SAE Formula Car

Data Acquisition & Display System

Ahmed Albitar
John Gertie
Justin Ibarra
Sean Lenz

Advisor: Professor Steven Gutschlag

Bradley University
Department of Electrical Engineering
April 30, 2015

BRADLEY
UNIVERSITY

Abstract

In the past, catastrophic failures have impeded Bradley University's ability to compete in the annual SAE formula car competition. Failures can occur suddenly and can cause harm to the vehicle and anyone around it unless appropriate action is taken to prevent it. Most failures can be attributed to the lack of an effective warning system utilized by the current vehicle display system. Through collaboration with the mechanical engineers who build the vehicle, the SAE Formula car data acquisition and display team has built a prototype of a universal system that will alert the driver and pit crew of imminent failures. The system retrieves data from the sensors and displays the vehicle data to the on-board Amulet touchscreen and the pit crew's laptop. These displays feature a built-in notification system to alert the driver if values exceed a specified limit. The Aerocomm transceiver sends packets of data to the pit crew's laptop where LabVIEW processes, displays, and stores the data. Additional work is required to finalize the system and install it on the formula car. All necessary parts are purchased except for a protective shell, which is needed to securely install the ATmega128 microcontroller and external circuitry inside the vehicle.

Table of Contents

I. Introduction and Overview

A. Problem Background

1. SAE Competition

Every year the Bradley University mechanical engineering department competes in the Society of Automotive Engineers (SAE) formula car competition. The competition requires each team to develop a prototype vehicle which is to be evaluated for production. Each vehicle is to be designed for the non-professional weekend competitor. The vehicle should be designed to be low in cost, reliable, and easy to maintain. Aesthetics are also included to enhance the car's marketability. Many awards are presented during the competition, with the criteria for the awards ranging from vehicle design to its performance.

2. Bradley SAE Formula Car

The annual Bradley SAE formula car build has a budget of approximately $10,000. Each Bradley car utilizes an engine from a 2007-2010 Honda CBR600RR motorcycle. The last several versions of the formula car has used a Mychron 3 display system. The Mychron 3 system has now been discontinued, but retailed for around $825 while available. The Mychron 3 display features a 3.3 inch x 2.0 inch display, and includes a tachometer, a single temperature sensor, and lap timer. The system supports a single temperature sensor that can be used to monitor exhaust gas temperature, cylinder head temperature, or water temperature. Another feature of the existing system is an internal data logger. However, due to the limited onboard memory, the data logger can only record around 30 minutes of data.

3. Current System Flaws

The current display system has many flaws and limitations. There are many sensor readings that are crucial to adequately monitor a vehicle during operation. Although the current data acquisition system does support one temperature sensor, other readings are ignored. For example, if oil pressure drops too low, catastrophic failure could result. Additionally, the current system relies on four small light-emitting diode (LED) indicators to notify the driver of problems. The layout of the display positions these lights outside of the driver's line of sight. Therefore, during a race it would be difficult for a driver to quickly glance at the tachometer and also look for other active warning indicators. The display must be designed to permit the driver to view all critical vehicle data with a single glance.

The display also lacks customizability. Without any customization, the driver can't arrange the data to his/her liking. This forces the driver to simply make do with what is there even though it may hinder their performance on the track. Also, while one display may be suitable for one environment, it may not be ideal for another. For example, in a racing environment, there is a tradeoff between the quantity of data displayed and the size to which it is

displayed. The driver may not need to see all the data from the system, and may opt to display certain values at a larger size making them easier to see while racing. On the other hand, if the driver is simply testing and tuning the vehicle the team may opt to have more data values shown, and may not be as concerned about value size. Visibility may also become an issue with the current notification system's indicator lights. In the case of direct sunlight, the driver may not be able to distinguish a lit bulb from an unlit bulb. These flaws prevent the current data acquisition system from effectively notifying the crew and driver of crucial engine and vehicle parameters.

With a past plagued by vehicle failures, it is evident that the Bradley SAE formula car is in need of a more sophisticated data acquisition system. A more advanced system will process data from the vehicle and recognize potential problems before the problems become severe. When values begin to exceed a programmed tolerance, the system will relay the alert to the driver through the touch screen display. Alerts will also be sent to the pit crew using radio frequency (RF) transmission to a remote display. By notifying the driver and crew early, the system can reduce the likelihood of catastrophic failure.

One past vehicle failure involved a driver who was not aware that the vehicle's engine was overheating. The driver continued to push the engine while practicing and within minutes the engine was destroyed. This failure was the direct result of the notification system failing to get the drivers attention to alert him of excessive coolant temperatures.

4. New Data Acquisition & Display System

A new system is a necessity for the mechanical engineering group. The new system would keep the driver and crew aware of the vehicles real-time status. With readings including oil pressure, water temperature, battery voltage, and the internal engine speed, the crew and driver will have peace of mind knowing that the vehicle is functioning properly. If a failure occurs, the driver and crew will be notified through an aggressive notification technique. The data logging feature, in conjunction with the gauge readings, will be useful in diagnosing the cause of the problem.

Implementation of the new system would be extremely beneficial for the mechanical engineering group. By preventing catastrophic failures, it could save the mechanical engineers substantial amounts of money in potential repair costs alone. The system will also provide the mechanical engineering team with a competitive edge. The "Demo" mode of the system is intended for use in the design portion of the competition. No other team has a system with such capabilities. The advanced data acquisition and display system also provides tools like the data logger that will aid in testing and tuning the vehicle.

B. Problem Statement

Every year the Mechanical Engineering department at Bradley University designs and constructs a formula racing car. Past performances have proven to be inconsistent due to engine failures and structural breakdowns. To improve future performance, an advanced data acquisition system will be employed to indicate problems before a failure occurs. Unlike the existing system, both the driver and the crew will monitor data. A touch screen mounted in the vehicle will display data and warning signals to the driver. The same data will also be transmitted to a computer, where it will be recorded for diagnostic evaluations. Multiple indicators will be used to warn the driver and crew if data readings exceed a safe limit. This system will provide the necessary information to optimize the formula cars performance, giving Bradley's mechanical engineering department an edge over the competition.

C. Constraints of the Solution

TABLE I - SYSTEM CONSTRAINTS

Constraints	
Wireless	Minimum range of 800 m
Size	Fits on vehicle dash
Safety	Does not pose additional safety concerns
Cost	Within senior project budget ($1000)

There were four key constraints to the data acquisition system. First, wireless transceivers were required to send the data to the pit-crew's display. The pit crew could then monitor the car's key functions to make sure the driver does not miss any errors. The next constraint in Table 1 is the size of the system. The SAE team that builds the car has to follow strict rules that limit the size of the formula car. Therefore, the data acquisition and display system has to be small enough to fit safely inside the confines of the car. Safety was also another constraint considered for the solution of the system. There are many parts that could cause injury to the driver such as loose wires or parts coming loose. All of the parts to the system will need to be mounted to the car so that in the case of an accident, nothing comes flying out to injure people. Also to be considered with the safety of the driver is the notification system that will alert the driver of any failures occurring in the car. If the notifications are too aggressive, then the driver could be distracted and lose control of the vehicle. This is obviously something that needed to be avoided. The notification system is designed to draw the driver's attention in a way that would not distract them from operating the formula car. The last constraint in Table 1 is cost. Many of the parts for the system are expensive. Software and hardware costs could have totaled in the thousands of dollars. However, because the team was able to acquire student versions of software, a lot of expenses were saved. Also the team had sensors that were bought by previous groups that the team was able to use to save money. These cost savings allowed for only $100 to be spent on extra equipment needed.

II. Statement of Work

A. Nonfunctional Requirements

TABLE II- PROJECT FUNCTIONAL REQUIREMENTS

Requirements		
Wireless	Minimum range of 0.5 to 1 mile	
Display	Safe	
Safety	No loose parts	
	No exposed wiring	
	Does not interfere with driver performance	
	Not a fire hazard	
	Does not pose threat in case of accident	
Construction	Inside cockpit	Withstand temperatures of 40 to 120 ˚F
		Withstand vibrations/forces of lateral magnitude 0 to 2G and forward of 5G
	Inside Engine Bay	Withstand temperatures of 40 to 250 ˚F
		Withstand vibrations/forces of magnitude up to 8G
	Undercarriage	Withstand temperatures of 40 to 250 ˚F
		Withstand vibrations/forces of magnitude up to 8G

B. Functional Requirements

TABLE III- PROJECT FUNCTIONAL REQUIREMENTS

Requirements	
Data Acquisition	Acquire data with max 5% error
	Send data for display
	Store data for review
Wireless	Accurate transmission
Display	Accessible to driver and pit crew

C. Design Overview

1. System Block Diagram or Black Box

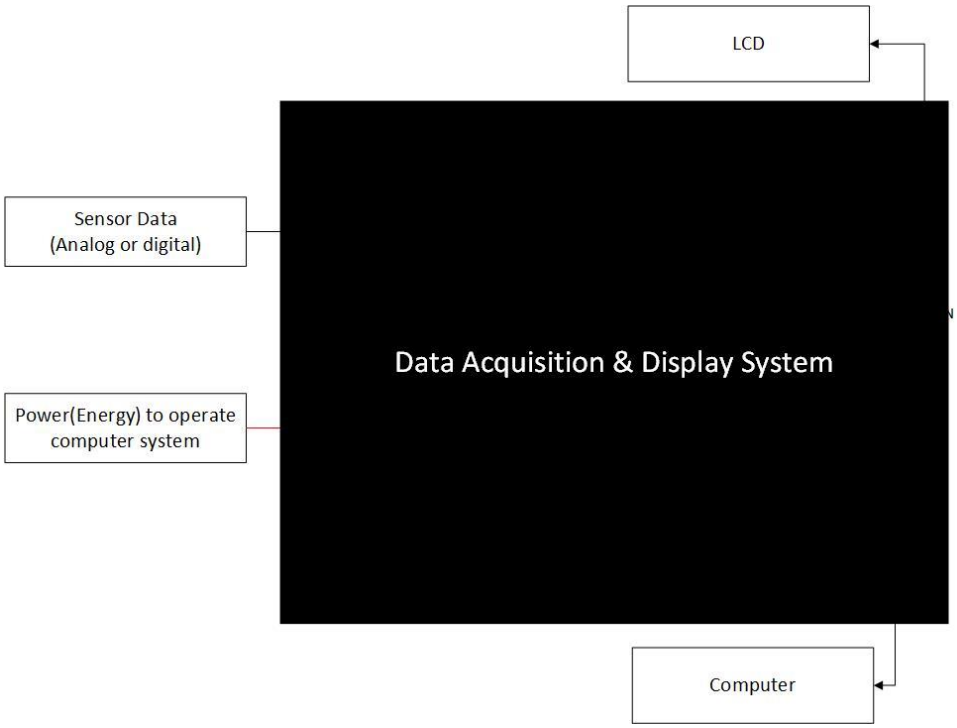

FIGURE I- BLACK BOX

Figure I is the black box used to represent the high level inputs and output of the data acquisition and display system. As shown in the figure, the inputs for the data acquisition system include sensor data and power. Sensor data can be either analog or digital depending on what data sending unit is chosen. The outputs will be a computer by the pit crew, and the LCD for the driver.

2. Subsystem Block Diagram or Glass Box

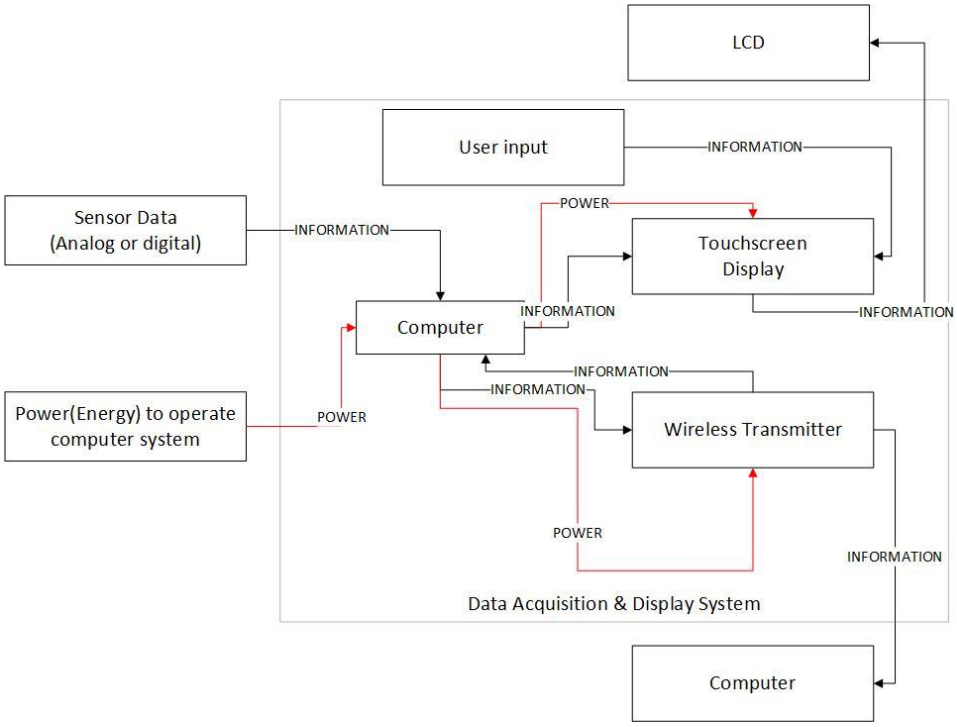

FIGURE II- GLASS BOX

Figure II shows the glass box of the data acquisition and display system. Similar to the black box, the glass box provides a high level functionality of the system. For the glass box, the internals of the black box system are also shown. As shown in the figure, information is taken from the sensors and passes through the computer. At this point, if an analog signal is received, it would be converted to a digital value. Digital values are then relayed to the touch screen display for the driver to view, and the wireless transmitter. After receiving the digital values, the transmitter will send values to the computer. Once data is transmitted, the wireless transmitter will send a signal back to the computer acknowledging that itset of values.

3. State Diagram or High-level Flowchart

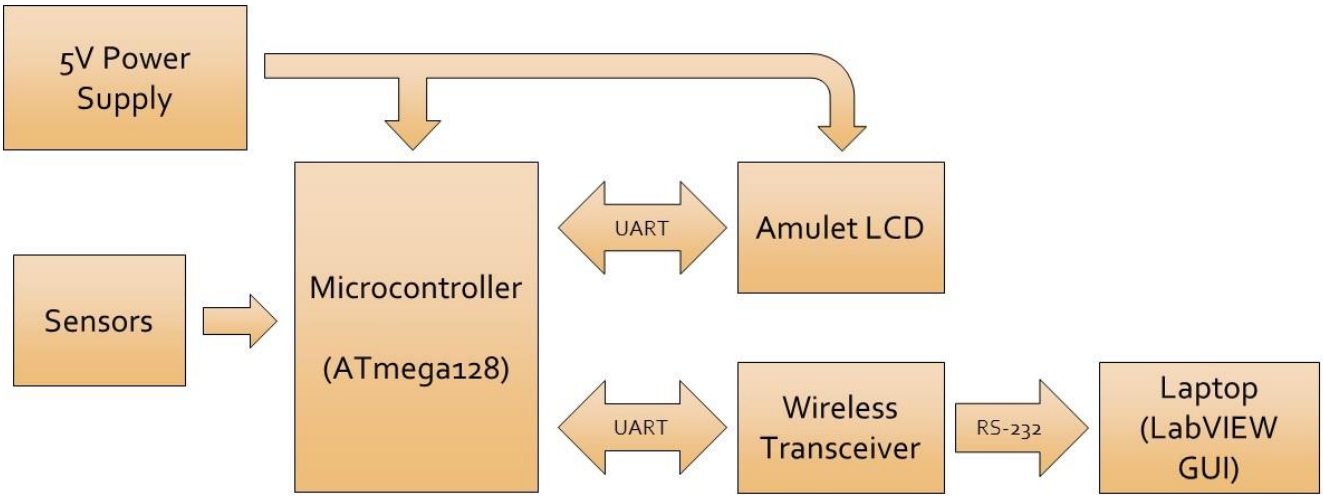

FIGURE III – HIGH-LEVEL BLOCK DIAGRAM

4. Division of Labor

TABLE IV-DIVISION OF LABOR

Ahmed	• Sensor selection • Sensor interfacing with the microcontroller
John	• Interface microcontroller with HyperTerminal • Test microcontroller with simulated sensor data • Interface microcontroller with LabVIEW
Justin	• Amulet display • Interface microcontroller with HyperTerminal • Test microcontroller with simulated sensor data • Interface microcontroller with LabVIEW
Sean	• Prepared LabVIEW to receive wireless data • Interface microcontroller with Amulet • Setup external power supplies for the microcontroller, Amulet, and Op-Amps

Table IV shows the division of labor of all the team members. Each individual worked on getting their tasks done throughout the project and then collaborated with each other to put together the design.

5. Hardware

TABLE V- HARDWARE USED IN SYSTEM

Equipment	**Description**
ATmega128	Microcontroller
Amulet LCD	Driver touch screen display
Aerocomm AC4790	Wireless transceiver

The reason we choose the ATmega128, Amulet LCD and the Aerocomm AC4790 is that these equipments satisfied all of our design constraints The remote display is connected to an Aerocomm AC4790 transceiver through a serial cable. The AC4790 chip for the remote display is left on the development board included in the Aerocomm kit. For the transceiver to be installed in the vehicle, the AC4790 chip was removed from the development board and wired to a breadboard. The transmit and receive pins from the transceiver are connected to the pins on the Atmega128 coordinating to UART0. In addition, PD1 was set to provide the transmit enable signals for the AC4790.

6. Software

TABLE VI-SOFTWARE USED IN SYSTEM

Programs	**Description**
Atmel Studio	Microcontroller programming software
GemStudio	Amulet LCD programming software
LabVIEW 2014	Pit crew display programming software
HyperTerminal	Serial communication software on Window's PCs

The software written for the microcontroller was done in Atmel Studio. The code is designed to retrieve data from the sensors, convert that data to an ASCII value and then send that data to two system displays.

Variables the team assigned for each sensor, as well as sensor maximum values. For coolant temperature, the maximum value was set to 250°F. Oil pressure used a maximum value of 100psi. Vehicle speed was set to 160, and engine speed was set to 15000. These maximum values were chosen because they are appropriate for the engine used in the SAE Formula Car.

In the main function, the program begins by initializing all ports. Ports C, A, and E are set as outputs. Port F is set as an input. The timers are initialized. The timers control when data is acquired for display. There are two separate counter variables in the timer ISR. *display_time* controls data being loaded into the Amulet display data array. "pc_time" controls data being loaded ino the LabVIEW data array. Readings like MPH and RPM are converted to an ASCII value within the timer ISR as well.

Next in the main function is the UART initializations. UART0 is set for the LabVIEW, and UART1 is coded for the Amulet. Both UART ISRs are responsible for transmitting data.

An analog to digital converter is used to convert incoming sensor data into digital values. Coolant temperature and oil pressure sensors are both included in the ADC. For each sensor, data is retrieved, and then converted into ASCII form.

The last step of our main function is a transmission rising edge which is set to PD1. This rising edge is necessary to control the Aerocomm transceiver and signal that a transmission can begin. This rising edge is created using a bitor operation with interrupt 1 controls.

The ASCII conversion function is designed to use case statements to translate data values into ASCII characters.

7. Interface

In order to interface the microcontroller with LabVIEW, the data needed to be converted into a format that could be read by LabVIEW. To accomplish this task, the built-in analog-to-digital converter on the microcontroller was used. This converter converts the analog sensor data to a digital value. After this conversion, an ASCII function is used to convert the digital value into its' ASCII equivalent. Since LabVIEW is able to read values in an ASCII format, the other portion of LabVIEW interfacing was done from the LabVIEW software.

D. Economic Analysis

TABLE VII- EQUIPMENT PRICES

Equipment	Price
Software Programs	$2000
Sensors	$320
Screen	$190
RF transceivers	$130
TOTAL	$2745

After summing up all the equipment's prices, the cost to finish the project from scratch is $2745. The estimated cost to finish the project would be $2745, this includes Software Programs estimated to cost $2000. Although these number sounds high, programs are going to be the biggest factor because they are the most expensive equipments. However, the software programs are for free to students and some Electrical Engineering Department lab's desktops already have some of those software programs installed, so it will cost the electrical engineering department less money to make this project.

III. Design Testing and Validation

A. Sensor Testing

For the temperature sensor, a LM741 Op-amp was used to convert the analog output current to analog output voltage. The output voltage is between 0V to 5V in order for the voltage to be compatible with the Analog to Digital converter on the ATmega128. The sensor was tested and a Linear Output was shown. Using a Thermometer to measure the temperature of water inside a hot plate, the results from the temperature sensor were within 2 percent error.

For the pressure sensor, a LM741 Op-amp was used to convert the analog output current to analog output voltage. The output voltage is between 0V to 5V in order for the voltage to be compatible with the Analog to Digital converter on the ATmega128. The sensor was tested and a Linear Output was shown. Using an air tank to measure the pressure, the results from the pressure sensor were within 2 percent error.

RPM and Velocity Sensor: No circuitry was needed for the velocity and RPM sensors. The testing showed that the sensors are linear with an output pulse voltage from 0 to 50 volts. 2 holes were drilled into a metal plate for the sensors to be implemented on, the metal plate was hooked to a rotating motor. Using an actual speedometer to measure the RPM, the results from the sensors were within 2 percent error.

B. Amulet UART Communication Testing

To test the functionality of communication between the ATmega128 microcontroller and the Amulet LCD, the values being sent need to be compared to oscilloscope readings and also HyperTerminal values. To read the oscilloscope readings, some calculations needed to be done to see how many microseconds it took for one bit to be sent. Since the baud rate is 9600 bps, one bit equates to a sending time of about 104us. Then the sending order needed to be researched. The ATmega128 datasheet specifies that most significant bit (MSB) is sent last. Also the byte that is sent would be started by a single start bit and ended with a single stop bit. The start bit is always logic low, and the stop bit is always logic high. Once these attributes are figured out, the oscilloscope data can be interpreted correctly. Every 104us there would be a new bit sent. After reading through 10 bits and marking down whether the bit is logic high or low, the first byte can be interpreted into meaningful data. The output would be a binary value, which can be converted into hexadecimal to be compared to the HyperTerminal values being sent from the microcontroller. If all of these values match what is trying to be sent, then the UART is working correctly.

C. Amulet and ATmega128 Power Supply Testing

The Amulet and ATmega128 both require 5V power supplies to operate correctly. The formula car will have a car battery of 12V that can be used to power the electronics. To get the required 5V for the electronics, a DC/DC converter was used. The DC/DC converter had two outputs, one for -5V and one for +5V. The -5V would be used for a different application in the system. The DC/DC converter also allowed for a maximum of 1.5A current draw. This would be plenty of current to supply both the Amulet and ATmega128. Once the DC/DC converter was wired up, it was connected to a variable power supply to imitate what the car battery would look like. Once testing the voltage output, it was ready to power the Amulet LCD. Included in testing procedures was to test the current draw from the Amulet. When no equations or applications were running on the Amulet, it drew about 200mA of current. According to the ATmega128 datasheet the maximum supply current ranged from 200-400 mA. Summing both of these currents results in being well below the maximum supply current for the DC/DC converter. If there is a momentary spike in the current there is at least 800mA of extra range for the current.

IV. Conclusion

Every year the mechanical engineering team at Bradley University builds a formula car for the SAE competition. The current data display system is inadequate at alerting the driver that there are problems with the specifically monitored data (e.g. oil pressure, water temperature, speed, RPM, battery voltage). Such failures at warning the driver has led to catastrophic failures and because of this, the mechanical engineers that build the car have expressed a need for a better system. The mechanical engineers have made a request for a system that has three goals: a multi-mode display system, wireless transmission of data to pit crew's laptop, and data logging abilities. This system has met all of these goals. The Amulet LCD will be an interactive touch screen and have 3 modes: racing, practice and demo. This LCD has aggressive notifications that will alert the driver of any problems. The wireless transmission of data is accomplished through RF communication. The communication is controlled from a microcontroller and is sent to a laptop where the crew can monitor the data. LabVIEW is used to program the display for the laptop. Also, LabVIEW allows all data to be logged for further analysis by the crew and driver. Through this system's more advanced notification system, there will be a safer environment for the driver and everyone around the car. As a result, it will also eliminate any catastrophic failures because the driver was not alerted of a problem; it will also reduce environmentally harmful spills from the car. This proposed system offers a universal application to other formula cars that will be built in the future. This flexibility to adapt to new cars offers to maximize the utility of the system. From our economic analysis, the system has an estimated cost of $3,000-3,700 to complete and install on the car. The system will be able to offer a competitive edge in the next competition.

V. Acknowledgments

Thanks to Jon Kellog from the Mechanical Engineering department for answering all of our questions regarding the SAE formula car.

VI. References

http://cegt201.bradley.edu/projects/proj2011/pjacher/SAEDAQ/Deliverables_files/SAEDAQ_final_report.pdf
http://www.atmel.com/images/doc2467.pdf
http://www.amulettechnologies.com/images/stories/Downloads/mk480272cdatasheet1112.pdf
https://www.dropbox.com/s/l8abp41iru83oqg/Datasheet_carspd_eng_101.pdf?dl=0
http://www.automationdirect.com/static/specs/prosensettrans.pdf
http://www.automationdirect.com/static/specs/prosensetransmitters.pdf

VII. Appendix

Parts List

TABLE VIII - SYSTEM PARTS LIST

Hardware	**Software**
Aerocomm AC4790	Atmel Studio
Atmega128	LabVIEW
Breadboard	HyperTerminal
Wire	GemStudio
Serial cable	
Amulet	

Detailed Test Procedures

Wireless Transmission

Testing began by establishing Aerocomm to Aerocomm communication. This was done through the use of the Aerocomm configuration utility. The baud rate used was 9600Hz. Once communication was established between the two transceivers, the receiving board was connected to a HyperTerminal through a serial cable. After data transmitted to the receiving Aerocomm transceiver started displaying in HyperTermial, the transmitting board was interfaced with the microcontroller. After interfacing the microcontroller to the Aerocomm transceiver, simulated sensor data was used to test data transmission to HyperTerminal. A power supply was used to simulate the linear output of coolant temperature, and oil pressure sensors. The pulse output of vehicle and engine speed was simulated with a wave generator. An oscilloscope was used to confirm the data being received was correct. The next step was to interface these values with LabVIEW.

Amulet Display

To test the Amulet display pseudo values were first used. This was done so the display can be worked on without waiting for the sensor interfacing to be completed. With these pseudo values the "Demo Mode" page was created. The testing done within this page included running the page in the simulator to make sure the notification system was in use and the data was sweeping from the minimum to the maximum values. Next, testing was done on the "Race Mode" and "Practice Mode" pages. To test these pages the microcontroller sent out fixed values via UART to the Amulet display. The Amulet was then viewed in the simulation mode for these pages to verify the Amulet was receiving the correct values. If the values were incorrect the receive pins on the Amulet were viewed and compared to the transmit pins on the microcontroller using an oscilloscope.

Detailed cost analysis

TABLE IX - EQUIPMENT PRICE

Equipment	Price
Pressure Sensor	$126
Temperature Sensor	$126
Amulet LCD	$190
RPM Sensor	$42
Vehicle Speed Sensor	$42
Aerocomm AC4790	$150
LabVIEW 2014	$1000
Atmel Studio	$500
GemStudio	$500
Atmega128	$25
DC/DC Converter ± 5V	$20
DC/DC Converter ± 15V	$20
Level Shifter	$4

Detailed schematics, models, flowcharts

Amulet Display

The Amulet display was successfully implemented. Four pages were created, "Home", "Demo", "Race", and "Practice", that included an aggressive notification system. These pages can be seen in FIGURE III- while the notification system can be seen in . The "Home" page successfully navigates the user to the three different modes by touching which mode the user would like to use on the screen. Also, the display can be easily navigated back to the "Home" page so the user can switch to a different mode. This was done using a widget that can be seen in the upper left of the pages as an arrow facing in the left direction. To ensure the safety of the driver an aggressive notification system was implemented. To make this system every image on the screen had to either be a numerical field or string field widget. The reason for this is to ensure the notification system adequately alerts the driver. Since every image on the screen is a widget they can be programmed to disappear. This is necessary because when a sensor value surpasses the allotted minimum or maximum it will flash red while every other image on the screen disappears.

FIGURE IV – HOME PAGE

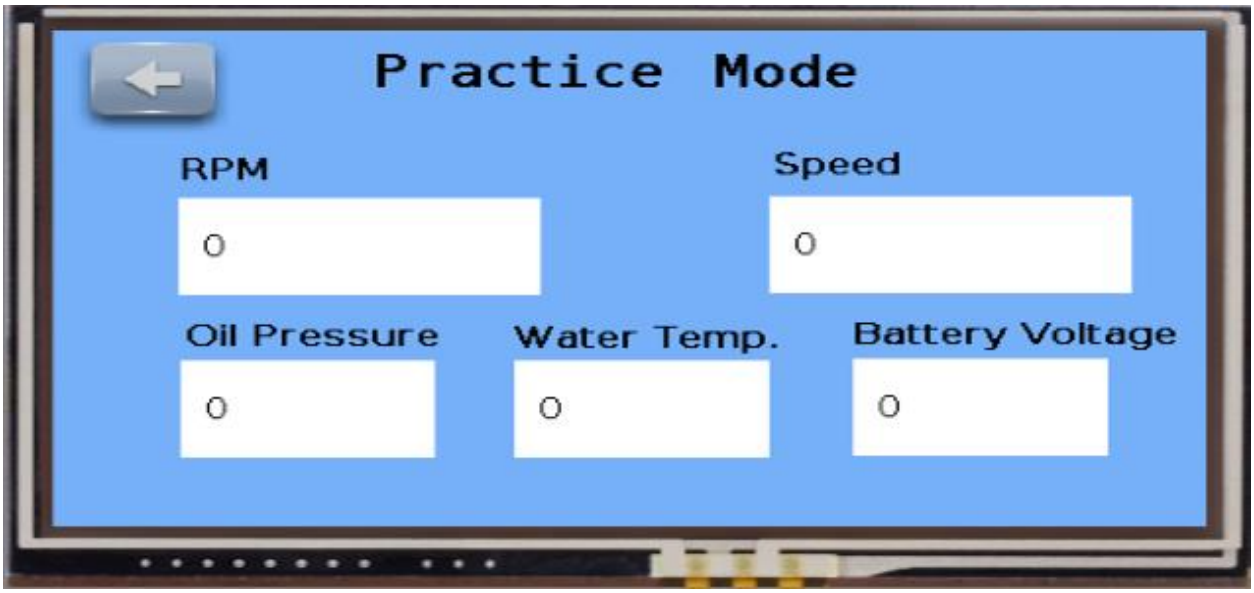

FIGURE V – PRACTICE MODE

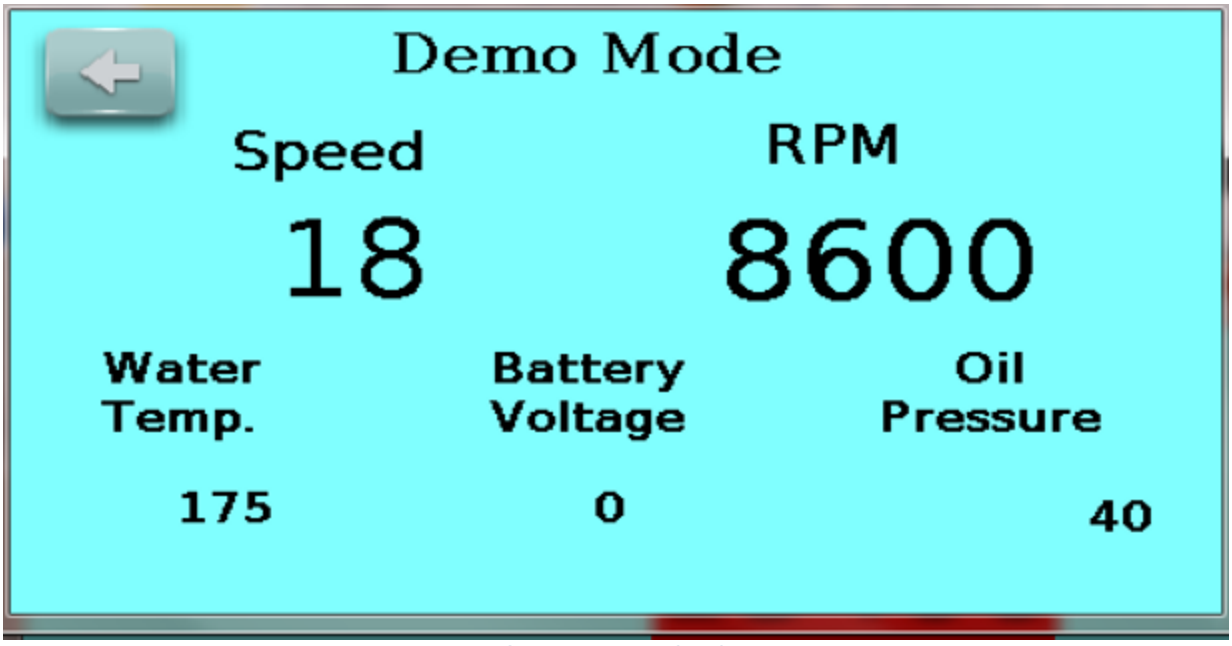

FIGURE VI – DEMO MODE

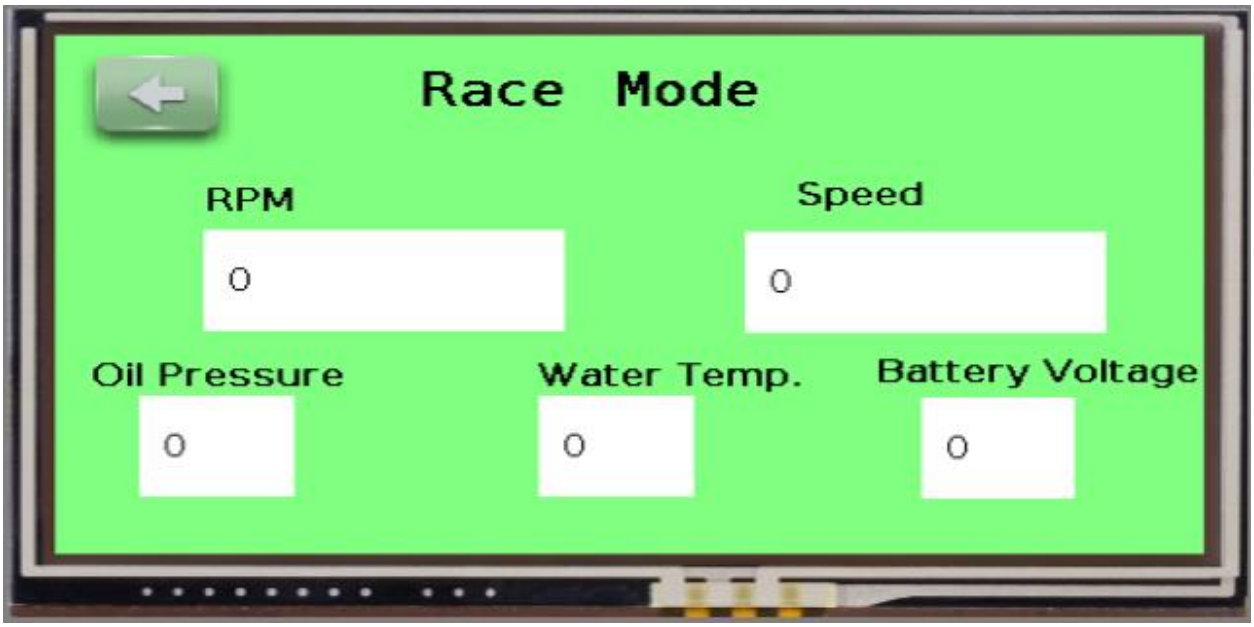

FIGURE VII – RACE MODE

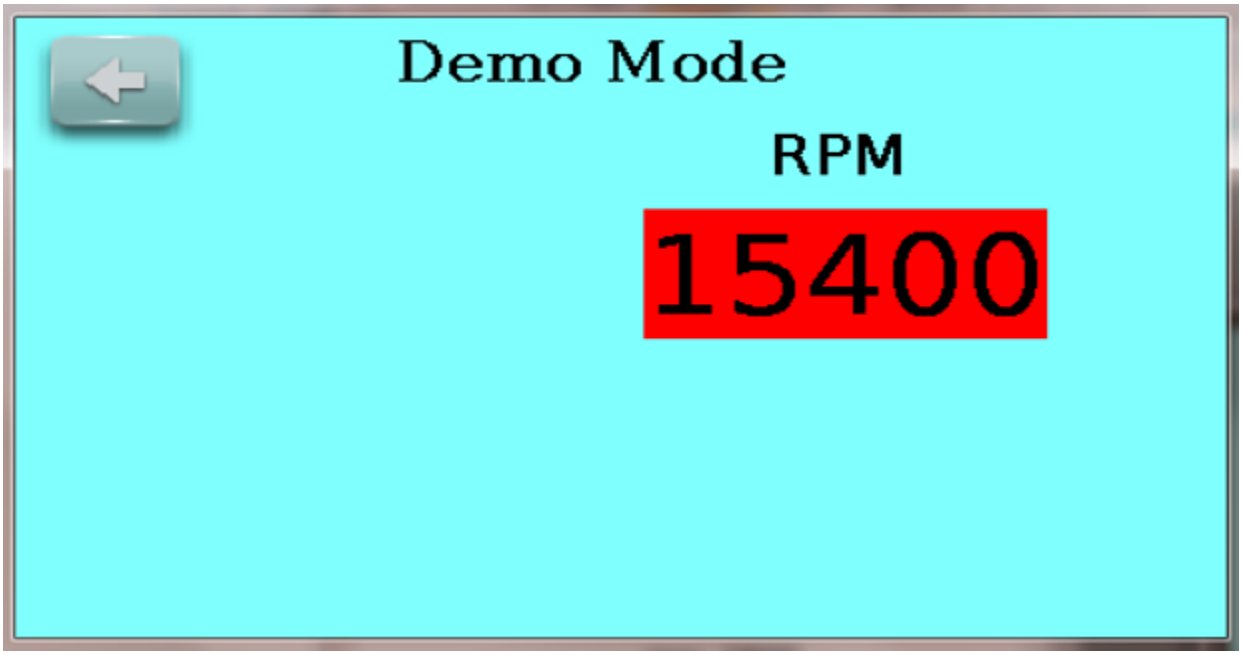

FIGURE VIII – NOTIFICATION SYSTEM IN USE

Amulet LCD UART Protocol

The communication method between the Amulet and the ATmega128 microcontroller is UART. It is sent in ASCII form. The ASCII communication protocol requires the message sent to the Amulet to be in a specific form, indicated in Table 7 below. The first byte tells the Amulet if a byte or word variable is being changed or copied. Bytes two and three tell the Amulet the specific location the variable is to be stored in the internal RAM of the Amulet. The rest of the bytes will contain what the variable value is to be changed to.

TABLE X - AMULET ASCII COMMUNICATION PROTOCOL TO CHANGE OR COPY VARIABLES

Message	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7 ..... Byte N	
Microcontroller Set Byte Variable	0xD5	Variable Hi Nibble	Variable Lo Nibble	Value Hi Nibble	Value Lo Nibble	None	None	None
Amulet Response	0xE5	Variable Hi Nibble	Variable Lo Nibble	Value Hi Nibble	Value Lo Nibble	None	None	None
Microcontroller Set Word Variable	0xD6	Variable Hi Nibble	Variable Lo Nibble	|-----MS Byte-----| Value Hi Hi Nibble	Value Hi Lo Nibble	|-----LS Byte-----| Value Lo Hi Nibble	Value Lo Lo Nibble	None
Amulet Response	0xE6	Variable Hi Nibble	Variable Lo Nibble	|-----MS Byte-----| Value Hi Hi Nibble	Value Hi Lo Nibble	|-----LS Byte-----| Value Lo Hi Nibble	Value Lo Lo Nibble	None

TABLE XI – EXAMPLE TO CHANGE WORD VARIABLE 76 TO HEXADECIMAL VALUE 0X02C9

Amulet LCD Module	Dir.	External Processor	Description
	<<	0xD6 0x37 0x36 0x30 0x32 0x43 0x39 '7' '6' '0' '2' 'C' '9'	Set IR word var 0x76
0xE6 0x37 0x36 0x30 0x32 0x43 0x39 '7' '6' '0' '2' 'C' '9'	>>		Confrim setting of IR word var 0x76

- A "Get Internal RAM byte variable" request.
- A "Get Internal RAM word variable" request. (word = 2 bytes)
- A "Get Internal RAM string variable" request.
- A "Get Internal RAM RPC buffer" request.
- A "Set Internal RAM byte variable" command.
- A "Set Internal RAM word variable" command.
- A "Set Internal RAM string variable" command.
- A "Set Internal RAM byte variable array" command.
- A "Set Internal RAM word variable array" command.
- A "Draw line" command.
- A "Draw rectangle" command.
- A "Draw filled rectangle" command.
- A "Draw pixel" command.
- A "Jump to specific page" command.

TABLE XII – LIST OF OPERATIONS THE MICROCONTROLLER CAN CONTROL ON THE AMULET LCD

LabVIEW Display

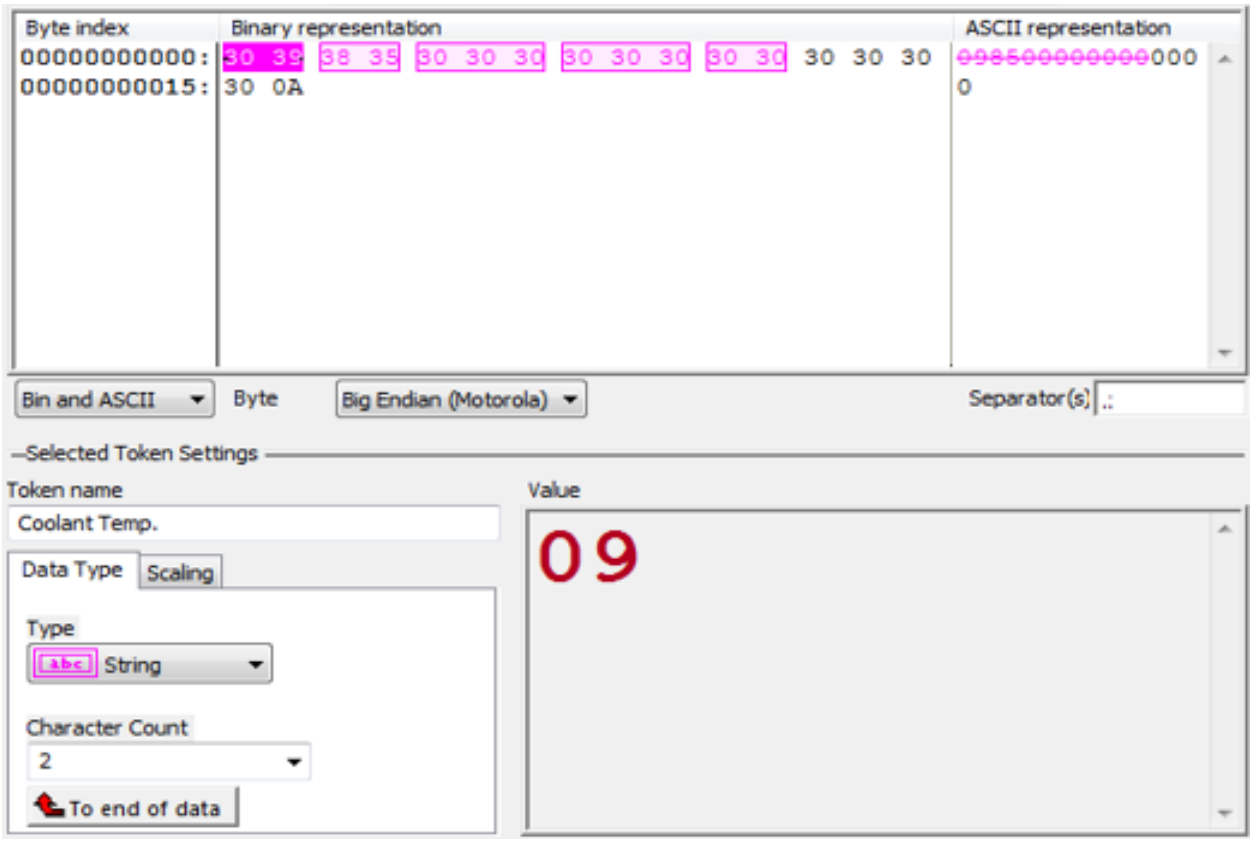

FIGURE IX - LabVIEW INSTRUMENT I/O ASSISTANT

The LabVIEW Instrument I/O Assistant is responsible for parsing through the data that is received from the wireless transceiver. The general user interface (GUI) shows the byte packet received. It allows the user to parse through the data and assign specific bytes to specific variables to be displayed to the gauges. The bytes are put into string variables in order to display the ASCII values to the display. Also another feature of the Instrument I/O Assistant is that it shows the binary and ASCII values received from the serial port. Also it shows the ASCII character value in the bottom right. This value would be what is shown on the gauge in the front panel.

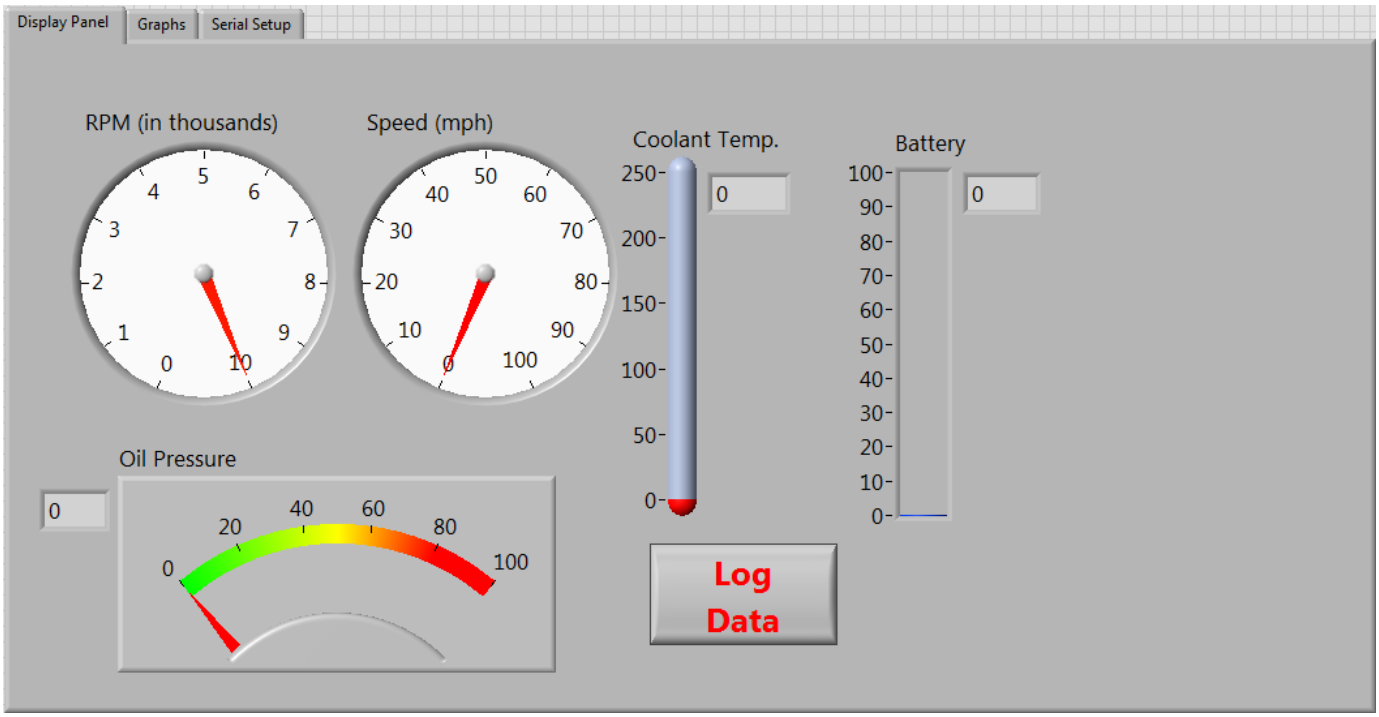

FIGURE X - LabVIEW FRONT PANEL DISPLAY

The front panel in LabVIEW is what is displayed to the user. This is where the gauges are displayed as well as graphs that plot the data over time and also a serial communication setup. The serial communication setup sets values such as which comm. port is in use, how many data bits, stop bits, and start bits are to be used. The log data button saves the data that has been received into a text file and then closes the comm. ports to stop displaying data. The block diagram of LabVIEW, shown in Figure 4, is where the front panel gets its functionality. This is where the Instrument I/O Assistant is located as well as all the blocks for displaying the correct values to the front panel.

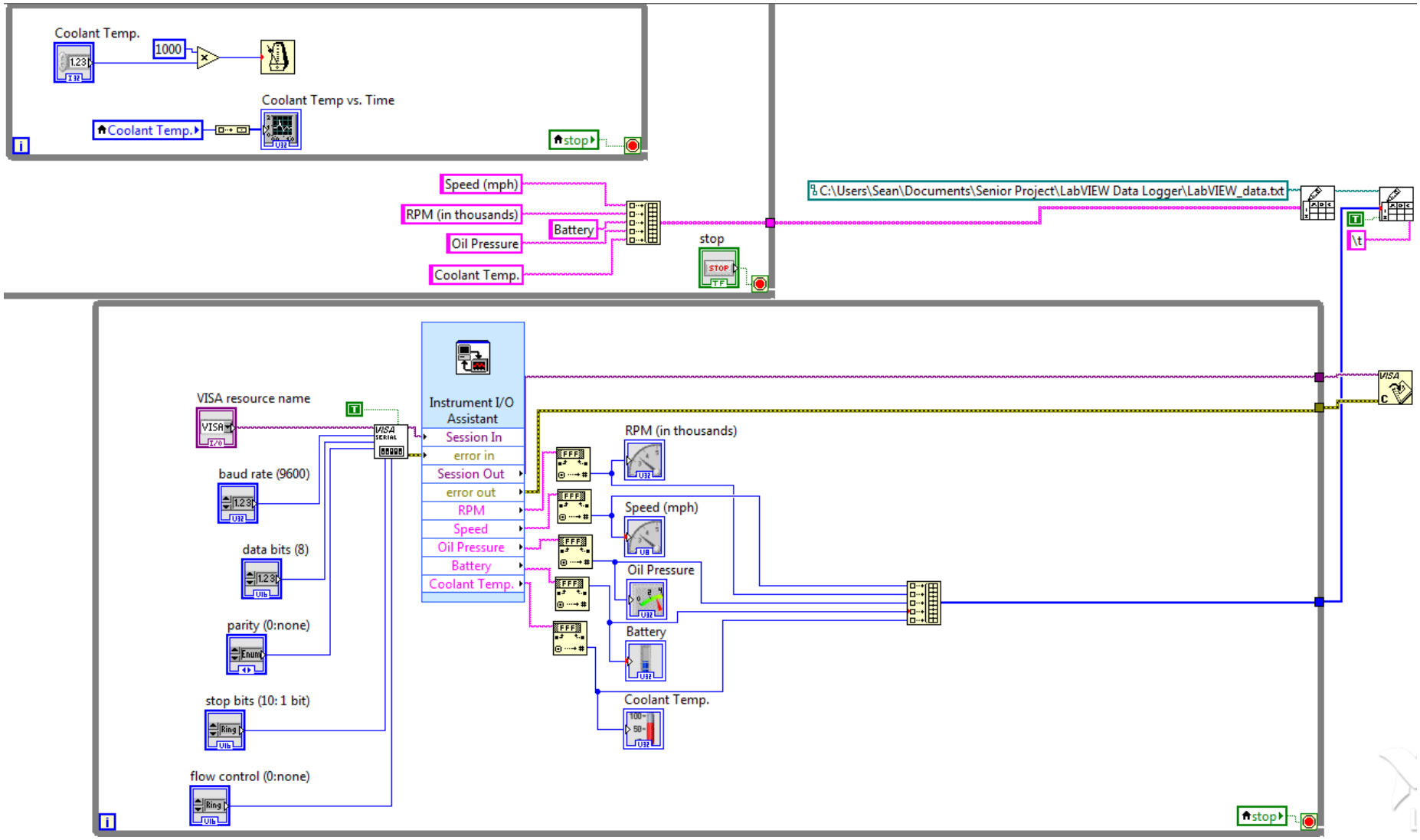

FIGURE XI - LabVIEW BACK PANEL BLOCK DIAGRAM

Aerocomm AC4790 Schematic

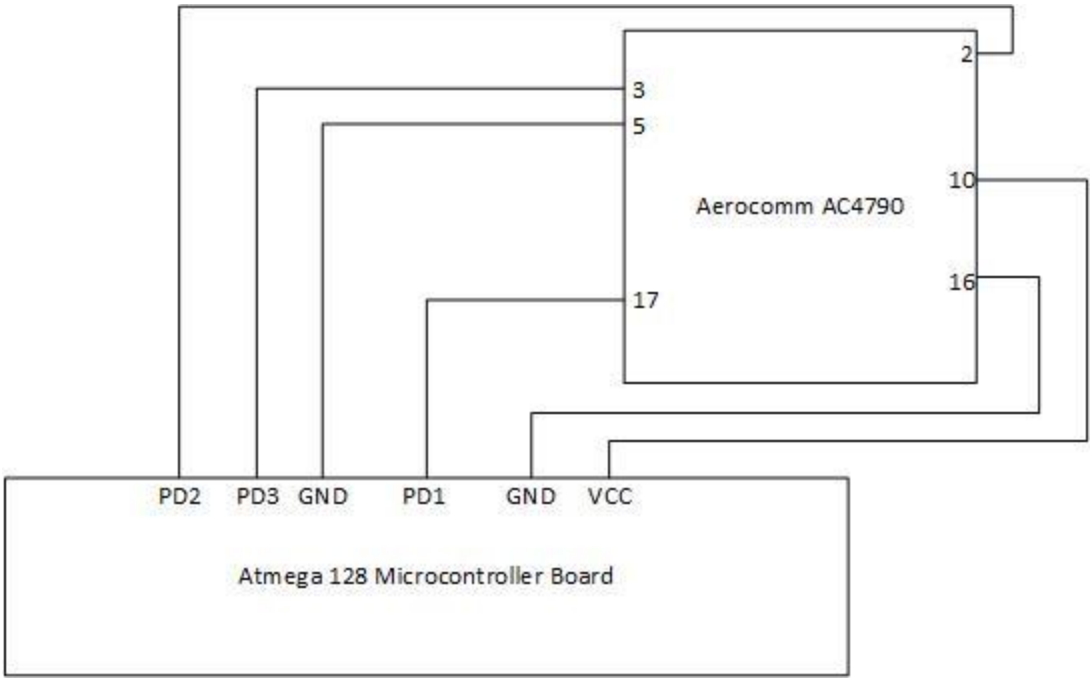

FIGURE XII - AC4790 SCHEMATIC

Initialization Flowchart

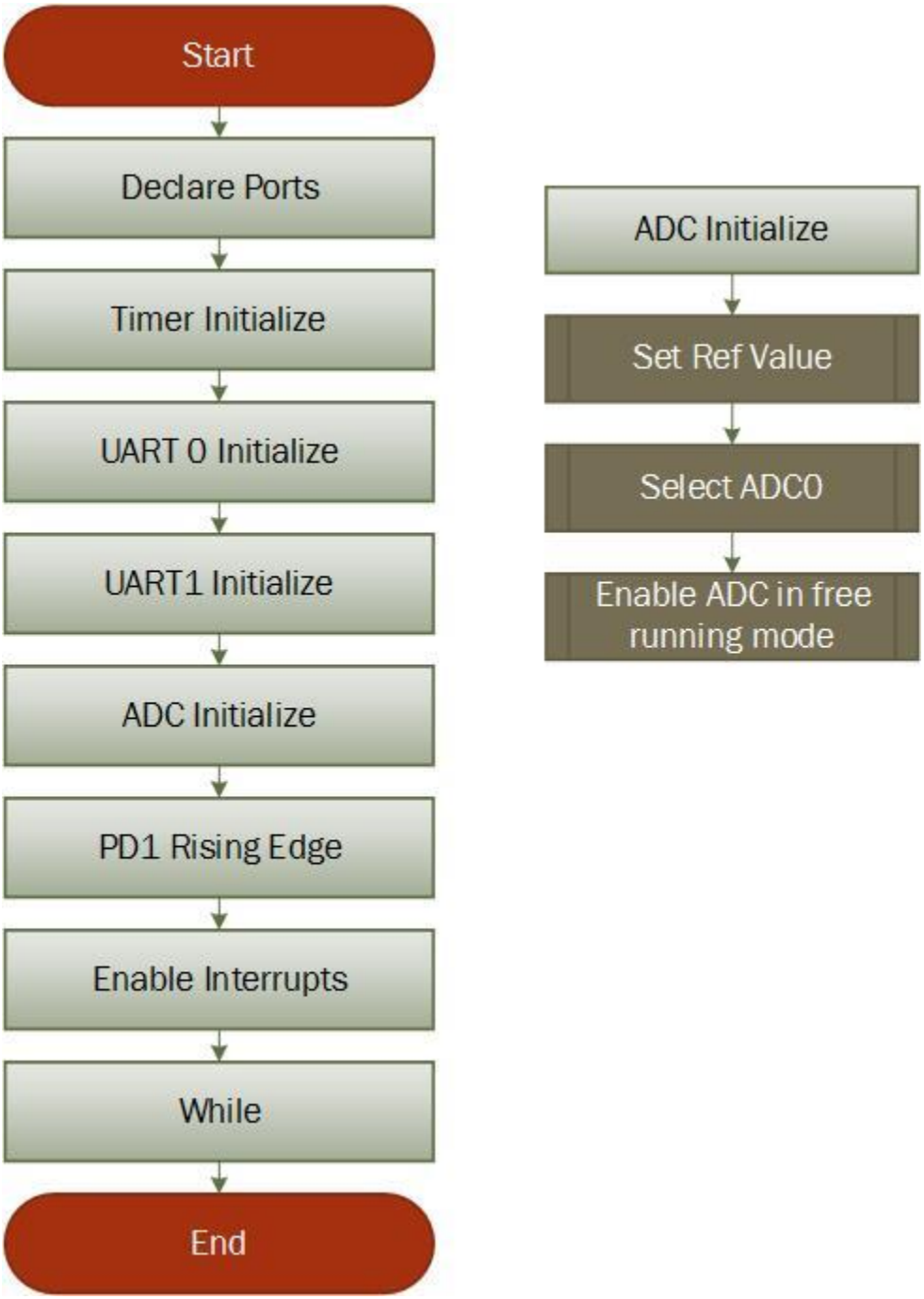

FIGURE XIII – SOFTWARE INITIALIZATION FLOWCHART

ISR Flowchart

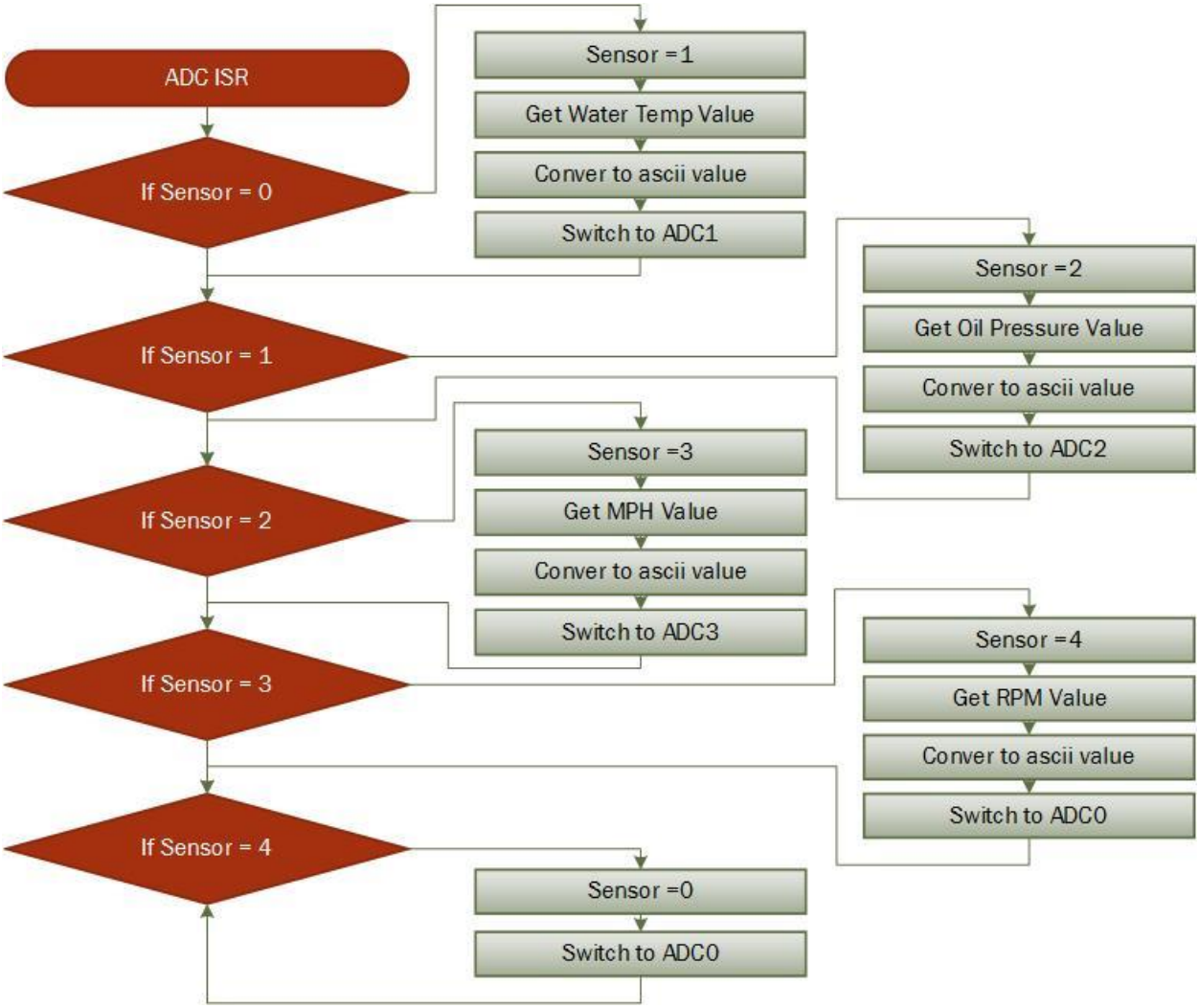

FIGURE XIV – ANALOG TO DIGITAL CONVERTER INTERRUPT SERVICE ROUTINE (ISR)

Sensor interface

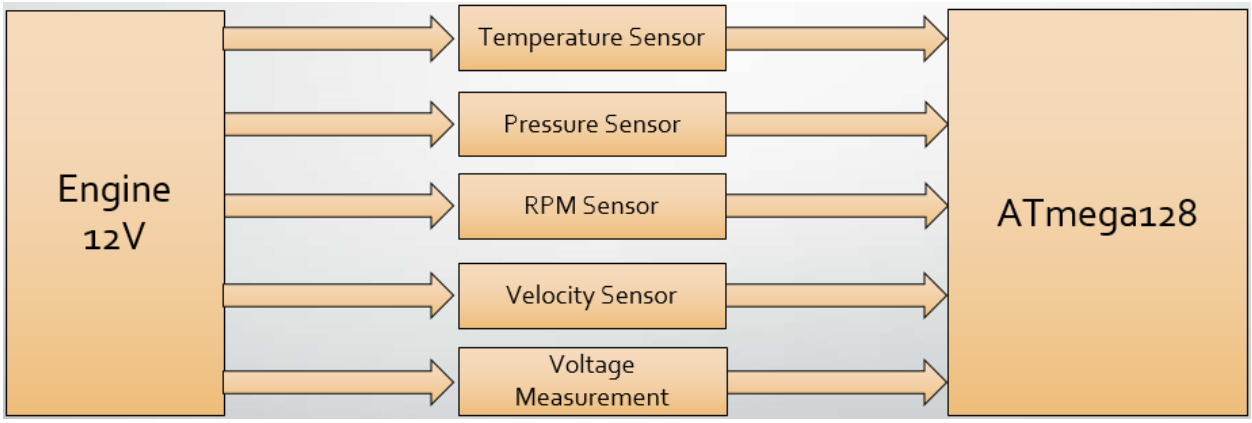

FIGURE XV – SENSOR INTERFACE BLOCK DIAGRAM

Pressure and Temperature sensors Op-amp Circuits

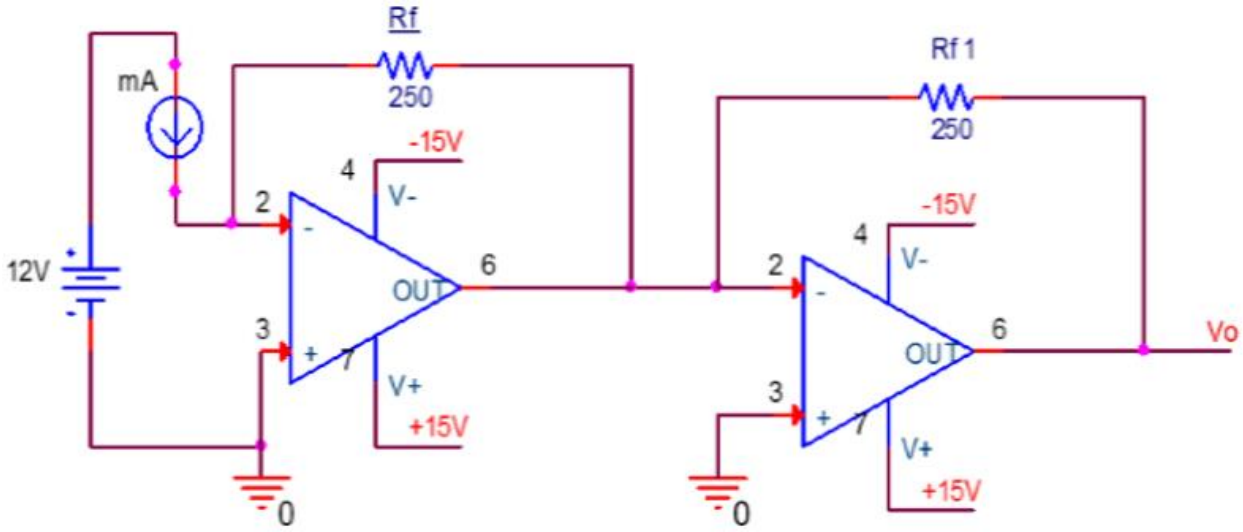

FIGURE XVI – INTERFACE CIRCUITRY FOR THE TEMPERATURE AND PRESSURE SENSOR

Detailed calculations

Temperature Sensor linearity:

T = m × Io +k — T= temperature

m = 10418.75 — m = slope

k = -59.48 ° C — k = Temperature

offset

To get the voltage output

V = Io x Rf (RF= 250 Ω)

Pressure Sensor Linearity

P = m × Io +k — P= Pressure

m = 6250 — m = slope

k = -25° C — k = Pressure

offset

To get the voltage output

V = Io x Rf (RF= 250 Ω)

RPM Sensor

$$RPM = F\left(\frac{cycle}{sec}\right)\left(\frac{60sec}{1min}\right)\left(\frac{1rev}{2cycles}\right)$$

Where F is Frequency

Detailed Experimental Results

Temperature Sensor Linear Output

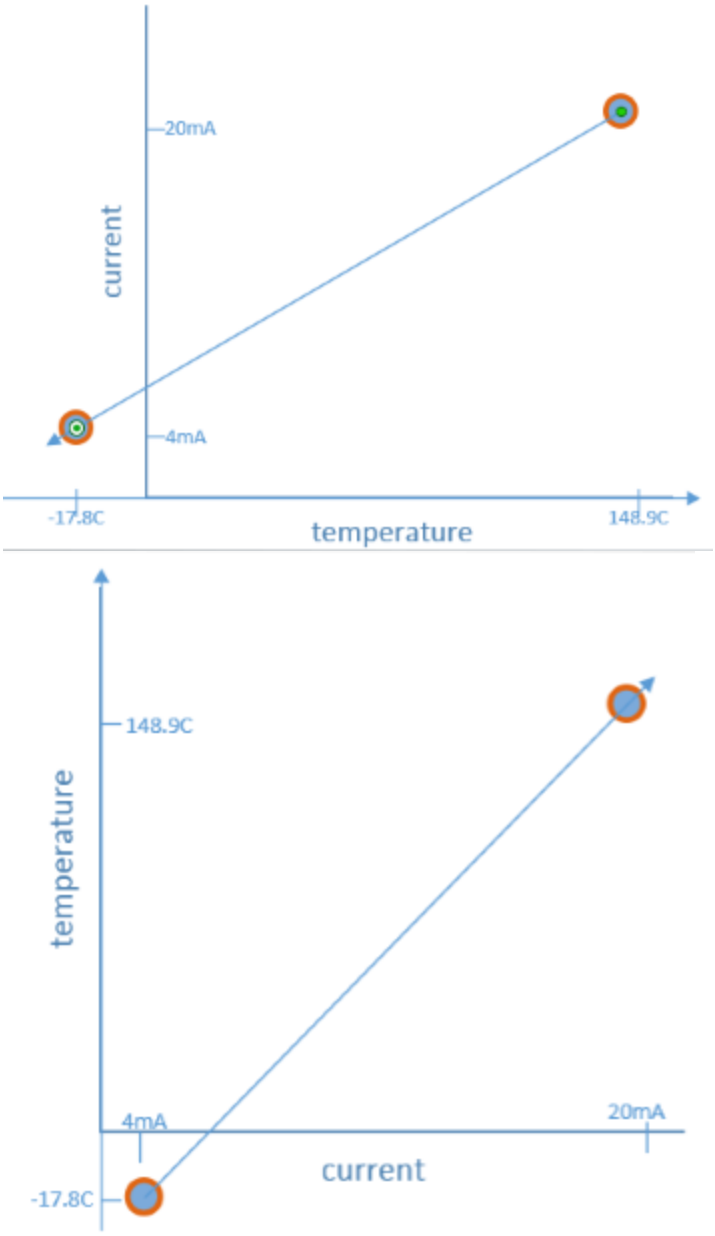

FIGURE XVII – TEMPERATURE SENSOR LINEAR OUTPUT

Temperature sensor results

TABLE XIII - TEMPERATURE SENSOR RESULTS

	Temperature	current	Voltage
2	-17.6	0.004	1
3	-12.390625	0.0045	1.125
4	-7.18125	0.005	1.25
5	-1.971875	0.0055	1.375
6	3.2375	0.006	1.5
7	8.446875	0.0065	1.625
8	13.65625	0.007	1.75
9	18.865625	0.0075	1.875
10	24.075	0.008	2
11	29.284375	0.0085	2.125
12	34.49375	0.009	2.25
13	39.703125	0.0095	2.375
14	44.9125	0.01	2.5
15	50.121875	0.0105	2.625
16	55.33125	0.011	2.75
17	60.540625	0.0115	2.875
18	65.75	0.012	3
19	70.959375	0.0125	3.125
20	76.16875	0.013	3.25
21	81.378125	0.0135	3.375
22	86.5875	0.014	3.5
23	91.796875	0.0145	3.625
24	97.00625	0.015	3.75
25	102.215625	0.0155	3.875
26	107.425	0.016	4
27	112.634375	0.0165	4.125
28	117.84375	0.017	4.25
29	123.053125	0.0175	4.375
30	128.2625	0.018	4.5
31	133.471875	0.0185	4.625
32	138.68125	0.019	4.75

Pressure sensor values

TABLE XIV - PRESSURE SENSOR RESULTS

Pressure	current	voltage
0	0.004	1
3.125	0.0045	1.125
6.25	0.005	1.25
9.375	0.0055	1.375
12.5	0.006	1.5
15.625	0.0065	1.625
18.75	0.007	1.75
21.875	0.0075	1.875
25	0.008	2
28.125	0.0085	2.125
31.25	0.009	2.25
34.375	0.0095	2.375
37.5	0.01	2.5
40.625	0.0105	2.625
43.75	0.011	2.75
46.875	0.0115	2.875
50	0.012	3
53.125	0.0125	3.125
56.25	0.013	3.25
59.375	0.0135	3.375
62.5	0.014	3.5
65.625	0.0145	3.625
68.75	0.015	3.75
71.875	0.0155	3.875
75	0.016	4
78.125	0.0165	4.125
81.25	0.017	4.25
84.375	0.0175	4.375
87.5	0.018	4.5
90.625	0.0185	4.625
93.75	0.019	4.75

RPM sensor values

TABLE XV – RPM SENSOR VALUES

RPM	frequency
300	10
480	16
660	22
840	28
1020	34
1200	40
1380	46
1560	52
1740	58
1920	64
2100	70
2280	76
2460	82
2640	88
2820	94
3000	100
3180	106
3360	112
3540	118
3720	124
3900	130
4080	136
4260	142
4440	148
4620	154
4800	160
4980	166
5160	172
5340	178
5520	184
5700	190

Microcontroller & Wireless Transmission

The microcontroller is able to acquire data from the sensors. Once acquired, it converted them into an ASCII value which could then be interfaced with LabVIEW. The ASCII value is then transmitted through the Aerocomm transceivers and is accurately received from wireless transmission. Data values were successfully interfaced with LabVIEW. As the values were changed from the Atmega128, the LabVIEW gauges adjusted accordingly.

Theory of Operation

In theory, the Data Acquisition and display system would acquire data from all sensors. This data would be displayed on both the remote laptop and Amulet. If a values exceeds the specified tolerance indicators will go off on both displays. This would allow the driver enough time to take the appropriate course of action. The remote display would have a data logger which will record the data and allow the pit crew and driver to review this data after track time. Also, since vehicles are judged on aesthetics, the Amulet demo display can be used to create a visual display to enhance the vehicles aesthetics.

Video, Web Links

http://cegt201.bradley.edu/projects/proj2015/saeformulacar/

Code


```
#include "bios_uart0.h"
#include "bios_uart1.h"
#include "bios_timer_int.h"
#include "bios_adc_int.h"
#include <stdlib.h>
#include <avr/io.h>
#include <inttypes.h>
#include <avr/interrupt.h>

int to_ascii (int aval);

volatile unsigned int arr_size = 0;
volatile unsigned char ready = 1;
volatile unsigned char i = 0;
//max values
volatile unsigned char CT_M = 250;
volatile unsigned char OP_M = 100;
volatile unsigned int  mph_m = 160;
volatile unsigned int  rpm_m = 15000;
volatile unsigned int  pc_time = 0;
volatile unsigned char val = 0;
volatile unsigned int  data = 0;
////transmit string
volatile unsigned char transmit[] = {   213,'0','0','0','0',//byte(0) -coolant temp
                                        213,'0','1','0','0',//byte(1) - pressure
                                        0xD6, 0x30, 0x31,'0','0','0','0',//word(1) - wheel speed
                                        214,'0','2','0','0','0','0',//word(2) - engine speed
                                        213,'0','5','0','0',//byte(5) - coolant temp warning
                                        213,'0','6','0','0',//byte(6) - oil pressure warning
                                        213,'0','8','0','0',//byte(8) - wheel speed warning
                                        213,'0','9','0','0',//byte(9) - engine speed warning
                                        160,2,0,32,62,//jump to screen
                                        19};//send XOFF for end of transmission
//LabVIEW String
volatile unsigned char trans_data[] = {'0','0','0','0','0','0','0','0','0','0','0','0','0','0',
'0','0',10};
volatile unsigned int  display_time = 0;
//declare ADC storage value
volatile unsigned char sensor = 0;
//Coolant Temp Values
volatile unsigned char CT_TEMP = 0;
volatile unsigned char CT_1 = '0';
volatile unsigned char CT_0 = '0';
//Oil Pressure Values
volatile unsigned char OP = 0;
volatile unsigned char OP_1 = '0';
volatile unsigned char OP_0 = '0';
//MPH Values
volatile unsigned int  mph = 0;
volatile unsigned char mph_4 = '0';
volatile unsigned char mph_3 = '0';
volatile unsigned char mph_2 = '0';
volatile unsigned char mph_1 = '0';
```

```
//RPM Values
volatile unsigned int  rpm = 0;
volatile unsigned char rpm_4 = '0';
volatile unsigned char rpm_3 = '0';
volatile unsigned char rpm_2 = '0';
volatile unsigned char rpm_1 = '0';
//external interrupt count value for rpm
volatile unsigned char count = 1;
volatile unsigned char update = 1;

void MyTimerFN (void){}

ISR(USART0_UDRE_vect){//ISR for RF transmission
    if(val == data){
        UCSR0B &= 223;//stop UDR0 interrupt
    }else if(val!=data){
        UDR0 = trans_data[val];
        val++;
    }}

ISR(INT1_vect){
    count = count + 1;
    }

ISR(TIMER1_COMPA_vect) {//ISR for timer 1
    display_time = display_time + 1;
    pc_time = pc_time +1;
    //record mph into 8-bit counter
    mph = TCNT2;
    TCNT2 = 0;
    //record engine rpm
    rpm = count;
    rpm = rpm*48;
    count = 0;
    //convert mph to display on screen
    mph_4 = to_ascii((mph >> 12) & 0x0F);
    mph_3 = to_ascii((mph >> 8) & 0x0F);
    mph_2 = to_ascii((mph >> 4) & 0x0F);
    mph_1 = to_ascii(mph & 0x0F);
    //convert rpm to display on screen
    rpm_4 = to_ascii((rpm >> 12) & 0x0F);
    rpm_3 = to_ascii((rpm >> 8) & 0x0F);
    rpm_2 = to_ascii((rpm >> 4) & 0x0F);
    rpm_1 = to_ascii(rpm & 0x0F);

    if(pc_time==5){//allow transmission after one second
        val = 0;
        pc_time = 0;
        trans_data[0]=CT_1;
        trans_data[1]=CT_0;
        trans_data[2]=OP_1;
        trans_data[3]=OP_0;
        trans_data[8]=mph_4;
```

```
        trans_data[9]=mph_3;
        trans_data[10]=mph_2;
        trans_data[11]=mph_1;
        trans_data[12]=rpm_4;
        trans_data[13]=rpm_3;
        trans_data[14]=rpm_2;
        trans_data[15]=rpm_1;
        UCSR0B ^= (1<<UDRIE0);
    }
    if(ready==1){
        if(display_time==5){
            display_time = 0;
            i=0;//start up transmission array position counter
            transmit[3]=CT_1;
            transmit[4]=CT_0;
            transmit[8]=OP_1;
            transmit[9]=OP_0;
            transmit[20]=mph_4;
            transmit[21]=mph_3;
            transmit[22]=mph_2;
            transmit[23]=mph_1;
            transmit[27]=rpm_4;
            transmit[28]=rpm_3;
            transmit[29]=rpm_2;
            transmit[30]=rpm_1;
        if(CT_TEMP > CT_M){
            transmit[34]='0';
            transmit[35]='1';
        }else{
            transmit[34]='0';
            transmit[35]='0';
        }
        if(OP > OP_M){
            transmit[39]='0';
            transmit[40]='1';
        }else{
            transmit[39]='0';
            transmit[40]='0';
        }
        if(mph>mph_m){
            transmit[49]='0';
            transmit[50]='1';
        }else{
            transmit[49]='0';
            transmit[50]='0';
        }
        if(rpm>rpm_m){
            transmit[54]='0';
            transmit[55]='1';
        }else{
            transmit[54]='0';
            transmit[55]='0';
        }
```

```
        if(CT_TEMP > CT_M || OP > OP_M || mph > mph_m || rpm > rpm_m){
            transmit[59]=36;
            transmit[60]=58;
        }else{
            transmit[59]=32;
            transmit[60]=62;
    }
    UCSR1B ^= (1<<TXEN);//enable transmitter
    }}}

ISR(ADC_vect){
    if(sensor==0){
        sensor = 1;
        CT_TEMP = ADCH;
        CT_1 = to_ascii(CT_TEMP >> 4);
        CT_0 = to_ascii(CT_TEMP & 0x0F);
        //switch to ADC1
        ADMUX &= 0xF0;
        ADMUX ^= (1<<MUX1);
    }else if(sensor==1){
        sensor = 2;
        OP = ADCH;
        OP_1 = to_ascii(OP >> 4);
        OP_0 = to_ascii(OP & 0x0F);
        //switch to ADC2
        ADMUX &= 0xF0;
    }else if(sensor==2){
        sensor = 3;
        mph = ADCH;
        mph_4 = to_ascii((mph >> 12) & 0x0F);
        mph_3 = to_ascii((mph >> 8) & 0x0F);
        mph_2 = to_ascii((mph >> 4) & 0x0F);
        mph_1 = to_ascii(mph & 0x0F);
        //switch to ADC3
        ADMUX &= 0xF0;
    }else if(sensor==3){
        sensor = 4;
        rpm = ADCH;
        rpm_4 = to_ascii((rpm >> 12) & 0x0F);
        rpm_3 = to_ascii((rpm >> 8) & 0x0F);
        rpm_2 = to_ascii((rpm >> 4) & 0x0F);
        rpm_1 = to_ascii(rpm & 0x0F);
        //switch to ADC4
        ADMUX &= 0xF0;
    }else if(sensor==4){
        sensor = 0;
        //switch to ADC0
        ADMUX &= 0xF0;
        ADMUX ^= (1<<MUX0);
    }}

//USART1_UDRE_vect
//interrupt for when the UDR0 transmit buffer is empty
```

```
ISR(USART1_UDRE_vect){
    if(i == arr_size){
        UCSR1B &= 247;//stop transmitter
    }
    UDR1 = transmit[i];
    if(i!=arr_size){
        i++;
    }
}
ISR(USART1_RX_vect){//receiver interrupt
//NOTE: In order to function properly, the UDR byte must always be read
// from in this interrupt; otherwise, the interrupt will instantly
// interrupt again upon exiting
if(UDR1==17){
    ready = 1;
}
}

int main(void){
    DDRC = 0xFF; //output
    DDRF = 0x00; //input
    DDRA = 0xFF; //output
    DDRE = 0xFF; //output
    //DDRD = 0xFF; //output

    Timer1_initialize( MyTimerFN);//Timer Initialization
    data = (sizeof(trans_data) / sizeof(trans_data[0]));
    uart0_initialize(uart_bps_9600);//RF UART0 Initialization
    arr_size = (sizeof(transmit) / sizeof(transmit[0]));
    uart1_initialize(uart_bps_9600);//RF UART1 Initialization
    adc_initialize();//ADC Initialization

    // Transmission rising edge PD1
    EICRA ^= (1 << ISC11) | (1 << ISC10);//bitor operation with interrupt 1 controls
    EIMSK ^= (1 << INT1);//clear enable bit

    sei();//Enables ALL Interrupts
    while(1);
    }

int to_ascii (int aval){
    switch(aval){
        case 0: aval='0'; break;
        case 1: aval='1'; break;
        case 2: aval='2'; break;
        case 3: aval='3'; break;
        case 4: aval='4'; break;
        case 5: aval='5'; break;
        case 6: aval='6'; break;
        case 7: aval='7'; break;
        case 8: aval='8'; break;
        case 9: aval='9'; break;
        case 10:aval='A'; break;
```

```
        case 11:aval='B'; break;
        case 12:aval='C'; break;
        case 13:aval='D'; break;
        case 14:aval='E'; break;
        case 15:aval='F'; break;
    }
    return(aval);
}
```

```
#ifndef BIOS_ADC_INT_H_
#define BIOS_ADC_INT_H_
#include <stdint.h>

void        adc_initialize  (void);

#endif /* BIOS_ADC_INT_H_ */
```

```
#include "bios_adc_int.h"
#include <stdlib.h>
#include <avr/io.h>
#include <inttypes.h>
#include <avr/interrupt.h>

void adc_initialize (void)
{
ADMUX ^= (1<<REFS0) | (1<<ADLAR);//use vcc
ADMUX &= 0xF0;
ADCSRA ^= (1<<ADEN) | (1<<ADSC) | (1<<ADFR) | (1<<ADIE) |(1<<ADPS2) | (1<<ADPS1) | (1<<ADPS0);
//set clock
}
```

```
#include "bios_timer_int.h"
#include <stdlib.h>
#include <avr/io.h>
#include <inttypes.h>
#include <avr/interrupt.h>

#define xtal 16000000L

static void (*Timer1_Overflow_function)(void) = 0;

void Timer1_initialize ( void (*handle_overflow)(void) )
{
    Timer1_Overflow_function = handle_overflow;

    TCNT1=0x00;                     // set timer0 counter initial value to 0

    OCR1A = 0x61A7;

    TCCR1B ^= (1<<WGM12);//enable timer 1 interrupt
    TIMSK ^= (1<<OCIE1A);//start timer
    TCCR1B &= ~(1<<CS12);//Clear CS12 bit
    TCCR1B = (1<<CS11)|(1<<CS10);
    TCCR2  = (1<<CS22)|(1<<CS21)|(1<<CS20);//Timer 2
    TCCR3A = (1<<CS32)|(1<<CS31)|(1<<CS30);//Timer 3
    TCNT3H = 0;
    TCNT3L = 0;
}
```

```
#ifndef BIOS_Timer1_INT_H_
#define BIOS_Timer1_INT_H_
#include <stdint.h>

void    Timer1_initialize   (void (*handle_fn)(void) );

#endif /* BIOS_Timer1_INT_H_ */
```

```c
//* USART0 library - implementation file *
#include "bios_uart0.h"
#include <avr/io.h>

#define xtal 16000000L

void uart0_initialize(uint16_t baud)
{
    uint32_t temp = xtal/16/baud-1;
    UBRR0H = (temp >> 8) & 0x0F;
    UBRR0L = (temp & 0xFF);

    UCSR0B ^= (1<<TXEN);//Enable TX
    UCSR0C = (1<<UCSZ01) | (1<<UCSZ00);//8 bit data
    UCSR0B ^= (1<<UDRIE0);//Data Register
}
```

```
#ifndef BIOS_UART0_H_
#define BIOS_UART0_H_
#include <stddef.h>
#include <stdint.h>

void    uart0_initialize    (uint16_t baud);

#ifndef _SER_BOUD
#define _SER_BOUD
    #define uart_bps_50          50
    #define uart_bps_75          75
    #define uart_bps_110        110
    #define uart_bps_134        134
    #define uart_bps_150        150
    #define uart_bps_200        200
    #define uart_bps_300        300
    #define uart_bps_600        600
    #define uart_bps_1200      1200
    #define uart_bps_1800      1800
    #define uart_bps_2400      2400
    #define uart_bps_4800      4800
    #define uart_bps_9600      9600
    #define uart_bps_19200    19200
    #define uart_bps_38400    38400
    #define uart_bps_57600    57600
    #define uart_bps_115200  115200
#endif


#endif /* BIOS_UART_H_ */
```

```
//* USART1 library - implementation file *

#include "bios_uart1.h"
#include <avr/io.h>

#define xtal 16000000L

void uart1_initialize(uint16_t baud)
{
    uint32_t temp = xtal/16/baud-1;
    UBRR1H = (temp >> 8) & 0x0F;
    UBRR1L = (temp & 0xFF);

    UCSR1B ^= (1<<TXEN)|(1<<RXEN); //enable the transmitter and receiver
    UCSR1C = (1<<UCSZ11)|(1<<UCSZ10);
    UCSR1B = (1<<UDRIE1)|(1<<RXCIE1); //enables receiver
}
```

```
#ifndef BIOS_UART1_H_
#define BIOS_UART1_H_
#include <stddef.h>
#include <stdint.h>

void    uart1_initialize    (uint16_t baud);

#ifndef _SER_BOUD
#define _SER_BOUD
    #define uart_bps_50          50
    #define uart_bps_75          75
    #define uart_bps_110        110
    #define uart_bps_134        134
    #define uart_bps_150        150
    #define uart_bps_200        200
    #define uart_bps_300        300
    #define uart_bps_600        600
    #define uart_bps_1200      1200
    #define uart_bps_1800      1800
    #define uart_bps_2400      2400
    #define uart_bps_4800      4800
    #define uart_bps_9600      9600
    #define uart_bps_19200    19200
    #define uart_bps_38400    38400
    #define uart_bps_57600    57600
    #define uart_bps_115200  115200
#endif


#endif /* BIOS_UART_H_ */
```

Recommendations for Future Work

Because the data acquisition system was never implemented on the formula car the first task that would be done is to install the system on the formula car. After this is done and testing has been completed more sensors can be added to be displayed on both the Amulet touchscreen and the pit crew's laptop. Some relevant data that can be acquired include: suspension information, airflow through both intake and exhaust, tire pressure, braking efficiency, and coolant temperature. Also, a tuning mode can be added to the touchscreen to allow the user to tune the formula car's internal timing. If all this work is completed the data acquisition system will give the driver a competitive advantage over competition and allow the students working on the car to focus on making the formula car as fast as possible without the worry of acquiring data.