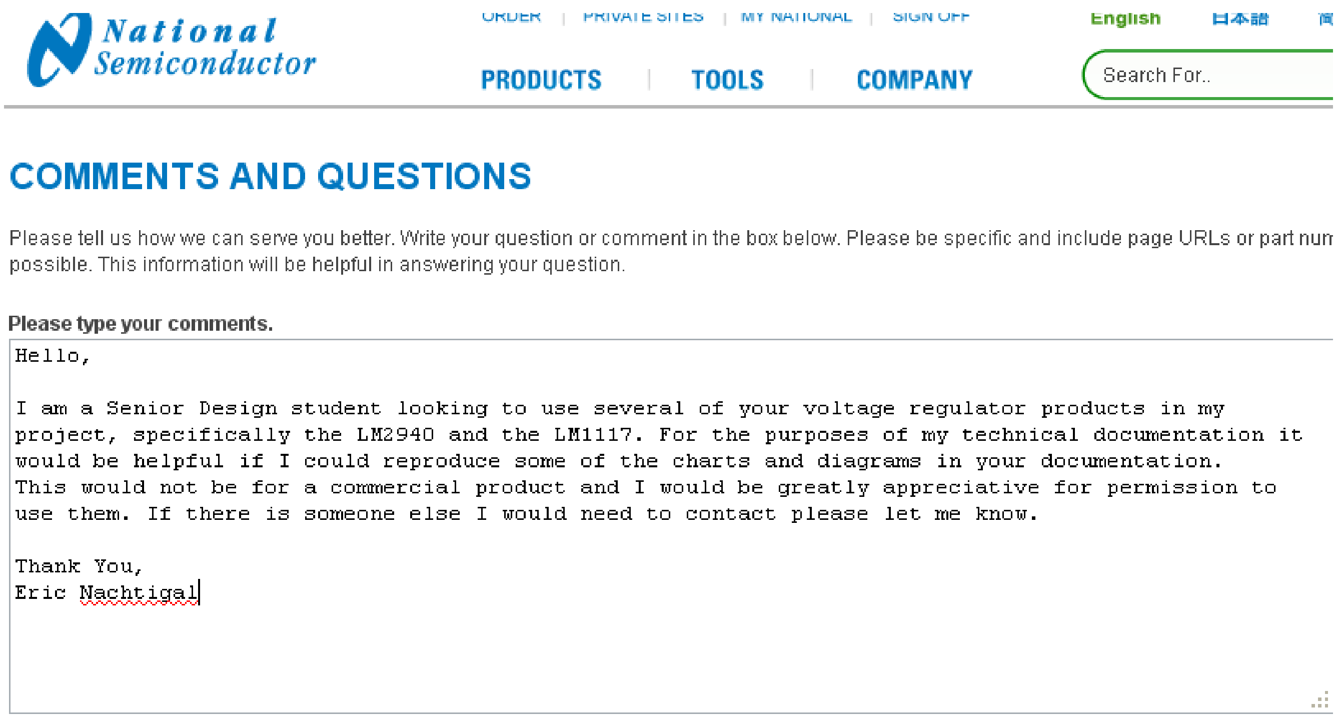
enachtigal@knights.ucf.edu
To university@microchip.com

Hello,

I am a Senior Design student looking to use several of your CAN products in my project, specifically the MCP2515 and the MCP2551. For the purposes of my technical documentation it would be helpful if I could reproduce some of the charts and diagrams in your documentation.
This would not be for a commercial product and I would be greatly appreciative for permission to use them. If there is someone else I would need to contact please let me know.

Thank You,
Eric Nachtigal

Microchip



National Semiconductor
ORDER | PRIVATE SITES | MY NATIONAL | SIGN OFF
English 日本語
PRODUCTS | TOOLS | COMPANY
Search For..

COMMENTS AND QUESTIONS

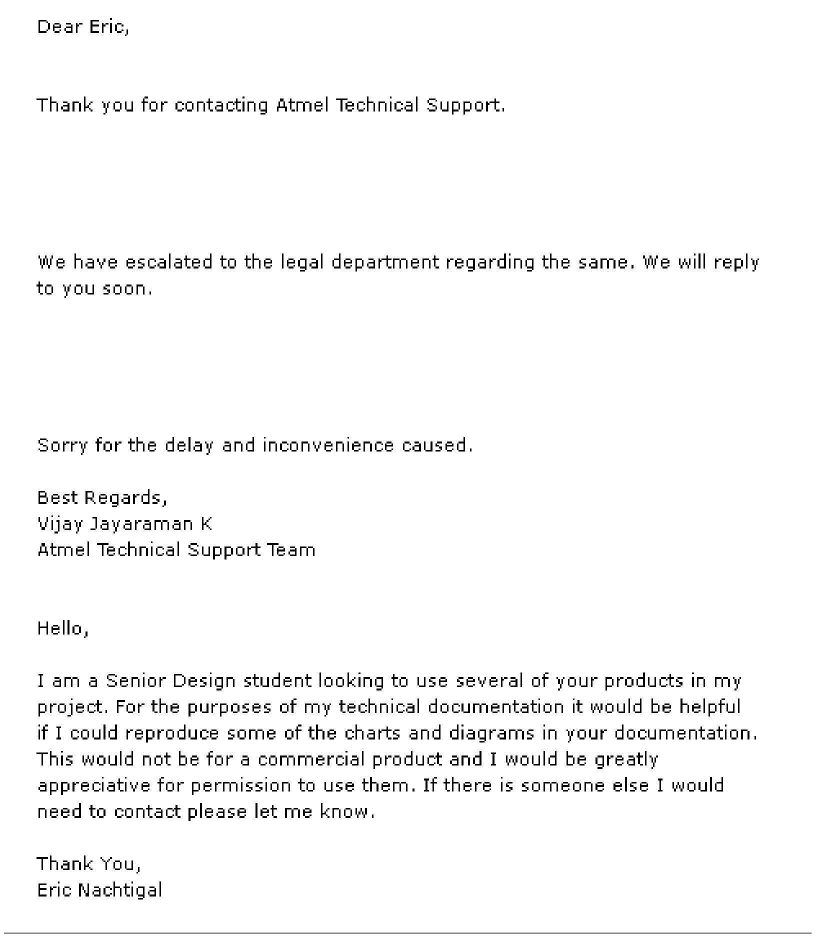
Please tell us how we can serve you better. Write your question or comment in the box below. Please be specific and include page URLs or part num
possible. This information will be helpful in answering your question.

Please type your comments.

Hello,

I am a Senior Design student looking to use several of your voltage regulator products in my project, specifically the LM2940 and the LM1117. For the purposes of my technical documentation it would be helpful if I could reproduce some of the charts and diagrams in your documentation.
This would not be for a commercial product and I would be greatly appreciative for permission to use them. If there is someone else I would need to contact please let me know.

Thank You,
Eric Nachtigal



Dear Eric,

Thank you for contacting Atmel Technical Support.

We have escalated to the legal department regarding the same. We will reply to you soon.

Sorry for the delay and inconvenience caused.

Best Regards,
Vijay Jayaraman K
Atmel Technical Support Team

Hello,

I am a Senior Design student looking to use several of your products in my project. For the purposes of my technical documentation it would be helpful if I could reproduce some of the charts and diagrams in your documentation. This would not be for a commercial product and I would be greatly appreciative for permission to use them. If there is someone else I would need to contact please let me know.

Thank You,
Eric Nachtigal

BOSCH SENSOR-SENSORTEC

Dear Sirs:

I am part of a team of students at the University of Central Florida working on our senior design project. We have chosen to use one of your accelerometers, BMA220, in our design. I would like to request permission to include some of the tables, figures and information from the data sheet into our report. Of course we will site all information used, as to give due credit.

Respectfully,

Jason Skopek
University of Central Florida
Department of Electrical Engineering
Jason@jasonskopek.com

Texas Instruments

Dear Sirs:

I am part of a team of students at the University of Central Florida working on our senior design project. We have chosen to incorporate one of your temperature sensors, TMP100, in our design. I would like to request permission to include some of the tables, figures and information from the data sheet into our report. Of course we will site all information used, as to give due credit.

Respectfully,

Jason Skopek
University of Central Florida
Department of Electrical Engineering
Jason@jasonskopek.com

Wahl Steven (BST/SNA)
To Jasonskopek@knights.ucf.edu

```
You can use...

-----Original Message-----
From: mailgenerator@de.bosch.com [mailto:mailgenerator@de.bosch.com]
Sent: Monday, November 21, 2011 12:51 PM
To: Wahl Steven (BST/SNA)
Subject: Request from Contact function: Permission

Permission

Title* : Mr.
Additional title :
First name : Jason
Name* : Skopek
Company : Student
Street* : 400 Central Florida BLVD
Postcode, Town* : 32816
Country* : USA
Your e-mail address* : Jasonskopek@knights.ucf.edu
Ref.* : Permission
Your Concern* :
I am a senior design student at University of Central Florida, I would
like to request permission to incorporate some information and figures
from the data sheets into our report. I am using an BMA220
```

```
Accelerometer.


Thank you,

Jason Skopek


Request from page        : http://www.bosch-sensortec.com/content/language1/html/3313.htm###http://www.bosch-sensortec.com/content/language1/html/index.htm
Selected options         :
Parameter                : Contact form - Contact function - Country: te
- language: en

----------------------------------------------------------------------
-----------------------------
This mail was generated by the Bosch contact window.
Please answer to the Reply-To address only.

Mail info:
_REPLY-TO                : Jasonskopek@knights.ucf.edu
_SUBJECT                 : Permission
```

RE: Permission to use

11:08 PM
Bassuk, Larry

To jasonskopek@knights.ucf.edu

Thank you for your interest in Texas Instruments. We grant the permission you request in your email below.

On each copy, please provide the following credit:

Courtesy Texas Instruments

Regards,

Larry Bassuk
Deputy General Patent Counsel &
Copyright Counsel
Texas Instruments Incorporated
214-479-1152

From: jasonskopek@knights.ucf.edu [mailto:jasonskopek@knights.ucf.edu]
Sent: Tuesday, December 06, 2011 4:48 PM
To: Bassuk, Larry
Subject: Permission to use
Hello Mr. Bassuk,

I wanted to request permission to use some of the information from the data sheet of the TMP100 temp sensor in our documentation for our senior design project at University of Central Florida. I am interested in the material contained in the data sheet; which consist mostly of the figures which show the pin layout and some of the tables, as well as an artist rendering of the sensor.

http://www.ti.com/lit/ds/sbos231g/sbos231g.pdf
http://www.ti.com/graphics/folders/partimages/TMP100.jpg

Thank you,

Jason Skopek