# Heterogeneous Automotive Response Apparatus Made for Broad Emergencies (H.A.R.A.M.B.E.)

Jacob Wurm, Tommy Goris,

Ismael Rivera, and Jihang Li

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

### *Abstract* — H.A.R.A.M.B.E. is a system that can extract information and generate data from a vehicle and then pairs with the user’s mobile device to display the data. In addition to displaying the data, the device will use this data to determine whether or not the user has been in an accident. In the event of an accident the accompanying mobile application will send a message to pre-registered contacts and will contact emergency services automatically. The product consists of a single PCB with all the circuitry required for the system and a mobile application to which the main module connects to. The mobile app allows users to track fault codes, speed, velocity, rotation, orientation, and much more.

### *Index Terms* — Vehicle safety, event detection, wireless communication, application software, mobile communication.

### I. Introduction.

A car crash is an event that should not be taken lightly. A single mistake behind the wheel could cripple someone for life, or even end in their demise. Throughout the years several different methods to mitigate such an event have been created. Contrary to existing products, we wanted our device to serve a single purpose which was to save user’s lives. Our main objective is to increase the speed at which emergency services is contacted to reduce the mortality rate of automotive accidents. H.A.R.A.M.B.E. constantly monitors the motor vehicle for any indicators that the user has been in an accident. These indicators will be checked against OBD-II, accelerometer, and gyroscope data. If detected that the user has been in an accident a signal will be sent to the driver’s mobile device.

1. *Constraints*

In its theoretical design H.A.R.A.M.B.E. needed to adhere to many constraints to ensure that it was a system that could function effectively and efficiently. In our initial discussion it had to be a compact, efficient, durable design, capable of pairing with a mobile device. This is primarily due to the environment and conditions the system will operate in. The main characteristics that were settled upon by the group can be seen in the list below.

* Wireless range of up to 20 feet
* Utilize 16 pin OBD-II interface
* Utilize 12v power provided by the OBD-II
* Understandable notification to driver within 1s
* Loud noises generated by user’s mobile device
* Secure Bluetooth data transmission
* 5V & 3.3V Power Supply to device components
* Signal mobile device within 2 seconds
* Device pair with a mobile device within 10s
* 4x4x1 inch PCB (including comp elevations.)

1. *Design*

The next task was to take all the variables that could be implemented and decide what should be prioritized in our system. After a series of thoughtful and conceptual problem solving, we decided on the following as a high-level overview.

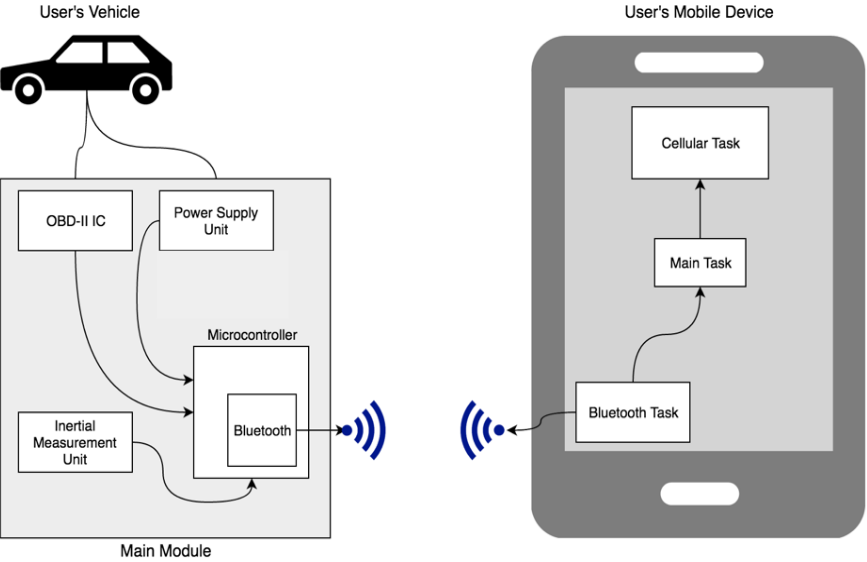


Fig. 1. Overview of Project Functionality

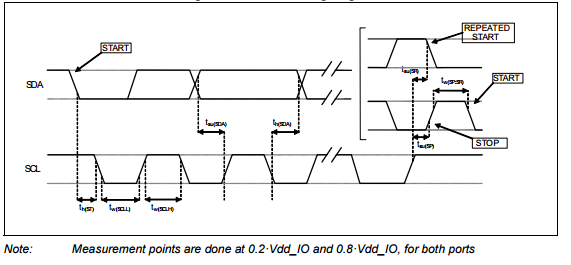
As shown in the figure above there will be 3 major systems connecting and communicating with each other. The first is the user’s vehicle which will feed the power supply and send data to the OBD-II IC. The second is the main module which will be a printed circuit board (PCB) that contains the OBD-II IC, the Power Supply Unit, the Inertial Measurement Unit (IMU) and the Microcontroller with integrated Bluetooth. The IMU will generate and provide data to the Microcontroller which will then use the integrated Bluetooth to connect to the user’s mobile device. The last system is the user’s mobile device. The mobile device will have 3 different types of tasks. The Bluetooth task will have highest priority compared to the other 2 tasks. The second task will be the user interface, since it has to notify the user that they have been in an accident. The final task will be the cellular task because this is when the system will contact emergency services as well as the user’s pre-registered contacts.

### II. Data Procurement

H.A.R.A.M.B.E. takes in data from 2 different submodules which is the Inertial Measurement Unit and the OBD-II IC. The Inertial Measurement Unit generates its own data which consists of acceleration, rotation, and orientation unlike the OBD-II IC which utilizes speed, engine RPM, throttle position, and fault codes that are generated from the car through the OBD-II port. These integrated circuits are used to provide data to the microcontroller will be discussed in further detail below.

1. *LSM9DS1 (Inertial Measurement Unit)*

The LSM9DS1 is a system-in-package featuring a 3D digital linear acceleration sensor, a 3D digital angular rate sensor, and a 3D digital magnetic sensor. It is capable of detecting up to 16G linearly and 2000 degrees per second. The magnetometer was not implemented in this project because it did not offer much more functionality for our project. This chip can also tolerate stress of up to 3000g for 0.5ms which is assumed to be durable enough for our application. Upon initial observation 16G seemed like much too small of a value for this application. For example a 35 mph collision against a wall can go up to 50G but in reality the minimum Gs needed to determine an accident is around 4G. This is due to the fact that the best brakes on a commercial car only experiences around 3G when braking. Severe accidents can experience up to 100G but detecting up to such a high value would be excessive for our application since the system requires a low threshold. The LSM9DS1 includes an I2C serial bus interface supporting standard and fast mode (SCL - 100 kHz and 400 kHz) and an SPI serial standard interface. We chose to use the I2C serial bus interface in fast mode due to the fact that the impact in a crash is a brief impulse voltage that is generated by the MEMS structure in the IMU chip and it also allows us to get more samples per second.

Fig. 2. I2C slave timing diagram.

The dimensions of the IMU are 3.5x3.0x1.0 mm.. This small footprint reduced the size of the main module but initially caused prototyping issues due to its uncommon LGA package. Fortunately there were development boards available that allowed us to prototype this IC.

1. *LSM9DS1 Operation*

In the LSM9DS1 the accelerometer and gyroscope have two operating modes available: only accelerometer active and gyroscope in power down or both accelerometer and gyroscope sensors active at the same ODR. Since we want to get orientation of the car and determine different types of accidents we chose to go with using the accelerometer and the gyroscope at the same time. To be more specific the mode of operation that was implemented was the accelerometer and gyroscope multiple reads (burst) operation. Essentially when both accelerometer and gyroscope sensors are activated at the same ODR, starting from OUT\_X\_G (18h - 19h) multiple reads can be performed. Once OUT\_Z\_XL (2Ch - 2Dh) is read, the system automatically will then restart from OUT\_X\_G (18h - 19h).

1. *STN1110 (OBD-II IC)*

The STN1110 is an OBD to UART interpreter IC designed to provide bi-directional half-duplex communication with the vehicle’s On-Board Diagnostic System (OBD-II). It supports all legislated OBD-II protocols. In other words if it contains the proper supporting circuitry it will be able to communicate with every vehicle made after 1996. As previously mentioned tapping into the OBD-II bus allows us to get a wealth of information, including the status of the malfunction indicator light (MIL), diagnostic trouble codes (DTCs), inspection and maintenance (I/M) information, freeze frames, VIN, hundreds of real-time parameters, and more. The STN1110 is fully compatible with the de-facto industry standard ELM327 command set. Based on a 16-bit processor core, the STN1110 offers more features and better performance than any other ELM327 compatible IC we looked at. To further compare the ELM327 v1.4 to the STN1110 we have provided a table in Figure 3.

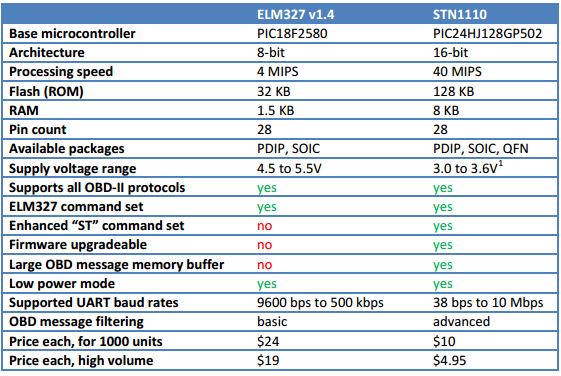


Fig. 3. ELM327 vs STN1110

When initially looking to integrate the STN1110 the circuitry seemed daunting because we did not have any previous experience with OBD-II protocols. In total there were three different transceivers. These three transceivers were for J1850, ISO and CAN. We chose to try to implement the CAN transceiver and the ISO transceiver because the 5 vehicles that we had available for prototyping and testing only supported these 2 transceivers.

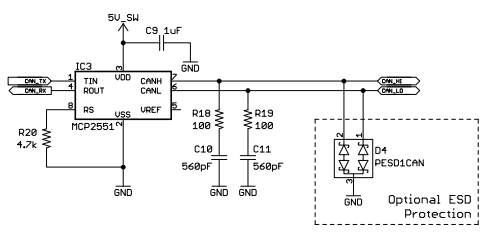


Fig. 4. CAN Transceiver

During the prototyping stage we attempted to use the ISO transceiver but encountered a series of issues. The components we had purchased could not function when connected to the car but worked effectively when tested individually. To reduce the amount of components needed overall in our PCB we chose to not use the ISO protocol in our final design. Upon further analysis we discovered that the BJT that was used to transfer data from the STN-1110 to the K-Line in the OBD-II port dissipated too much power. This as a result immediately did not allow the Main Module to communicate with the ISO supported vehicle.

### III. Data AggreGation and Analysis

1. *Design Architecture*

Initially when coming up with a system design there were two possible architectures. The first was that there was a microcontroller which purely performed calculations on the data collected from the other ICs on the board, and when it was time to communicate with the mobile device it would send data to a dedicated Bluetooth IC. The second architecture was to have a dedicated SoC which contains both the microcontroller and the Bluetooth IC on one chip which greatly simplifies the design.

After careful consideration, we decided to choose the latter architecture because it simplified the electrical design by a large margin.

In addition to the necessary Bluetooth functionality we needed a device that contained at least one I2C bus and one UART bus to be able to communicate with our onboard ICs.

1. *Microcontroller*

Since we decided to choose an SoC for our design our options were limited. There were offerings from TI, Nordic Semiconductor, and ST however the most powerful options were available from TI. Their most significant offering was the TI CC26XX series of chips. These contain an ARM Cortex M3 for the main computation portion and an onboard Cortex M0 for the Bluetooth portion. The M0 is not accessible by the user as it is used to maintain the Bluetooth connection, but the M3 is what the user programs. One issue that came about during the design stage was that these SoCs require an antenna to be run on the PCB. None of the group members had any experience creating an antenna much less adding one to a PCB, so we looked for third-party TI 26XX offerings. The most promising version we found was the SaBLE-X, which contained an TI CC2640 and an antenna that was already run. Figure 5 contains a block diagram of the CC2640.

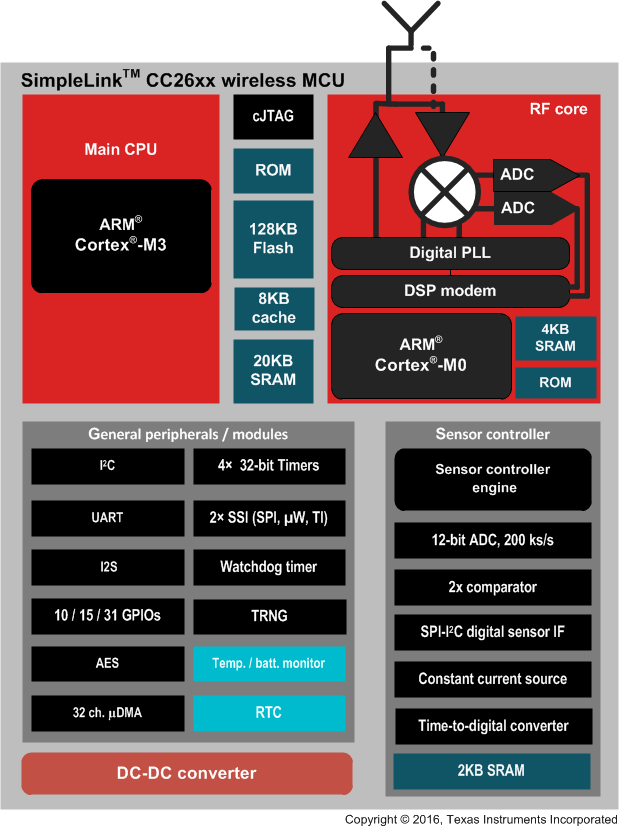


Fig. 5. TI CC2640

1. *CC2640 Design Quirks*

When designing software for the CC2640 we were aware that the development environment would be unique since it utilizes TI’s version of a real-time operating system, TI-RTOS. This introduces the concept of tasks and scheduling to the software development process, which is not usually present in microcontroller software design. Traditionally microcontrollers solely utilized interrupts for scheduling different types of functionality.

One other quirk that was found was that for the Bluetooth to function properly it needed a Bluetooth stack which takes up half of the available 128KB of code space. This meant that a significant amount of optimization had to be performed on the program in order to minimize code size while implementing the necessary functionality.

1. *Development Environment*

In order to develop software for the CC2640 we needed TI’s Code Composer Studio, Bluetooth Developer Studio, TI’s BLE SDK, and the TI-RTOS SDK.

1. *Software Design*

When designing the software, we needed to achieve the following functionality; Collect data from OBD-II bus and IMU, perform calculations on that data, send real-time data to the mobile device, and send results of calculations to the mobile device. Utilizing the concepts of a real-time operating system we divided the functionality up into different tasks. The OBD-II data collection, IMU, and calculation portions of the application were split up into their own tasks.

When the system first turns on the program will initialize the IMU and the OBD-II, checking to make sure the test parameters we are sending are getting the proper results. If the results are valid the tasks for the OBD-II and the IMU ICs will take turns gathering data and sending it to the mailbox of the calculation task. Once the calculation task received enough data in its mailbox it will perform the required calculations and will send a parameter over Bluetooth saying whether the user has been in an accident. In addition to sending the result of this calculation the calculation task will be sending data in real-time to the mobile application as it’s receiving it from the IMU and the OBD-II tasks.

This process will continuously be occurring while the device is powered on.

### IV. Power.

There were many different things considered for the power system since we will be getting power from the car battery. We chose to go with a switching regulator power supply because they are relatively simple and offered more efficiency than a linear regulator. This is because instead of drawing power from the supply the whole time it instead switches on and off for a period of time to draw less power.

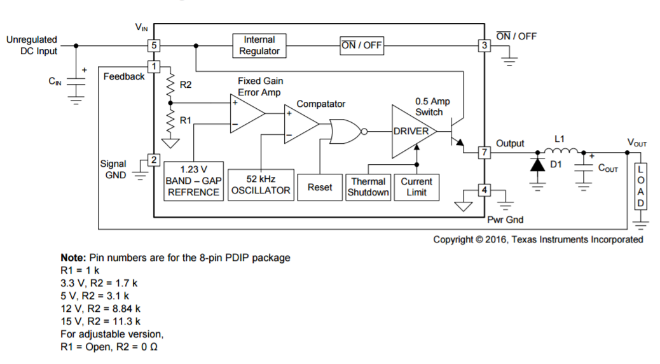


Fig. 7. LM2574 Block Diagram

A linear regulator at maximum efficiency would have only offered 41.7% and 27.5% efficiency for the 5 volt supply and the 3.3 volt supply respectively.

1. *LM2574*

The chip we chose to use was the LM2574x. The LM2574x series offers a high-efficiency replacement for popular three-terminal linear regulators. Due to its high efficiency, the copper traces on the printed-circuit board (PCB) are normally the only heat sinking needed. The variants we used were the 5V and 3.3V fixed regulators. The circuit required for this chip is actually very simple compared to most switching regulators due to the fact it only requires 4 external components.

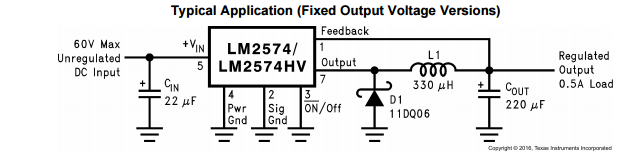


Fig. 6. Typical Application of LM2574

This chip only recommended for up to 0.5 Amps but this is more than satisfactory for our application. Unfortunately the minimum current supplied in continuous mode is 0.1 Amps.

This is fine for our 3.3V power line but not for our 5V power line. This is due to the fact that only the MCP2551-I/SN and a couple pins in the stn1110 will use this. So due to this our only option was to hook these regulators in series instead of in parallel. Since this was the case our total efficiency was reduced from 72% to around 60% efficiency.

In order to ensure the system works well on the first try we had to pay close attention to the equations and graphs to determine the value of our inductor and requirements for power supply circuitry.

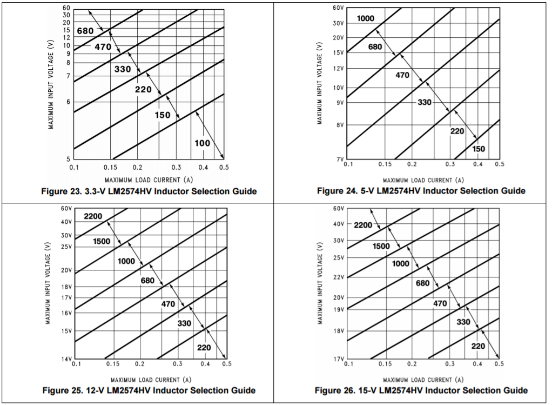


Fig. 8. Inductor Selection Guide

Of the 4 components the inductor was the most important. This is in part due to the fact that this is what determines if whether or not the output is continuous or discontinuous. When the system is in continuous mode it offers better load regulation, lower peak switch, inductor, and diode currents, and it can even have a lower output ripple voltage

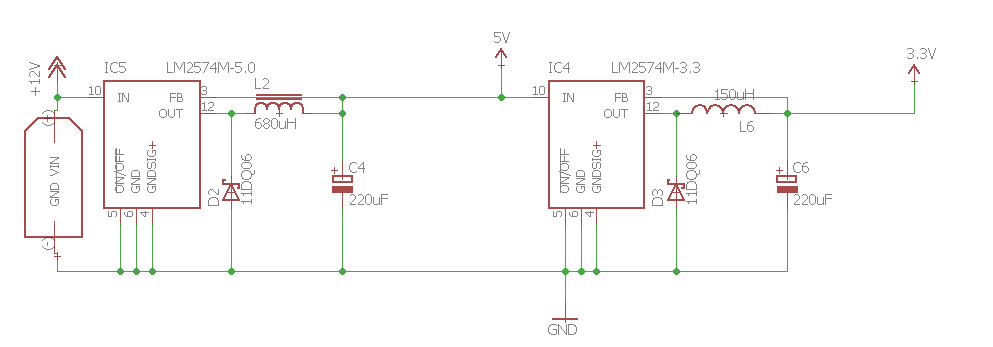


Fig. 9. Initial Power Circuitry

1. *Power Testing*

Initial testing had mixed results. This is primarily due to the fact that although it did provide 5V and 3.3V the output from the regulators were not clean and did not regulate well. Upon further inspecting the data sheet we noticed that a low-pass filter could be implemented to clean the output voltage. This does not actually help with regulation due to the fact that the feedback loop does not connect to the new node. The updated schematic for the power supply can be seen below.

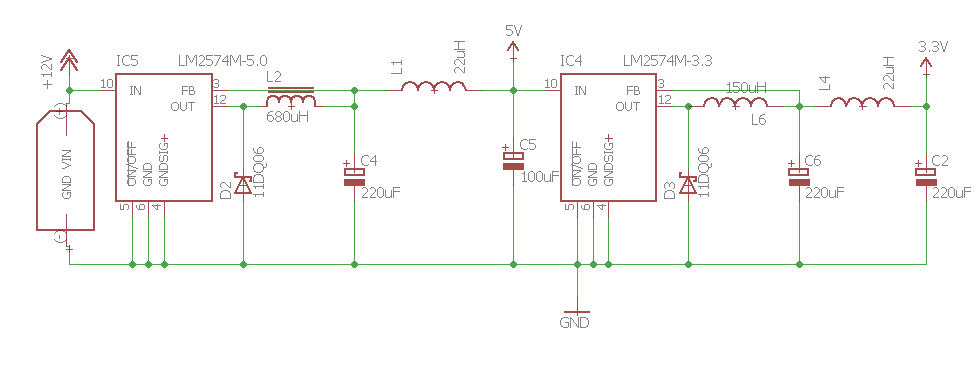


Fig. 10. Final Power Circuiting

This new schematic helped greatly reduce the amount of noise generated at both outputs. After testing with a power supply the voltage only fluctuated +/- 0.01Volts compared to the +/- 0.15 Volts we originally had. So, due to this we chose to use this schematic as our final design for the PCB.

### V. PCB And Schematic Diagrams.

The PCB schematics and board layout was done in Cadsoft Eagle 8.0.1. We chose to use this program because of the broad community and the ease it provided in quick production. Not all libraries were readily available for all components used, so some time was spent looking for footprints. Eventually almost all footprints were found but we had to still had to spend some time drawing up custom footprints for some parts. The PCB fabrication house that the group opted to use did not accept Eagle’s .brd files and thus there was a need to create Gerber files for manufacturing.

The final PCB design was chosen so that it would fit all necessary components in the main module. This incorporated all the needed components including the SaBLEx, LSM9DS1, STN1110, MCP2551, power supply and all the supporting circuitry. This all went in a package that is 4” by 3.15” with a thickness of an inch, perfect for our design constraints.

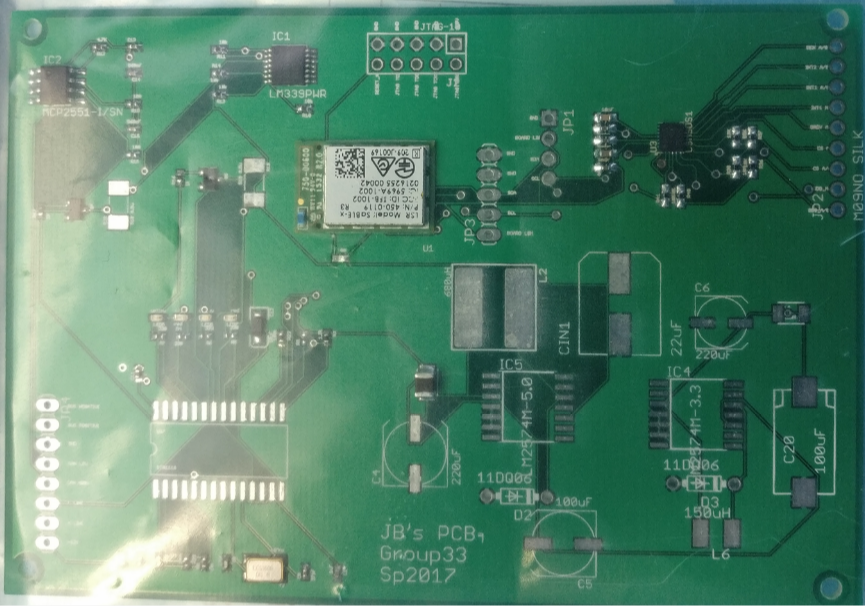


Fig. 11. The finished PCB.

The boards were manufactured by PCBWay at a price of five for $77. The price is very cheap, compare to other company of more than $100, however, the quality of the PCB from PCBWay is not really good, we were having troubles of copper on the PCB. This in the end resulted in being a good choice because we had to change some components or remove them due to initial miscommunication on the design. Components were obtained separately by the group mainly through DIGI-Key. The components which were too small to solder by hand were then soldered at Quality Manufacturing Services.

The enclosure is protecting the PCB from any possible damage can be apply to the PCB. The size of the box is 5 in x 4 in x 3 in, and the material of the box is plastic with a lid of metal. A OBD-II port is connecting with a 4 feet wires (8 each 22 AWG inside of the wire) to 6 pins on the PCB, pins are including power supply, ground, Can transceiver and ISO transceiver. With 4 feet wires, the box can be routed and placed in anywhere that is safe in the car. There is Velcro at the bottom of the box, which will secure the box under the car sit.

The following picture is the inside of the box:

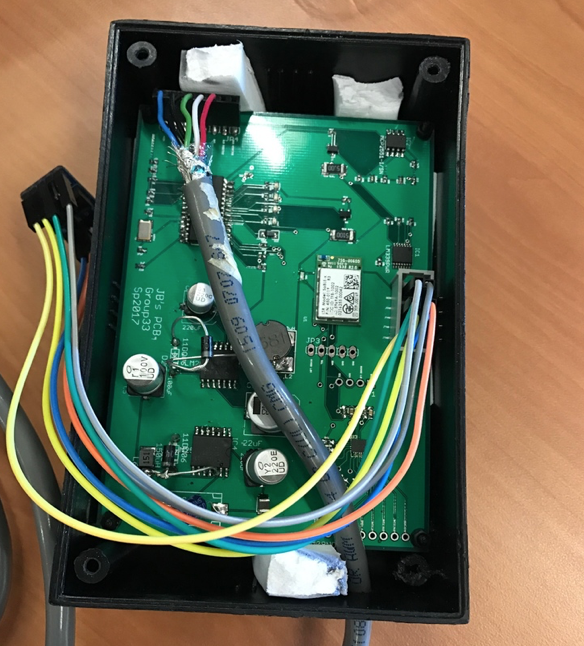


Fig. 12. Enclosure with PCB

### VI. Mobile Application.

In order for the user to communicate with the main module we needed to provide an interface for them. Since most people have mobile devices we decided to use Bluetooth for the communication interface along with a mobile application. After performing a cursory market analysis we came to the conclusion that developing an application for the Android device market was ideal. Essentially, as to why we chose Android, it has a much larger market share and is ultimately, less restrictive on the development process.

*A. Bluetooth Integration*

To integrate Bluetooth Low Energy into the Android application, an example Bluetooth Low Energy Android application was followed. In the mobile application, the user must first be asked whether his Android phone has Bluetooth functionality. If the user does not have Bluetooth functionality, the Android application will not let him connect to the main module. If the user has a Android phone with Bluetooth functionality, the application will ask the user to turn on Bluetooth on his phone. This will allow the Android application to display all nearby Bluetooth Low energy applications. A ListView is used to display the names of all nearby Bluetooth Low Energy applications. In this ListView, the user has the option to choose which of the nearby Bluetooth Low Energy applications he wants to connect to by simply clicking on it. To connect to our BLE application, the user must simply click on the BLE connection named as “HARAMBE”. Once clicked, a few things immediately happen on the Android application. The name of the BLE application and it’s address are sent to the Bluetooth Low Energy interface in the Android application. To actually connect our main device and the Android application, the BLE service on the phone must be started and binded to an activity in the Android application. The main device will be broadcasting the Bluetooth signal waiting for a device to pair with it. In this case, the main device acts as the server while the Android application acts as the client. The end all process of this is that we are finally connected to our main device in our mobile application. From here, we can write code to receive/send data from either BLE device. The BLE service will be running at all times in the background, so that BLE connectivity will not be lost once the Android application has paused or is sent to the background. It is important to keep the BLE device binded to activities in the Android application as you move to different activities. This way, you will be connected to the main module of our device at all times.

*B. Early-Stage Design & Development*

*Early design for our Android application mostly dealt with ways that we could inform others that the user has gotten into a car crash. The first activity that was created was the Main Activity which is the first thing the user sees when he opens the Android application. As previously stated, we implemented Bluetooth functionality into our Android application in the early stages in order to understand how BLE worked and to make sure we could sustain a connection between both devices. Once BLE functionality was implemented into our Android application, we decided to create an activity so that different contacts could be specified and contacted in the event of an accident. For this, we created a contacts activity which allows the user to specify a custom contact and/or get a contact from his phone’s contact list. To do this, we interfaced with two different list views, as well as an alert dialog to allow the user to input custom contacts. The next milestone in early stage design was to send a message to all the contacts in the contact list in the contacts activity. We created a send message service that is started in the main activity to do this. An Android service allows the phone to always be checking whether the user has gotten into a car crash in the background. This was needed because if the user puts the application on pause, it will still be checking for an accident. If an event is triggered that says that the user has gotten into a car crash, a signal is read in the send message service which then allows the Android application to send a message to everyone in the contact list of the user. That is the end of our early stage design for our mobile application. As development progressed, we added a popup asking the user if he has gotten into a car crash, real time data from the main module, and a terms of service right when the Android application is opened.*

*C. Running a Session*

The goal of the Android application was to constantly be checking in the background whether the user has gotten into a car crash. When a user connects the main module to the car, he will be able to connect to it via BlueTooth Low Energy. It will be constantly be checking in the background whether the user has gotten into a car crash. The user will have the option of adding different contacts to his contact list to be contacted during a car crash. Additionally, the user also has the option of viewing real time data obtained directly from the main module. These data, as of right now, is speed, gyro, acceleration, and messages from the BLE for debugging purposes. If the Android application determines that the user has gotten into a car accident, the user will be notified and a popup will show up. In this popup the user will have the option of either pressing yes or no. If the user presses yes then the app will immediately send a message to his contacts, as well as contact emergency services. If the user presses no, then the popup will close and nothing will be done. For precautionary measures, we added a 1 minute timer to the popup. If the 1 minute timer times out, in the case the user is unable to press yes or no, then his contacts and emergency service will be contacted. This is what the user can expect from our Android application when he is running a session.

*D. Testing*

*A ton of testing was required to make the Android application run smoothly. The first test conducted was on the BLE functionality. We needed to make sure the Android application stayed connected to the main module at all times. We tested this by repeatedly sending different amounts of data from the main module to the Android application at different time periods. We found out that the BLE functionality on the Android application was working properly, although there were a few quirks we needed to fix to make it better. Another aspect of the Android application that needed testing was the messaging system. The messaging systems needs to read all the contacts from the contact list and send a message to all these contacts. A few problems arose where the last person in the contact list was not properly receiving a message from the Android application. The problem was that the last contact’s number was getting overridden every time we tried to read it. Once we figured out why it was happening, we immediately fixed it and the messaging system started working perfectly.*

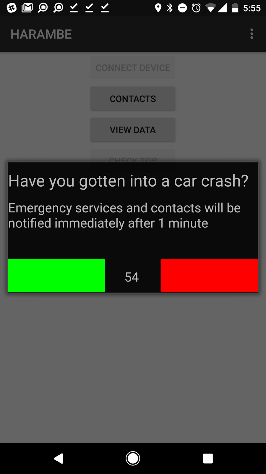


Fig. 13. Popup Message in Case of Accident

*The popup message also required a lot of testing since it was one of the most crucial aspects of our Android application. If the Android application determines that the user has gotten into a car accident, the Android application needs to open a popup message telling the user this. At first, the popup message was opening too many times which made the interface awkward. The reason for this was that for some reason the Android application was calling it twice. To fix this, we made sure to limit the number of times it could open by using a threshold. This seems to have fixed the issue. Getting the real time data from the main module was a whole problem on its own. We used fragments to display the real time data on the Android application which increased the difficulty of the implementation. Every time a fragment was trying to read from the main module, the BLE functionality would fail as it would be reading from a null reference. To fix this, we were required to send a reference of the Bluetooth Low Energy Service from the Data Activity which connects all the fragments which display data. The fragments must read this reference so that they can collect data from the main module without crashing. A few other problems arose while testing, but they were eventually fixed. Ultimately, most of the problems arose from either BLE or services and how threads must be initialized and started. There were also problems with using SharedPreferences to make sure the whole Android application was connected and could talk to each other. Testing is important for any type of project, in this case, we did a lot of testing to make sure our Android application works flawlessly.*

### VII. Conclusion.

This project has proven to be an incredibly valuable experience for each person in our group. We have applied the concepts and ideas that were introduced in class to bring our idea into reality. Additionally, the group came to understand the all the aspects of engineering that come together to bring a project into fruition.

All in all, this experience has shown us about ourselves, our communication skills, our relative skill sets, and engineering as a whole. Our device is fully functioning and works well in solving the problem that the group set out to fix.

### References.

The Engineers.

Jacob Wurm is a graduating Computer Engineering student who is taking a job with Raytheon Centers of Innovation as a Cyber Engineer in Melbourne, FL specializing in embedded systems cybersecurity research.

Tommy Goris is a senior at the University of Central Florida.  He is graduating with his Bachelor’s of Science in Computer Engineering in May of 2017. He is currently deciding whether he wants to attend graduate school or start working in the field. He has interest in artificial intelligence and algorithms.

Ismael Rivera is a senior at the University of Central Florida pursuing a major in Electrical Engineering. He has spent the last year working as an Electrical Engineering Intern at AVCON. Projects worked on consist of Low Voltage Systems, Airfield Lighting, and Lightning Protection. After graduation, he hopes to continue working in Power Electronics.

Jihang Li is graduating in May 2017 with a Bachelors in Electrical Engineering. He is currently working as Co-Op position in Contractor Company at NASA. In the work, he had opportunities to learn real engineering knowledges on power system and electronics. He is looking for more opportunities to gain more experience that related to electrical engineering filed.