# Gasoline Economy Management

Pedro Betancourt, Alexander Patino, and Mohhamad Pulliam

Dept. of Electrical Engineering and Computer Science, University of Central Florida, Orlando, Florida, 32816-2450

***Abstract* — The purpose of this project is to develop a small consumer electronic device to assist the consumer in making better decisions in order to improve their fuel economy. This device is called the GEM. The GEM is intended to connect to the engine control unit (ECU) of a vehicle via the standard OBD-II interface found in all vehicles sold in the United States after 1996. The device is designed to safely provide feedback to a driver in a non-intrusive way so as not to be a distraction or nuisance. To provide feedback to the driver the GEM connects to a custom Android OS application. All data processing and display is taken care of by the driver's Android device. The GEM has a small footprint, consumes very little power when in standby mode and is user friendly. It's designed to be plugged into the OBD-II port permanently and require no further user intervention.**

***Index Terms* — Automotive, fuel economy, engine control unity (ECU).**

## I. Introduction

In 2013 the U.S. consumed over 134 billion gallons of gasoline. On that scale even slight improvements in efficiency can save millions of dollars and prevent tons of carbon dioxide from entering the atmosphere. Automobile manufacturers are constantly striving to develop more efficient engines in order to meet the needs of consumers as well as regulatory requirements. Unfortunately purchasing a new vehicle in order to realize better fuel economy isn't always viable or practical. In fact purchasing a new vehicle in order to receive an incremental increase in efficiency is actually a bad idea. It is rare that the efficiency gains pay for themselves except over an exceptionally long period of time. Not to mention the large environmental impact and high energy consumption required to build a single car. In order to make progress in this realm what we need is a simple, low-cost solution that is available to every driver.

This is why we design the "Gasoline Economy Management" tool or GEM. GEM is a simple system that is designed to be available to nearly any driver who uses a smart phone. GEM is a small device that plugs into the OBD-II port that is standard on all vehicles sold in the U.S. after 1996. The OBD-II port is a standardized hardware port through which vehicle diagnostic codes and other vehicle information can be read. We can use this information to help develop more fuel efficient driving habits. While it's prohibitively expensive to retrofit older vehicles with modern fuel efficient systems it can be very cheap and very effective to reprogram an individual's driving habits. Simple changes such as driving the correct speed and avoiding excessive acceleration and braking can increase fuel efficiency by up to 5%. For a driver that drives 15,000 miles a year if they can go from 20 miles per gallon to 21 they save 36 gallons per year. At three dollars a gallon that's a yearly savings of $108. Not only will it save money and resources it's designed to be easy for anyone to use. When a person gets a GEM it will be very simple for them to connect the device to their vehicle and pair it with their phone. All they have to do is connect the GEM to the OBD-II port, turn on their car and start the phone application. The phone application provides a simple, hands off user interface that provides feedback to help a driver adjust their driving habits in order to enjoy greater fuel economy. The device itself is powered by the vehicle's battery and when the vehicle is turned off the GEM enters a sleep mode so that it won't drain the battery. When active the GEM automatically pairs with the driver's phone via Bluetooth LE or, if the driver doesn't have their phone available, the GEM will store driving information so that the next time the driver connects their smart phone fuel economy and other statistics will be transferred to the application. While the GEM is designed for semi-permanent installation it also has the ability to store vehicle profiles. If a driver finds themselves behind the wheel of a different car every day they'll still be able to take advantage of the provided feedback.

## II. PROJECT DESCRIPTION

Several key objectives were identified when developing this product. It needs to be low power, with a small footprint, and a simple and safe interface to be used when driving.

The hardware systems for GEM can be broken down into the following subsystems

1. Power supply
2. OBD-II to serial interface
3. Onboard processing
4. Data storage
5. Data transmission

Each of these subsystems was analyzed based on our key objectives and specifications were developed.

## A. Power

Power is a challenging aspect of the GEM. The primary requirement is that the GEM should be plug and play. The user should be able to plug the device into the OBD-II port on the car then leave the device plugged in without drawing down the battery when the vehicle is not in use. A goal was set to be able to leave the GEM connected to a vehicle for 183 days (half a year) without drawing down the battery so far as to cause the vehicle to be unable to start.

We reviewed different batteries to find out what the average storage capacity in amp hours is in order to find out what the maximum current draw we could have while the device is in low power mode. The energy storage of batteries manufactured in the U.S. is not specified as automotive batteries are designed for starting the vehicle only. Starting batteries are specified by cold cranking amps (CCA). Because approximations for current capacity for U.S. batteries range anywhere from 30 to 60 Ah, we relied on European Union labeling standards for which amp hour capacity is a requirement. Varta brand batteries were taken as a typical example of an automotive battery. The midrange Varta "Black Dynamic" battery has a range of capacities from 40 to 90 Ah. There is no way of knowing what battery is being used so the minimum capacity was chosen as the baseline. The Varta "Black Dynamic" type A16 battery has a capacity of 40 Ah. The GEM should not draw down more than half the capacity within six months. In order to achieve this the power draw when the device is idle must be less than 5 mA as shown in (1).

$$\frac{20{,}000\ mAh}{5\ mA} = 4000\ h \approx 167\ days \tag{1}$$

## B. OBD-II to Serial Interface

While the OBD-II physical port configuration is standardized there are many different communications protocols available for use. The GEM must be able to communicate using all legislated protocols. This means that the GEM will be able to interface with any consumer vehicle sold within the U.S. since 1996. The required protocols are

1. SAE J1850 PWM
2. SAE J1850 VPW
3. ISO 9141-2
4. ISO 14230 (KWP2000)
5. ISO 15765 (CAN)

Each of these protocols will be discussed in detail in Section IV.

## C. Onboard Processing

In order to keep the vehicle hardware simple, low cost, and low power on board data processing will be minimal. The primary function of the MCU is to interconnect the various on-board components and prepare the data to be sent to the smart phone for processing. The key features of the microcontroller are very low current draw, multiple serial interfaces, and simple to program.

The microcontroller unit in the GEM system serves primarily as a bridge between other onboard modules. It transmits and receives data from an OBD-II transceiver, communicates with a Bluetooth module, and handles a cache memory in an SD card. First off, the MCU must be able to request and store the vehicles identification number (VIN). Based on the VIN, the software must determine the correct OBD-II protocol and use the appropriate subset of PID's. While the vehicle is turned on, the MCU must poll the ECU, through a serial interface, continuously, with a minimum delay of 1 second and a maximum delay of 2 seconds in between polls.

The MCU will determine if the Bluetooth module has established a connection to a device, and provide a continuous data stream if a connection has been established. If it is determined that no device is in range of the Bluetooth module, the data will not be flushed. Instead, a frame of data will be cached in an SD card.

## D. Data Storage

Data sent to the Android device will be stored on internal storage by writing to application files. These data files can be read from by the application enabling the calculation of fuel economy statistics. To allow for more permanent data to be saved and recalled later the application will manage data across sessions using shared preferences. This will allow the application to track a user's fuel economy as long as the data is stored on their device. A user must be able to recall and clear the "history" of data from device storage from within the application. To accommodate users with multiple vehicles the application will allow data to be stored accordingly for up to five user defined profiles. OBD-II message lengths range from 12 to 255 bytes, the minimum delay between messages received is 1 second. We expect GEM to be able to store up to 24 hours of driving data, in cases when users forget their phones or similar scenarios. Considering the most conservative scenario, where all messages are the maximum length, and are delayed by the minimum length of time we calculate the maximum frame length in (2).

$$255\ B \cdot 60\frac{polls}{m} \cdot 60\frac{m}{h} \cdot 24\ h = 22.03\ MB \tag{2}$$

Thus, allowing some leeway, the maximum frame length shall be 25 MB, and the on-board storage must be a minimum of 25 MB.

*E. Data Transmission*

In order to get data from our device to the smart phone application we chose to use Bluetooth. Bluetooth is a short range wireless standard originally intended for wireless RS-232 serial data. It operates in the 2.4 GHz range. Bluetooth devices operate as a master and a slave. Typically in a Bluetooth use case a user wants the two devices in communication to communicate securely but at the same time avoid having to manually connect the devices whenever they are in range. The two devices should automatically connect and start sharing data when in range. In order to achieve this devices go through a pairing operation which is usually started by a specific request from the user. Once the devices have gone through the pairing operation they are securely bonded and can connect to each other whenever they're in range without further interaction from the user.

To setup a Bluetooth connection our Android application must search for devices in range and also display previously connected devices. Once the devices have been discovered they may be paired by using "Passkey Entry." It is also important that a user may terminate a Bluetooth connection with GEM from within the application. All of these functions must be accessible from the first screen of the application. In the event that a Bluetooth connection is lost the application must handle the error by displaying a notification and attempt to repair the connection. Once a connection is established it will remain connected and receive broadcasts until GEM is unplugged or the vehicle is turned off.

## III. FUEL ECONOMY

To determine the fuel economy of a vehicle the Environmental Protection Agency has designed a series of tests to estimate fuel consumption. The fuel economy of a vehicle based on the tests performed by the EPA is weighted such that the final value is 55% city and 45% highway. Because the EPA uses a dynamometer or "dyno" which neglects real world driving conditions, they expect actual fuel economy to be less than calculated. The Urban Dynamometer Driving Schedule (UDDS) shown in Figure 1 represents the driving cycle used to estimate city driving. To simulate highway fuel economy the EPA uses the Highway Fuel Economy Driving Cycle (HWFET).

In 2011 the EPA and National Highway Traffic Safety Administration (NHTSA) adopted a new fuel economy label that is displayed on new cars. The most notable change is the addition of a vehicle's fuel consumption rate which is the gallons consumed per 100 miles. Figure 2 shows the relationship between miles per gallon and gallons per 1,000 miles. The takeaway from this chart is that there is a notable difference in fuel consumption from 10 mpg to 15 mpg (about 33 gallons) and a less significant improvement from 30 mpg to 35 mpg (about 5 gallons). This new method of measuring fuel economy is used to allow more accurate comparisons among vehicles. Our device will ultimately display mpg and gallons per 100 miles because our primary goal is for consumer to increase their fuel economy no matter how they measure it.

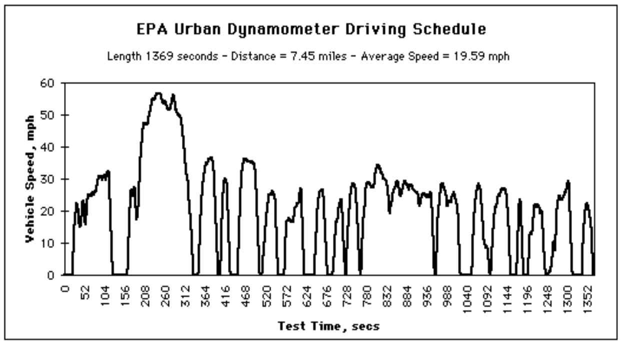


Figure 1: Urban Dynamometer Driving Schedule [1]

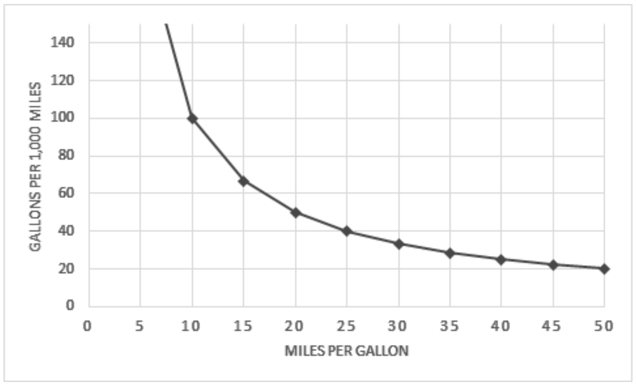


Figure 2: MPG & Gallons per thousand miles

## IV. DEVICE INTERFACE

In 1996 the OBD-II port was made standard on all vehicles sold in the U.S. The OBD-II port is a 16 pin female SAE J1962 connector. The connector is required to be located in the passenger cabin and within 0.69m of the steering column, except where requested by exemption. There are five possible signaling protocols used by the OBD-II port. In vehicles 2008 and newer the standard is the CAN Bus (ISO 15765) and is required by law as one of the protocols.

### *A. SAE J1850 PWM/VPW*

SAE J1850 can be broken down into two sub classes. The first is variable pulse width (VPW). VPW is a single wire bus protocol that utilizes only pin 2 of the connector. It operates at either 10.4 Kbps or 41.6 Kbps. High signal voltage is a nominal 7V with a minimum and maximum of 6.25V and 8V respectively. Low signal voltage is a nominal 0V with a minimum and maximum of 0V and 1.2V respectively. Start of frame is issued by a 200µS high signal. A 1 bit is issued by a 128µS low signal or a 64µS high signal. A 0 bit is issued by a 64µS low signal or a 128µS high signal. Messages may be up to 12 bytes.

The second class of SAE J1850 is pulse width modulation (PWM). PWM is a two wire protocol that utilizes both pin 2 and pin 10 of the connector. PWM supports a speed of 41.6 Kbps. High signal voltage is a nominal 5V with a minimum and maximum of 3.80V and 5.25V respectively. Low signal voltage is a nominal 0V with a minimum and maximum of 0V and 1.20V respectively. The active bus state occurs when pin 2 (BUS+) is pulled high and pin 10 (BUS-) is pulled low. Start of frame is issued by an active bus state for 48µS. A 1 bit is issued by an 8µS bus active state within a 24µS period. A 0 bit is issued by a 16µS bus active state within a 24µS period. Messages may be up to 12 bytes.

### *B. ISO 9141-2*

The ISO 9141-2 standard is a two wire serial communication protocol. It operates at 10.4 Kbps. This protocol utilizes pins 7 and 15 on the connector. Pin 7 is referred to as the K-line. Pin 15 is referred to as the L-line and is optional. The K-line is the communication line and is bidirectional. The L-line is used to send a signal to the ECU on older cars as a wake-up so that communication could start on the K-line. In newer cars the L-line is not used and all signaling occurs on the K-line. A high voltage signal is a nominal 12V with a minimum and maximum of 9.60V and 13.5V respectively. Signaling is similar to RS-232 (Though with the obvious difference in voltage level). The serial settings are 10.4K baud, 8 data bits, no parity, and 1 stop bit. Messages may be up to 12 bytes.

### *C. ISO 14230 (KWP2000)*

The KWP2000 protocol is the same as the ISO 9141-2 protocol in all respects except that the data rate is variable from 1.2K baud to 10.4K baud. Messages may also be up to 255 bytes in length.

### *D. ISO 15765 (CAN)*

The CAN protocol is modern standard for vehicle messaging. It is required in all vehicles sold in the United States since 2008. CAN utilizes pin 6 and pin 14 on the connector. Pin 6 is CAN high. Pin 14 is CAN low. The CAN high signal voltage is a nominal 3.5V with a minimum and maximum of 2.75V and 4.5V respectively. CAN low signal voltage is a nominal 1.5V with a minimum and maximum of 0.5V and 2.25V respectively. CAN is in the recessive state when neither pin is being driven. In this state both lines sit at around 2.5V. CAN is in the dominant state when both lines are being driven and there is a difference of 2V between the lines.

### *E. OBD-II PID's*

Once the messaging protocol has been determined and the GEM can communicate with the vehicle messages will be sent and received using the OBD communication protocol. Most of the data that is available via OBD-II is related to emissions as the OBD-II protocol was mandated for use in emissions inspections. The GEM will be able to request data related to vehicle speed, engine RPM, fuel level, and other information useful to calculate the current efficiency of the vehicle. In order to retrieve the data from the vehicle ECU a message needs to be sent to the data bus. This message takes the form of a "Parameter ID" or PID. The PID takes the form of four bytes. The first two bytes identify the mode of the query as shown in Table 1 and the second two bytes indicate the actual query if applicable. For example, to request the current diagnostic trouble code (DTC) a value of 0300 is sent to the data bus. The ECU responsible for that code will respond with the currently stored DTC.

Table 1: OBD-II Modes

| Mode | Description | Mode | Description |
|---|---|---|---|
| 01 | Show Current Data | 06 | Test Results |
| 02 | Show Freeze Frame Data | 07 | Show Pending DTC |
| 03 | Show Stored DTC's | 08 | Special Control Mode |
| 04 | Clear DTC | 09 | Request Vehicle Information |
| 05 | Test Results | 10 | Permanent DTC's |

Since not all vehicles support all modes the GEM is limited to a specific subset. The only two modes that will be used by the smart phone software will be modes 01 and 09. Mode 1 is used to retrieve specific data related to fuel efficiency as shown in Table *2* and mode 9 is used to retrieve the VIN (Vehicle Identification Number) to verify that the GEM has not been moved to another vehicle.

Table 2: Mode 01 PID's

| PID | Description |
|---|---|
| 00 | Supported PID’s (1-20) |
| 04 | Engine Load % |
| 0C | Engine RPM |
| 0D | Vehicle Speed |
| 20 | Supported PID’s (21-40) |
| 2F | Fuel Remaining % |
| 40 | Supported PID’s (41-60) |
| 5E | Engine Fuel Rate |

## V. Device Design

### *A. Serial Interface*

In order to send and receive signals the GEM must be able to receive a signal using any one of the five protocols. This means that there needs to be four different transceivers that can take the voltages and signals and convert them to something that can be read by the digital input on a microcontroller. The microcontroller must then be able to take the signal, determine what protocol is being used and then receive and transmit data using that protocol. For the SAE J1850, ISO 9141-2, and ISO 14230 buses a simple comparator can be used to compare the line voltages and then output a 3.3V signal to the OBD-II to serial interpreter. An example of the typical transceiver is shown in Figure 3.

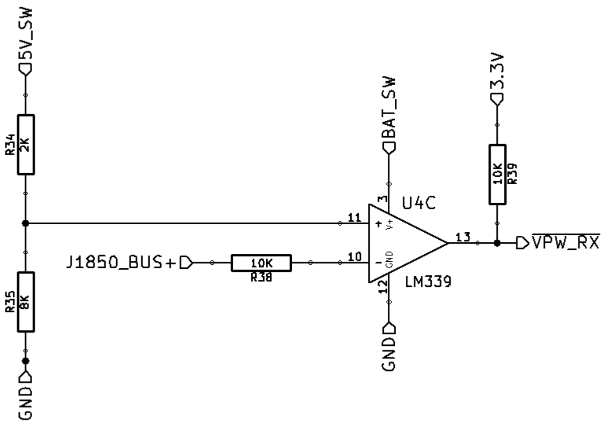


Figure 3: VPW Transceiver

The remaining challenge is the CAN bus. Because the CAN bus allows for speeds of up to 1 Mbps timing and control is critical. In order to transmit and receive messages on the CAN bus a commercial transceiver is necessary. There are a wide variety of CAN receivers due to the fact that CAN is used in both automotive and industrial applications. For our design we have opted for the MCP2551 as it meets our stringent power requirements and is one of the least expensive options.

Once the ECU signal voltages have converted to 3.3V they are then passed to a serial interface processor. This processor automatically negotiates the protocol and then converts serial input and output to match that protocol. The benefits of using the preconfigured automotive OBD-II to serial interface are a simplified software ecosystem at the cost of hardware complexity and additional device cost. We found this tradeoff to be acceptable as adding this component while more expensive allowed us to save countless hours developing the interface code ourselves.

### *B. Power System*

The power requirements for this design are somewhat complicated. The system requires a vehicle battery voltage, nominally 12V, a 7V supply for the comparators for the SAE J1850 VPW system, a 5V supply for the comparators for the SAE J1850 PWM system, and a 3.3V supply for the microcontrollers and Bluetooth module. Since the two SAE J1850 protocols use the same input and output and only differ in their voltages it is required that the power supply on those inputs is switchable between 5V and 7V

In addition because of the nature of the power supply on a vehicle we also need to have filtering and transient voltage suppression. Filtering can be done by capacitors on the input to the power stage. Transient voltage suppression can be done with the addition of TVS diode pairs on the input stage to shunt current when the voltage rises beyond a set point. An example of the filtering circuit used is shown in Figure 4.

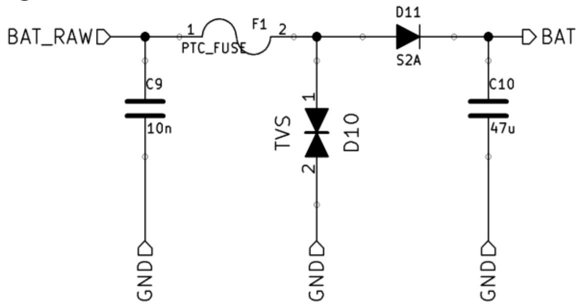


Figure 4: Power Filter Stage

To meet the power requirements multiple voltage regulators will need to be used. For these stages we need high efficiency as we have ultra-low power requirements when the vehicle is powered off and we have limited heat dissipation when the vehicle is on and the device is powered.

Given the typical operating current for the various on board devices we arrive at a current consumption of 223mA, add a 50% safety margin and we have 334mA consumed while the device is fully active. If we were to drop the battery voltage to 3.3V via a linear regulator we would have to dissipate 2.9W.

We opted to go with a set of switching regulators to provide the 5V and 3.3V rails. In addition, to support the SAE J1850 bus we use a linear regulator to achieve the nominal 7V required. Using a transistor as a switch we change the resistor network attached the linear regulator so that it can also provide the nominal 5V. This allows us to keep the device relatively simple.

### *C. Bluetooth*

Since Bluetooth is such a common standard there are many ways that we can implement it on our device. We had to make the choice between Bluetooth and Bluetooth Low Energy (BLE) we found BLE to be ideal for our system as it is low power and is active over short ranges. We then had two options. We could either integrate a prebuilt Bluetooth module or we could develop a module based upon an existing Bluetooth chip. Due to the complexities of antenna design and the fact that wireless communication is a core feature of our device we decided to reduce the risk of a failure by using a prebuilt module. We selected the Microchip RN4020. Using this module allows us to prototype and iterate our design quickly and easily. The RN4020 is a complete Bluetooth module with an onboard Bluetooth Low Energy 4.1 stack. Using the RN4020 reduces Bluetooth related MCU software to the task of interfacing with the module. The RN4020 has a command API, which are issued by the host microcontroller as ASCII characters. Commands are sent from the host MCU through a UART control interface.

### *D. Storage*

The GEM needs to be able to store data when the user does not have their smart phone available. On board storage in the MCU memory is not sufficient because should the device become unplugged or otherwise disconnected from the battery the memory will be lost. A simple solution is a cheap, easily available, easy to use, non-volatile memory. SD cards meet all of these requirements. In order to interface with the SD card the MSP430 can utilize the SPI bus. Texas Instruments provides a SD card library which can be used to write data to the card.

## VI. SOFTWARE DESIGN

### *A. Overview and Considerations*

Our goal is to produce software that meets the expectations of mobile application users with performance and presentation similar to other fuel economy applications. The Android application displays vehicle metrics received from the GEM device in real-time. Some of which are:

1. Vehicle speed
2. RPM
3. Instant fuel economy
4. Trip meter

This software is able to display these values for the user within range of the GEM device. Users may customize which information is displayed by the application and its location on screen.

The GEM Android application was developed based on the best practices suggested by the Android Development Team[1]

Code documentation is generated using Javadoc because of its availability in Android Studio and our team's familiarity with this documentation generator. All documentation will follow the conventions provided by Oracle.[2]

The GEM Android application will try to adhere to the code style guidelines distributed by the Android Development Team[3]

Many design considerations shaped the development of the GEM Android application. We will discuss some of these considerations here.

The reusability of the system was a top priority from the beginning. The system is designed to take information from the GEM device and provide the application with the data necessary to produce meaningful statistics to the user. The design architecture must be flexible enough to allow any subsystem or feature to be modified/improved in future releases.

Maintainability is crucial to any system that may be modified or examined in the future, and proper programming practices must be employed throughout the application development lifecycle. The Agile software development model will be the most effective as design challenges may force us to change directions quickly, but proper documentation through the build is absolutely necessary.

[1] https://developer.android.com/training/index.html

[2]http://www.oracle.com/technetwork/articles/java/index-137868.html

[3] https://source.android.com/source/code-style.html

Testing the system on mobile devices will be require access to several different Android devices with varying hardware. The Android Virtual Device (AVD) emulator increases productivity, but cannot guarantee that the real device will work as shown. Our chosen IDE, Android Studio, is built specifically for the creation or Android apps and offers much of the same testing frameworks as Eclipse.

Our system is designed to be relatively light and requiring Android 4.3 will guarantee that hardware used by the mobile device can handle our application. Any performance issues that arise will be dealt with on a case by case basis.

Developing the system for mobile devices introduces many variables (i.e. screen size, processor, RAM), but the languages and APIs used to program our system are standardized and subsume previous releases which makes portability a relatively simple feat to achieve.

The information stored and transmitted by our device is not considered sensitive information. The range of the Bluetooth transmission is short enough that it is unlikely that unintended users will have access to the information transmitted. In the event that data is intercepted, there is no danger that a user's vehicle can be altered or harmed in any way. The biggest concern for safety that our team has is that users will not be distracted by the application while operating their motor vehicle. Careful consideration must be put into the design of the application so as not to require input from a user while driving.

### *B. Fuel Optimization Algorithm*

The firmware on the GEM system deals with polling the vehicle for as much fuel data as possible on every trip a user makes. This raw data is then transmitted to a user's mobile device with some basic organization. An analysis of the data is left to the mobile application with the use of high level programing. It is our intent to design a high level fuel optimization algorithm to efficiently process the data and achieve our team goal of providing practical fuel advice to users. While some vehicles provide unique PID's to measure specific fuel injection, others don't, therefore we can't rely on this data when designing fuel algorithms. We instead will opt to use widely available ECU data to meet the GEM system's requirements. The useful messages for this task are engine RPM, speed, air flow rate, throttle position, and fuel rate.

By synchronizing instantaneous data with fixed-interval time polls, average rates can be efficiently calculated. For instance, instantaneous air flow rate and fuel rate can be used to calculate an instantaneous miles per gallons (MPG) measurement. Synchronizing several measurements over equally divided time segments would provide an average MPG measurement. This technique can be generalized for acceleration and de-acceleration data as well. Thus, the first stage of the algorithm deals with synchronizing instantaneous data and time. The data is then normalized to lower error percentages and placed in a histogram-like data structure. The device will then compare this to, static, "optimal rates" pre-stored in the application software. These optimal rates will be derived from readily available fuel research. A point-by-point distance formula will be used to generate a correlation coefficient between the gathered average rates and optimal driving rates. A set of thresholds will then be used to trigger fuel advice based on several correlation coefficients for each driving parameter measured by the GEM system.

### *C. Firmware Design*

The MCU firmware was designed with modularity, portability, and reusability in mind. The design revolves around five interacting subsystems, gelled together by a main subroutine. A finite state machine best describes the design of the main routine. The subsystems are implemented as modular subroutines and each handle an independent task as follows:

1. Bluetooth Connection
2. Vehicle Profiles
3. Onboard Storage
4. OBD-II Data Pipelining
5. Power Control

Upon powering on the device for the first time, an initial setup state is entered by the MCU firmware. During this initialization stage, the MSP430 establishes a connection, and configures the STN1110, RN4020, and microSD chips to enable the core functionality of GEM. An "initial state" flag is cleared to ensure that these operations don't have to be repeated. Upon all subsequent device powering's a default operation state is executed instead. In this default state the VIN is always retrieved and passed on to the Profiles subroutine. The Profiles subroutine passes profile data to the Bluetooth subroutine. The main subroutine decides whether to proceed to either the Storage or Data subroutines based on connection status parameters provided by the Bluetooth subroutine. If a connection is not available or if there are previously unsent packets in the microSD card, the Storage subroutine is initiated. The Data subroutine is only initiated if a connection is available and there isn't any unsent data. Finally, the Power subroutine interrupts any executing routine when the Vehicle is turned in order to cache important data and power off the device.

There is a small footprint profile system onboard that serves two primary functions. The first is to permit GEM to be a portable system. The second is to enable quicker start up times when the vehicle is already known. The system software operates in either a Fetch state or a Create state. Up to 5 vehicle profiles may be stored, identified by the VIN and some other parameters. The system software compares the VIN to existing profiles. If there is a match, the corresponding profiles cached data is provided to the main subroutine. If the VIN doesn't match an existing profile, a new profile is created, or an older one is replaced if 5 unique profiles already exist.

The Bluetooth subroutine is triggered based on the Profile system's parameters, or status flags for non-transmitted packets provided by the Data system. When the system software in its default state it scans for connectable devices, if none are discovered a flag is set and execution is passed to the Storage system. If a device is found, the system proceeds to a Paring state which determines if a previously connected device is available. If an unknown device is paired with the GEM system, an authentication step is required, which is handled in a Connection state. Once authentication is achieved or if a recognized device is connected, the MLDP state is triggered. In this state the device prepares the RN4020 to blast OBD data to the mobile device. Once this is complete, a connection report frame is used to initiate the Data subroutine.

The Storage subroutine is triggered after the Bluetooth subroutine finishes execution. This subroutine operates in either a Write or Read state, based on flags set by other routines. When a connection is not available the system is placed in a write state, storing all OBD data in microSD blocks. The read state is entered when there are blocks in the microSD card that have not been transmitted yet. When a connection is available, and there aren't any non-transmitted blocks the system cedes execution.

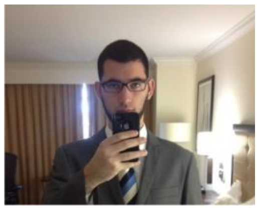
When several conditions have been met the main routine transfers execution to the Data subroutine. These conditions are passed in through several status flags. The system software alternates between two states, a Request state and a Pipeline state. In the Request state the ECU is constantly polled through the STN1110 and a small local frame is built. Once the frame is ready, it is sent to the Pipeline stage, which deals with transmitting to the RN4020 in MLDP mode via UART. Both states monitor for any status flags changes, such as loosing Bluetooth connection, and terminate the subroutine if there are any changes, passing control back to the main routine.

The Power subroutines main task is to handle the power states of all peripherals. Any of the other subroutines execution can be interrupted by the Power subroutine, when it is determined that the vehicle has been turned off. Coming from an interrupted routine the Power system initiates in the Interrupt state. In this state, important data (status flags) necessary for the next startup is cached, peripheral devices power states are altered, and any other tasks are interrupted. Then, an Idle no power state assumed. Once the device is powered on again, the idle state wakes up peripheral devices and returns execution to the main routine.

## ACKNOWLEDGEMENT

The authors wish to acknowledge Boeing for generously providing funding to design and develop the GEM.

## BIOGRAPHY



Pedro Betancourt is graduating from the University of Central Florida in May of 2015 with a Bachelor of Science in Computer Engineering. Pedro will be working for Capital One, as a Technology Development Associate, following graduation.

Alexander Patino will receive his Bachelors of Science in Computer Engineering from the University of Central Florida in May of 2015. He enjoys developing Android applications because the applications are endless. From guitar tuners and fitness tracking to social media and games, mobile technology is the future and he intends to be a part of it.

Mohhamad "Jake" Pulliam will be graduating with honors and receive his BSEE from the University of Central Florida. He is currently working to develop web and mobile applications that help connect students and employers at career fairs. His interests lie in hardware development, especially in the areas of filtering and signal processing. He has accepted a position with Texas Instruments as a field application engineer and will join them following graduation.

## REFERENCES

[1] United States Environmental Protection Agency <http://www.epa.gov/nvfel/testing/dynamometer.htm>