**

*Heterogeneous Automotive Response Apparatus for Broad Emergencies*

(H.A.R.A.M.B.E.)

Senior Design I – Group 33

|  |  |
| --- | --- |
| Jacob Wurm | Computer Engineering |
| Tommy Goris | Computer Engineering |
| Ismael Rivera | Electrical Engineering |
| Jihang Li | Electrical Engineering |

1. Executive Summary 1

2.0 Introduction 1

2.1 Project Motivation 1

2.2 Project Overview 2

2.3 Project Objectives 3

2.4 Existing Product Research 4

2.4.1 OnStar and OnStar FMV 4

2.4.2 Automatic 5

2.4.3 Mercedes-Benz Mbrace 5

2.4.4 BMW ConnectedDrive 5

2.4.5 Freematics ONE 6

2.4.6 eCall 6

2.4.7 Lexus Link 6

3. Design Constraints 7

3.1 Engineering Requirements 7

3.2 Economic Constraints 7

3.3 Time Constraints 7

3.4 Political Constraints 8

3.5 Ethical Constraints 8

3.6 Environmental Constraints 8

3.7 Sustainability Constraints 8

3.8 Manufacturability Constraints 9

3.9 Safety and Security 9

3.10 Spectrum Considerations 9

4. System Design 10

4.1 Main Module 10

4.1.1. Functionality 10

4.1.2 Design Architecture 10

4.2 Mobile Device 11

5. Summary of Related Standards 12

5.1 Bluetooth 12

5.2 I2C 13

5.3 SPI 15

5.4 UART 15

5.5 Android Development Guidelines 16

5.6 OBD-II 17

5.8 RS-232 18

5.9 git 18

6 Hardware and Software Design 19

6.1 Microcontroller 19

6.1.1 Microcontroller Choice 19

6.1.2 Hardware Considerations 25

6.1.3 Development Environment 25

6.2 Bluetooth Transceiver 27

6.2.1 Nordic Semiconductor nrf8001 28

6.2.3 Operating Principles and Usability of CC2640 29

6.2.4 Security Constraints 29

6.3 Mobile Application 31

6.3.1 Platform Choice / Development Environment 32

6.3.2 Use Case 33

6.3.3 User Interface Design 34

6.4 OBD-II IC 38

6.4.1 Functionality 38

6.4.2 Hardware Evaluation 39

6.4.3 Hardware Selection 40

6.4.4 Transceiver Configuration 41

6.4.5 Interaction 43

6.4.6 Data Collection 44

6.5 Accelerometer and Gyroscope IC 45

6.5.1 Functionality 45

6.5.2 Interaction 48

6.5.3 Data Collection 48

6.5.4 Chip Selection 48

6.6 Main Module Software 50

6.6.1 High-Level Overview 50

6.6.2 MCU Software Flow 51

7.0 Summary of Power Supply 55

7.1 Main Power Supply 55

7.1.1 OBD-II Adapter 55

7.1.2 Cigarette Lighter Receptacle (DC outlet) 56

7.1.3 Conclusion 57

7.2 Main Power Regulation 57

7.2.1 Linear regulator 58

7.2.2 Switching regulator 59

7.2.3 Conclusion 64

7.3 Back-up battery supply 65

7.3.1 Battery Characteristics 66

7.3.2 Discharge topologies 67

7.3.3 Battery and Topology selection 72

7.3.4 Battery Control 73

7.4 Overcharge Protection 74

7.4.1 Overvoltage Protection 74

7.4.2 Overcurrent Protection 75

7.5 LED 75

7.6 Simulation 76

7.6.1 Simulation Program 76

7.6.2 Circuit configuration 77

7.7 Design Summary 79

8. Printed Circuit Board 81

8.1 Components and Chip Families Purchased 81

8.1.1 Suppliers for Components and Chip 81

SRR1208-221KL 82

UPW1J121MPD 82

EEU-FR1E680B 82

1N5817 82

LSM9DS1TR 82

8.2 Software Considerations 82

8.2.1 KiCad 82

8.2.2 EAGLE 83

8.2.3 ExpressPCB 83

8.2.4 Altium CircuitMaker 84

8.3 Software Decision 84

8.4 PCB Layout Consideration 84

8.4.1 Copper Thickness 84

8.4.2 Components Placement 84

8.4.2.1 Power Supply Placement 85

8.4.2.2 Capacitors Placement 85

8.5 Schematic Design Plan 85

8.5.1 Patterns of Components 86

8.5.2 Arrangement of regulators 86

8.6 Layout Design Plan 87

8.6.1 Single Layers 87

8.6.2 Multiple Layers 87

8.6.3 Via On PCB 88

8.7 PCB Routing Design 88

9. Project Prototypes 94

9.1 Prototyping Components 94

9.1.1 CC2650 Launchpad 94

9.1.2 LSM9DS0 Accelerometer/Gyroscope/Magnetometer Breakout Board 95

9.1.3 STN1110 Sparkfun OBD-II Board 95

9.1.4 SaBLE-x Breakout Board 96

9.1.4 LM2574 Buck Switching Regulator 97

9.2 Breadboard 97

10. Testing 99

10.1 Hardware Testing 99

10.1.1 Hardware Testing Environment 99

10.1.2 Hardware Chips Testing 100

10.1.3 System Testing 100

10.1.4 PCB Testing 101

10.1.5 Power Supply 101

10.1.6 OBD Port 102

10.1.7 Microcontroller 102

10.1.8 Accelerometer 103

10.1.9 STN1110 Multiprotocol OBD 104

10.2 Main Module 106

10.2.1 UART and Software Serial 106

10.2.2 Bluetooth profile testing and communication 107

10.2.3 JTAG debugging 108

10.2.4 Memory Constraints 108

10.2.5 Extended use effects 108

10.3 Sensor Module 109

10.4 OBD-II IC 109

10.4.1 Command Execution 110

10.4.2 Data validation 110

10.5 Android to Main Module communication 110

10.5.1 Bluetooth listener 111

11. Project Management 112

11.1 Management Plan 112

11.2 Milestones 113

Tommy Goris 114

Jacob Wurm 115

Ismael Rivera 115

Jihang (Bruce) Li 116

12.3 Division of Labor 118

12.4 Finance 119

12.5 Team Organization 119

12.5.1 Overview 119

12.5.2 Communication 120

12.5.3 Information Sharing 120

12.5.4 Version Control 120

Appendices I

Appendix A - Abbreviations I

Appendix B - List of Tables II

Appendix C - List of Figures III

Appendix D - Datasheets IV

Appendix E - References V

Appendix F - Copyright Permission Requests VII

# 1. Executive Summary

Car crashes are all too common nowadays for a variety of reasons. A single mistake behind the wheel could cripple someone for life, or even end in their demise. The speed at which emergency services is contacted and arrives at the scene of a car crash could mean life or death in many situations. In 2014 a study conducted by the National Highway Traffic Safety Administration showed that approximately 32,765 people are killed in car crashes in the United States every year. This is a staggering amount considering car manufacturers allocate a significant amount of resources to improve the safety and security designs of their motor vehicles. Safety and security provided by a vehicle, however, can only get you so far. If a driver decides to drive under the influence or refuses to stop texting while driving, the risk of getting in an accident is much higher. Anyone is prone to getting in an accident and there is no way to predict it. Our goal is to prevent the deaths of drivers by ensuring help arrives at the scene of the accident in a timely manner. Our system will constantly monitor 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 it is detected that the user has been in an accident a signal will be sent to the driver’s mobile phone. Next, an application that the user has installed will capture the signal and will contact emergency services as well as the user’s defined emergency contacts. In addition to providing these features we will also provide a user-serviceable backup power source for the system in the event that power is lost in an accident (i.e. the battery is disconnected from a front-end impact). This functionality is essential to ensuring the safety of the user in any accident and increasing the reliability of the product. Existing products on the market are relegated to specific vehicle manufacturers, have a large price tag, or both. We seek to minimize the cost of our design while keeping the core functionality of those devices present. We will reach a larger audience than a majority of existing implementations by only depending on the user having an Android device and a vehicle that was manufactured after 1996. When utilizing our product users can be sure that they have a chance to get the help they need in the event of an accident.

# 2.0 Introduction

## 2.1 Project Motivation

The primary motivation for developing H.A.R.A.M.B.E. is to improve the survival rate of drivers in the event of an automobile accident. Drivers can get themselves into situations where they are unable to reach their mobile devices and thus are unable to contact emergency services. We would like to mitigate this problem by introducing a device which will detect whether the user has been in an accident and if so, it will contact emergency services and their emergency contacts on their behalf. By creating a device like this we will be able to get drivers the help that they desperately need in the event of an accident. The quicker emergency responders are dispatched the faster the driver is able to get help.

Products already exist on the market that have a similar functionality but there are large barriers to entry such as the cost or that it’s relegated to a specific manufacturer such as General Motors or Mercedes-Benz. We wanted to expand the audience of drivers that can be helped by building a device that would work in any vehicle that contains an Onboard Diagnostic Port (OBD-II). These are present in every vehicle manufactured after 1996, and as such this is standardized. By providing the bare essential functionality we are able to keep the cost of the device down while still saving the most amount of lives. One feature we haven’t seen from third party competitors is a battery backup in the event that power is lost in the vehicle. By implementing this solutions we hope to make our system more reliable.

In addition, other products such as the systems implemented in certain vehicles require a significant amount of proprietary, manufacturer-specific hardware which is costly. We wanted to utilize technology that most drivers already have, smartphones. According to a PEW study 64% of American adults have a smartphone. Smartphones are essentially powerful computers that people carry around. Since the adoption of smartphones is widespread we wanted to utilize what that demographic has. In terms of operating system, according to IDC the market share for Android is 87.6% and iOS is 11.7%. A majority of these devices contain wireless technologies such as Wi-Fi and Bluetooth which can be used to extend the functionality of the device. Common examples of utilizing this technology include using wireless headsets to make phone calls and transferring data between devices such as sensors. We seek to utilize an extremely common wireless interface, Bluetooth, in order to connect with the main module of our system.

In addition, as a group we wanted to learn more about embedded systems development. After taking the Embedded Systems course at UCF the group members wanted to learn about writing software and building hardware for more complex microprocessors such as those that are ARM-based. Next, the group members wanted to be able to implement serial communication technologies such as UART, SPI, and I2C. The group hopes to gain valuable and practical experience developing embedded systems which this project will provide.

## 2.2 Project Overview

The project will be split up into two separate components, the mobile application and the main module. The purpose of the main module is to gather data from the vehicle’s OBD-II port (i.e. speed, engine RPM), an accelerometer, and a gyroscope then perform comparisons on that data to detect whether the user has been an accident. This will require separate ICs for each part of the functionality (OBD-II, accelerometer), and will require a powerful microcontroller to handle the data and perform comparisons. In order to fulfill the requirement of contacting emergency services the main module will connect to the user’s mobile device through a Bluetooth interface, which will also require its own IC. The main module will only send a signal to the mobile device in the event that the user has been in an accident.

As for the mobile application, at a high level it will initially pair with the main module to establish a wireless connection. After the main module is connected it will ask for the user’s emergency contact information. Once the user has entered all of their information the application will simply sleep and wait for a signal from the main module. In the event that it does receive a signal from the main module it will prompt the user as to whether they need help. If the user is unable to respond the application has a built-in timer which will contact emergency services after a set period of time. In addition to contacting emergency services it will send a text message to the user’s emergency contacts that they specified earlier. Figure 1 shows a high-level overview of the project’s architecture.

Project_overview (1).png

Figure : Overview of Project Functionality

## 2.3 Project Objectives

The following is a list of the group’s objective for the H.A.R.A.M.B.E project. This list was created based on our goal of developing a product with will increase the likelihood of our users living in the event of an automobile accident. The group also took into consideration the available in-vehicle mounting options and space constraints, as well as the safety of the driver while the device is operating. The module will be appropriately sized for under-dash mounting in a passenger vehicle.

The primary objectives of the H.A.R.A.M.B.E project are summarized as follows**:**

1. Driver notifications must be easy to understand and interpret
2. The module should have a low enough power draw to be able to safely operate off of the power provided by the vehicle’s OBD-II interface
3. The module should switch to a backup battery in the event of a power failure
4. The module should be able to constantly gather data from the included sensors
5. The unit should process information fast enough to detect the signs of an accident
6. The unit should configure itself and the user will only have to communicate with it through the mobile application
7. The device should operate reliably for an extended period of time
8. The device should operate reliably on the included backup power supply
9. The device will be able to securely communicate with a mobile phone
10. The device will not transmit sensitive data without the user’s consent
11. The device will do its best to eliminate false positives
12. The device should utilize a cost-efficient design to encourage user adoption
13. The device should have a fail-safe mode of operation in the event of an accident
14. The device should be able to communicate with any vehicle that contains an OBD-II bus
15. The device should be rugged enough to handle typical vehicle conditions

## 2.4 Existing Product Research

As a significant part of this project involves research into solutions that can be utilized in our design the team decided to look at other existing products. This encouraged the team to look at different solutions to a similar problem, possibly exposing them to different technologies that have not yet been considered. Examining existing products is an effective way to gain inspiration on how to implement a solution to a problem. This research involves functionally similar projects and products that are already on the mass-market.

### 2.4.1 OnStar and OnStar FMV

OnStar is a built-in vehicle safety system for vehicles manufactured by General Motors. This subscription-based system allows users to contact OnStar call centers in the event of an emergency. Emergencies can be detected through airbag deployment, or other sensors in the body of the vehicle. In order to contact the call centers they partnered with Verizon Wireless, a CDMA mobile voice provider, in the United States and Bell Mobility in Canada. In addition to contacting emergency medical services in the event of an accident they also provide functionality such as stolen vehicle assistance such as vehicle tracking, vehicle slowdown, and remote ignition block. Also, OnStar provides a “vehicle manager” which provides advanced diagnostics letting the user know if there are any problems with their vehicle and whether they are in need of routine maintenance. This information is relayed to the user through their companion mobile application. If the user wanted to contact emergency services for other reasons such as needing emergency assistance, finding escape routes, or other disasters they can press a red button that is located on their rearview mirror. In addition to an emergency button there is an OnStar button which will contact the OnStar call center where an agent will handle your call. In order for the OnStar system to gather data about the vehicle it utilizes the vehicle’s CAN bus.

An aftermarket version of OnStar called OnStar “For My Vehicle” or FMV replaces the user’s vehicle’s rearview mirror, bringing the core safety features of regular OnStar to non-GM vehicles. This links the user to OnStar advisors through a built-in CDMA cellular device. When the accelerometer in the mirror detects an impact, a call is automatically placed. Drivers are also able to contact emergency services by pressing a red button located on the replacement mirror. The built-in GPS allows OnStar to locate the user’s vehicle in the event that it’s stolen.

### 2.4.2 Automatic

Automatic is an aftermarket vehicle information gatherer and aid. The system involved an OBD-II plug which connects to your vehicle’s OBD-II port and connects wirelessly to the user’s smartphone through Bluetooth or in Pro versions through a 3G cellular connection. Crash detection is only available in the Pro version of this system, but it also provides a slew of features such as trip logging, business tagging, engine light diagnostics, a web dashboard, parking tracking, live vehicle tracking, streaming applications, and external application integration. The objective of this device is to be the end-all-be-all for car technology. One device provides a majority of the functionality a user will need in their car. By connecting the user’s vehicle through OBD-II their product applied to a massive audience since a majority of vehicles on the road were manufactured after 1996, which is when OBD-II became a mandatory standard.

### 2.4.3 Mercedes-Benz Mbrace

Mercedes-Benz provides a similar service to OnStar with its products. This will allow the driver to communicate with emergency services in the event of an accident.  In addition it allows the user to remotely start their vehicle from their phone or computer, locate and track their vehicle, provides remote diagnostics, and provides in-vehicle support for applications like Yelp and other local search tools.

### 2.4.4 BMW ConnectedDrive

This is a telematics roadside assistance service offered by BMW. It utilizes both a cellular network and Global Positioning Telemetry to locate or guide the vehicle. This system allows the user to get aid in the event of an emergency along with sending diagnostic data to BMW about their vehicle.

### 2.4.5 Freematics ONE

The Freematics ONE comes in the form of an OBD-II dongle which plugs into the vehicle’s OBD-II port. The objective of this device is to be a vehicle data logger. The data collected includes reading car trouble codes, measuring car battery voltage, and GPS positioning using an external GPS receiver. This data is stored on a microSD card which is inserted into the included microSD slot on the device. In order to connect to the user’s mobile device it utilizes Bluetooth low energy.

### 2.4.6 eCall

eCall is a European initiative to provide rapid assistance to motorists involved in a collision anywhere in the European Union. The idea is that this device will be installed in all vehicles and they will call the European equivalent of 911, “112”, in the event of a road accident, and wirelessly send airbag deployment and impact sensor information, as well as GPS coordinates to local authorities. The European Union agreed to have this in every motor vehicle by 2018.

### 2.4.7 Lexus Link

Lexus Link is a subscription-based safety and security service offered by Lexus that is available on specific Lexus models. This service offers call-center-based telematics services to owners of equipped vehicles in the United States and Canada. For telecommunication it utilizes a dedicated CDMA device and GPS. In the event that a driver’s airbag is deployed a call is automatically made to a Lexus Link call center. The Lexus link also offers the following features: stolen vehicle location, roadside assistance, remote door unlocking, remote horn and lights, accident assistance, and driving directions

# 3. Design Constraints

The following pages outline the design parameters of the project as well as the considerations that were made in the final design. These parameters are based on the project objectives as well as realistic constraints such as the electrical characteristics of the available power source and industry regulations and standards. Other items are considered such as ethical and environmental constraints.

## 3.1 Engineering Requirements

The following requirements were developed for the project’s functionality:

* The device will have a wireless range of up to 20 feet
* The device must use the 16 pin OBD-II interface
* The device must utilize the 12v provided by the OBD-II interface for power
* Driver notifications must be understandable to the driver within 1 second
* In the event of an accident, loud noises must be generated to draw attention to the user’s mobile device
* Software must be able to securely handle Bluetooth data transmission
* The device’s power supply must supply 3.3V to the device’s components
* The backup battery must be able to sustain the device for at least 30 minutes
* The device must be able to send a signal to the user’s mobile device if the user has gotten into an accident within 2 seconds
* The device must pair with a mobile device within 10 seconds.

## 3.2 Economic Constraints

Economic constraints is something we are well aware of with the creation of our project. It will be one of the bigger problems when designing our project, H.A.R.A.M.B.E. Since there will be a significant amount of modules developed into one board, multiple layers of PCB might be required. This project is being developed by college students so the amount of reserve money that we have is low. Being full-time students will prevent us from having enough time to earn extra money. To help alleviate the lack of funds, the group applied for funding through a contact at Raytheon. We were fortunate enough to get the funding for the project approved. We were given an overhead of 20%, above or below our budget. This will give us wiggle room in the event that something goes wrong during the development process. For example, if we have to repurchase certain pieces of hardware that will drive the cost of development up. Our research and development budget is small compared to mass produced products similar to H.A.R.A.M.B.E. We have recognized this and have planned accordingly.

## 3.3 Time Constraints

The amount of time that we have to develop H.A.R.A.M.B.E is a time constraint for what we are trying to create. First of all, the technologies we are using in this project are foreign to the members of the group. There will be a learning period for becoming proficient with these technologies. As a group, we also have to figure out what kinds of technologies we require to build H.A.R.A.M.B.E. In addition to this, we will need some time to plan on how we will implement these technologies. This will not include the research time necessary to determine of which chips we will need to use to meet our design specifications. The development process of the project will require us to not just spend time researching for technologies, but understand the process of creating PCB layouts and programming for real-time operating systems. Furthermore, we will need to learn on how to implement regulators and learn how to implement wireless technologies. The time spent on learning how to implement these technologies will take a significant portion of our allotted time for the project.

## 3.4 Political Constraints

There are no relevant political constraints that we have attributed to the development of H.A.R.A.M.B.E. We believe the development of this project is not aligned with the ideals of any one political system or party. Our research has also indicated that there is not any noticeable political involvement in products that are similar to ours.

## 3.5 Ethical Constraints

The only ethical constraint that we can think of is the choice of resuscitation. In the event that the driver gets into a fatal accident it is unlikely that they will want to be revived. Since emergency personnel have to abide by the driver’s choice we feel that our product does not encroach on the rights of the individual.

## 3.6 Environmental Constraints

There is no doubt that the power usage from this product will have a negative impact on the environment. First of all, the product will be used in a motor vehicle. The functionalities of our tool will rely on the premise that the vehicle is turned on and the consumer is driving it. If the driver is driving safely, they will be less likely to get into an accident. Getting into an accident can have negative environmental consequences such as leaking gasoline and vehicle fluids into the environment and burning harmful elements.

On a different note, the product will be drawing power from the car’s battery, so it must draw as little power as possible to prevent draining it. As car batteries are commonly lead-acid, it would be beneficial to drain it very slowly.

While developing H.A.R.A.M.B.E our goal was to develop something that was extremely efficient on energy.

## 3.7 Sustainability Constraints

In the development of this project, we have decided to use commonly available parts to increase the efficiency and reliability of the product. By using this design methodology of using common parts, repairs will cost less. It will also have a small impact on the global supply of the parts that we will be using.

## 3.8 Manufacturability Constraints

H.A.R.A.M.B.E has the potential of being implemented in a significant amount of vehicles. In this case, the manufacturer would need to create a plan on how to implement our device into their vehicles. To make integration easier for the manufacturer, we are always reflecting on how a change in our design might affect the overall ease of reproduction. Since the system is composed of a main MCB, we have identified this as the place where we can maximize manufacturability. To maximize manufacturability, we have constrained ourselves to using only highly available parts to create our design. If we follow through with this commitment, it will ensure that H.A.R.A.M.B.E will be easy to replicate.

## 3.9 Safety and Security

Safety and security is our primary constraint as we continue the development of H.A.R.A.M.B.E.. Since our project will be integrated into a motor vehicle it must not negatively affect the driver’s ability to drive. More specifically, it must not obstruct the driver’s feet or affect the drivability of the vehicle. In addition, this device is intended to aid the driver in the event of an accident. The device must be able to detect an accident when it happens and it must communicate with emergency services within a very small time frame. Seconds matter when it comes to saving someone that has gotten into an accident.

There is a tradeoff that needs to be made in the event that the driver is in an accident. If the accident is very minor (e.g. bumping into another vehicle), there is no need for emergency services to be called. Since the system may interpret this as an accident we must have some kind of user interaction to prevent emergency services from being notified. This is where the mobile device comes in. The user is able to turn off the system in the event of a small accident. Although if they do not respond on their mobile device within a certain amount of time emergency services will be notified.

The device failing should not prevent any major systems from functioning properly. In general, some principles for making sure that safety and security problems are taken into consideration are

* Analyzing potential problem areas
* Developing solutions for problem areas
* Anticipating failures and handling them accordingly

Through analysis beforehand we may be able to determine problem areas. However during development and testing we will find more problem areas which need to be remedied.

## 3.10 Spectrum Considerations

For H.A.R.A.M.B.E we will need to utilize Bluetooth for over the air communication between the main device and a mobile Android device. Since Bluetooth utilizes the 2.4Ghz bands we will use these bands. We need to make sure that when we

# 4. System Design

This section contains a high-level overview of the general architecture of the main module and the mobile application without going too far into implementation details. This chapter includes an overview of the basic design architecture and the components that are necessary for the functionality of the system. The more low-level implementation details and the decision making process for choosing ICs is in Chapter 5.

## 4.1 Main Module

One of the most important pieces of HARAMBE is the main module. The main module serves as the intermediary between a paired mobile device and the vehicle’s OBD-II interface. This device is also responsible for detecting if the user has gotten into an accident. The following sections will detail the functionality of the main module, as well as detailing the design for the main module’s architecture. Diagrams will be presented that will show how the components within the main module shall interact with each other.

### 4.1.1. Functionality

The purpose of the main module is to act as an intermediary between a paired mobile device and the vehicle’s OBD-II interface. This module will contain an IMU along with an OBD-II interpreter IC and a microcontroller that will be able to gather data from the two ICs. In addition, the data will be transferred between the mobile phone and the main module through a Bluetooth connection.

The main objective of HARAMBE is to detect whether the user has gotten into an accident in real time. If the user has gotten into a car accident, our tool will contact emergency services on their behalf. The main module will poll the OBD-II IC and the IMU for data, interpret that data to determine whether the user has been in an accident. As said before, if the user has gotten into an accident, our tool will send a signal to their mobile device which will automatically contact emergency services.

### 4.1.2 Design Architecture

There are a multitude of possible architectures that could be used in the creation of the main module for HARAMBE. The first solution that came to mind was attaching a PCB to a pre-existing microcontroller solution such as the Arduino, or Raspberry PI. In terms of development speed this would be beneficial, however the amount of processing power we will have is overkill and the amount of power the system would draw is too large. Another architecture would involve creating a custom PCB that contains all of the necessary components. This will exclude some components that would be connected to the PCB externally. The final design considered for this project will have all components implemented on a custom PCB or multiple PCBs.

The main module for HARAMBE will require interaction between several different components in order to realize the desired functionality. The functionality of the primary component is characterized by determining the user’s speed, acceleration, and position in space. This device will also require a power source, so the device will be integrated with its own power supply unit.

The most useful components in the main module are the OBD-II IC and the IMU. Through the OBD-II interface we will be able to gather information about the vehicle such as its speed and the vehicle’s current error state. As for the IMU the goal is to get the acceleration of the vehicle along with its orientation.

In order to handle the collection of the information from these ICs a single microcontroller will be used. Polling for information from the OBD-II interface and the IMU is asynchronous so these processes can be threaded on the microcontroller. The only portion of the main module that could be asynchronous is the Bluetooth communication. Many different implementations have a separate Bluetooth module that takes in serial data and sends the signal out on its own, meaning the host who sent the signal does not have to be concerned about the timing of sending the signal since it is abstracted away. Ideally we would like to have a microcontroller that can gather and interpret this data as quickly and efficiently as possible while also being power efficient.

## 4.2 Mobile Device

The purpose of the mobile device in HARAMBE is to prompt the user for confirmation on whether they were involved in a car accident. If the user confirms that, yes, they have gotten into an accident. Our tool will immediately notify emergency services. The user will also have the option of adding personal contacts that will be notified in the event of a car crash. If the user is unable to respond to the alert within a certain amount of time the mobile application will assume that they were involved in an accident. In terms of components the user will only have to supply a single mobile device that is Bluetooth-capable and is able to communicate with emergency services.

# 5. Summary of Related Standards

In this section, we list and discuss the standards and regulations our device is affected by. The device is been created to be used for motor vehicles, so it is important to strictly adhere to standards and regulations involved in automobiles. Standards for devices such as the OBD-II and Bluetooth already exist and have a long history behind them. For standards that relate to communication, such as, UART, I2C, etc., most of them relate to how they need to be configured so that they work properly. Since we will be creating a phone application for the project, the standards and guidelines of Android programming will be considered when developing the application. Although there are many different standards relating to the automobile industry and the tools needed to make our project function, this section will only provide a small list of all the standards that apply to our project.

## 5.1 Bluetooth

The specification for the Bluetooth core defines the technology buildings blocks that developers use to create the devices making up the Bluetooth system. The Bluetooth Special Interest Group (SIG) oversees the Bluetooth specification. It is often updated and enhanced by the Bluetooth SIG Working Groups to meet the evolving needs. The Bluetooth Basic Rate/Enhanced Data Rate (BR/EDR) adopted as version 2.0/2.1 and Bluetooth with low energy (LE) adopted as version 4.0/4.1/4.2 are the two most widely used implementations of the specification. Implementations like these both have different uses and both use different chipset to meet hardware requirements. Even though each implementation is created to satisfy different needs, the Bluetooth core system architecture defines the elements each implementation is required to have. These elements includes an RF transceiver, baseband and protocol stack. Bluetooth enabled devices exchange protocol signaling is based on the Bluetooth specification. Core protocols such as the radio (RF) protocol, link (LC) protocol, link manager (LM) protocol, and logical link control and adaptation protocol (L2CAP). The radio, link control, and the link manager protocols define the lowest three system layers and are often grouped into a subsystem known as the Bluetooth controller. A common implementation which uses an optional standard interface, the Host to Controller Interface (HCI), enables two-way communication with the remainder of the Bluetooth system known as the Bluetooth host. By using the protocol messages that are exchanged between equivalent layers, it enables Bluetooth specification interoperability between systems.

Bluetooth is associated with the IEEE 802.15.1 wireless communication standard that operates the 2.4 GHz ISM band. Bluetooth was created in 1994 as a wireless alternative to data cables through the use of radio transmissions as the method of transmission. To be more specific, the standard was originally designed to be a wireless replacement for RS-232 cables and has come a long way since then.

Since Bluetooth is an open standard, different products and companies can freely use Bluetooth to connect their services. Bluetooth is used in many different fields, but it is widely used for the transmission of wireless audio. Exchanging data using radio transmissions allows for a hands-free experience, meaning you can simply connect your object through Bluetooth is it’s supported and not worry about a physical connection. Bluetooth also has functionality for low energy use. Low energy use allows for increased flexibility and functionality a developer can take into account when creating his product.

A Bluetooth connected device works by sending radio waves to connect to another Bluetooth connected device, such as a phone or computer. A tiny chip with Bluetooth is added to a product, alongside its software, allowing for the connection of two devices to be possible. The process of connecting to another device is usually described as pair/pairing. A downside of pairing is that both devices need to be close enough to each other, so they can talk to each other and pair. Bluetooth technology uses piconet as the root of connection between two devices. This is done dynamically and allows for up to two to eight devices to connect to one piconet. Although Bluetooth uses piconet technology, Bluetooth is a packet-based protocol. This allows for data to be transmitted between devices in discrete data units called packets. The most common Bluetooth network architecture is a master-slave configuration. This means that, at most, there will be one master and up to seven-eight slaves in a Bluetooth connection. This type of connection will suffice for our project. Ideally, the use of Bluetooth in our product will mainly rely on pairing one phone to our device. Bluetooth can get a maximum data rate of 24 Mbps at V3.0+ HD. The latest version of Bluetooth (V4.0) now has better functionality called Bluetooth smart/BLE, Bluetooth low Energy that allows for low energy use. BLE lowers range and data throughput for power consumption savings. BLE offers 0.27 Mbps data transfer rate and a range of 50 meters. The Bluetooth module, TI CC2450, when compared to other modules such as, ZigBee and ANT standards, performed significantly better. As a Bluetooth low energy module, it had a power usage of 0.7800 uA and only an awake power usage of 4.5 mA on a power supply of 3.3 volts. Each Bluetooth connection has a 48-bit address that is used to create the connections. This means every Bluetooth device will have a unique address. When creating a connection using Bluetooth, it is necessary to first start the device in the discoverable mode, so that other Bluetooth devices can locate it. Once the Bluetooth device is in discoverable mode and other devices are able to locate it, the next step is to pair both devices. When two devices are paired, they create and share the same link key. The link key creates the connection between the two devices so that they can communicate together.

## 5.2 I2C

Inter-Integrated Circuit (I2C) is a powerful and popular bus used for communications between a master or multiple masters and a single or multiple slave devices. I2C allows for a single data line to be used for bidirectional data flow by using an open-drain/open-collector with an input buffer on the same line. The standard bidirectional I2C interface uses a controller, also known as the master, to communicate with slave devices. In the I2C interface, slaves may not transmit data unless it has been addressed by the master. Devices on the same I2C bus will have a specific device address to differentiate between other devices. Generally, slave devices will require configuration once started to create the behavior of the device. This behavior will be specified when the master accesses a slave’s internal register maps, based on unique register addresses. Physically the I2C interface consists of the serial clock (SCL) and serial data (SDA) lines. SDA and SCL lines must be connected to Vcc through a pull-up resistor.

For a master device to access a slave device, the procedure below is followed:

1. If data needs to be sent from the master to a slave:
   * Start condition is received and the slave responds to it.
   * Data is sent to the slave from the master.
   * Stop condition ends the transfer of data.
2. If data from the slave needs to read or received by the master:
   * Start condition is received that addresses the slave transfer.
   * Register requested is sent to the slave.
   * Data is received by the master from the slave.
   * Stop condition ends the transfer of data.

The way data is sent and received from or to subsequent slave devices is through the use of reading or writing to or from registers in the slave device. The slave’s register are located in the memory useful for containing information. In this case, this information can either be configuration information or data sampled that needs to be sent back to the master. Some slave devices may have a limited amount of registers, if not just 1 register or no registers at all. If the slave device only have 1 register, data can be written directly to by sending the register data right after the slave address instead of just addressing the slave’s register.

When writing to the slave device on the I2C bus, the master will direct a start condition with the slave’s address. The register address the master wishes to write to will then be sent. Acknowledgement will be received by the slave once it is ready. The master will start sending the data to the slave. Once all the data has been sent by the master, the stop condition will be received to acknowledge the termination of the transmission of data.

The same procedures are following to read from a slave. To read from a slave, the master instructs the slave of which register it wishes to read from. Once this register is acknowledged, then the master will send a start condition again. The slave will acknowledge the read request, and the master will release its SDA bus, as well as continue supplying the clock to the slave. The master will come the master receiver and the slave will become the slave transmitter. Clock pulses will continue to be sent by the master, with the SDA line released, so the slave device can continue to transmit data. The stop condition will occur once the expected number of bytes has been received by the master.

## 5.3 SPI

SPI, Serial Peripheral Interface is a synchronous serial data protocol used by microcontrollers for communicating with one or more peripheral devices quickly over short distances. SPI can also be used to communicate between two microcontrollers. The most common SPI connection involves a master device which controls the peripheral devices. SPI has three lines common for all devices: MISO, MOSI, SCK, and then there is the SS.

* The MISO, or Master In Slave Out, refers to the slave line that sends data to the master.
* The MOSI, or Master Out Slave in, refers to the master sending data to other devices.
* The SCK, or Serial Clock, is used to synchronize all the devices in the SPI connection.
* The SS, or Slave Select, is the select line that turns on or turns off slave devices in the connection.

The way SPI is implemented may differ from device to device, so it is important to closely follow what is listed on the device’s datasheet. Code needs to written specifically for the type of device you currently have available.

## 5.4 UART

UART, or the Universal Asynchronous Receiver/Transmitter, is based on the industry standard TL16C550 asynchronous communications element. Serial-to-parallel conversions on data received from a peripheral device is performed by the the UART and parallel-to-serial conversion on data received from the CPU.

The UART is used to transfer bytes of data and send the individual bits in the same order as it was received. Asynchronous and synchronous are the two types of transmission supported by the UART. Hardware wise, the UART is a block of circuitry responsible for integrating serial communication. In other words, the job of the UART is to be an intermediary between parallel and serial interfaces. Usually, the UART is a bus composed or about eight data lines and control pins, while the the other two serial wires are denoted as RX and TX.  When the UART is transmitting data, the UART must create a data packet and sending this data packet out of the TX line at a specific time. This timing is based on the set baud rate. When the UART is receiving data, the baud rate of the UART determines how the UART samples the data from the RX line. Overall, the UART is useful for interacting with input/output devices connected to a microcontroller. Most microcontrollers have the UART implemented as a feature and is it important as some devices can only communicate through an UART. This is the only way data can be transferred and received when the UART is the only available communication device.

## 5.5 Android Development Guidelines

The goal of the Android development guideline is to ensure that proper coding procedures are followed and all the criteria for good Java development is considered.

A great starting place to understand proper Java development are the strict rules created for Android contributors. These rules are based on how a contributor should write Java code and how a contributor should not write their Android Java code. The instructions offers bad coding practice as well as good coding practice that a contributor to Android is required to follow.

Listed below are some of the Android Java language rules for contributors:

* Don’t ignore Exceptions
  + It is important to not ignore error checking when writing code where an exception might occur. For example, if you are parsing a string into an integer value, a few things need to be considered. Whether the string is actually a number. Even if the string is a number, it is crucial to check whether an overflow might occur if the value is either too small or too large to hold in a 32-bit variable. Proper exception handling lets the user know in a friendly manner that something has gone wrong, so that it can be fixed by reporting the issue. It is also helpful when you need to debug your program.
* Don’t Catch Generic Exception
  + It is helpful to specify the type of exception a specific problem has caused instead of simple denoting as a general exception. Catching generic exceptions can be very dangerous because Exceptions you never expected get caught in application-level error handling. Using generic exceptions in the code obscures the failure handling properties of the code. It makes it impossible for the compiler to help when it doesn’t understand what type of error it really is.
* Don’t use Finalizers
  + Finalizers are useful when a chunk of code needs to be executed when an object is garbage collected. Although it is useful, it is unknown when exactly a finalizer will be called in the code. Android does not use finalizers, but can still be implemented if needed.
* Fully Qualify imports
  + Always explicitly denote the full import statement of your library instead of importing the whole library relevant to the classes you require in your code.
* Use Spaces for indentation
  + Android contributors use four (4) space indents for blocks and never tabs. Eight (8) space indents for line wraps, including function calls and assignments.
* Field Naming Conventions
  + Non-public, non-static field names start with m.
  + Static field names start with s.
  + Other fields start with a lowercase letter.
  + Public static final fields (constants) are
  + ALL\_CAPS\_WITH\_UNDERSCORES’
* Use Standard Brace Style
  + Braces do not go on their own line; they go on the same line as the code before them.
* Limit Line Length
  + Each line of text in the code should be at most 100 characters long. Anything exceeding this limit is required to be continued on the next line.

These are some of the rules required for Android development and can be seen as a standard for some of the proper Java convection and development.

## 5.6 OBD-II

The OBD, also known as the on-board diagnostics, usually refers to a vehicle’s tool for self-diagnostic and reporting capability. It is used as a gateway for diagnostic information amongst the various vehicle subsystems. Implementations of the OBD may vary between vehicles but, nowadays, the OBD must be able to act as a standardized digital communication port, which provides real time data from the vehicle. The OBD port should also provide diagnostic trouble codes, known as DTCs, allowing technicians to quickly identify and fix errors found in the vehicle. The DTCs are kept under a standard to help manufacturers produce the same diagnostic tool for every vehicle. The Data Link Connector (DLC) is used to access the OBD for vehicles, trucks, and motorcycles. It is necessary to interface the DLC with a scan tool and the known codes of a given vehicle. For vehicles made after 2008, the DLC will connect with the vehicle’s Controller-area network instead of the OBD directly. By sending a PID, or On-board diagnostics parameter IDs through to the DLC and its scan tool, to the CAN or OBD, data can be retrieved about a vehicle’s diagnostic depending on the PID sent. For the latest OBD-II standard, the SAE J1979, there are ten different modes of operation. The first three being, show current data, show freeze frame data, and show stored diagnostic trouble codes. Each mode of operation has their own corresponding PID that corresponds to different types of data. For example, for the first mode of operation, for the standard defined by SAE J1979, there are more than 30 PIDs that can be sent through the DLC to the CAN or OBD to retrieve information from. Among these PIDs are engine RPM, vehicle speed, timing advance, throttle position, etc. Some manufacturers, while implementing the required PIDs, have the option of implementing their own custom PIDs. Some of these custom PIDs are released to the public while others are kept hidden. These hidden PIDs are known as non-standard PIDs.

The SAE J1962 standard of the OBD-II defines the specification for two standardized interfaces, called type A and type B. Both type A and type B are female, 16 pin (2x8), D-shaped connectors, both with a groove between the two rows of pins. The type A connector, the type of connector we will be using, supplies a vehicle with 12V, while type B supplies the vehicle with 24 voltage. As a standard, the OBD-II connector is required to be within 2 feet of the steering wheel. Manufacturers have the option of implementing among five signaling protocols, the only ones permitted with the OBD-II interface. For example, one of the five protocols, the SAE J1850 VPW, known as, variable pulse width is the standard of General Motors. This standard includes a high voltage of +5 V, a message length restriction of 12 bytes, and a multi-master arbitration scheme.

## 5.8 RS-232

In order to establish the connection between the OBD-II IC and the OBD-II connector on the user’s vehicle we will be utilizing an RS232 interface. This interface will be responsible for establishing the serial connection present in the vehicle. In this standard user data is sent as a time-series of bits. As this is a form of serial communication it is similar to UART. Neither of these devices will serve as a slave or master, and the clock must be defined on both devices in order for the communication to be successful. Communication using this ranges from +3 to +15 volts. For data transmission lines logic 1 is defined as a negative voltage. Logic 0 is positive and the signal condition is termed “space”.

## 5.9 git

Git is free and open source version control system. We will be able to host our own git server on a virtual machine or we will be able to create a repository on a website such as Github or BitBucket. Utilizing git will allow us to collaborate on the paper and keep track of changes. In addition, this will give us a centralized place to work on the paper, so in the event of an accident the paper will still be intact. We will also be utilizing version control for the microcontroller software development process.

# 6 Hardware and Software Design

The main hardware component of HARAMBE is the Main Module which contains the Main Microcontroller Unit, the IMU unit, the OBD-II interface, and the internal Bluetooth communication. This section describes the decision making process that took place within the group and provides sub circuit diagrams for each.

## 6.1 Microcontroller

In order for H.A.R.A.M.B.E. to operate properly a microcontroller will need to be integrated into the design of the main module. The microcontroller will be receiving all of the external data from the IMU sensors and the OBD-II interface. As such a significant amount of consideration was be taken when chose the MCU to make sure that it is fast enough to handle and parse the incoming data as well as perform computations and send data to a mobile device over Bluetooth. It is crucial that the microcontroller does not fail. If the microcontroller fails, the system and the mobile device will not be able to determine if the user has gotten into an accident, and subsequently, they will not be able to get help.

### 6.1.1 Microcontroller Choice

Architectures differ between different families of microcontrollers and as such they are optimized for different use cases like needing a faster clock speed, having increased power efficiency, and passing different types of standards. In our case, we wanted to choose a microcontroller that is power efficient, is powerful enough to manage a bluetooth signal while also gathering information from other submodules, and a microcontroller that is easy to develop in.

Once we understood the required architecture specifications that our project required, we decided to take a look at different design architectures. For the Bluetooth system in our project, we had the option in either separating the microcontroller and the Bluetooth module or choosing a microcontroller with a built in Bluetooth module. For the case of a microcontroller with a separate Bluetooth module, the microcontroller would handle gathering and analyzing the data received from the sensors. The Bluetooth module would simply be used as the communication device between the microcontroller and the mobile application. If the microcontroller detects an accident it will send a signal to the bluetooth module. The Bluetooth module will then send the signal from the microcontroller to the mobile application. In this architecture model we would have to account for powering both modules and incorporating them into the PCB layout. In addition, we would have to build a library to communicate with the bluetooth module. If we were to go down this route, we have a plethora of choices to choose from for both the microcontroller and the Bluetooth module. One of the most common choices is the ATMega family of microcontrollers which are most notably found in the popular Arduino development boards. These development boards have a wide appeal due to their user-friendly development environment and abundance of libraries that support many different submodules. One of the microcontrollers we specifically were looking at was the ATMega 328p which is used on the Arduino UNO board. This board contains 32 KB of flash memory, two SPI connections, one UART connection, and one I2C connection. This type of microcontroller completely satisfies our architecture specifications. The ATMega 328p microcontroller is widely used and so, it contains many different software libraries which we can use to speed up the software development in our project. For example, a library exists that can help us to configure and initialize a Bluetooth module with a few lines of code. Being able to utilize these libraries would significantly decrease development time and would increase the amount of functionality we would be able to introduce into the system. Another advantage is that we would be able to flash an ATMega328p with an Arduino bootloader which would allow us to program using the user-friendly Arduino development environment.

Another popular choice for microcontrollers is the TI MSP430 family. Most of our group members are familiar with this family of microcontrollers because we have taken Embedded Systems at UCF which solely focuses on this family of microcontrollers. More specifically, we looked at the TI MSP430FR2433. This microcontroller contains two UART connections, two SPI connections, and one I2C connection which will be suitable for our application. In addition this device contains 15.5 KB of non-volatile memory which will be used for program storage. In addition to being familiar with the TI MSP430 family of microcontrollers. The group is also familiar with the development environment used to program these microcontrollers. Another advantage of using this microcontroller is that it was designed with power efficiency in mind. At the absolute maximum load this microcontroller will only use 3.4uA of current at 3.3V. We are not putting a whole lot of consideration into power efficiency into our project as we will be drawing power from a car battery. The car battery is able to support multiple amps at 12V. In comparison our system will be drawing such a small amount of power. A major disadvantage of using this platform is that the user community is much smaller for these chips as they are not widely used in online communities. Most of the time people will gravitate toward a platform that has more user support such as the ATMega family of microcontrollers. Since this is the case these users will continue to support that platform and draw even more users. We are well aware of the limited audience in the MSP430 user community. This will require us to spend considerable time implementing configuring parameters in the software that is already done in the ATMega microcontrollers as libraries. If a developer for this platform is in need of help they can only turn to the TI forums which are not as comprehensive as other competitors’ forums. The user will be forced to take a closer look at the microcontroller’s datasheet. Even with the understandable failures that comes with using a TI microcontroller. Ti’s documentation is known for being comprehensive to help developers find solutions to problems that might arise when working with a microcontroller. As said before, another disadvantage is the lack of library support for the necessary submodules that we will be integrating into our project. The TI microcontrollers do not have pre-written libraries for our submodules, such as Bluetooth, OBD-II IC, and accelerometers. The lack of pre-written libraries will increase the development time and decrease the amount of functionality we will be able to implement into our device. When compared to the AtMega family of microcontrollers. They have a large amount of support from the open-source community. These community creates open source libraries that are free to use and greatly speeds up software development.

One problem that can negatively affect these microcontrollers is program concurrency. In the event of a car accident the system would need to respond as quickly as possible meaning the microcontroller would need to gather data from the sensors as quickly and efficiently as possible. Since these microcontrollers do not support concurrency through traditional means seen in larger microcontrollers, it can only rely on the speed of the CPU to offset this. Another way this lack of concurrency can be mitigated is through interrupts. If all of the submodules are able to support triggering interrupts then the microcontroller would be able to service the submodules based when the interrupt was triggered. The priorities of the interrupts could also be set in order to give other submodules the opportunity to get their data processes first. This idea is very good in getting the appropriate data quickly, but it starts to break if submodules do not support interrupt functionality. If we were to use this design architecture then the microcontroller would be required to constantly poll the state of the sensors to receive the necessary data. The problem is that this would involve having to constantly re-initiate connections between the microcontroller and submodules. The burden of time loss plays a huge role on why we do not want to be continuously re-initiating connections in our microcontroller. A better design architecture is to use a microcontroller that has multitasking functionality. Multitasking functionality will allow different threads to perform different jobs allowing for faster polling of the sensors. There will still be a burden of time with multitasking but this loss is greatly alleviated. The main problem of multitasking is the complexity involved. Implementing multitasking will require intricate software development and a lot of effort.

Another disadvantage of this architecture is that the Bluetooth module is completely separate from the microcontroller. The microcontroller will only be communicating with the Bluetooth module over UART. This means it will only be receiving data from a data buffer on the module. The Bluetooth setup will be abstracted away from the microcontroller which would simplify the development process. However, this architecture doesn’t allow for further introspection into changing how the Bluetooth connection will be used to speak with the Android application. Also, utilizing this architecture will leave our entire system at the mercy of the Bluetooth module when notifying emergency services. If the Bluetooth module malfunctions or is unable to communicate with the mobile device the system will be rendered useless. If we were able to have more control over the Bluetooth module, such as having it physically built into the microcontroller we would have a better idea of the state of the system and whether it is functioning or not. Furthermore, if the Bluetooth module was already built into the microcontroller, we could easily develop a way to fix the issue. Relying on another manufacturer’s implementation of Bluetooth profiles and communication does not seem wise on a project like ours. One other major disadvantage of using this system is security. When relying on a third-party Bluetooth module we will have to utilize the security measures used onboard when communicating with another device. Security is a major concern in the development of this module because if a hacker is able to hijack or block the communication between the main module and the mobile device. The user may not be able to get the help that they need in the event of an accident. The hacker also has the option of sending wrong data to the mobile application. This might cause the mobile application to constantly prompt the user that he has gotten into a car crash. In the worst case, emergency services might be notified for no apparent reason. Security considerations will be further detailed in the Bluetooth section of this document. Overall the disadvantages of this architecture have led us to choose one that is more reliable.

The architecture we decided to implement is one in which the Bluetooth module is integrated onto the microcontroller die which is more commonly known as a System-on-Chip (SoC). With the use of the System-on-chip architecture, we hope to be able to achieve a higher level of introspection on how the Bluetooth module operates. We would also like a higher level of control over the communication between the mobile application and the main module. Another benefit that is included in the System-on-chip architecture is the use of a real-time operating system (RTOS). The RTOS will be very useful in our design because our system depends wholly on being able to identify a car crash or not. The RTOS will give us more leeway in developing the software that will be detecting an accident. Specifically, the main goal of using an RTOS is the fact that the RTOS gives us the ability to program enforceable deadlines for execution time on programs that are currently running in the operating system. For example, if a program was unable to finish executing within a certain amount of time the system will hard reset due to the operating system noticing a failure in the program’s execution. In comparison, regular operating systems do not have enforceable deadlines, they simply schedule processes for certain blocks of execution time called quantum. Then based on the scheduling algorithm being used and the priority of the process these processes will be able to utilize this quantum. For a regular operating system there are no enforceable deadlines, so in the event of an accident the system would not be accountable for not getting an execution done in time as this is not their intended purpose. Through the use of an RTOS we aim to have the system respond within a certifiable amount of time. This will allow us to increase the speed at which the signal is sent to the mobile application. We wish to develop software that is efficient and powerful, so that the system stays reliable at all times. If the system is reliable, then it is guaranteed to function properly at all times.

Another advantage of using an RTOS is that it supports multitasking functionality. This means that we will be able to create separate processes for separate jobs. One process can read data from the OBD-II IC while another process can be used to receive data from the accelerometer. We have the option of scheduling these processes at any time.  Scheduling will give us the ability to create a rudimentary version of multicasting. With the multitasking functionality we will be able to achieve “concurrently”. What this means is that we will able to gather data from the sensors and the OBD-II IC and make decisions based on the data collected all at the same time. In addition to having these processes running concurrently we will be able to set priorities for these processes. In the event of an accident communication with the internal Bluetooth controller will take the highest priority.

By using the System-on-Chip architecture we realize that there will be significantly smaller amount of options we can choose from, however we feel that the benefits outweigh these disadvantages. We wish to make our system safe and efficient. As we continued our research on TI microcontrollers, we came to the end conclusion of simply using a wireless MCU. This wireless MCU would follow the System-on-Chip architecture we wanted to use. After looking through the TI available inventory we found the TI CC2XXX line of SoCs. More specifically, we took a look at the TI CC2640. One of the main advantages of using this chip is that the Bluetooth module is integrated into the SoC. We would not have to worry about implementing the communication channel between the Bluetooth module and the microcontroller inside. Another advantage of using this chip is that the internal Bluetooth module is controlled by an ARM Cortex M0 chip. The chip is solely dedicated to it meaning to perform computations so that the sensor data communication will not interfere with the Bluetooth communication. The addition of the M0 also allows for decreased power consumption. When the main internal microprocessor is in a power saving mode the M0 can still perform its necessary computations without drawing nearly as much power. In addition to containing a Cortex M0, the processing power from this SoC comes from the integrated ARM Cortex M3 processor. The ARM architecture is very popular and is used in almost every mobile device on the market. The ARM instruction set is widely known and thus there are a significant amount of resources available online if assistance is needed. In addition, the usage of ARM in this SoC allows third-party software developers to build development environments for this platform that allow for debugging since this is an open platform to some extent. So instead of just having development environments from TI, there are third party development environment developers who include support for this chip.

Another advantage of using the CC2640 is that it contains an AES-128 security module and a True Random Number Generator. These components will allow the system to remain secure by encrypting the data that is being sent between the phone and the main module, and the communication channel. In addition to being secure the CC2640 is very power efficient. When the device is not communicating and the device is running at 100% CPU usage it only draws 1.45mA of current. When the device is on standby it will only use 1 uA of current. Another large advantage of using this SoC is that it supports the use of an RTOS, more specifically TI-RTOS. By using TI-RTOS it will eliminate the need to write and maintain system software such as schedulers, protocol stacks, and drivers. This operating system combines a real-time multitasking kernel with TCP/IP and USB Stacks, a FAT file system, and device drivers. By utilizing this operating system in our design we will be able to have enforceable deadlines on process execution which will in turn ensure the safety of the driver and the frequency and reliability of emergency service response in the event of an accident.

While researching for another SoC that met our needs we were able to find the STMicroelectronics BlueNRG-1. The advantage of this chip is that it contains the Bluetooth Low Energy controller on the die itself so we will not have to interfere or implement the communication channel between the onboard Cortex M0 and the Bluetooth controller. In addition this device contains a secure Bluetooth low energy stack in order for the communications between the mobile device and the main module. In addition this SoC is power efficient, utilizing only 15.1mA at its maximum power draw and as little as 0.9 uA when in sleep mode. One of the disadvantages of this SoC is that it does technically support the use of an RTOS as it is not present in the documentation. For the purposes of our project we prefer to use an RTOS.

Throughout the course of our analysis we weighed the positives and the negatives of each architecture and each corresponding chip within those architectures. In the end we decided to choose the TI CC2640 as our primary choice as the chip for our main module.  A block diagram for the C2640 is shown in the figure below.

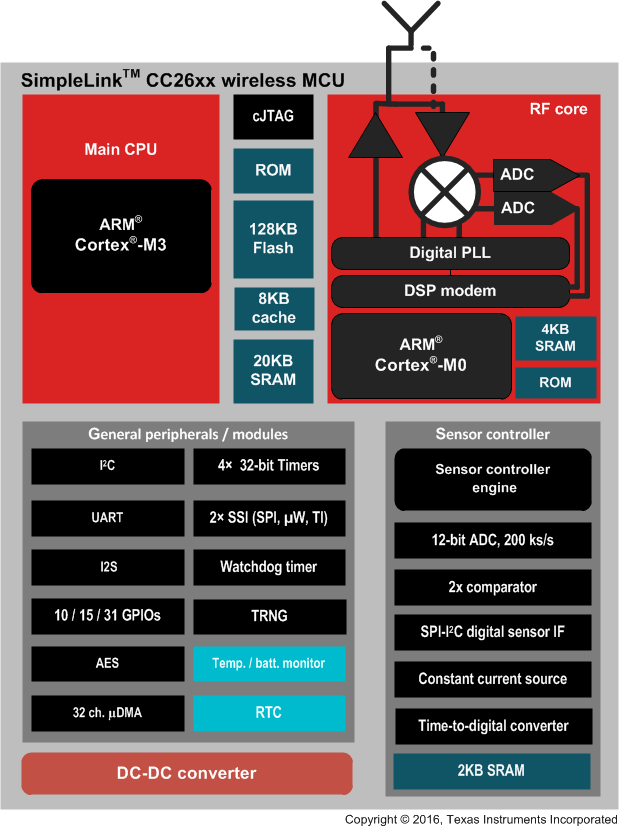


Figure : CC26XX Functional Block Diagram (Courtesy of Texas Instruments)

### 6.1.2 Hardware Considerations

One of the downsides of using the TI CC2640 is that we would have to implement our own antenna for the design. As we would like to save time during the development process we decided to take a look at pre-packaged solutions that already had an antenna. The reason we decided to obtain a product with a built in antenna is because it would make the debugging and testing phase much easier. For example, if the antenna is working properly, but the Bluetooth module is not. We can pinpoint the cause of the error to be in the software. After considering our options and doing further research, we discovered the Sable-X BLE module already built in the TI CC2640. In addition to having a Bluetooth module, the TI CC2640 also contains a built in antenna.

### 6.1.3 Development Environment

To begin using our chosen microcontroller, it is also important to choose the right development environment for the job. The manufacturer of the microcontroller has information on their website which states the two development environments that can be used for the microcontroller. These two development environments are TI’s own Code Composer Studio and the IAR Embedded Workbench.

We considered several aspects when the decision of choosing a development environment came into fruition. The first aspect we considered when choosing a development environment was the cost associated with the environment. Due to our limited amount of funds, we cannot afford to spend money on expensive development licenses. Furthermore, since each member of the group would require a license, this would multiply the costs significantly. The same development environment will be required because we intend to use GitHub as the source of version control. If we were to use different development environments, this would cause project file conflicts which is annoying to deal with. The developer document of the CC2640 microcontroller states that the full-featured version of IAR is required to begin software development of the microcontroller. In addition to having to buy the full-featured version of IAR, they state that the size-limited Kickstart evaluation option is not compatible with the SDK. According to embedded development forums we could be expected to pay around $2000 for a license. This amount is far beyond the funds we are willing to spend for a license. According to TI’s website, to obtain a license of their development environment, the cost for a single license is only $445. In addition to this, the license is included with the purchase of a development kit. We decided to take advantage of this offer and bought a $31 development kit from TI to obtain the license at a lesser price. In terms of cost TI’s Code Composer Studio is advantageous. IAR workbench offers a variety of features, including but not limited to, an efficient compiler, power debugging, multicore debugging, hardware debugging support, C-Spy, and an instruction simulator.

The second aspect we considered when choosing a development environment was its included features. IAR workbench offers a variety of features, including but not limited to, an efficient compiler, power debugging, multicore debugging, hardware debugging support, C-Spy, and an instruction simulator. If we simply consider the power efficiency testing of both environments, the power debugging function of IAR is extremely useful. The functionality will allow us to sample the system’s power draw and calculate the power usage of the code. It will allow the developer to locate areas in the code that require optimization due to its power consumption. IAR claims that their proprietary compiler is considered one of the best in the industry due to its support of excellent code optimization. Using this compiler would theoretically allow us to reduce the size of our binary which would allow for more functionality to be implemented compared to alternative compilers. The IAR development environment also has a unique functionality called C-Spy, which helps simplify the debugging process of software development. The C-Spy functionality will speed up the debugging process as we will be able to narrow down the locations in the source code where the problem might be occurring. We deem this functionality as unnecessary, for our software development needs, because we will not be dealing with complex code and thus won’t be using the debugging function too much.

The competing development environment that we have the option of using, TI Code Composer Studio, also offers a variety of features. This features include but are not limited to, an instrumentation trace module, a code optimization assistant, and an optimized C compiler. A unique feature of this development environment is the instrumentation trace module. The instrumentation trace module allows the user to see a historical account of code execution, timing and data accesses. This provides a higher amount of introspection into how the system is working, which will allow us to identify areas where we can optimize the code when we are debugging. This feature is also useful for detecting intermittent bugs as well as profiling to improve code performance. In addition to all of these features TI Code Composer Studio was built from the ground up to be of use for their own chips. This means that Code Composer Studio contains more chip-specific support out of the box. Overall, in terms of features we feel that TI Code Composer studio is the best option.

In the end we decided to choose TI’s Code Composer Studio due to its competitive price, variety of features, and our desire to stay within TI’s ecosystem.

Table : Embedded Development Environment Comparison

|  |  |  |
| --- | --- | --- |
| Development Environment | IAR Workbench | TI Code Composer Studio |
| Cost | $2000 | $0 for basic version |
| Guaranteed support from TI | No | Yes |
| Ability to import sample projects from the cloud | No | Yes |
| Code Optimization | Yes | Yes |
| Built-in debugger | Yes | Yes |
| C-level debugging | Yes | No |

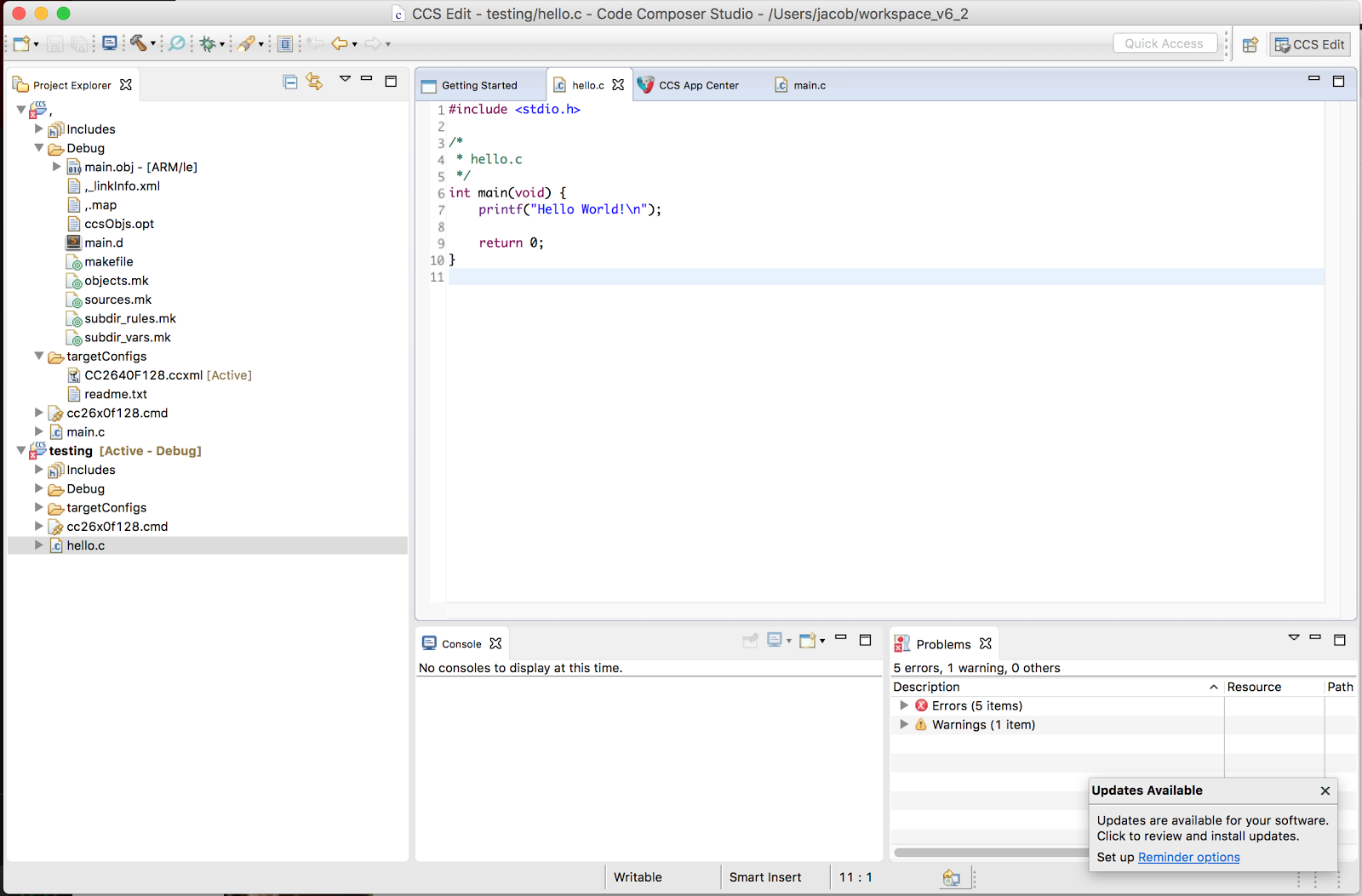


Figure : Photo of Code Composer Studio Version 6

## 6.2 Bluetooth Transceiver

The purpose of integrating Bluetooth into our design is because we need the main module to be able to communicate with a mobile device in the event that the user gets into an accident. In addition, we want to be able to send vehicle data to the mobile device for the purpose of data analysis. Utilizing another wireless standard such as Wi-Fi or ZigBee would reduce the audience of our product since most mobile devices communicate to peripherals through the Bluetooth protocol. We had the option of choosing between many different Bluetooth chips. We narrowed down the Bluetooth modules to the classic Bluetooth and the Bluetooth Low Energy.

Though the throughput of classic Bluetooth is higher mainly due to its increased power, we realized that in our design we will only need the amount of throughput present in the BLE standard. In addition, the range for classic Bluetooth was overkill for our design as well as the number of active slaves. The main disadvantage of classic Bluetooth is that the latency from a non-connected state is 100ms, while for Bluetooth low energy, it is only 6ms. Table 1 shows a comparison between classic Bluetooth and Bluetooth Low Energy.

|  |  |  |
| --- | --- | --- |
| Technical Specification | Classic Bluetooth | Bluetooth Low Energy |
| Range | 100m | >100m |
| Data Rate | 1-3 Mbit/s | 1 Mbits/s |
| Application throughput | 0.7-2.1Mbit/s | 0.27 Mbits/s |
| Latency from non-connected state | 100ms | 6ms |
| Voice capable | Yes | No |
| Power Consumption | 1W | 0.01 - 0.5W |

Table : Technical Comparison between Classic Bluetooth and Bluetooth Low Energy

Overall since we are sending such a small amount of data and we intend for our design to be as power efficient as possible we decided to utilize Bluetooth Low Energy. As we are using the TI CC2640 the Bluetooth module is already integrated into the SoC, so it would make sense to utilize the on-board module. One of the most important factors in choosing the CC2640 was its security capabilities when it came to the Bluetooth module. Having a secure connection between the phone and the main module is crucial as any interruption from a third party could put the user’s life in danger. When comparing the different design architectures for the main module, the first that we considered was one where the microcontroller and bluetooth module were in separate packages. Upon searching for modules we came across the following options listed below.

### 6.2.1 Nordic Semiconductor nrf8001

The nrf8001 is a very popular chip which is found in many development kits for Bluetooth Low Energy development. One of the features that draws people to this chip is its simplicity of integration. It is able to communicate with microcontroller using only a serial connection meaning a multitude of microcontrollers would be able to support this type of connection. In addition, this chip was specifically designed for chips in the peripheral role, meaning they advertise that they have data that can be collected. This would directly apply to the main module since its primary role is to send data to the mobile device. Another appealing feature of this chip is its low power consumption. Overall, the peak current of the device which occurs when the receiver is active is 14.6 mA.  One of the major disadvantages of using this chip is the security measures that are implemented which are Just Works and Passkey, both of which are susceptible to man in the middle attacks. These security measures will be described in more detail in the security constraints section. Another disadvantage of using this chip is that in order to change the hardware and protocol parameters we must use nRFgo studio which would add another layer of complexity to our design.

### 6.2.3 Operating Principles and Usability of CC2640

After deciding against the initial architecture where the microcontroller and the Bluetooth module were in separate packages we narrowed our focus onto the TI CC2640. This IC utilizes an ARM Cortex M0 to control the RF core. This interfaces the analog RF and baseband circuitries, handles data to and from the system-side (Cortex M3), and assembles the information bits in a given packet structure. One of the main advantages of the RF core is that it offers a high level, command-based API to the main CPU which will decrease development time for the main module’s software.

The RF core is also responsible for automatically handling the time-critical aspects of the radio protocols, thus offloading the main CPU and leaving more resources for the user application.

### 6.2.4 Security Constraints

Security is a large concern for this project because if the integrity of the system is compromised the user will suffer from getting his data stolen. Bluetooth security has recently undergone a number of improvements. In the Bluetooth 4.0 and 4.1 specifications they utilized the Simple Pairing Model, which is now referred to as “LE Legacy”. This model utilized three methods for authentication: Just Works, Passkey Entry, and OOB pairing. “Just Works” does not involve invoking any action on the device in order to pair. The device simply advertises and connects to the first device that initiates the connection. In order to defeat this security an attacker would simply have to approach the device and attempt to connect to it. In order to fool the user’s mobile device the attacker can create a “clone” of the device and trick the mobile phone application to connect to the phone. If this attack were possible on our system the user’s mobile device would not receive a signal in the event that an accident occurs. This attack serves as a denial-of-service and a spoofing attack. Another scenario is that the attacker acts as the main module and triggers the user’s phone, making it think that an accident has occurred. At the very least this would act as an annoyance to the user, and at the worst public resources could be wasted on sending emergency services out on a fake call.

The second security method that is widely used is “Passkey Entry”. This involves the user having to enter a passkey on both the Bluetooth device and their mobile device. An attacker could not directly pair with the device, however they would be able to get the user to re-initiate the pairing process. Next, the attacker would be able to clone the device with its MAC address, the application will try to connect with the cloned device and the keys will not match. The attacker will then turn off their cloned device and the user will reconnect with the original device. This time the attacker will be able to eavesdrop on the Bluetooth communication. Because the attacker has eavesdropped they will be able to determine the Long Term Key, which is used to detect the pattern of frequencies the device is transmitting on. This will allow the attacker to continue with active interception of the data transferred by the modules. This will allow them to act as the original device and provide false information.

The third security method that was used for Bluetooth security was “Out of Band Pairing”. Instead of sharing the secret keys over the 2.4 GHz band used by Bluetooth they are exchanged over different mediums such as Near Field Communication. Once these keys are exchanged it will encrypt the channel to ensure that the highest level of security is in use. This attack is only susceptible to attack if the attacker gets a hold of the secret keys, which are usually 128-bit. The only issue with this pairing method is that it isn’t widely adopted as it requires multiple steps to initiate a connection.

The introduction of the Bluetooth 4.2 specification greatly enhanced the level of security by introducing a numeric comparison method using the Elliptical Curve Diffie-Hellman algorithm. This algorithm involves securely exchanging cryptographic keys over a public channel, which in this case is the space between the devices. Once the keys have been established between both parties they will be able to encrypt their communications. If the attacker wanted to eavesdrop on this communication they would have to brute-force a 128-bit key, which is utilized in the Bluetooth 4.2 specification and would require the testing of up to 2^128 keys. The idea is that by establishing these keys, anyone who is trying to intercept the communication will have to figure out the key, which is computationally expensive, and will likely take longer than the length of communication was. This method is only susceptible to those that have a large amount of resources such as the government or military. A graphical representation of the Diffie-Hellman key exchange process is shown in Figure 4.

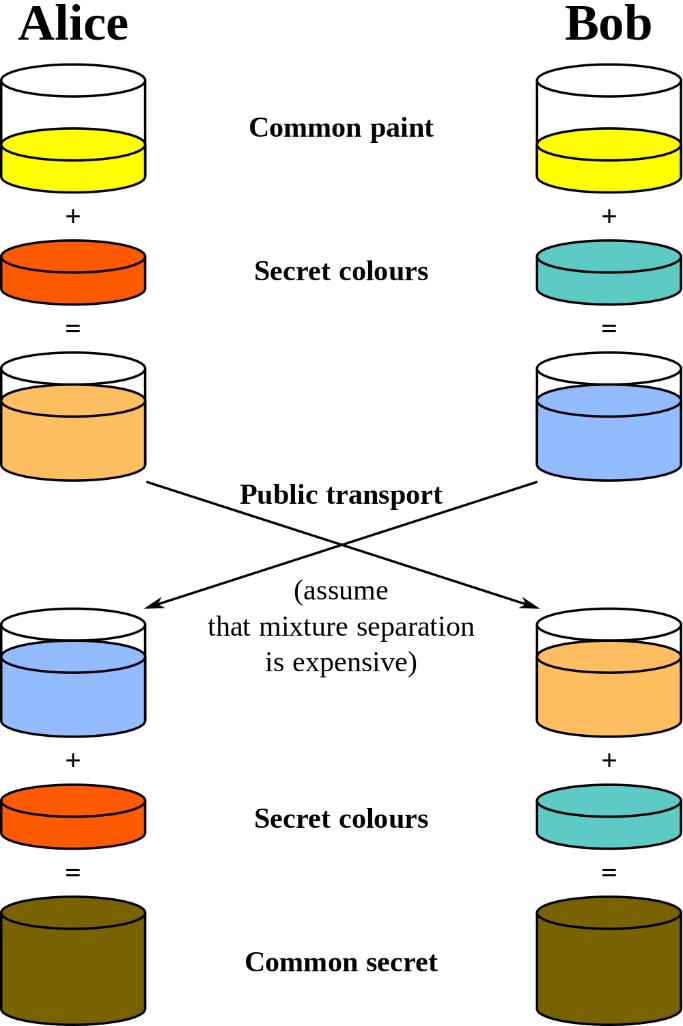


Figure : Graphical Representation of the Diffie-Hellman Key Exchange Process

Many options that were considered for the project did not conform to the Bluetooth 4.2 standard such as the Nordic Semiconductor NRF8001. Since security is such a large constraint for this project, we decided to utilize the TI CC2640 which conforms to the Bluetooth 4.2 standard, and thus utilizes this key exchange.

## 6.3 Mobile Application

The mobile application will prompt the user to check whether they have gotten into an accident or not. If the user has gotten into a car accident, the mobile application will then contact emergency services on their behalf.

In order for the functionality of HARAMBE to be realized completely there must be a mobile application that is able to connect with the main module. The mobile application will prompt the user with a popup message to check whether they have gotten into an accident or not. If the user has gotten into a car accident, the mobile application will then contact emergency services on their behalf. When we set out to develop this application we wanted to make something that was simple and to the point. We didn’t want the user to be confused by the interface because they might be using the mobile application in one of the most dangerous situations of their life. Usability and stability were emphasized in the development of this application.

### 6.3.1 Platform Choice / Development Environment

Android is the mobile operating system that we have chosen to utilize for the development of the companion application for the main module of HARAMBE. We decided to choose Android because of its 87.6% market share as reported by the IDC. The development environment for Android has many useful functionalities such as having the ability to create Android emulators to test your code. Android can also be developed for any other platform. In comparison, developing for iOS requires that developers have Mac computers and have purchased an iOS development license which costs $99 per year and is limited to 100 devices unless an enterprise license is purchased which is much more costly.  Another advantage of developing for Android is that the operating system and all of its applications run in the Dalvik virtual machine, meaning that if Android devices have a different architecture the Dalvik binaries or Android applications will run on it. Application portability is what makes the Android operating system unique and allows for such a wide range of applications to be developed for it.

With our target platform having been selected we needed to research what the best environment for developing this application would be. There is only one officially supported development environment for Android which is Android Studio. Since this is the only officially supported development environment and there is a large user community for it, we have decided to utilize this environment.

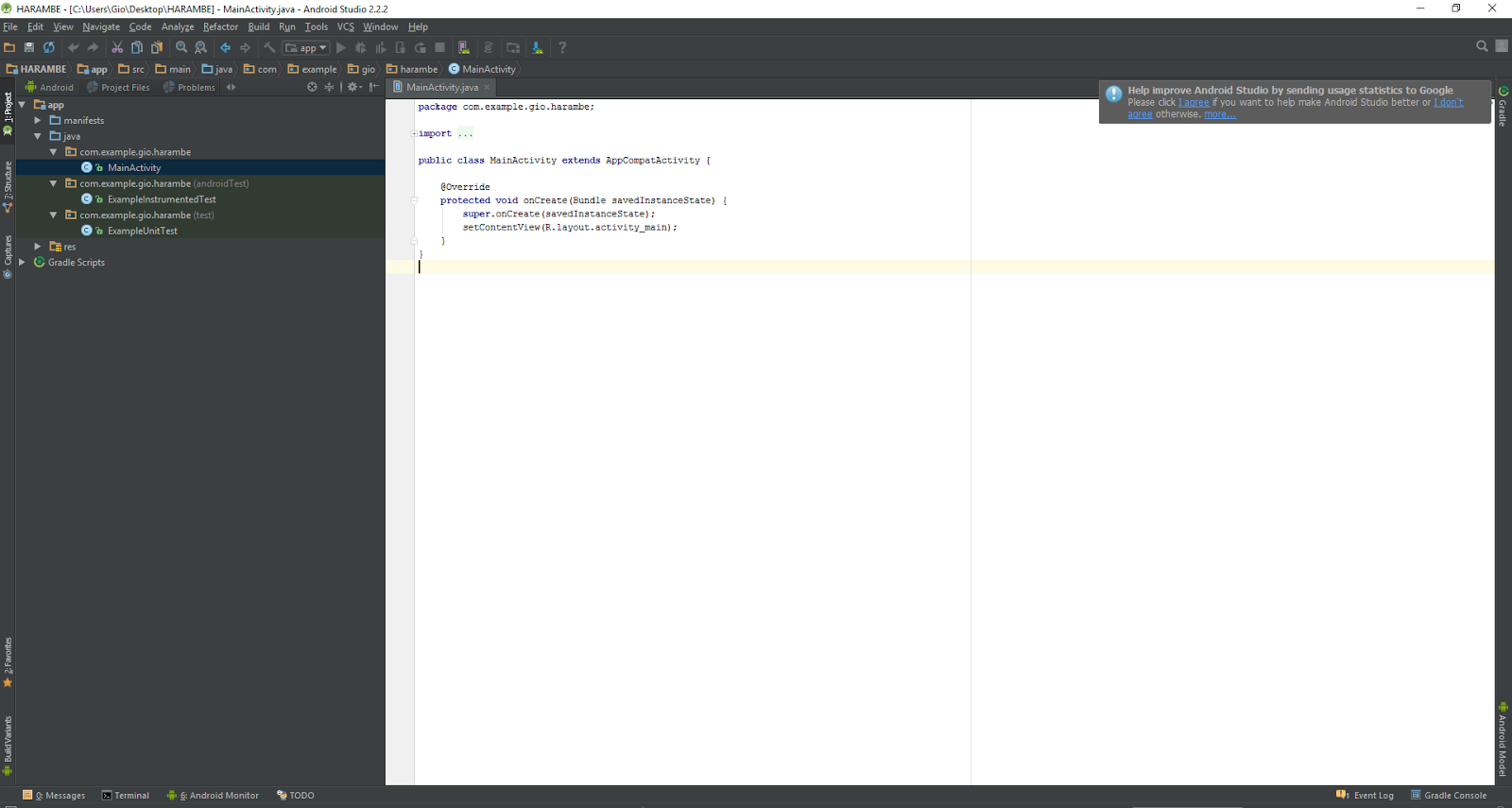


Figure : Android Studio’s Development Environment

### 6.3.2 Use Case

The primary use case for HARAMBE is in its ability to detect whether the user has gotten to an accident. HARAMBE will then forward an alert to a mobile device. On the mobile device the user will be prompted on whether they have gotten into an accident or are in need of assistance. If the user is in need of help, they can press “Yes” which will contact emergency services. If they are unable to operate their phone there will be a timeout functionality which will contact emergency services if the user has not responded within a certain amount of time. The main objectives in developing this mobile application are for usability and reliability. We have to make sure that if the main module detects that the user has gotten into an accident the mobile application will provide an alert. If the system fails to respond to an accident appropriately the user may succumb to fatal injuries.

The first time the user opens the application they will be presented with a user agreement which they must read and agree to before continuing using the application. If the user does not agree to the agreement, the application will close. Otherwise, they will be brought to the next step which is where they will enter their emergency contacts. These contacts will be sent a text message in the event of an accident. After the user has entered their desired emergency contacts the user will be prompted to connect the main module to their device. There will be a button on the prompt that the user can press which will take them to their Bluetooth settings. Once the user has paired the main module to their mobile device the application will bring the user back to the application. Here the system will show the checks being performed on the main module to ensure that it is in working order. These tests include gathering data about the accelerometer and the OBD-II port to make sure that they are working.

Once the testing is finished the user will simply minimize the application. In the event that the user gets into an accident the application will cause the phone to emit a loud alert and it will prompt the user to say whether they are in an accident or not. This will be shown in a prompt message on the phone application. If the user is unable to respond within 30 seconds the application will assume that the user has gotten in an accident and will contact emergency services automatically. This will trigger a phone call to take place to emergency services. The call will automatically be placed on speakerphone and they will be able to talk to the user along with getting their location data from the phone call. Figure 6 contains the use case diagram for the mobile application.

mobile_use_case_harambe (1).png

Figure : HARAMBE Mobile Application Use Case Diagram

### 6.3.3 User Interface Design

The main goal when developing the mobile application for HARAMBE is our emphasis on usability over everything else. When the user is in a life-threatening situation they should not have to spend time figuring out how the application works. The interface should be intuitive enough that anyone can respond to an emergency without having to think.

#### 6.3.3.1 Initial Setup

The goal for this portion of the application is that the user will be able to input contact information and will be presented with a user agreement. The user interface for this portion seems straightforward, we simply have to provide an interface for users to import contacts from their contacts list and we will have to present an interactive scrolling user agreement. Figures 7 and 8 show mockups of the user interface for this portion of the application.

choose_contacts (1).png

Figure : User interface mockup for selecting emergency contacts

user_agreement.png

Figure : User interface mockup for user agreement

#### 6.3.3.2 Tutorial

The goal for this portion of the application is to help the user learn how to use the application in the event that an emergency occurs. We would like to be able to present the basics of the application to the user without it being too complicated or overwhelming. The user will be presented with an example emergency alert and will be instructed to respond to the alert. Once the user has proven that they are able to respond to the alert, the tutorial will be completed.  Figure 9 shows a mockup for the tutorial interface.

tutorial.png

Figure : User interface mockup for tutorial

#### 6.3.3.3 Emergency Alert

Since this application will mainly be running in the background the user will not have much interaction with it after the setup and main tutorial. This means that the user will not have time to get accustomed to the application and will mainly only use the application in an emergency. When designing the alert that will pop up when the main module detects an emergency we wanted to make sure that the purpose and the choice the user has to make is clear. We did not want to include any extraneous features or options that would draw attention away from the event that is occurring. Though this alert will only stay on the screen for 30 seconds, it is important that the point is clear and the user understands it. Figure 10 shows a mockup for the emergency alert screen.

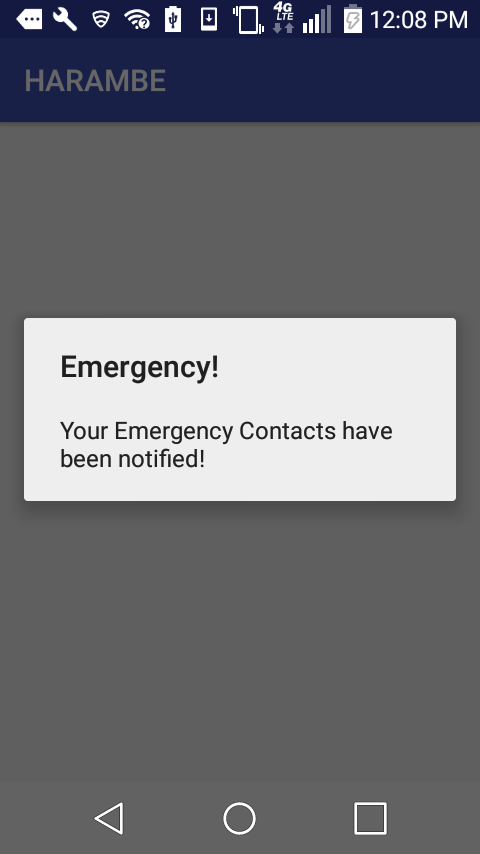


Figure : Sample emergency alert screen

## 6.4 OBD-II IC

This project will require a connection to the OBD-II interface of the vehicle in which it is installed in. The availability of a standardized connector is what allows HARAMBE to interface with a wide variety of vehicles which increases its value as a viable aftermarket solution. The OBD-II interface is valuable because we are able to gather data from the car such as the speed and the engine RPM as well as draw power from the connector since it has a 12V passthrough.

### 6.4.1 Functionality

The OBD-II IC that we will be using is the STN1110. The STN1110 chip is a multiprotocol OBD-II to UART interpreter IC. This means that it can receive information from the OBD and by acting as a UART interpreter, the STN1110 chip can convert any of these messages into the necessary information we require. Compared to the industry standard ELM327 chip set, the 16-bit processor of the STN1110 offers more features and better performance. Some of the features the STN1110 chip has are, a UART interface, secure bootloader, superior automatic detection algorithm, large memory buffer, and much more.

The primary functionality of OBD-II is to be able to provide diagnostic information about the vehicle it is connected to. The intention of this system is to give the vehicle owner or repair technician access to the status of the vehicle’s subsystems. This diagnostic information can range from speed and engine RPM to O2 sensor monitoring. The advantage of having this information is that we will be able to see changes in the vehicle’s speed which will give us insight into the situation the driver is in. For example, if the vehicle suddenly goes from 80 miles per hour to 10 miles per hour we can assume that the user has been in a dangerous situation. In addition, if a large amount of the vehicle’s subsystems fail at the same time we will be able to assume that the vehicle has been in an accident.

In the early stages of the project we wanted to be able to tell if the airbags in the vehicle had been deployed. Though this information is able to be gathered through the OBD-II interface, the codes for retrieving that information are proprietary. In fact, a majority of OBD-II PIDs are nonstandard and there is a very limited amount of overlap between manufacturers when it comes to these PIDs. This means that these nonstandard PIDs are unique between each vehicle manufacturer. If we wanted to be able to see these PIDs we would have to become a member of the Equipment and Tool Institute which would cost $5000 for annual dues. As we are unable to afford this and even if we became members of this institute we would not get access to all of the non-standard PIDs, we have decided to only use the PIDs that are a part of the standard.

### 6.4.2 Hardware Evaluation

Making a choice for OBD hardware is not a simple decision. The most important parameters for our selection of the OBD-II IC are ease of implementation, cost, number of protocols supported, and space taken up by its implementation. When it comes to implementing OBD-II functionality into our device, there are two possible routes. The first route we can take involves the purchase of an OBD-II communication IC and all of the associated circuitry on the board. The second route we can take involved using a premade OBD-II connector that already has the associated circuitry on a PCB. There are a variety of options for communicating with these devices including Bluetooth, USB, and even WiFi. The devices examined below all support having a hard-wired serial connection. This is important because HARAMBE will be powered by the OBD device.

The STN1110 chip supports all legislated OBD-II protocols, is fully compatible with the ELM327 command set, and is available in SPDIP, SOIC, and QFN-S packages. In addition this contains functionality for power saving. This feature would be useful in that we can power down our unit when the vehicle is not on. One disadvantage of this design is that we would have to implement ISO and CAN transceivers into the design along with 16 conductive lines for the OBD-II connection.

The ScanTool OBDLink SX scan tool is an example of the second route. This is an OBD-II product that connects to the OBD-II port of a vehicle and allows the user to connect the device to a computer over a USB connection. The main advantage of this platform is that we would not need to implement the CAN and ISO transceivers and we would not need to implement the connector for the OBD connection. The disadvantage of using this platform is that we would be at the mercy of the manufacturers in terms of the size of the board and the components that are used on the board.

After considering the advantages and disadvantages of the different architectures we have decided to choose the former. The constraints of our design necessitate a smaller surface area and more creative control over the layout of our design.

### 6.4.3 Hardware Selection

From the research we have conducted we have found two possible solutions for communicating with the OBD-II interface, the ELM327 and the STN1110.

The ELM327 is a multiprotocol OBD to RS232 interpreter IC which comes in DIP and SOIC packages. It contains a PIC microcontroller with custom firmware which allows it to automatically recognize nine OBD protocols. In addition it has power control which allows it to stay in a low-power standby mode. This chip provides a serial interface for a user to be able to enter AT commands to get data from the vehicle’s OBD-II interface. One of the main disadvantages of this chip is the supply voltage which is 4.5 to 5V. Since most of the components in our design are using 3.3V, we would like to have all of our components using that supply voltage. Having the same supply voltage will simplify our design and will prevent us from having to implement more regulators. Another disadvantage of this chip is that it does not have a large amount of RAM, meaning if multiple OBD messages come in at once, it will have a “BUFFER FULL” error. When this error occurs data can be lost.

Another competing chip is the STN1110 which is an OBD to UART interpreter IC that comes in DIP, SOIC, and QFN packages. This device fully supports the ELM327 command set and also supports the Extended ST command set. This command set allows for the setup of multiple advanced OBD message pass/block filters, supports “raw” CAN alongside ISO 15765, improved flow control, better control over timing parameters for ISO9141, and support for a wider range of non-standard baud rates. The STN1110 support baud rates as low as 39 bps and as high as 10 Mbps versus the EML327’s 9600 bps to 500 kbps. The STN1110 is able to support this higher throughput due to its more powerful microcontroller that has a 16-bit architecture. Another advantage of this chip is that it can process up to 40 MIPS while the ELM327 only support 4 MIPS. In terms of price the STN1110 is also cheaper than the ELM327 at $10 compared to $24. We have created a detailed comparison table to show the difference between the two chips (Table 2).

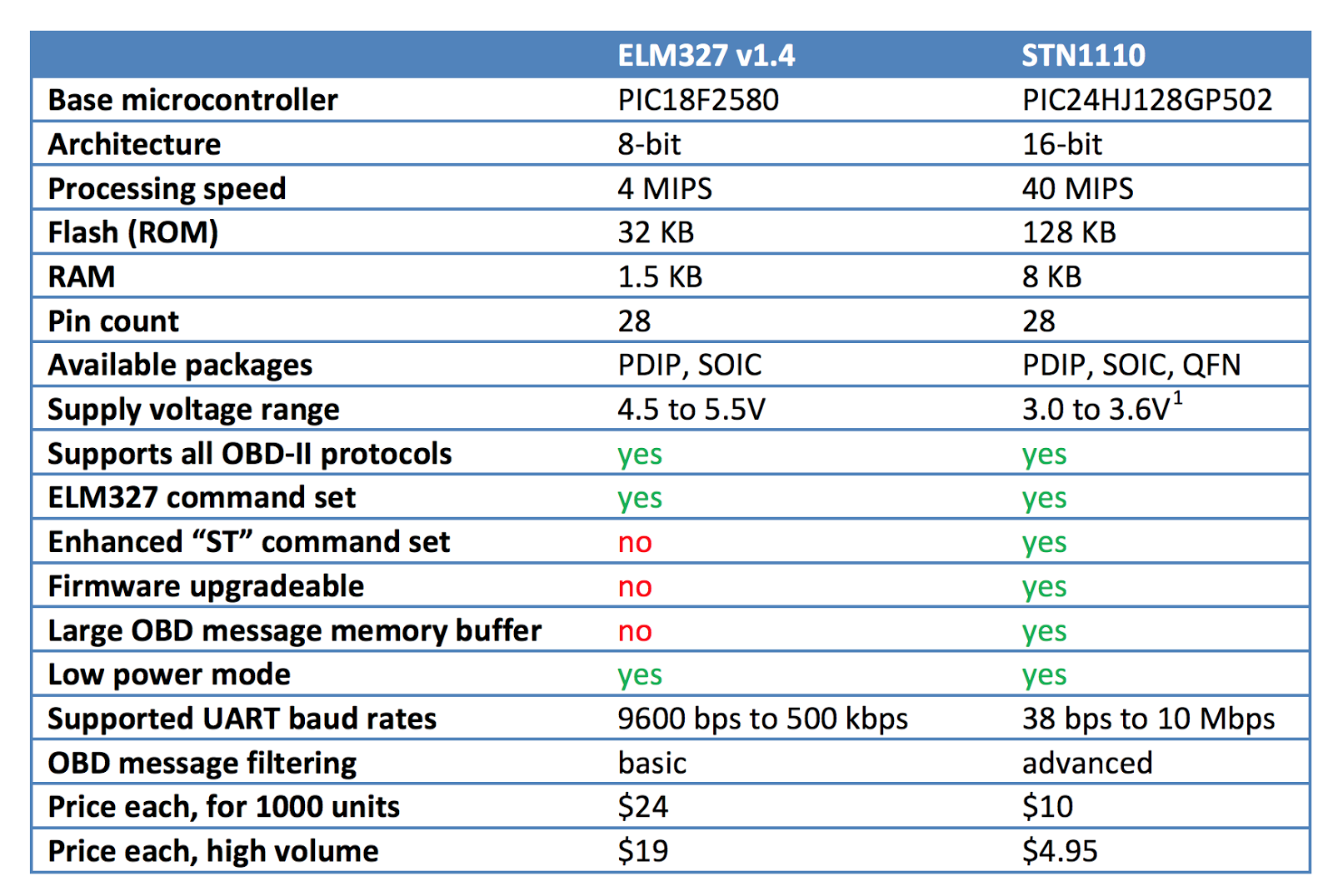


Table : STN1110 and ELM327 Comparison

After carefully comparing the ELM327 and the STN1110 we have decided to go with the latter option. It seems that in every category the STN1110 is an improved version of the ELM327. The ELM327 has been produced since 2005, meaning the internals have not changed since then. The STN1110 in comparison has been in production since 2010, meaning the internals are much newer. One other reason that we chose the STN1110 is that we utilized a breakout board containing this chip from Sparkfun for our viability testing.

### 6.4.4 Transceiver Configuration

In order to communicate with the CAN bus we will need to implement a CAN transceiver onto the PCB. According to the reference schematics for the STN1110 they utilize the MCP2551 CAN transceiver. Additionally, the STN1110 breakout board from SparkFun utilizes the same CAN transceiver. The purpose of a CAN transceiver is to serve as the interface between a CAN protocol controller, i.e. the STN1110, and the physical bus. This chip provides differential transmit and receive capabilities for the CAN protocol controller. Each node in a CAN system must have a device to convert the digital signal generated by a CAN controller to signals suitable for transmission over the bus cabling. This provides a buffer between the CAN controller and the high-voltage spikes that can be generated on the CAN bus by outside sources such as ESD. In addition the voltage differences on the OBD pins would not be suitable to connect directly to the STN1110, these transceivers will normalize the voltage. In terms of package size the MCP2551 comes in an 8-Lead PDIP and an 8-Lead SOIC package.

In addition to having a transceiver for CAN the reference documents for the STN1110 include transceivers for ISO and J1850. The reference schematics for the CAN, ISO, and J1850 are shown in the following figures.

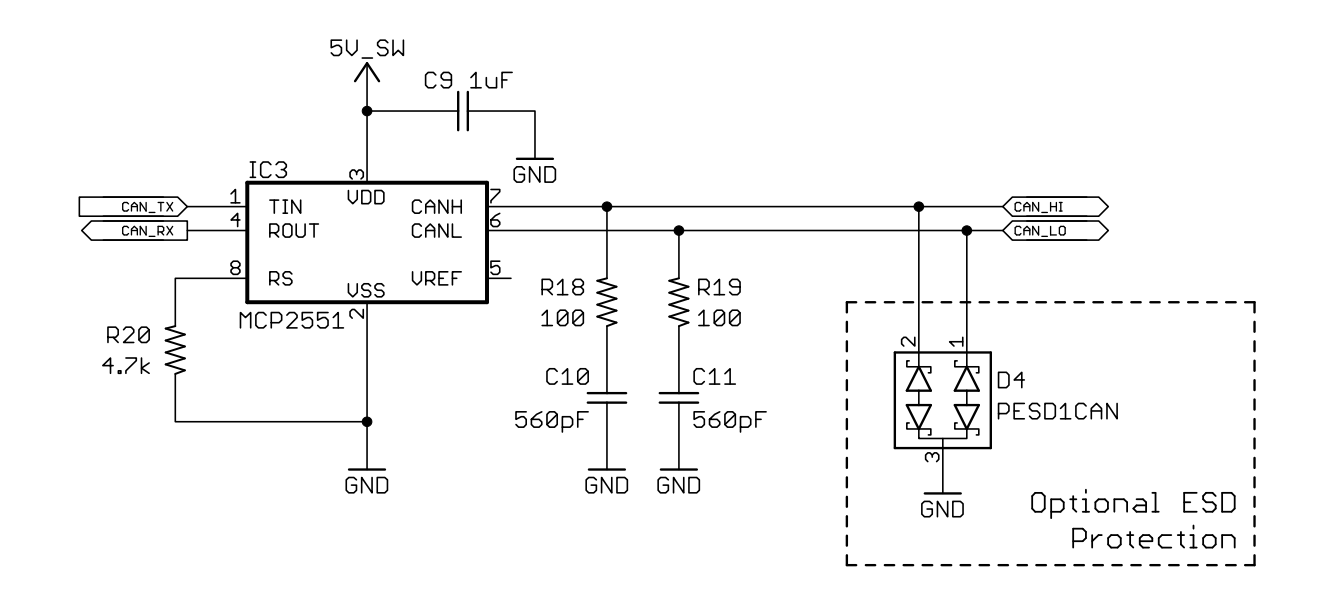


Figure : MCP2551 CAN Transceiver:

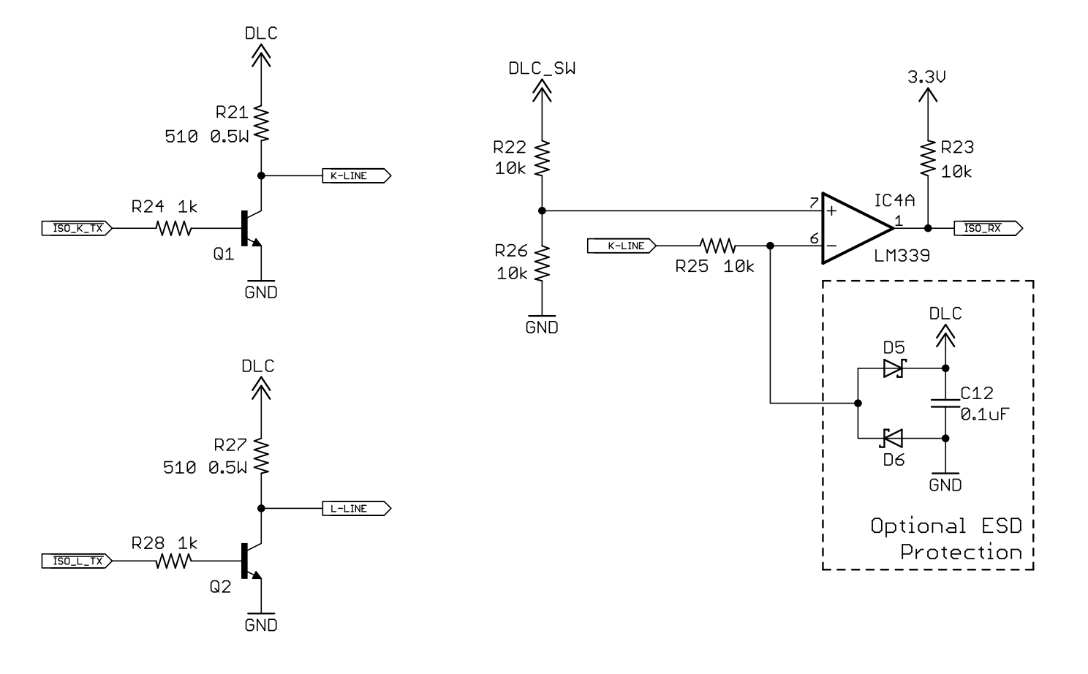


Figure : ISO Transceiver

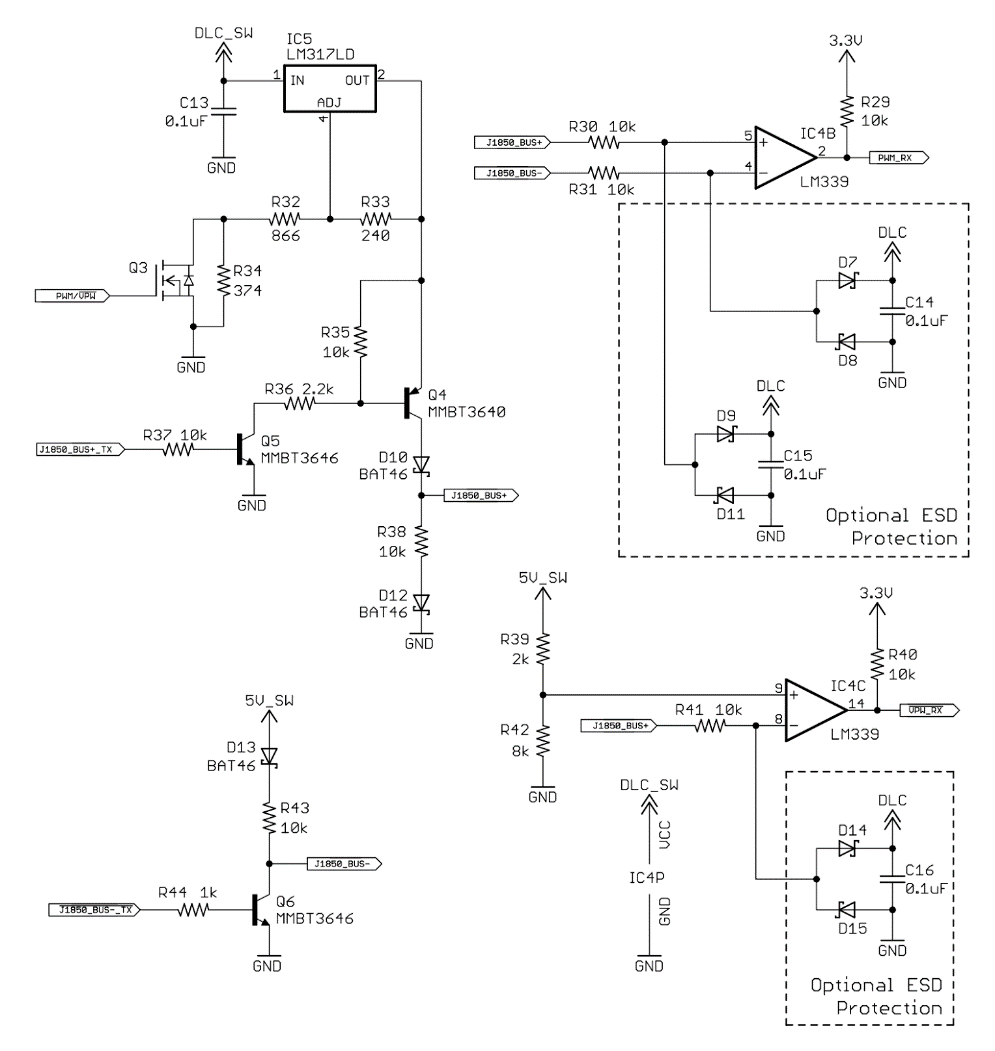


Figure : J1850 Transceiver

### 6.4.5 Interaction

Communication with the STN1110 is done exclusively through a four-wire UART connection. The default connection settings for UART are a baud rate of 38400, 8 data bits, no parity bit, one stop bit, and no handshaking. For different applications the baud rate can be changed as it is software-selectable. Once the connection to UART is initialized there will be a prompt for entering commands. There are three types of commands that are recognized by the STN1110: AT commands, ST commands, and OBD requests. The implemented AT command set is fully compatible with the ELM327 AT command set. AT commands begin with “AT” and are used to cause the STN1110 to carry out an action such as changing display settings, performing a reset, etc. To provide additional functionality the STN1110 implement the ST command set. This command set is also used to make the IC carry out an action such as switching the UART baud rate, change the Power Save configuration, etc.

The most relevant command set we will be dealing with is the OBD requests. These are messages that are transmitted on the OBD bus. The constraints for these messages are that only ASCII hexadecimal digits (A-F and 0-9) are allowed. When sending a command over UART all commands that are sent must end with a carriage return byte (0x0D). The two OBD requests that we will be sending are for the vehicle’s speed and the engine RPM. Further details on the OBD-II communication protocol are detailed in Section 7.

The OBD-II IC will be interacting with the OBD-II port and the MCU. The IC is required to request and receive information from the OBD-II port. In vehicles made after 2008 the OBD-II IC will interact with the CAN bus, which will then request the information from the vehicle’s OBD. The MCU will be doing the background work and controlling most of the pieces in the device. The MCU has a built in Bluetooth function which we will use to complete the interaction between the device and an Android compatible device. We intend to send some of the information from the OBD-II to the Android application. The android application will mostly be used to contact emergency services in the case a car crash is detected.

### 6.4.6 Data Collection

The goals of collecting data from the user will be to understand their driving habits and give them feedback. From OBD specifically we would be able to poll the vehicle speed and see how fast their speed is changing. From this information we would be able to tell how fast they are stopping and how fast they are leaving from a stop. The feedback provided would aid them in being a safer driver. Accessing a vehicle’s OBD-II port will allow us to obtain crucial information about the vehicle in real time. A request for information can be sent to the OBD-II by sending it a specific PID code. Depending on the OBD-II port’s current mode of operation, a PID code request will return different types of vehicle data. According to the ISO 15031 standard, an OBD communication protocol should have at least 10 modes of operation. The first mode of operation is used to identify current powertrain information. In this mode of operation, information such as engine rpm, vehicle speed, throttle position, fuel tank level, and many more can be received with a simple PID request sent to the OBD-II port. A vehicle’s engine rpm can be obtained if a PID value of 0C, in hex, is sent to the OBD-II port. The throttle position can be obtained with a PID value of 11 and the vehicle position can be obtained with a PID value of OD, both in hex format. Although a lot of a vehicle’s information can be retrieved through the OBD-II port, a lot of it is of no use to us as it does not help in detecting a car crash. In the future we might look to implement additional functionally to the data collection to provide additional information to the user about the current state of their vehicle. For now, the information we are looking to retrieve from the OBD-II port is mainly the vehicle’s current velocity and the vehicle’s rpm. Other critical information can be obtained from the OBD-2, but this information is proprietary and is hidden by the manufacturer. Information such as the airbag codes and crash sensors can easily be accessed by the OBD-II or a car’s CAN, but the information is not publically available online. The only possible way to extract the non-standard information from the OBD-II port is through trial and error of sending different PIDs in hex value to the OBD-II port and analyzing the information the OBD sends back. The same procedure would need to be done for all the different car manufactures. For now, we have decided not to tackle this problem as it would be too time consuming and focus on only collecting the publicly available data from the OBD-II port.

We have no intention of storing the information received from the OBD-II port in a database as of now. It is highly possible that we will be storing the data collected from the phone locally in an Android compatible phone. Since we are focusing on collecting mainly the vehicle’s speed from the OBD-II port, it would be relatively simple to store the data locally as the data would never get too overwhelming in terms of size. The Bluetooth functionality in our device will allow for the transfer of the data from the IC to the Android phone. Ideally, the Android application will be able to continuously read streams of data from the OBD-II port to the IC and from the IC to the Android phone. It is crucial to program the IC to be constantly requesting and receiving data from the OBD-II port. We will be processing the velocity of the vehicle, but we will also sending other types of PIDs to the vehicle’s OBD port to scan for errors in the vehicle. Modes 5, 6,8 in the modes of operation have specific PID codes that help scan for errors in the vehicle. Checking whether error codes in the vehicle have been engaged can help us determine better if the vehicle has gotten into a car crash. Of course, this won’t be the only information we will be using to determine if a car has gotten into a car crash. We intend to use the data from the OBD-II port as well as the data we get from the other sensors to determine if a vehicle has gotten into a car crash. If the data we obtain from the OBD-II port and the other sensors seem to conclude that a car crash has occurred, the Android application will respond by sending a notice to emergency services and the user’s contacts.

## 6.5 Accelerometer and Gyroscope IC

For our accelerometer and gyroscope IC we will be using an IMU which is a combination of both. We chose to use this type of device because it has a very small footprint, can provide ample functionality and we do not have to worry about communicating between several devices. The acronym stands for Inertial Measurement Unit. We will further explain what is inside of this device and how it works in the following section. We have considered three different IMU’s and will pick the one that best satisfies our needs. This integrated circuit typically has more devices built in besides the accelerometer and gyroscope. In most cases it includes a magnetic sensor (magnetometer) and an embedded temperature sensor.

### 6.5.1 Functionality

In terms of functionality the IMU will be one of the most important devices in the system. This IMU is one of the few devices that will generate data and information for the system to use so it is paramount that this device is accurate and responsive. Its ability to determine linear acceleration and angular velocity will determine a wide variety of conditions the car could be in. The possibility of having a magnetometer and an embedded temperature sensor on the other hand will have limited application but will still be useful in certain scenarios.

Since the only major components that are from the IMU are the accelerometers and the gyroscopes I will briefly talk about how they are built. Since the IMU has to measure physical forces such as rotation and acceleration small mechanical devices must be implemented in this IC that can generate electrical signals. These types of devices are normally referred to as MEMS (Microelectromechanical systems) devices. The way that a MEMS accelerometer and a gyroscope are set up can be seen in the figures below.

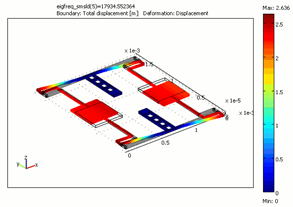


Figure : Initial setup for MEMS accelerometer and a gyroscope

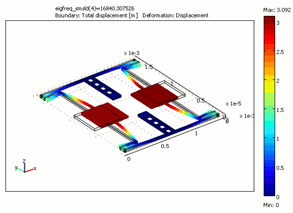


Figure : Ending setup for MEMS accelerometer and a gyroscope

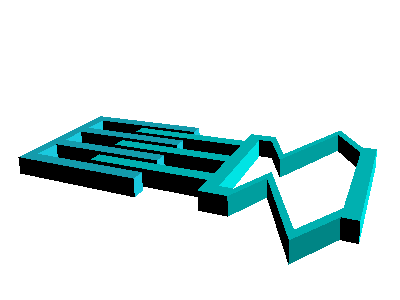


Figure : Closer picture of MEMS

The accelerometer gages acceleration by measuring change in capacitance. As seen above this microstructure consists of a mass, springs and fixed plates. The mass is attached to a spring which allows the mass to move along one dimension while the outer plates are fixed. So when an acceleration in a particular direction is applied the mass will move and the capacitance between the plates and the mass will change. This change in capacitance is then measured, processed and will correspond to a particular acceleration value. So the purpose of the accelerometers are to measure the linear acceleration of the car

The microstructure of the gyroscope is somewhat similar to the accelerometer. Like the accelerometer a mass is allowed to move. In fact it is actually constantly oscillating but when an external angular rate is applied a flexible part of the mass would move and allow perpendicular displacement. This displacement is caused by a force that is due to the Coriolis Effect.

Now that we know how these devices work, inside of the IMU there are three accelerometers. This is due to the fact that the accelerometers can only determine acceleration in one dimension. The measured signal from each accelerometer is split into three components (x,y,z) because they are placed orthogonally to each other and as a result can tell us several different types of conditions the car may be in and can determine the total acceleration. If there was a large acceleration that went above a certain threshold in the opposite direction that the driver was headed, we could assume that there was a head-on collision. If this acceleration was oriented in the same direction the driver was headed the system should assume a rear end collision occurred. If this acceleration is perpendicular to the driver’s velocity, the system should assume that a t-bone collision occurred. Of course more conditions may be applied and tested but this is what we shall initially test for.

The gyroscope may also provide a wide variety of conditions and events the car could have experienced. Just like the accelerometers there are 3 gyroscopes. Each Gyroscope can only determine Coriolis force in one axis so like the accelerometers they are placed orthogonally to each other. This will then determine if there was any rotational velocity in any of the three particular axises and will split it into individual components. The z-axis will determine if the vehicle turned left/right or even if it has spun out of control. The x-axis will then determine if the vehicle is experiencing a flip and the y-axis will then determine if the vehicle is experiencing a roll.

The magnetometer as previously mentioned will have very little application. Regardless this does not mean it is not utterly useless. This magnetometer can determine if the car is facing north, east, west, south, etc. and as a result could help out with directions. If the quality of the GPS in a user's phone is not up to par with other standards.

Lastly the embedded temperature sensor could help determine if cabin of the car is on fire. This is clearly not ideal because in order for the sensor to trigger it will have to sense a large temperature and the car at that supposed point could already be engulfed in flames.

### 6.5.2 Interaction

This IC has several different features when interacting with the Main Microcontroller. It has a 16-bit data output, SPI and I2C serial interfaces, interrupt generators, embedded self-test, embedded FIFO (first in, first out) and a low power mode. How this device will interact with the Main Microcontroller or the other devices in the system will be discussed below.

Of the two optional serial interfaces we will use SPI (Serial Peripheral Interface Bus). The advantage of using SPI is that it will allow us to have a simpler protocol, and typically lower processing overheads. It also has SPI has faster speeds and longer ranges compared to I2C. We will need more wires but since this is the only device that will be on the bus the pros outweigh the cons.

### 6.5.3 Data Collection

The data collection for the accelerometer and gyroscope IC will mostly consist of retrieving information from the environment in terms of vehicle acceleration and position. The data we collect from this sensor will be very important to us because it will be one of the measures we will be using to determine whether a vehicle has gotten into a crash. We are still uncertain if we will be storing this data in a storage device. It's still uncertain how exactly we will be using the acceleration and position of the vehicle as a factor for a car crash. We are considering using a few algorithms that will make assumptions based on the data collected by the sensors.

### 6.5.4 Chip Selection

Some initial testing was done to see how these devices work before considering the five options you will see below. The initial chip we used to test was the LSM9DS0. Once we found that we could implement IMU products to determine acceleration and rotation the LSM9DSX series were the first chip that we further researched after verifying that the project was viable. When comparing these chips we will consider as many variables as possible. These variables will include and are not limited to the following:

Accelerometer

* Sensitivity
* Accuracy

Gyroscopes

* Sensitivity
* Accuracy

Operation

* Input Voltage
* Input Current
* Data Rate
* Communication

Footprint

* Length
* Width
* Height

#### LSM9DS0

The LSM9DS0 is a system-in-package which contains a 3D linear acceleration sensor, a 3D digital angular rate sensor, and a magnetometer. It is able to detect up to +-16gs of force, a magnetic field of up to +-12 gauss and an angular rate of +- 2000 dps. This IC is able to communicate with microcontrollers through either an I2C or SPI serial interface. This IC can also be configured to generate interrupt signal on dedicated pins and is capable of motion and magnetic field detection. This could be useful for our project in that when the vehicle is in motion it will be able to alert the microcontroller that it is moving and it can start collecting data.

In terms of voltage the supply voltage is 2.4 - 3.6v, and the max current draw is 6.1mA.

#### LSM9DS1

The LSM9DS1 is the successor to the LSM9DS0 and its applications are extremely similar. It too contains a gyroscope, accelerometer, and a magnetometer. In terms of the accelerometer it still is sensitive up to +-16g, however the device is now able to detect up to a +- 16 gauss magnetic field. In addition it is able to detect an angular rate of +- 2000 dps. This device also contains an I2C and a SPI serial standard interface and is able to generate interrupt signals on dedicated pins. When it comes to power draw it is more power efficient in that it only draws 4mA at its peak and it able to operate from 1.9 - 3.6V.

#### MPU-9250

The MPU-9250 has a few benefits compared to the other devices considered. First, it has over 10 times the precision in its accelerometer than the

BNO055 and nearly 50 times the ADXL312. It also has the smallest footprint of the three, meaning it will occupy less space on a PCB. The MPU-9250, however, takes more current than the other devices, even with just the accelerometer operating.

#### BNO055

The main benefit of the BNO055 is that it has the lowest noise output of the three devices compared. The BNO055 was easy to dismiss, however, when taking cost into consideration. The BNO055 costs more than twice as much as the cheapest option (the ADXL312), and still nearly twice as much as the MPU-9250.

Not to mention, the BNO055 also has the worst temperature sensitivity, meaning an increase in temperature will be accompanied by a larger increase in error than with the other two devices. Furthermore, the BNO055 has the slowest wakeup time, taking nearly a half-second to get started.

## 6.6 Main Module Software

This section details the software algorithms that will be written for the MCU in detail. All necessary third party software and libraries are covered in this document as well. The source code for this project will be stored in a GitHub repository located at https://github.com/TheWurminator/harambe\_mcu. Utilizing a git repository will enable collaboration between team members and will allow us to utilize version control.

### 6.6.1 High-Level Overview

The MCU is one of the most important aspects of the design as it will be doing a majority of the calculations. The microcontroller inside of the TI CC2640 is an ARM Cortex M3. At a high-level the microcontroller will first have to perform a series of tests to make sure that it is functioning properly. This will be implemented through the use of predetermined parameters that will be calculated beforehand. After the testing has been completed the main module must connect to the user’s mobile device over a Bluetooth connection. The establishing of this connection will take place through the use of the onboard Bluetooth IC. After the connection has been established then the MCU will continuously gather data from the OBD-II IC and the IMU. Once the main module has gathered a sufficient set of data from these ICs it will compare the results against a set of fixed values. Based on the calculated results the MCU will determine whether the user has been in an accident. If the MCU determines that the user has been in an accident then it will communicate with the Bluetooth IC and will send a signal to the user’s mobile device. If the user has not been in an accident then the system will clear out the previous set of data and will continue gathering data from the sensors. Figure 17 shows a graphical representation of the high-level software overview.

highlevelsoftwareoverview (1).png

Figure : High-Level Software Functionality Overview

### 6.6.2 MCU Software Flow

This section details the components that go into the software for the main module as well as the software design flow for the main module.

#### 6.6.2.1 TI RTOS

The software that is running on the MCU will utilize a Real-Time Operating System (RTOS). Since a majority of the operations that the MCU will be performing are asynchronous we will will be able to schedule them as tasks. Each task will be able to be assigned a priority, meaning that tasks that we deem more important will get a larger portion of execution time on the CPU. In addition, TI RTOS utilizes a real-time multitasking kernel, so tasks will be able to be executed in parallel. This parallel execution will allow for a more efficient flow of execution than with non-multitasking systems, which many microcontrollers utilize. For example, if the system is waiting on a response from an IC on the SPI bus, it can better utilize that time by performing another operation in the meantime such as getting data from a device connected over UART. This example illustrated why a multitasking kernel would be extremely useful in a use case like ours. In addition, since this is a *real-time* operating system, there are enforceable deadlines for execution, meaning the execution time of operations is deterministic. We will be able to set enforceable deadlines on the execution of certain tasks, such as figuring out whether the user has been in an accident. This gives us a concrete idea of how long the execution will take.

In order for tasks to exchange information a series of queues are used. Each process can have a queue which has multiple writers (i.e. other tasks) and a single reader (i.e. the task possessing the queue). This functionality is important because once tasks receive the data they need they can send data to another task’s queue, allowing the current task to reuse the memory they have allocated.

The RTOS that TI recommends for the CC2640 is TI RTOS. They offer example code for use in Code Composer Studio, so development will be able to start relatively smoothly. TI RTOS is limited for use within the TI family of chips, so if we needed to switch to another microcontroller that was not supported by TI RTOS, a significant engineering effort would be required to restore functionality.

#### 6.6.1.2 Development Environment Setup

In order to set up TI RTOS it requires downloading and installing the appropriate files in the CCS project. First the user needs to install the latest version of Code Composer Studio which can be found at http://www.ti.com/tool/ccstudio. At the very least the version of Code Composer Studio should be version 6.0. Next, the user will have to go into the “App Center”, select the version of TI-RTOS for the CC26XX family, and click the “Install Software” button. After answering the prompts and completing the installation the user will restart Code Composer Studio.

#### 6.6.1.3 Software Design Flow

From a high level the first action that occurs in a TI RTOS application is that it initializes the supporting tasks for the Stack and GAP role, then calls the user’s application’s “createTask()” function, and starts the TI-RTOS scheduler. Initializing the device’s Bluetooth role is important because it will be advertising its capabilities to other devices, which will understand how to communicate with our device. The specific GAP role that our device will realize is the “Peripheral role”. In this role our device will be advertising its presence to central devices will be able to establish a connection. After connecting to a device the peripheral will no longer broadcast data to other central devices and they will stay connected to the device that accepted the connection request.

Once the Stack and GAP role have been initialized our application will be able to register itself as a task with TI-RTOS. When we are registering our application with TI-RTOS we will give it the address of our main task thread. This thread is what will be constantly running in an infinite loop, and will wait for callbacks to perform a certain function. In the case of initialization our task will configure the BLE stack, the services (UART, SPI) and hardware peripherals (STN1110, LSMDS0) used by our application. It will also register a large amount of callbacks for system events, which will cover the functionality of our application. After the initialization the main task thread will be sitting in an infinite loop waiting for messages to process.

Specifically relating to the structure of the application we plan to have three processes running, the main calculation tasks, the OBD-II task, and the LSM task. The main calculation task will be spinning and waiting for messages from the OBD-II and LSM tasks. When it receives a message from either of the tasks it will gather the data from the message that is sent and store it inside of a local variable within the task. When data from both the OBD-II and the LSM tasks is present it will perform a calculation on the data, comparing it to determine whether the user has been in an accident. If the user has been in an accident the system will send a data packet over Bluetooth to the user’s mobile phone, then clear the data and keep listening. If the user has not been in an accident the system will clear the values in the local variables for the LSM and OBD-II data and will continue to spin and wait for data.

The OBD task will be sending commands to the OBD-II interface and will be waiting for a response over UART. The commands it will be sending will request the vehicle’s speed and the vehicle’s RPM. Once the OBD task has received all of the information it needs it will send the data in the form of a message to the main calculation task. After it has put the message in the queue of the main calculation task it will reuse the memory it allocated for the variables in its task. The LSM task contains similar functionality in that it gathers data from the accelerometer IC, accounts for the forces that are already acting on the sensors (i.e. gravity), stores the data in local variables, sends a message containing that data to the main calculation task, then proceeds to gather more data from the LSM9DS0 IC.

These tasks will run indefinitely once they are started. In terms of power saving the module will not be able to sleep that much since it will be constantly requesting and parsing data from the OBD-II sensor and the LSM9DS0, in addition it will be performing calculations on the data from these sensors and making decision as to whether the user has been in an accident. This amount of work does not leave a lot of room for the system to go into a low-power state. Once way we could get the system to go into a low power state is to only request data from the sensors after a certain interval such as 100ms. That way we can sleep in between processing data. Another option is that we set up interrupts for UART and SPI after we request the data and sleep until we receive data on the line. Since the communication will trigger an interrupt the system will be able to sleep and save power. In terms of the Bluetooth communication, the process of keeping the connection alive is automated by the Cortex M0 attached to the RF core which is inaccessible to the user. This means that the power drawn by keeping the connection alive and transferring data is unable to be controlled. This will have a negative effect on power consumption, however even at the peak current draw this system is more efficient than an architecture where the microcontroller and the Bluetooth module are separated.

Figure 18 details the software flow for the main module.

RTOS_diagram.png

Figure : Visual representation of software flow for main module of HARAMBE

#### 6.6.1.4 Software Considerations for CC2640

When developing for the Sable-X BLE module which contains the TI CC2640 different board parameters that are present in the “Board.c” and “Board.h” files will need to be replaced in the project. Fortunately Sable-X provides these files to the user. We will simply need to get the required files from Sable-X and swap them with the default “Board” files in the project directory. We believe this this is the only modification that needs to be made in order to get the Sable-X module to work with our project. For development of the software we will be using the TI CC2650 Launchpad which is binary-compatible with the CC2640 and CC2630, it contains the circuitry for Wi-Fi and ZigBee along with Bluetooth Low Energy.

# 7.0 Summary of Power Supply

This section contains the explanation of the power supply for our project H.A.R.A.M.B.E. Since our device will be inside of a car, we have the option of using the car’s battery to provide power to our device. We will also use a backup battery to provide power to our device in the case where the car’s battery fails. The primary power supply for our device will the car’s battery. The second power supply we will be using is a backup battery. There are three possible ways we could get power for the system when considering our main power supply. One method is to use an USB adapter that connects to the DC outlet on a car. You can also connect directly to the DC outlet itself. The DC outlet is usually located next to the ash tray and/or next to the auxiliary port in modern cars. The third option is connecting our device to the OBD-II port in the car. One of the test vehicles which is a 2007 Honda Accord has all three of these options. It includes a cigarette lighter receptacle (DC outlet) with an USB adapter and an OBD-II Port.

Our system as previously mentioned will consist of several ICs. As a result, it will only need a very small amount of power. The integrated circuits that will be used on our system are the SaBLE-x, STN1110 and the LSM9DS1. Each of these ICs only need 3.3 Volts to operate and draw a maximum total of 110 mA when these devices are hooked up in parallel, meaning the ICs could draw a maximum of 363 mW. However, this does not represent the total power consumption of the system, in fact it will be more. As a result exact power consumption will be difficult to calculate since it could vary greatly from how we choose to design our power supply and from where we choose to draw power.

## 7.1 Main Power Supply

As previously mentioned we will consider using three different ports in a car which will allow the system to use the car battery as the main power supply. The advantage of using the OBD-II port as a power supply for our device is that we can extract information from the car itself. This information can complement the information that we will be getting from the IMU (Inertia Measurement Unit). When using an OBD-II IC we can get the car's velocity, throttle position, and other relevant information. The DC outlet on the other hand is more convenient than using the OBD-II in terms of design. This is because, for the DC outlet, we have the option of using a USB adapter to connect our device to the car’s battery. We wouldn’t have to worry about stepping down the voltage because it is essentially already done for us.

### 7.1.1 OBD-II Adapter

As previously mentioned the ideal total power that could be supplied by the car battery through the OBD-II port is 48 Watts. Due to this many people assume that the input voltage is always 12 Volts and the current is 4 Amps. These individual numbers are good for design but unfortunately the input supply can change making this statement untrue. What actually happens is that the input voltage we will have can vary from 12 volts to 14.6 volts. The 14.6 volts is obtained when the car battery is at full power and the car has been turned “on”. Since the output voltage from the car is not regulated we will assume that the output current is not regulated as well. As a result we will assume that this output current to be load dependent. Table 4 shows the readings we measured from the OBD-II ports from our vehicles using a multimeter.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Car Model | Voltage (off)1 | Voltage (on)2 | Current3 |  |
| Honda Accord 2006 Lx | 12.36 Volts | 14.19 Volts | 2.54 Amps |  |
| Toyota Camry 2009 | 12.51 Volts | 14.35 Volts | 2.45 Amps |  |

Table : OBD-II Voltage and Current Readings

In order to get these values you must understand the layout for the pins in the OBD-II port. There are a total of 16 (2x8) pins on the OBD-II. The two pins used to get direct power from the battery are pin number 16 which is positive for the battery voltage and pin number 4 which is the chassis ground. The other pins will be discussed later on to explain what they will be used for. If we do decide to use the OBD-II as the power supply for our device we will need to use a male connection. This connection is rather bulky compared to the other connections and are supposed to only have a clearance of 48mm in length and 25mm in width. Since the OBD-II port is located in a cramped space we will have to account for leg clearance and port connection clearance in our design.

Now that we know what types of parameters we have to deal with when using the OBD-II, this will narrow down what type of topologies and size constraints we will have to deal with, in order to achieve the best efficiency and ensure our device can work. Since we only need around 22.5% to 27.5% of the input voltage and less than 1% of total possible power from the input we will need to carefully consider what we should use to save power and dissipate less heat.

### 7.1.2 Cigarette Lighter Receptacle (DC outlet)

Today, most cars have at least one cigarette lighter receptacle which is able to accept several different types of adapters including USB. The standard rating of cigarette lighter receptacle is 12 Volts with 10 Amps. The car models that we will be using to test our system adhere to the same standard, so this will allows us to use any car out of the four available that we have during testing and/or demo.

One of the advantages of using a cigarette lighter receptacle is that we could also pair our system with a commercial USB adapter which would make our design simpler to build than using an OBD-II connector. The USB port is also relatively small compared to the OBD-II port and is cheaper and simpler to implement. Another advantage of using the cigarette lighter receptacle with a USB adapter is that it already has a regulator built in compared to the OBD-II port which does not. When connected to either one, there is going to be a wide output voltage varying between 12V to 14.6V compared to the 4.95 Volts and 5.33 Volts of our available adapters when the voltage from the car battery itself is stable. This actually changes greatly when the car is turned “on”. The standard USB port has a max output current of 1A - 2.4A which is a large difference when compared to the original 10 A that could be drawn from DC outlet on the car. Below are the values we measured from the commercial USB adapters that we have available.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| USB Adapter Brand | Vout OFF | Vout Nominal | Vout Impulse | Io Nominal |
| Honda Accord 2006 Lx | 0 V | 5.33 V | 7.45 V | 330 mA |
| Toyota Camry 2009 | 0 V | 4.96 V | 7.5 V | 330 mA |

Table : Differences in USB voltage between a Honda Accord and a Toyota Camry

So even though the USB adapters provide many advantages they do not come without disadvantages. For example when the car is turned “on” the output voltage can increase up to 7.5 Volts. Thankfully this is only an impulse voltage but regardless this could still bring many problems if our system can't tolerate this voltage.

### 7.1.3 Conclusion

In the end we decided to use the OBD-II port as our power supply. The DC Outlet would have been a good option if we decided to use the USB adapter but there is a major issue with this. The issue is that we do not know exactly what is going on inside the available adapters without breaking them open and even still it would be difficult to determine what is going on. So this uncertainty could turn out to be a safety hazard. After careful consideration we’ve decided that the OBD-II port is much better for our design because we can extract information that can be used to determine a crash. Since this information comes directly from the car we assume that it would be hard for someone without direct access to tamper with this information. Some of the variables to be considered from the OBD-II are velocity, throttle position and more. Another advantage is that we would only have to regulate a total possible 48 Watts instead of the 120 Watts from the DC outlet. We did find out that some cars come with a built-in USB port but none of our test vehicles have it and we want to make sure our system works with a wide variety of cars.

## 7.2 Main Power Regulation

Now that we know that we are going to use the OBD-II Port as our main power supply, the main problem we have to deal with is the high voltage and high current that comes from the car’s battery. The acceptable range of input voltage for our system as previously mentioned is 3.0 V to 3.6 V. With the OBD-II connector, the output voltage is 12V to 14.6V with a max current draw of 4 amps, so we need a regulator that reduces the voltage and current from the supply in order to power and not damage the system. There are two main types of regulators that we can use, which are linear regulators and switching regulators.

### 7.2.1 Linear regulator

Linear regulators are a type of circuit that can maintain a constant output voltage. The circuit behaves similarly to a voltage divider. In other words the regulator cannot provide more power to the system and it dissipates power. The biggest disadvantage of voltage dividers are that they cannot keep a constant output voltage if the input voltage varies greatly. So to compensate for this the linear regulator behaves like a “variable” resistance. In order to do this the regulator usually comprises of a resistance, a reference voltage, an operational amplifier and a Bipolar Junction Transistor (BJT). The process of regulation usually begins with 2 resistances put in parallel to the output voltage. One of the resistances lets the regulator “see” part of the output voltage. This voltage is then compared to a reference voltage in one of the leads then an operational amplifier is set to amplify the difference. The output of the operational amplifier is then sent to the base of the BJT. The BJT then controls the amount of current allowed to pass through the load which will regulate the output voltage. So the BJT is the component acting like a variable resistance and the effect of the “variable” resistance will allow the input voltage and the output resistance to change but not affect the output voltage.

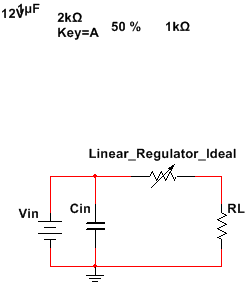


Figure : Ideal linear regulator

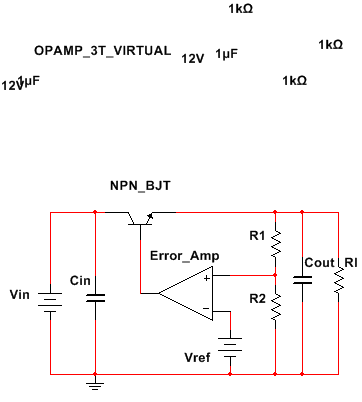


Figure : Non ideal linear regulator

There are few advantages of using a linear regulator for our project. The first advantage is that it is cheap and simple. The average cost of a linear regulator is about one or two dollars and the IC inside the package is much simpler when comparing it to a switching regulator. The second advantage is that it has a fast transient response because it has low ripple at the output when comparing it to the switching regulator. For applications involving a signal, the linear regulator is a great component to use. For example, as previously mentioned in our project, some of the pins on the OBD-II are receiving the signals from the car, and they have a voltage of 7V and draw less current than the supply; so we can use a linear regulator to lower the voltage to 3V or 5V depending on the tolerance of the pins. Since electrical power is defined as current times voltage, it is going to have less total power consumption and efficiency will not be as important. However, there are a few disadvantages of using a linear regulator. One disadvantage is that it has a low efficiency when regulating to low or medium voltages when the input voltage that we want to regulate is high. So when there is a smaller difference between the input and output voltage the efficiency will be higher.  Another disadvantage is that the dissipated power which is the difference between the input voltage and regulated output voltage will be converted into heat, so as the amount of heat increases our device will be at a greater risk of failure. This heat might damage our device over an extended period of time. Therefore, we will use the switching regulator as a component on the PCB to lower the high input voltage from the OBD-II connector.

### 7.2.2 Switching regulator

A switching regulator can come in various topologies. The three most common topologies are buck, boost and buck-boost. We will only be considering a buck topology because the primary goal of this regulator is to lower the input voltage. A simple buck converter typically consists of a power switch, a diode, an inductor and a capacitor. The switching regulator’s primary component is the internal power switch. The switch is an SPST (single pole / single throw) which in other words turns ON or OFF. The “average” output voltage is proportional to the input voltage based on the duty cycle of the frequency that the switch operates in. The duty cycle is usually controlled by a PWM, a transistor driver or other devices but since we will not be building a switching regulator we will only discuss how it operates. The diode is used to block any current that may travel in the opposite direction. The inductor and the capacitor are used to store charge and “smoothen” the voltage from the switch to the output. The nominal output voltage displayed on a datasheet is not the actual voltage in fact it is the average of the output voltage. In reality what happens is that there is typically a ripple voltage due to the inductor and the capacitor interacting with each other. This ripple voltage is generally not an issue unless it is high. The higher the ripple the more it may damage the system. This is due to harmonics and minimum and peak voltages operating outside of the system voltage input range. Since the regulator requires the capacitor and the inductor to charge to reach output voltage it takes a small period of time for the system to operate. The image below is an example of a buck regulator. This is actually not the whole topology since there is no feedback from the output.

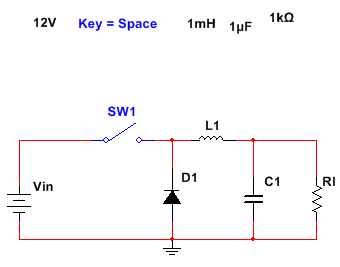


Figure : Buck Topology

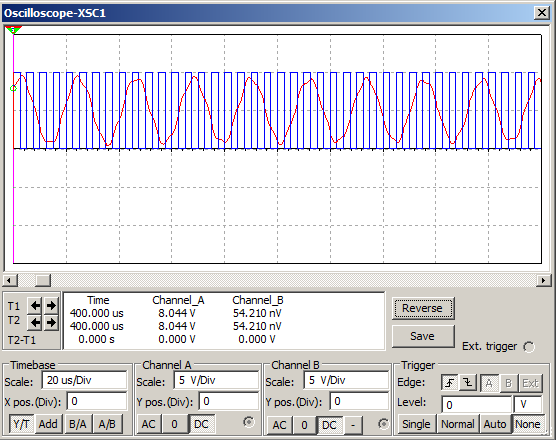


Figure : Buck Regulator Simulation

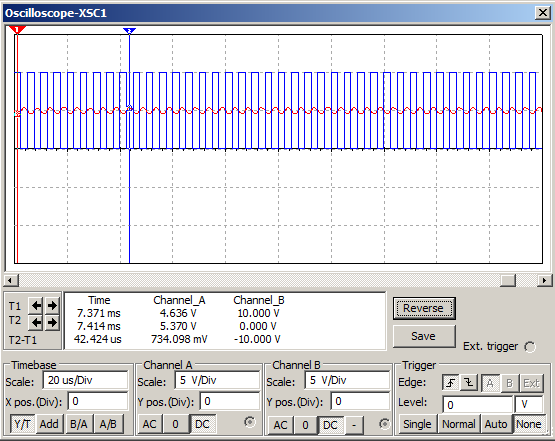


Figure : Buck Regulator Simulation

There are many advantages and disadvantages of using a switching regulator. The biggest and most clear advantage it has is that the output voltage is directly dependent on a switch and not a resistor which is why it is much better than the linear regulator when the input voltage has to be reduced by a large amount. One of the other several advantages of using a switching regulator is that it not only reduces the voltage but also reduces the input power. This is due to the fact that it only takes in power at certain periods of time due to the duty cycle. It is also great because ideally it is supposed to have no resistance besides the load. All of these advantages lead to better efficiency which as a result reduces the amount of dissipated power. As previously mentioned this power turns into heat which reduces the need for a heat sink on our PCB. Not only that but this low dissipated power may be important to our system because the temperature inside of the car can get really high without air conditioning, so we do not want too much heat to be generated by the power supply. The disadvantage of using a switching regulator is that the output voltage ripple can be really high. This is due to the fact that it needs a switching frequency to charge and discharge the inductor and capacitor. This as a result can cause the output to look like a DC output with a ramp waveform. This causes the output voltage to vary and exceed the operating voltage range of the system. For example, in our project, we need 3.3 volts to supply our device, but if a high ripple occurs around 4.5 volts it will create overvoltage and it will damage our device. However if the switch operates at a high frequency where the period of the signal is much lower than the time constant of the circuit this in effect lowers the ripple voltage. At the same time since the circuit needs time to respond because of the time constant it will also take the circuit longer to turn “on” so as a result there is a tradeoff between these two advantages.

#### TPS62177

This is one of the DC to DC converters that we considered. It is a switching regulator that regulates an output voltage of 3.3V from a range of input voltage between 4.75V to 28V. That is one of reasons we might use it, since it fits both of our power sources which are either 12V to 14V from car battery or the 5V from the USB adapter. Another reason that makes TPS62177 switching regulator fit our power supply requirement is the output current of 500mA. The operating input current that goes into the microcontroller has a range of 300mA to 600mA.  In the section of switching regulator, we described that a high duty cycle and switching frequency will decrease the output ripple. The TPS62177 fits this requirement of having a 100% duty cycle and a 1MHz switching frequency. Another key advantage is the range of operating temperatures which is between -40C and 125C, so it includes the highest temperature of 75C that our device might have to face. In addition, the size of the IC is about 1.36 cm^3 and 3.89 cm^2, so it is not going to take up a significant amount of space on the PCB. Finally, the price of this switching regulator is about $1.59. Unfortunately, the manufacturer does not produce the package size that we need. Based on the packaging information datasheet provided all of the package types are WSON, which we do not want to use because it requires the IC to be soldered from the manufacturer. Therefore, it is prohibitive to use it for testing and prototyping.

#### TPS62172

The TPS62172 is another of the switching regulators in our consideration list. It is very similar to the TPS62177, but with a few differences. One of the differences is that the TPS62172 regulates the range of input voltage between 4.75V - 28V to an adjustable output range of 0.9V to 6V. Another difference is that the TPS62172 has a switching frequency of 2.25 MHz, which is about two times higher than the TPS62177. In other words, the output ripple of the TPS62172 is lower than the TPS62177. The size of the TPS62172 is 1.353 cm^3 and 4.1 cm^2, which is very similar to the TPS62177. Moreover, the price of three TPS62172 ICs has a total price of $4.68. Compared to the TPS62177 there is a monetary difference of only a few cents. Besides these two differences, other parameters are very close to the TPS62177. Unfortunately, it has the same packaging problem that the TPS62177 has which is that they only provide WSON packaging.

#### LM2574 & LM2574HV

The switching regulators LM2574 and LM2574HV are the best options that we are considering to use for our power supply. The parameters of switching regulators are very close to each other, the only difference of the LM2574HV (4.75V to 60V) can regulate a higher input voltage than LM2574 (4.75V to 40V). There are few main reasons: the first, it can adjust the output voltage into 3.3V, 5V, 12V and 15V. Although we only need a regulated output voltage of 3.3V, but the microcontroller might demand a slightly different voltage from what we currently expect. Another reason is the range of input voltage of both regulators meet the required input voltage of 3.3V goes into the microcontroller. The output current of both regulators is 500mA, so that also meets the requirement of our device. The main reason that we chose this type of regulator instead of the TPS62172 and the TPS62177 is because the packaging of the LM2574 is DIP, so we can test it before designing it in the PCB. Even though the switching frequency of this switching regulator is much lower than the TPS62172 & TPS62177, so it is easier to do the testing. However, the price of this switching regulator is twice more expensive than the TPS62172 & TPS62177. The size of it is about 5.12 cm^3 and 13.47 cm^2 of so it has bigger area when comparing with TPS62172 & TPS62177.

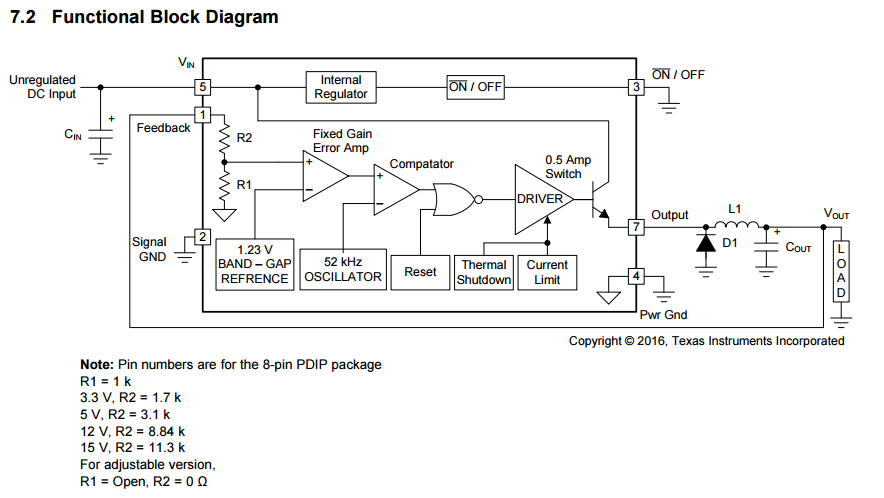


Figure : LM2574 Block Diagram

#### LM2594 & LM2594HV

The only difference between these two switching regulators is that the LM2594HV can regulate the input voltage between 4.5V and 60V, but the LM2594 can only regulate from a range of 4.5V to 40V. However, the car battery that we are going to be using does not go beyond 15V. This makes the LM2594 a better choice if we decide to use this as our switching regulator. One benefit of using this regulator is that it has 3 times the switching frequency when compared to the LM2574. Another reason we decided to use this switching regulator is because it can adjust the input voltage into several different output voltages, such as 3.3V, 5V, 12V.. Finally, the range of input voltages of this regulator includes 14V for the car battery and 5V for the USB, with an output current of 0.5A. The price of 3 of this switching regulator is about $9.54, which is about $1.26 more expensive than the LM2574.

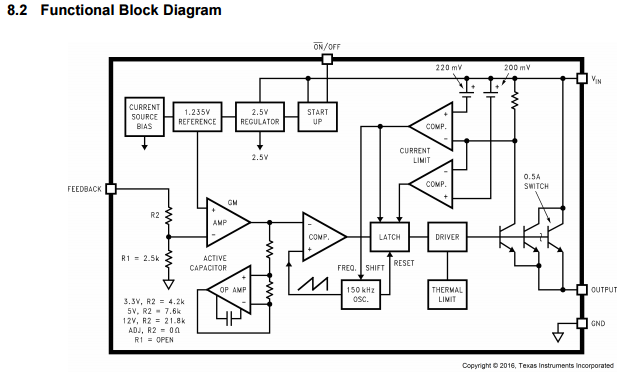


Figure : LM2954 Block Diagram

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | TPS62172 | TPS62177 | LM2574 | LM2574HV | LM2594 | LM2595HV |
| Vinmin | 4.75 V | 4.75 V | 4.75 V | 4.75 V | 4.5 V | 4.5 V |
| Vinmax | 28 V | 28 V | 40 V | 60 V | 40 V | 60 V |
| Vout | 0.9 V- 6 V | 3.3 V | 3.3V, 5V, 12V, 15V | 3.3V, 5V, 12V, 15V | 3.3V, 5V, 12V | 3.3V, 5V, 12V |
| Iout | 500 mA | 500 mA | 500 mA | 500 mA | 500 mA | 500 mA |
| Freq. | 2.25 MHz | 1 MHz | 52 KHz | 52 KHz | 150 KHz | 150 KHz |

Table : Electrical Characteristic Comparison for Switching Regulators

### 7.2.3 Conclusion

We have decided that the IC that should be used to regulate the output voltage will be a switching regulator. The main reason we have decided to implement this is because of the power efficiency that it provides. Below are a series of equations that are used to determine input power for a switching regulator and a linear regulator.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Linear Regulator | [https://lh6.googleusercontent.com/vmu3UQJLaEQDZl9XnusZHfBamFKnkkvo4VsTZ5o8pBY-wqi3wTvgVwibxfX0FXt_C1hluAWBsU8nkU0tqX0LrnJ9jUeriTWeg7KktS1dd-PZLe_f1Duqyv72_7A_qbxnqt2vYhO3](NULL) | [https://lh5.googleusercontent.com/h3M94Sxl0FGkl62ziLC5RrjKnpmrCGosz2o4argKg8d6Wf8z43v_VKRxq8TzWCegnzvTq7rCg8HRg5xEG8p0jHYi3CcH-1XLnRoKBbQZfoznMf9wiaOAloPebPMmuR44aLDwECqF](NULL) | [https://lh6.googleusercontent.com/CLYjCfDsXo9DninWSA1TtlNAUmIsWpBl_pPxzg4BSupU17oayIDo3M2S4Kz6cOW6N4EJbQKzSWb_2kqwHzQz5tDaWjeNXPoI8FLBm--bYF_lTmeR5S8uhErlVIf29oN36tfVHcvP](NULL) | [https://lh6.googleusercontent.com/be8mJ4kjWCSttFhUvp8eqfR9Ua3-YiyB9z1FHjFNWDx8eS4pHDeduFUzdzCy2hyld2W1uKCxo_HxbcZ1xqItYNeiINKbwQA02I5Zot8HPKVXh9cvtHKEpBboyVYHPGSqK0hpIGju](NULL) | [https://lh3.googleusercontent.com/86sVxHAhJMWP2sdt0KUyQxLE9mR4VNlwEEEwaSSMfcxU7FJScLSeRZeE2c0QwlTdi8flw148r3CjlRZkZ7IktdBbd-TRWU5GEWkXcXubUOOT25EgjqNlpOQ05LjJy1ZOk1dq789F](NULL) |
| Switching Regulator1 | [https://lh4.googleusercontent.com/EsiB9YivX6GKPC9f3ZnxZ1NWXzBBfw7qf2twvAaJjByaoeI4szHyDUQGulyrBIq3-fVANmdkjpSMoGGrE_KOCP7Woorutwb3b1xMssU9CJFDzoQU6zbfyOGbuCDARMNMdPxIdLbe](NULL) | [https://lh3.googleusercontent.com/ndbdjb5nE4dlZTJ0sm10MHRAGMsB1V_IMhXDlK_0JxVI5WaSps2c5xh3eFSexRmEFJF2fjdQIcqsOBjpK3V6VnFniKi-H3eeQhmG59_3nNocHEcuoeSpb34CZZxR3vu5asCEGa5E](NULL) | [https://lh5.googleusercontent.com/BbF4bXrBqfMgaIUkVaHYnK_FHul4mFW3sylYWHCdCWSG1gJVvLNyOQbbdKzU0Si5IG7EyxfBb9GUZoXBFIYOp6E99e27mQ6i9MmJdV8K1dl3W5ttQ95ywhP_bZcmSoYc4J2gaRDw](NULL) | [https://lh4.googleusercontent.com/xcbdhvB62KrhUuMwb3VgIIKW6BLkjgLXORAcGH-wIpFjbH59wEIy6YJy2ZLxwYlUo5BJff6H-svZ9yjxNRwL_dk-sBQKUAfqT5nDUSQyKPqfVW9gWYeZFACtxxd_3nsQyYECAzgM](NULL) | [https://lh4.googleusercontent.com/EbMwCMmvMUPErqBF2n1k3M2sqnDZlERhf0SpXT29ACEy3GkzyxkjM1vpyqpkZoqQ4vX-i0t-hfBvqW8suS6oY0e8Mpfs2iSmpYDfx6-Kk4NcasJSgZrl4QnxHN1jWukw7Xc2vZmz](NULL) |

Table : Equations for linear and switching regulator

As previously mentioned before, all of this efficiency will lead to less power dissipation. This will lower the total amount of heat in the system. With better efficiency it will not be necessary to include a heat sink in our system. Not having to include a heat sink in our system lowers the footprint of the overall power supply. We will need to be extremely careful of how much current we draw because the switching regulator increases the amount of current supplied at smaller duty cycles as seen from the Iout./Iin equation provided above. Thankfully the current draw ideally is only determined by the load in the circuit which will be the load in our system. Since there is a ripple voltage there will also be a current ripple. If the system does not work due to this we may need another IC that is able to draw this volatile current from the system.

Of the 5 switching regulators mentioned above we decided to use two of them for prototyping and testing. These two regulators are the LM2574 and the LM2594. Their characteristics are very similar since both are developed by TI. They are able to take in the same range of input voltage of 4.5 Volts to 40 Volts. Their ripple voltage is similar at higher loads but at our range of operation the LM2574 has a slightly better ripple voltage but it will dissipate more heat because of its lower efficiency. The primary reason we decided to implement these two devices are because they both provide a DIP packaging and a SOIC packaging. The DIP package will allows us to test out the IC in a breadboard while the SOIC will let us implement the IC with a smaller footprint on our PCB.

## 7.3 Back-up battery supply

Integrating a good backup battery into our system is important. This is primarily due to the fact that we need to ensure that the system still works after the primary power supply is lost. In order to ensure that the user stays safe the system needs to restore power back to itself as soon as possible. Unfortunately due to time and space constraints it may be near impossible to have uninterruptable power.

There are a vast amount of types of batteries that we can use as a backup battery supply. They all operate in a wide variety of ways. In order to make a decision the usual characteristics of these batteries were looked at. Our options for batteries will be limited to the commonly used batteries that have been used and experimented in order to limit any uncommon intricacies. These batteries types are Nickel Cadmium (NiCd), Nickel-Metal Hydride (NiMH), Lead Acid, Lithium Ion (Li-ion), and Lithium Ion Polymer (Li-ion polymer).

Nickel Cadmium (NiCd) is a mature and well understood battery type. Unfortunately, it has a relatively low energy density so on our PCB design it may have a relatively high footprint. NiCd is used in applications where long life, high discharge rate and economical price are important to the design. This is also a good battery to use in rechargeable applications since it is the only chemistry that allows ultra-fast charging with minimal stress. Common applications of this battery type are two-way radios, biomedical equipment, professional video cameras and power tools. The NiCd contains toxic metals and is environmentally unfriendly. NiCd has been replaced with other batteries due to this concern. Ultimately, since we wanted to be RoHS compliant, we decided not to use this type of battery.

Nickel-Metal Hydride (NiMH) has a higher energy density compared to NiCd but unfortunately that comes at the expense of a reduced cycle life. NiMH contains only mildly toxic metals. This leads to the assumption that a battery of this type would make it easier to be RoHS compliant but in fact it is the opposite. This is due to the fact that the rechargeable batteries may be volatile if configured incorrectly. Applications include mobile phones and laptop computers. Nickel-Metal Hydride is also available commercially as AA and AAA cells.

Lead Acid is the most economical battery type for larger power applications where weight is of little concern. Since we are dealing with a small-scale system where a very high density may disconnect the male to female OBD-II connection and ultimately cutoff power. We decided not to choose this type of battery because of its weight. The lead acid battery is the preferred choice for hospital equipment, wheelchairs, emergency lighting and UPS systems.

Lithium Ion (Li‑ion) out of all these battery types is the fastest growing battery system. Li‑ion is used where high-energy density and lightweight systems are of prime importance. One of the flaws of these types of batteries unfortunately is that the technology is fragile and a protection circuit is required to assure safety. Other issues are that Lithium Ion batteries are more expensive than most other batteries, but high cycle count and low maintenance reduces the cost per cycle over many other batteries over time. Depending on the complexity of our protection circuit the footprint may be larger. Applications include notebook computers and cellular phones.

### 7.3.1 Battery Characteristics

As previously mentioned before space constraints is a notable issue in our design. Having a small footprint will reduce several issues involving how we implement our battery. The battery parameters most affected by this is the capacity of the battery and its nominal voltage.

All of the IC’s implemented in this system can have a nominal voltage supply of 3.3 Volts and have a max tolerance of 3.6 Volts. The minimum voltage supply to power the OBD-II IC is 3 volts, whereas the microcontroller only needs 1.8 Volts, and the IMU only needs a minimum 1.9 Volts. Ideally, the nominal voltage for the battery will be 3.3V due to battery voltage increasing at max charge. If the backup battery only uses 3 volts or less it should not be much of an issue because if we cannot get power from the OBD-II port it is safe to assume that we should not be able to get any of the signals from the OBD-II.

Battery capacity and maximum discharge will be an extremely important parameter for our backup battery. The MMU itself can draw up to 18mA in pulses so the battery should be able to withstand that. So we will not be considering thin film batteries due to the fact that they only have a capacity generally around 50 uAh so the discharge current would end up being far too large for it to handle it.

So to meet these characteristics we have narrowed down our options to two types of batteries. These two battery types are coin cell batteries, and battery packs. If we do decide to use battery packs we will be limited to AAA batteries or smaller. This is because we have a small clearance to deal with since we will be plugging into the OBD-II port. There are some circuit topologies that could maintain a constant voltage and current regardless of the backup battery nominal voltage so we could also get single cells if multiple cells take up too much space but first we will discuss the Discharge topology section.

### 7.3.2 Discharge topologies

In this section we will briefly talk about discharge topologies for our backup battery. A discharge topology is extremely important since we have decided to not implement rechargeable circuits since they can consume a large amount of space and are difficult to design. So in the end we will only be able to use our backup battery for one cycle. We could also opt to not have any topologies for the backup battery but the downside is that it will operate for a smaller period of time because as soon as it nears full discharge the voltage in the battery drastically drops. This will be one of our testing conditions but we will also test some of the proposed topologies to see if the other advantages we could possibly have are true.

We will not consider just using a linear voltage regulator because as previously mentioned they are very inefficient with their power consumption and they cannot regulate the output voltage if the input voltage is lower than the required output.

A common topology used for batteries are charge pumps. Charge pumps simply consist of regulators and SPST switches. We will consider this type of topology more than the linear regulators because it is able to amplify the input voltage. This one of our main requirements because amplification will be required if the battery voltage is lower than 3.3 volts. Now I will briefly discuss about the IC that we may consider to get for the backup power supply which is the TPS6020X.

#### TPS60204

This charge pump provides a regulated 3.3V output, and a maximum 100mA output current. The input voltage can range from 1.8Volts to 3.6 Volts. The output voltage ripple can be as low as 5mV which is much better than some of the other devices we will be using. It also comes with an integrated Low-Battery and Power-Good Detector and can achieve an efficiency of 90%. The quiescent supply current of 35-uA minimum will be good to have since the device will only work periodically. This device is also pretty good as a supply because it also has a low Electromagnetic interference since it does not use a inductor. The shutdown current also very good since it is at minimum 0.05 micro Amperes. The enable pin is very important to have in our application because this will allow us to disable the device whenever we may need to. When the enable pin is high the device runs off the internal oscillator so we will not have to worry about having an external clock for this device but if we did choose to have the Enable pin clocked the device will be able to synchronize. The block diagrams for this device can be seen below.

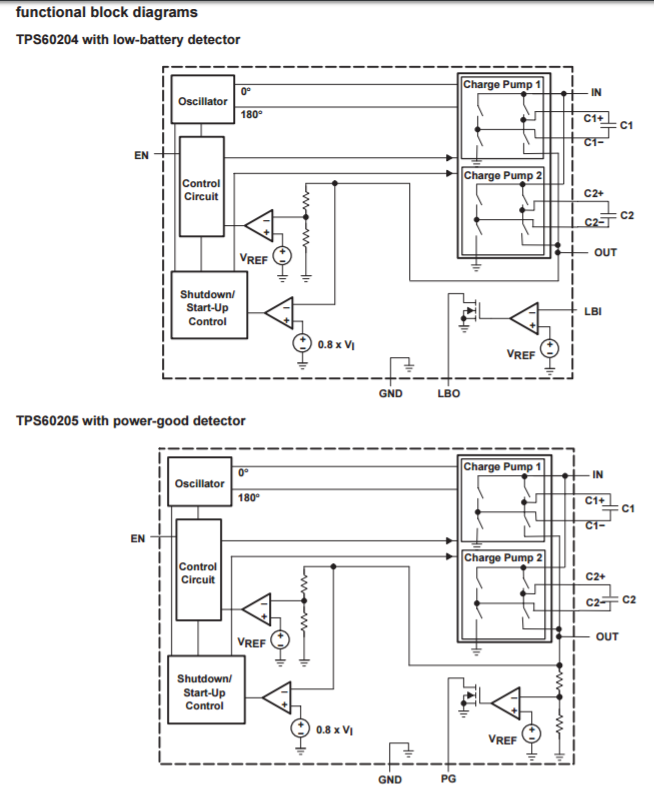


Figure : TPS6020X Family Block Diagram

Of the two possible chips we could use we will test the TPS 60204. The TPS 60205 is very similar but we think that the feature of being able to determine whenever the battery will be low could be a great feature to include in our project. The only external parts we will need to include are 2 more resistors which should not really affect our budget.

A switching regulator could be used again for the supply for the backup battery. The only thing that changes compared to the main power supply is that the topology the regulator will need for the backup battery will no longer be a buck topology. When the voltage supply from the battery is lower than the required output voltage which we know to be 3.3 Volts the circuit will then have to amplify its input voltage. The topology capable of amplifying the input voltage is a Boost regulator.

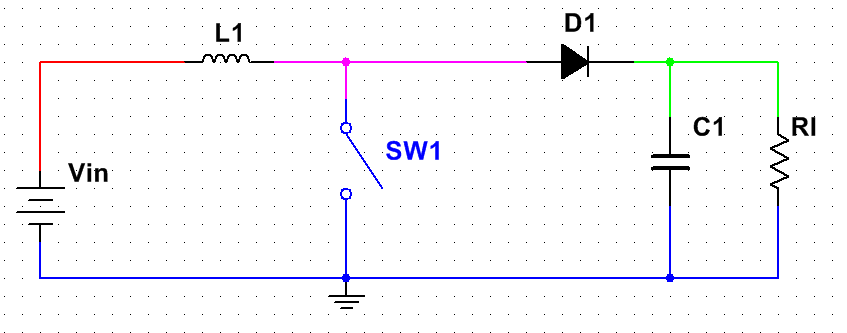


Figure : Boost Regulator Common Topology

The difference from Boost regulators and Buck regulators are that Boost topologies draw current from the input constantly and it charges an inductor to create the amplified voltage. This can be seen in the figure above. The disadvantages of this type of topology are similar yet different from a Buck regulator. The boost regulator like the buck regulator depends highly on the power switch. Unlike the regulator the relationship between input and output is not linear. You can look at the set of equations below to understand why this is the case.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Switching Regulator1 | [https://lh6.googleusercontent.com/d7qsBRjR3d88LMT5hWmKlYBh2tn2OabxzBrVJlFQxupO1tJQ8bZzS8g37xYZ4E2YjupAnayDzLQ34N-zk-0WR4LiTrgeYXrRPLEYb0qt_HPrnGBneiq5xsV1CcEmJLh-oaX9y89I](NULL) | [https://lh4.googleusercontent.com/xfdSPuomnUqbycacmKaqxASn-MpzpdYLmYlIV7a-Wd3l1i98hYqiCw3QVJwtuu1Q0cmiMCNMgpzifdK-QVRXsJr4qeOrgpn_mJ2F6AAjAghDyNjiDkxRNc5SU0qzfN2Vn59z94Il](NULL) | [https://lh5.googleusercontent.com/M6bCoWAH33ozbkKuHZoA2D681OsQ2NPFXu2NI1NksJ93uuvAmtarJK2oMbhjxoLN6J8ef9_l8uvq6hbDrDONwof58OKiFkvUy2JFPjsBRZ5OXz9jVLK7cJWYbyUvHyPb0Y_hr6W0](NULL) | [https://lh6.googleusercontent.com/QY6gkBrAEKEJGyI2psvGBM7oSX1pOn-oQDhjrZmuKX7JVCxoJsqXoxlm-aWzq62H28iMQahP-9Zrk8NZZVSSEu1EPUi7INzsLpiw-rMZ_bGS_FvAmLQ0yjvI9fWQ8lJblnhTK7il](NULL) | [https://lh4.googleusercontent.com/EbMwCMmvMUPErqBF2n1k3M2sqnDZlERhf0SpXT29ACEy3GkzyxkjM1vpyqpkZoqQ4vX-i0t-hfBvqW8suS6oY0e8Mpfs2iSmpYDfx6-Kk4NcasJSgZrl4QnxHN1jWukw7Xc2vZmz](NULL) |

Table : Boost Switching Regulator Equations

Out of all these cases the most important disadvantage is that the boost regulator can never really be turned “off”. Since this device can only amplify the input it can't go lower than the voltage provided in the input. This effect can be clearly seen in the waveform below.

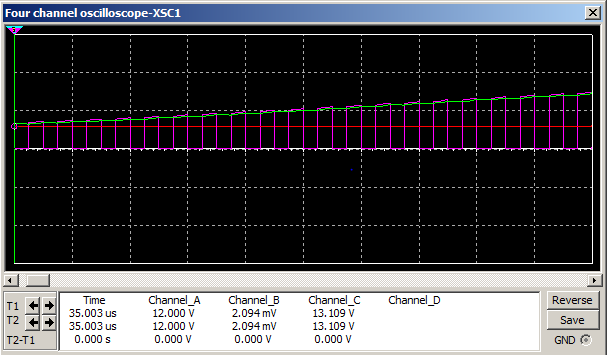


Figure : Boost Waveform During Power On

The waveform clearly depicts that the voltage at the output increases as the inductor is periodically charged but it is never below the input voltage. Now that we have finished talking about the disadvantages and advantages we will take a look at the Boost regulators that we may be considering.

#### TLV61225

The TLV 61225 can reach up to 94 percent efficiency in typical operating conditions. This is great considering that we will have to be as efficient as possible when using the backup power supply. This device operates with an input range of 0.7 Volts to 3.3 Volts. So in other words the battery could be at a very low depth of discharge and it will still work. The quiescent current of this device is 5 micro amps. The output current can exceed more than 40mA when the input voltage is around 1.2 Volts which is a good thing but we may need much more if we choose to power the OBD-II IC. This IC provides output over voltage protection which will be important to have just to ensure our system doesn't get damaged. For this protection the TLV61225 output voltage is monitored internally. If the output voltage of the device reaches the internally programmed threshold, the voltage amplifier regulates the output voltage to this value.

The output is fixed to 3.3 Volts which is exactly where we want it to be. Unfortunately this only comes in a package type which will make it difficult for us to test and prototype but thankfully if it works the way we want it to the footprint this device will take up will be really small. Below is a block diagram of this device.

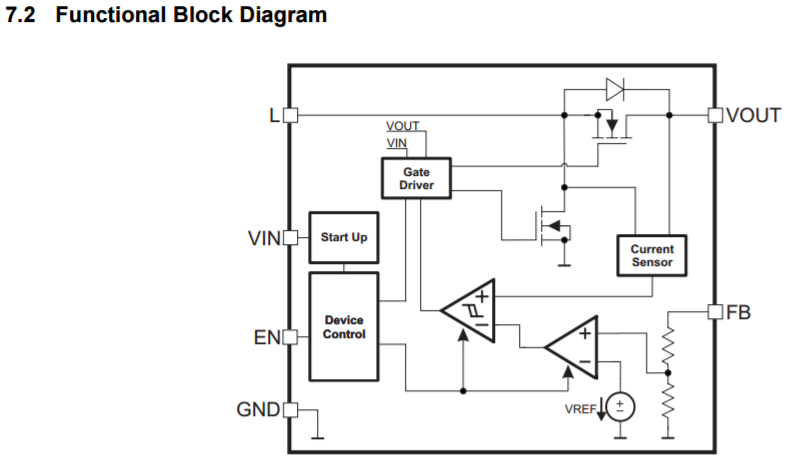


Figure : TLV61225 Block Diagram

#### TPS61260

The TPS 61260 has a large input voltage range that goes from 0.8 Volts to 4 Volts. This is good for our system because it allows the batteries to operate in large depths of discharge. The primary reason we want this is because the batteries we have are not rechargeable. The efficiency of this IC is the highest of all the ICs we have looked at so far which to be exact is 95%. This family of boost regulators unlike the TLV 61225 allows us to have adjustable and fixed output voltage options from 1.8 Volts to 4 Volts. This is a nice feature to have but not really a necessity for our project so if there is a fixed 3.3 output voltage option out of these series of ICs we will use that boost converter. The output current for these series of IC’s  can be 100 mA when the input voltage is higher than 1 Volt and the output is set to 3.3 Volts. Another feature that sounds great but we may not need for our project is the ability of this device to have a programmable output current from 10mA to 100mA. The one downside of this is that it comes in an WSON package. This package is ridiculously hard to solder so this device will be used of the other devices do not work.

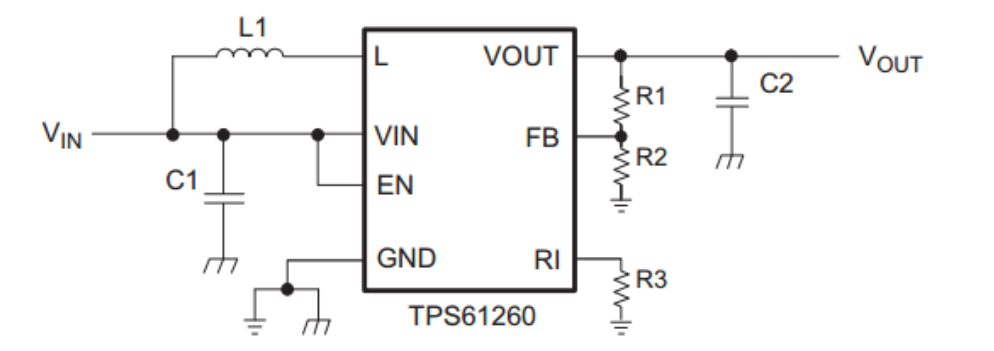


Figure : TPS61260 Reference Schematic Diagram

The final type of topology we may be able to use is a Buck-Boost topology. This could be implemented if the initial battery voltage is higher than the required output voltage but eventually switches and the input is lower than the output. The topology alone has an inverted output so the design of the buck boost converter may be larger and more expensive than the boost or the buck regulators since it will need an inverter. So to keep our device simple and inexpensive we decided to not use this type of regulator.

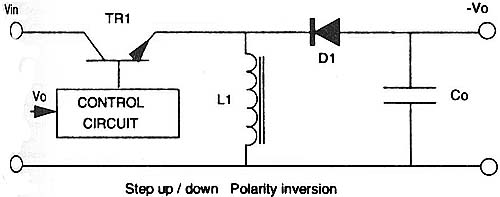


Figure : Buck-Boost Common Topology

### 7.3.3 Battery and Topology selection

For the backup battery we decided to utilize two AAA batteries. There are several advantages to using AAA batteries when compared to the coin cell. The system in total is capable of drawing around 100mA and a single coin cell only has only around 100mAh to 200mAh. On the other hand AAA batteries tend to have up to 1000mAh in capacity. This allows the system to operate on the backup battery ideally up to 10 hours compared to the 2 hours the coin cell will provide. On top of that most coin cells are not able to discharge that much current at a constant rate or even in pulses. Combine this with the fact that lithium batteries have many transportation restrictions in airplanes and are dangerous without a safety circuits due to short-circuiting and overheating, Lithium ion cells no longer seem like a viable option. Short-circuiting may not be an issue for our device but overheating will be an issue because the system will be inside of a car. AAA batteries on the other hand are typically made out of zinc and manganese oxide. These batteries are also known as alkaline batteries. These batteries are typically cheaper and do not require maintenance. They are made through dry-cell construction which as a result makes them completely sealed and explains why they do not require maintenance. The cells do have some disadvantages such as a limited cycle life, environmental issues and leaks but we believe that the chance of the system exploding or catching on fire while driving is a risk that we should be avoided at all costs. The cycle life is mainly affected by deep discharge; the first cycle gives the greatest capacity, and if deeply discharged a cell may only provide 20 cycles. The available energy on each cycle decreases. This is not an issue because alkaline batteries account for 80% of manufactured batteries in the US and the user will have access to the holder so they could replace the battery. Alkaline cells unfortunately have a relatively high internal resistance, making them unsuitable for high discharge current so we will have to be careful when the system is drawing a lot of current.

As far as the topologies themselves we decided to not use linear regulators. The reason we decided to not use this type of topology is because linear regulators are not efficient and cannot amplify the input voltage. We decided to not consider 2 of the 3 switching regulator topologies due to the fact that they do not meet our requirements for operation. We chose to not implement the buck-boost regulators because of the fact that it is not able to provide positive voltage. The buck regulator will not be implemented because the battery we choose to use cannot be supply more than 3.3 Volts so as a result we will need amplification and we know that the buck regulator cannot provide that. So in the end we will only use the boost regulators or the charge controller. Of the two boost regulators that we proposed above we will use the TPS60205.

### 7.3.4 Battery Control

In order to toggle between the backup battery supply and the main power supply we choose to build our own circuit topology due to constraints we will experience. Some of these constraints are:

* There should be no voltage drop across the Main regulated supply
* No reverse current should flow on each supply (battery, regulated buck output)
* A square wave signal will be supplied to enable and disable supply lines

As a result we will need several different components. For example since we cannot drop voltage across the 3.3V regulated power supply we cannot use a simple diode in series with the load to prevent reverse current. So instead we will be using a specific pin on both regulators to turn them on and off. This will help minimize the amount of losses seen by the system. In order to get a signal for the enable pin we will have a comparator. This comparator will then give us a high or a low to send to both of these pins since we can't leave them floating. Now the only problem with the comparator is that it needs constant power. So to do this we will supply a raw battery voltage to the comparator. The comparator we will be using only needs to provide a positive high output voltage when the battery voltage is higher than the line voltage. To do this we will use the CD74HC85E.

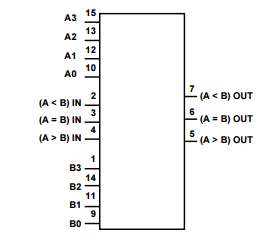


Figure : CD74HC85E Comparator Pinout

#### CD74HC85E

This device more than satisfies our application. It only needs 2 to 6 volts to operate which is great because we only have 3 volts from the raw battery power for this device to operate. The output voltage high can be as high as 1.5 volts which is just what we need to turn off the buck regulator. It also allows us to have more flexibility due to the fact that it has multiple outputs. These could be sent to the microcontroller as status pins for the user to know what issues may be going on in the power supply.

## 7.4 Overcharge Protection

Overcharge is a serious problem that all electronic devices have to face. Moreover, since we are dealing with high voltage from car battery compared to our device, our device only needs a small voltage and current in order to operate; so we will definitely need overcharge protection. With the overcharge, it might cause our device to crash. There are two main types of overcharge protection: overcharge protection and overcurrent protection.

### 7.4.1 Overvoltage Protection

In our project, there are many ways can cause a overvoltage. The first, we might have short in our breadboard testing when we solder the components. The components are really small, so they might solder too close or connect to each other and cause a short. The second, power supply itself might causes an overvoltage. Since our power supply is coming from the OBD-II port that is supplied from the car battery, we need to be careful in a case that a dead 12 volt car battery needs a booster cable from a 24 volt truck battery.

Overvoltage protection is going to be located at beginning of the power supply pin on the PCB, so it will avoid damage if overvoltage comes into the PCB. The way that it works is by setting a threshold voltage which is no more than the maximum input voltage of 3.6 volts, it’s either controlled manually or programmed remotely. Since the threshold voltage is a fixed value, any voltage higher than this threshold voltage will trigger the overvoltage protection circuit and force the high voltage to go through.

### 7.4.2 Overcurrent Protection

The overcurrent happens when there is a high unwanted current goes into device. There are several ways that will cause an unwanted current in our project. The first, there is a situation that the car battery is dead, and it might needs a jumper from a 24V truck car battery, so it will increase the input current to the connector and causes an unwanted current to pass through device. If this is the case, then we need overcurrent protection to avoid this type of situation. Another situation that causes an unwanted current, when we start up the car, it usually increases the voltage from 12V to 14V on the OBD-II connector, and it jumps from 5V to 7V instantly on USB adapter connector. Therefore, it is good to have a overcurrent protection for our PCB to avoid an unwanted high current.

In order to protect our device from the overcurrent, we need a fuse in the circuit. A fuse is a type of low resistance resistor that will stop high current from passing through. The reason that it has low resistor value because when the high current pass through a low resistor, it is easy to break up a wire and becomes a open circuit. For example, in our project, if a high input current goes into the microcontroller, with a fuse of limited current 500mA before the current goes in, the high current such 5A will melt the wire quickly, so the wire become open circuit and protect the device from high current. The large difference between limited current and overcurrent is, the quicker that the wire will melt.

## 7.5 LED

The following section will cover the possible components chosen to build LED lights which display the status of our device. LED light for our project has few different functions. There are two main colors of LED lights, which are red and green. The red light represents the negative side of device, such as, the device is not working properly. The green light represents the positive side of device, such the device is ON, and it’s running correctly.

Here is the explanation of performance of LED, and the key parameters that related to LED lights. The figure 33 is the schematic of two LED lights, and both LEDs will connect in parallel to the pin output on the microcontroller. LED is a diode, it becomes illuminated when it is forward biased. The different types of LED lights supplied by different voltages, and the voltage increases when using the bright LED lights. The turn on voltage for red LED is usually around 1.9V to 2.1 V, and the forward current is 20mA. The green LED has a higher voltage than red LED, and it’s about 3.2V to 3.3V. The luminous intensity of both red and green is 20 lumens, which is able to see at the daytime and nighttime.

Microcontroller is supplying a 3.3V to both LEDs, and the reason of not connecting the power supply pin of OBD-II to the LEDs is the voltage is around 13V, so it’s too big. From the schematic, 3.3V is the supply from microcontroller, and connecting with two resistors R3 with LED1, and R4 with LED2. With the calculation, the value of R3 is equal to R4, it’s around 200 ohms. With 200 ohms resistor, it’s easily to supply both LEDs. In order to turn on LED red, we are setting the pin out register bit to zero. In the opposite, we are setting the pin to 1, so the LED green light will turn on.  Controlling with LEDs with a microcontroller is a very common application and easy to implement.

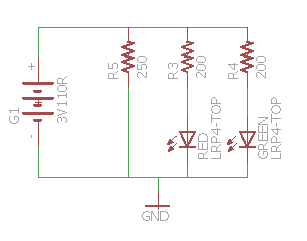


Figure : Schematic for LED

## 7.6 Simulation

This section will explain how we simulated the power system as a whole. Not all intricacies and devices have been considered. For example the comparator in this simulation is not existent but the general effect of it such as toggling between both regulators Effectively meaning that during the prototyping and testing phase the system may give different characteristics. This may be due to impurities in the devices, components and several defects that may tamper the system. Such examples include but are not limited to EMF, Internal resistance, ESR, heat dissipation and much more. This is unfortunate because simulation of these defects are important but extremely difficult to do. So unfortunately in order to save time we may have to fabricate several different systems and test them in harsh conditions to ensure the device works they way we want it to.

### 7.6.1 Simulation Program

The simulation program we chose to utilize was Multisim 13.0. The primary reasons we chose to use this program was because it was already available to us in the computers in the Engineering Department and because most of us were familiar with it due to the labs we had to do. The only issue with this program is that we could not find any libraries with the devices we wanted to implement. There were some comparators already available but the issue is that they did not have similar properties to the actual comparator to be tested. For example when supplying 2 volts the comparator is capable of supplying an output with a high of 1.5 when the correct case is triggered. Unfortunately the comparators we found were not capable of doing this. Instead they either behaved as a linear comparator when we were looking for a logic comparator. When we did manage to find a logic level comparator what occurred was that the device did not encounter a voltage drop from 2 volts to 1.5 volts. More issues arose when attempting to find switching voltage regulators so as a result we decided to keep the circuit simulation as simple and as basic as possible.

### 7.6.2 Circuit configuration

We chose to first build the buck regulator since this is our main power supply. As previously mentioned the DC source in this device is 12 Volts. In figures 34, 35 and 36**,** this is the circuit on top. The boost regulator was then placed below. This device instead had a supply of 3 volts. When we finished building both of these circuits and connected them to the same load we choose to add a SPST switch to both of the DC sources. This ensures neither the boost nor the buck regulator were on at the same time. The main advantage of this was to ensure that all current passed along the correct path and the sources did not short.

Once the design was completed we began to work on the simulation. First we tested the Buck regulator alone. The waveform as you can tell below did overshoot the voltage. After several different simulations the voltage managed to get as high as 4.2 Volts for a brief period of time. This as a result meant that the devices would have been damaged. Thankfully some of the feedback in the LM2574 is supposed to account for this overshoot.

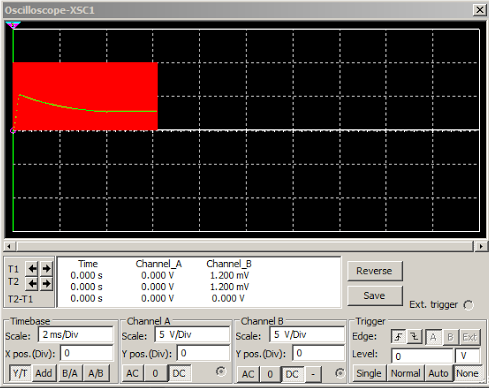


Figure : Buck Converter Simulation

We then tested the Buck regulator alone. The waveform as you can tell below did not have a voltage overshoot unlike the buck regulator. After several different simulations the voltage managed to only get as high as 3.3 Volts and then plateau. This is exactly what we want because it will ensure that neither the battery nor the load gets damaged; the load voltage would remain at 3.3 Volts.

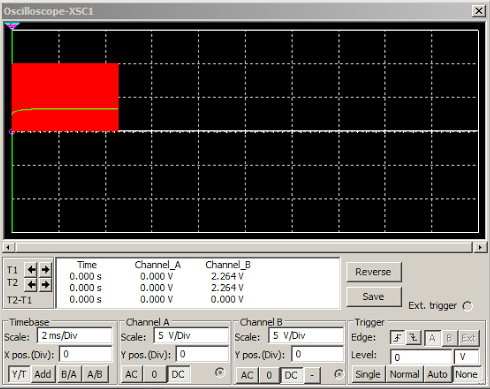


Figure : Boost Converter Simulation

Once we ensured that each regulator worked alone we proceeded to test the whole system. The initial condition had the buck regulator connected to the car battery and thus providing power to the load. After the voltage reached the desired 3.3 Volts we cut power from the buck regulator effectively simulating disconnect. We waited until the voltage seen at the load was 3 volts. When the voltage reached this point we added a millisecond to the “disconnect” time. This is because there will be a brief period of time that the comparator will need to process the voltage change at its input and then provide a high voltage to the enable pin on the boost regulator. Once this time elapsed we turned on the boost regulator. The waveform below provides

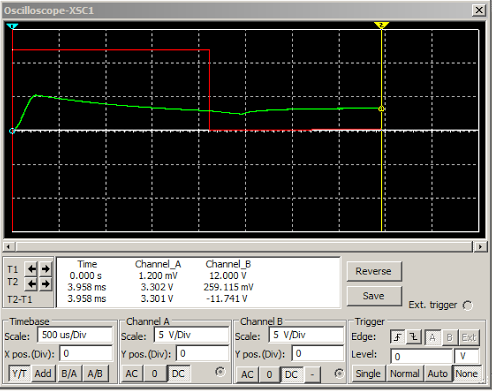


Figure : Power Test of Buck to Boost Conversion

.

## 7.7 Design Summary

Table 9, which can be seen below features the components that will be tested. The first row will be the primary components to be used. The second row will be the components to be tested if the primary devices do not work.

|  |  |  |  |
| --- | --- | --- | --- |
| Comparator | Main power regulation | Back-up Supply | Backup topology |
| CD74HC85E | LM2574 | AAA batteries | TLV61225 |
| N/A | LM2594 | N/A | TPS60204 |

Table : Component Consideration Table

The flow of power to our system will begin from the OBD-II port. As previously mentioned the amount of voltage being taken in by the system is 12 Volts. The amount of current that may pass is a maximum of 4 Amps so to ensure the battery doesn't get damaged our total resistance should not be lower than 3 Ohms. This unregulated supply will go to the LM2574. The LM2574 can provide up to 3.3 Volts to a load of 0.5 Amps so this means that the output power can be a maximum of 1.65 Watts. The total efficiency of this device is around 72% so the total amount of power being drawn is around 2.3 watts. This means that the LM2574 takes in up to around 190 mA from the car battery. This then gets supplied to both the comparator and the load. The comparator will then compare the regulated voltage from the Buck regulator to the raw backup battery voltage. Assuming there are no issues the power should still be functional. Now that the circuit is fully operating we assume that the driver is either turning on the car or the power pin of the OBD-II gets disconnected. If this occurs the voltage seen by the LM2574 will drop down to ranges where we know it will not be able to operate. The output voltage will also reflect this change so the comparator will be able to see this change. When the output voltage from the buck regulator is lower than the raw battery voltage a signal high from the comparator will be sent to both of the regulators. This signal high will then turn off the buck regulator and the boost regulator for the battery will then be enabled. The main issue that we may run into is that the system as a whole may encounter a brief period where power is lost. In order to prevent this a series of capacitors will be put in parallel to briefly retain power during this period. A diagram below shall visually depict how this occurs.

thing.png

Figure : Power System Overview for the Main Module:

# 8. Printed Circuit Board

This section covers the overall structure of the our project H.A.R.A.M.B.E.. The overall of picture includes the layout and construction of our device on the breadboard testing and printed circuit board. But first, we are going to explain the purchase of each components and chip families that required by our device.

## 8.1 Components and Chip Families Purchased

So far, with our final decision, we purchased OBD-II Scan, accelerometer, gyroscope and backup battery, and the components included two types of switching regulators. There are other few components and chips that we have not order, which includes Bluetooth.

### 8.1.1 Suppliers for Components and Chip

There are different locations that we ordered our components and other components that we are going to order. All of the components that we ordered and have not order are going to purchase from the two main suppliers TI and DigiKey. The related chips such as accelerometer and gyroscope purchased from the Sparkfun and Adafruit, and we purchased other components such switching regulators from the DigiKey.

There are several reasons that we choose these suppliers for our components and chips. The first reason is some components we cannot find on other supplier. For example, when we searched the switching regulator components such as TPS62177, there are a lot of different options of this component on the DigiKey, such as different types of packaging, different input regulators. However, we searched the same type of component on the Adafruit, there are related components that we need, but they were built in with a chip in an expensive price; it looks fancy, and the chip can do a lot of different things; however, save money is also our goal for our project, so that is why we do not choose the components from the Sparkfun or Adafruit. Some components is out of stock at TI website, you might wonder why the company design the components but sold out before other companies, because other companies bought all of the components form TI, that’s why some components showed not available.  The second reason is some components are cheaper from Digikey than purchase from TI. For instance, we purchased the switching regulators LM2574 with unit price of $2.76, and LM2594 with unit price of $3.18. Comparing with the supplier TI, LM2574 is $2.9, so it is a little bit expensive than Digikey. Third reason is some suppliers delivery components faster than others. The components we bought from the Digikey is about 3 days shipping. Some chips purchased from the Adafruit, and it took 3-5 days to arrive.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Part | Quantity | Cost | Mfg part | Supplier | Total |
| 330 uH | 1 | $.75 | BOU-0000850 | Digikey | $.75 |
| 220uF | 1 | $.90 | SRR1208-221KL | Digikey | $.90 |
| SaBLE-x Bluetooth Module | 1 | $16.52 | 450-0119C | Digikey | $16.52 |
| 11DQ06 | 1 | $.46 | SB160-E3/54 | Digikey | $.46 |
| 120uF | 1 | $.55 | UPW1J121MPD | Mouser | $.55 |
| 68uF | 1 | $.36 | EEU-FR1E680B | Mouser | $.36 |
| 1N5817 | 1 | $.43 | 1N5817 | Digikey | $.43 |
| CC2640 | 1 | $3.64 | CC2640F128RGZR | Digikey | $3.64 |
| STN1110 | 1 | $9.99 | STN1110 | Scantool | $9.99 |
| LSM9DS1 | 1 | $6.40 | LSM9DS1TR | Digikey | $6.40 |
| LED red | 1 | $.27 | LTST-C191KRKT | Digikey | $.27 |
| LED green | 1 | $.27 | LTST-C191KGKT | Digikey | $.27 |
| 30 pF | 1 | $.25 | DC33ATR | JAMECO | $.25 |

Table : Hardware Component Cost Table

## 8.2 Software Considerations

The design of our device H.A.R.A.M.B.E is not really complicate when we design all of the chips and components on the PCB although there are a lot of modules on our PCB. Before designing any of our printed circuit boards we decided to analyze which software would allow us to do the job in the most efficient and cost effective way. The team does not have any previous experience with PCB design, however EAGLE is one of the most talked about board software among students due to there being an EAGLE Express version which is free. An alternative solution which is open source is KiCad. In this section we will be comparing the two options.

### 8.2.1 KiCad

As an alternative to EAGLE, KiCad performs well, it contains all of the primary features that EAGLE has, and is free and open-source. Because this software is open-source versions have been built for all operating systems which would allow all group members to collaborate on the design on different platforms.

One disadvantage of KiCad it that it does not come with a built-in autorouter. This functionality was taken out due to legal trouble that the creators were in. An autorouter can be installed and used separately though discussions on forums online cast doubt on the efficiency of the router. For a small design such as ours it would make sense for us to router the design by hand.

A unique feature of KiCad is the 3D board view. This will allow us to get an idea of what our board would look like with the components on the board. By looking at this representation we will be able to get a better idea of where to place the components and which routes are most efficient.

### 8.2.2 EAGLE

EAGLE PCB is commercial software that enables the user to capture schematics and create board layouts. This software provides a variety of features that would aid us in creating the board. As with most commercial software there are a variety of versions. The free or “Express” limitations are that the PCB can only have two layers, the routing area can be at maximum 100x80mm. There is also an Education version of the software which is free for students and allows up to 6 PCB layers and a 160x100mm routing area. The Educational version of the software seems to be identical to the “Maker” version which is $160. Another benefit of using EAGLE software as our PCB design is that it is easier to debug, because when we select the component in layer board, at the same time, the components in the schematic will also select with highlight. So it is easy to see which component that we are debugging.

For the purposes of our project I believe we would only need two layers and the routing area would be sufficient. Another advantage of this platform is that the software is cross-platform meaning team members who are working on different operating systems would be able to share and work on the same files with EAGLE. In addition to being cross-platform EAGLE contains an autorouter which would definitely aid in the design process.

### 8.2.3 ExpressPCB

ExpressPCB is another layout software for most PCB design. It can many unique tools that other software does not have. For example, there is tool such as net connection tool. It is really useful because none of our group members are familiar with PCB, so before we become professional on PCB layout, this tool will help us checking any error on each net. Another benefit is that there is an overall checker tool, so we can use it to check our entire layout after our design is finished.

### 8.2.4 Altium CircuitMaker

Altium CircuitMaker is also a layout software. All the tools are really easy to use with the help tool. Overall, it is really close to Eagle, such as the tools and components. Moreover, it has a unique tool as KiCad has, which is the group design. It’s like all of my members can working on the entire PCB layout together. The reason it is good because all of us are new to PCB design, and each of us are working on the different section, so some section are relating to software, and other are relating to hardware. Therefore, one of us can working on software design on PCB, at the same time, others can work on hardware design on the PCB. Overall, we can check each other to avoid any mistakes.

## 8.3 Software Decision

After careful consideration we have decided to utilize EAGLE CAD design software. Its competitive features at a low cost to students, widespread support, and the abundance of tutorials greatly influenced our decision. Overall, we feel like this is the best option.

## 8.4 PCB Layout Consideration

### 8.4.1 Copper Thickness

Copper thickness is a chemical element, and it has a very high thermal and electrical conductivity. It will melt to the top layout of the PCB, for example, a ounce of copper will cover an area of 1 square foot, and it has a thickness of 1.4 mils. The more copper on the PCB, the higher thermal and electrical conductivity, with that, it helps to increase the current-carrying capacity and the resistance to thermal strains. So it relates to our project because when we solder our components to the PCB, we have to make sure that we have our components connect to other components properly. For instance, two LED lights will be soldered on the PCB, if the copper thickness is not enough, then less current will go into both LEDs, and that will cause the function of lights.

### 8.4.2 Components Placement

To place all of the components in order is very important. The shape of PCB is going to be rectangle or other special shapes that our case of device required, and the size of it depends the number of components that we are going to use for our project. There are different sizes of the each chips; the types of size can be DIP, WSON, SIP, SOIC, and SDIP. For our project, the component of stn1110 has two different types of packaging DIP and WSON, stn1110 is the OBD chip that we are using for receiving the data from the car. The difference between two packaging is that WSON has a benefit of smaller size. The size of WSON is usually around 1.27 mm, comparing with the DIP, the size of DIP is triple bigger than WSON. Some components have a closed size when comparing with different types of packaging. For instance, the switching regulator LM2574 has two types of packages, SOIC and PDIP. SOIC has a body size of 8.992mm \* 7.498mm, and PDIP is about 6.35mm \* 9.81 mm. LM2574 with SOIC has 14 pins, and that is why it’s bigger. LM2594 with PDIP has 8 pins. With these two examples, we have to be careful with choosing the packaging of components when we are working on the schematic design.

### 8.4.2.1 Power Supply Placement

The placement of a power supply of our device should be place at the somewhere close to the load. Our power supply is DC to DC from OBD-II to our device, because in order to minimize the interconnection impedance and the conduction voltage drop across the PCB traces, and that will help our switching regulator to reach the best regulated voltage to supply our device.

### 8.4.2.2 Capacitors Placement

In our regulation area, since we picked a switching regulator as a buck converter for our power supply, there are input capacitor for an input bypass, and output capacitor for a fixed output. When we design our schematic for PCB, we have to connect both capacitors directly to the pins, which pin 1 connect to output capacitor and pin 5 connect to the input capacitor without vias, and it’s because it will reduce the connection impedance.

Learn how to design and organize a PCB layout is really important to our project. The size of each component and chip needs to be calculate for our PCB design, so when we design our PCB, we can find a best solution to fit all of the components that we need for our PCB layout. Some components we need to be careful when we design our PCB layout. There are several basic things related to our project that we need to be careful when we are design our PCB layout. The project libraries include three different types of drawing, schematic, layout, and 3D modeling.

## 8.5 Schematic Design Plan

Schematic is the first step of our project H.A.R.A.M.B.E. on the PCB, it control the layout of our project on PCB. The accuracy and perfect of entire layout is the key mission to our project. The schematic show the information of each components on the PCB, and the information will help us to understand the function of each components. The information can be include the pin numbers, names, components values, and ratings. Moreover, the package of each components has the related price and other specifications, such as the size of the footprint for each components. It is really help because when we design our components on the PCB, it helps us to understand the functionality of each components. For instance, the components of switching regulator is on the PCB, it is easy to see the specification of each components, such as the size and types of switching regulators. Sometimes, the component has so many different types and sizes. With the package of the switching regulators, it includes the size and type that we need, so it helps us save a lot of time by looking it online.

The first step of creating a PCB schematic for our project H.A.R.A.M.B.E. is to create a new schematic on a new project file. Once we have the new schematics open, we add our components to create our circuit. Inside of the Add tool, we search the parts we need, and then we can add the parts with the specific type, name, and numbers.

After we add all of the parts that we need, we use the line tool to line each part properly, and connect each part in a shortest wire. And combine them into a circuit. And then, we add the values for each parts such as resistor value, capacitors and inductors. If its chip, and we have to connect the chip to choose parts. Also, make sure the wire connect the part to another part has a proper angle such as 45 degree, but not 90 degree.

Finally, after double checking with the circuit, we open the property of each chip, and change the pin number and pin name of the chip as the datasheet. The reason is that it helps us to see and check the circuit when they have the same pins, otherwise it will get confused. The second reason is that it will be easier for the EAGLE software to check if there is any errors when they have the same pin number and name.

### 8.5.1 Patterns of Components

It gets to a limit when we are using the EAGLE libraries, there are some components that project libraries do not have. Therefore, we have to build or design this part and add it to the libraries. When we design a new part to the project libraries, we have to make sure that we have the same pin number, pin type, and pin name as datasheet of the part.

There are project libraries provided most electronic components in the PCB software, and that means there are footprint, schematic, and the functionality of each components. Therefore, we do not have to design the schematic of the exist components, but we do not have any issues when using these schematics. However, if somehow there are components that the libraries do not have the components that we are choosing, then we have to actually design the schematic of the picked components. If that is the case, we have to be careful the pad spacing with another pad because even with a very small difference, but it still will affect the connection of the components. For instance, the pin of components will have a problem of align properly, because it will be very difficult when we are trying to solder the components to the PCB.

### 8.5.2 Arrangement of regulators

Since we are using switching regulators as a buck DC-DC converter for our power supply, we have to draw or layout the regulator carefully. The switching regulator has complicated design than linear regulator does, but it is more efficient and does not have a lot of wasted power. However, when it gets to layout, we have to lay out the switching regulator step by step, and it is great idea to follow the datasheet closely. With the discussion of spacing early, the switching regulator will be layout at the area around the power supply, which is the connection of OBD-II. Comparing with the linear regulator, it is very easy to implement the circuit inside of it, therefore it is easy to lay out on the PCB, and however, it is not really efficient due to the equation of efficiency.

## 8.6 Layout Design Plan

After checking all of the parts that is correct, we design a layout by using our schematic design. In EAGLE, there is a tool that can generate a layout for the schematic, the tool is called Generate/Switch to Board; that is another benefit by using EAGLE software as our PCB layout. After click the tool, we have all of the parts connected but not place as a circuit layout. Therefore, we need to replace each parts in a shortest distance but not overlap each other, and then move the entire circuit into a specific area to save the area for other circuit. In order to fit all of the chips into one PCB for our project, we have to connect each part with chip in a most efficient way. Also, the layout board is making sure to place each chips in a right area.

### 8.6.1 Single Layers

The number of layers is using depends the size of the device. For our project H.A.R.A.M.B.E., it can be design as one layer or multiple layers. For one layer, it combines every chip and power supply together on one single layer. In another word, single layer includes mechanical layer, ground layer, power layer, and routing layer and solder layer. Mechanical layer is dealing with the information of a specific component. One benefit of that is not dealing with the multiple layers when we are debugging. It is easier to debug PCB since we do not need to worry about other extra layers, such as ground layer, power layers, and additional routing layers. However, one disadvantage of using single layer is hard to find out the errors if there is something wrong, since every part is mixed together, so it might get confused where to look for the error.

### 8.6.2 Multiple Layers

Multiple layers includes everything that a single layer has but with extra ground layer, power layer, and additional routing layers. One reason that we might use the multiple layers is that there is not enough space for all of our parts or Vcc or ground. For instance, some of our chip might need more power, so we might make via through the board in order to supply extra power to the specific chip; and it might causes a case of not enough space, so let ground and power supply Vcc in a another layer will solve the problem. One benefit of using multiple layers is it gives us flexibility of designing the circuit in each layer since there are plenty of space. For instance, the entire power supply of our project is in single layer, and all of the microcontroller chip will be in the second layer, and there is ground layer for all of the parts, and so on. It is very clear to see and easier to design when we are working on the PCB layers. One disadvantage of multiple layers is that it gets tedious when soldering the parts to PCB. Moreover, we are considering to solder the parts by yourself, so it might even harder to solder for a multiple layers. Even with a machine soldering, still it is harder to solder. Also, with the multiple layers, we have a choice to place the last layer as ground layer, and second layer as the microcontroller layer, and other layers which make our device easy to function properly.

### 8.6.3 Via On PCB

In our project, if we decide to use multiple layers for our PCB design, then we have to understand how vias works, and how it will help our project to be succeeded. Via is a hole between two layout boards, and it allows a conductive connection between both layers. There are several different types of vias that we might use while we are working on PCB design, the first is blind vias, it is uncovered on one side of the layout board, which means the other side has vias inside of the layout board. For our PCB, if we are using a multiple layers, then we might use blind vias between ground layer and microcontroller layer, so ground layer will be covered inside of the board but microcontroller does not. The second type is buried vias, which is connecting two layers by using via inside of both layers, which means we can not see it on the PCB. The third type is the thermal vias. It is a vias that will show on both sides of the layer board, it is carries heat away from the power devices.

## 8.7 PCB Routing Design

Once we have our PCB schematic design and layer design, we can start design the ground layer and router layers. Like what we discuss in the previous session, we can have the ground layer and router layer in either single layer or multiple layer. In a single layer, we will use use the auto-routing layer tool to design the entire routing layer for our PCB. In order to make a ground layer for our PCB, we will have to name the selected area as ground, and use the automatic routing tool to draw a ground layer. Once we have our router layer and ground layer, we can use the design rule check tool to check our PCB in case there is any errors. If we do have errors, the EAGLE will tell us what the error is about, so we can fix the error easily.

Tool called auto routing that enables us to route the entire connections and footprint considerations automatically. The main benefit of this tool is that it allows us to save time with simple routings and the tool may be able to change traces automatically if it is allows to do so. For example for the IMU chip there are a total of 24 pins however the footprint for the chip in the library is difficult to route due to its small footprint. So it will be able to give you all possible errors. When using the DRC tool to check the design it is smart enough to display warning and errors. As the design become more complex the tool is susceptible to error. Some errors may include misconnected traces or inefficient routes being generated.



Figure : SaBLE-X Sample Schematic

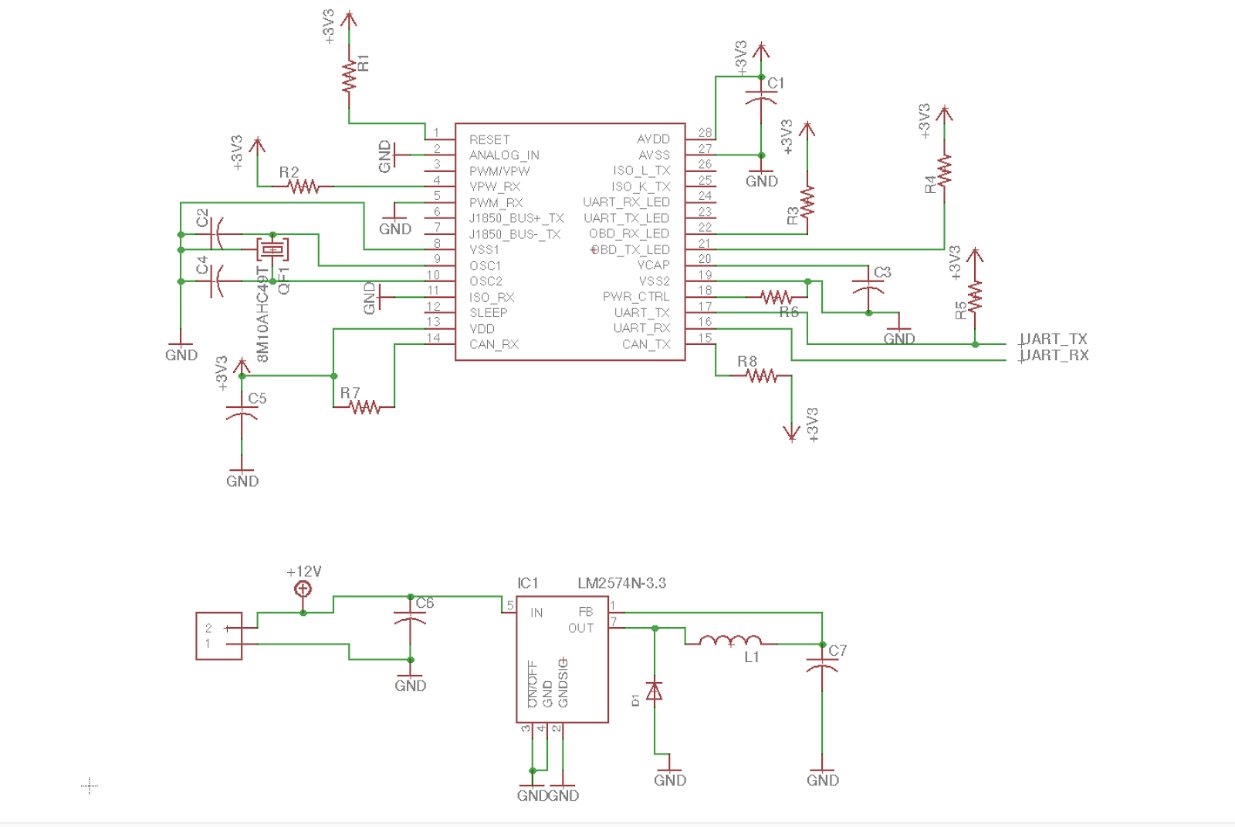


Figure : STN1110 chip schematic

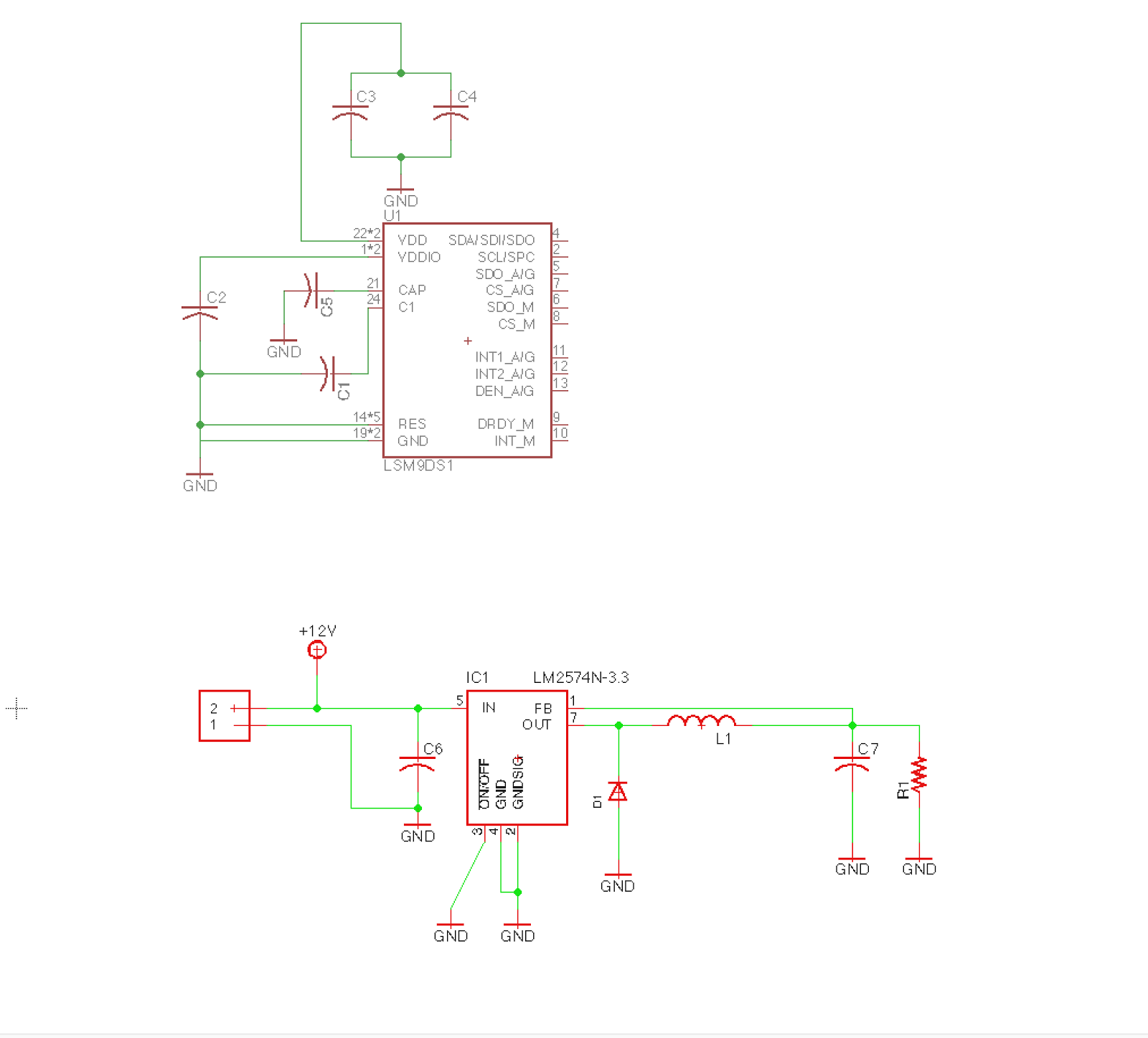
****

Figure : The lMS9DS1 chip configuration. Next to the chip is the buck regulator power supply.

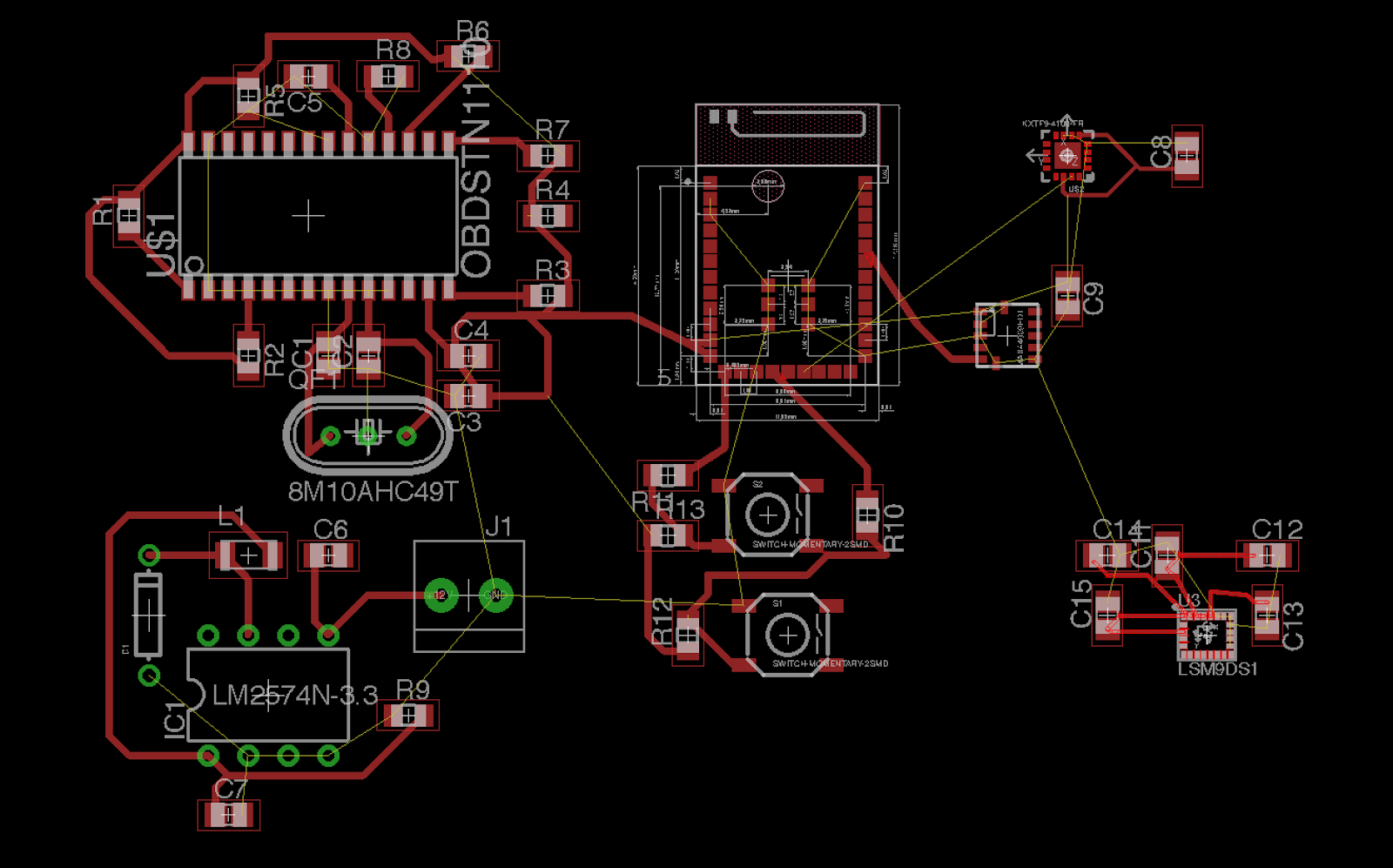
****

Figure : Overall PCB Design

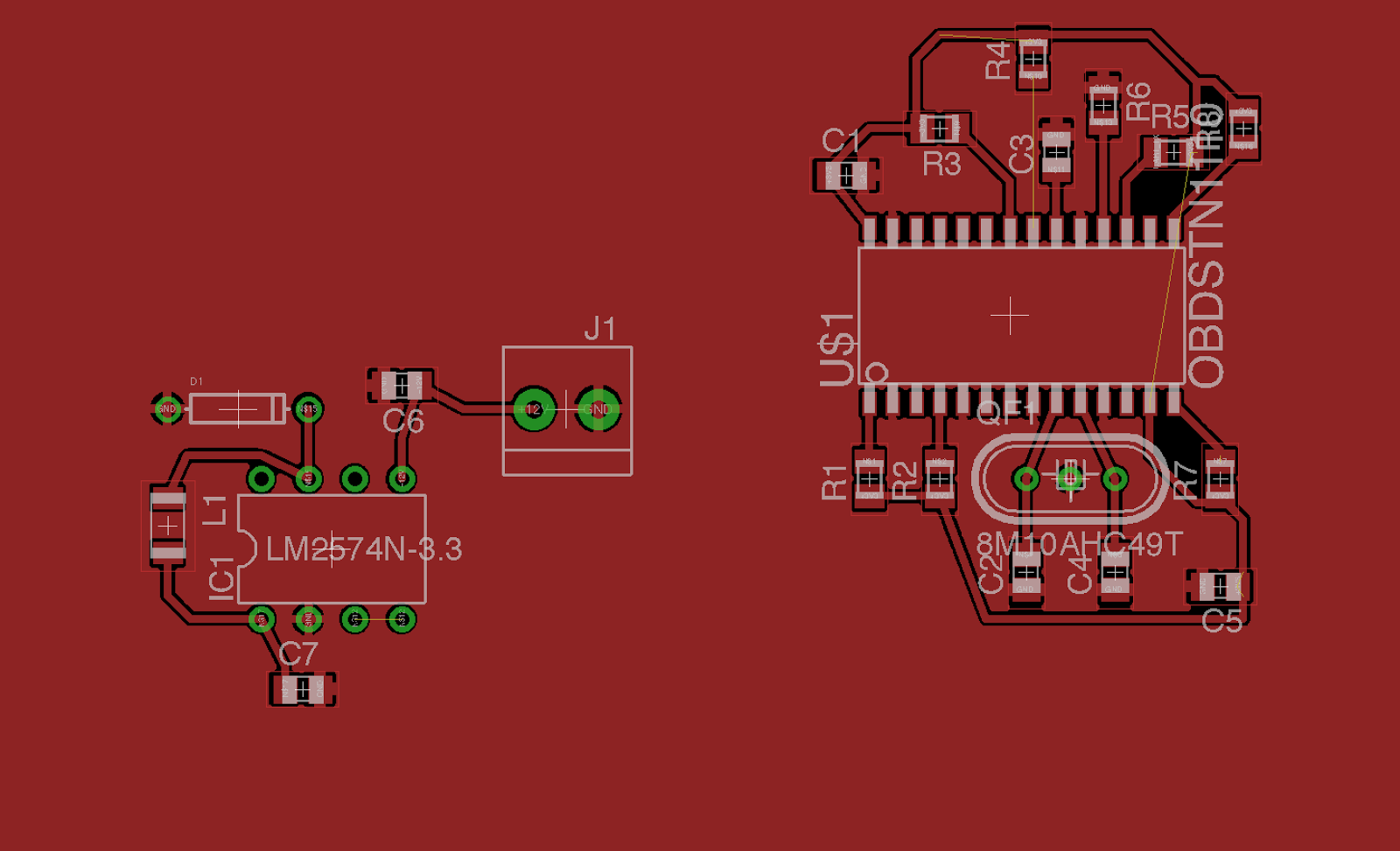


Figure : STN1110 PCB Design

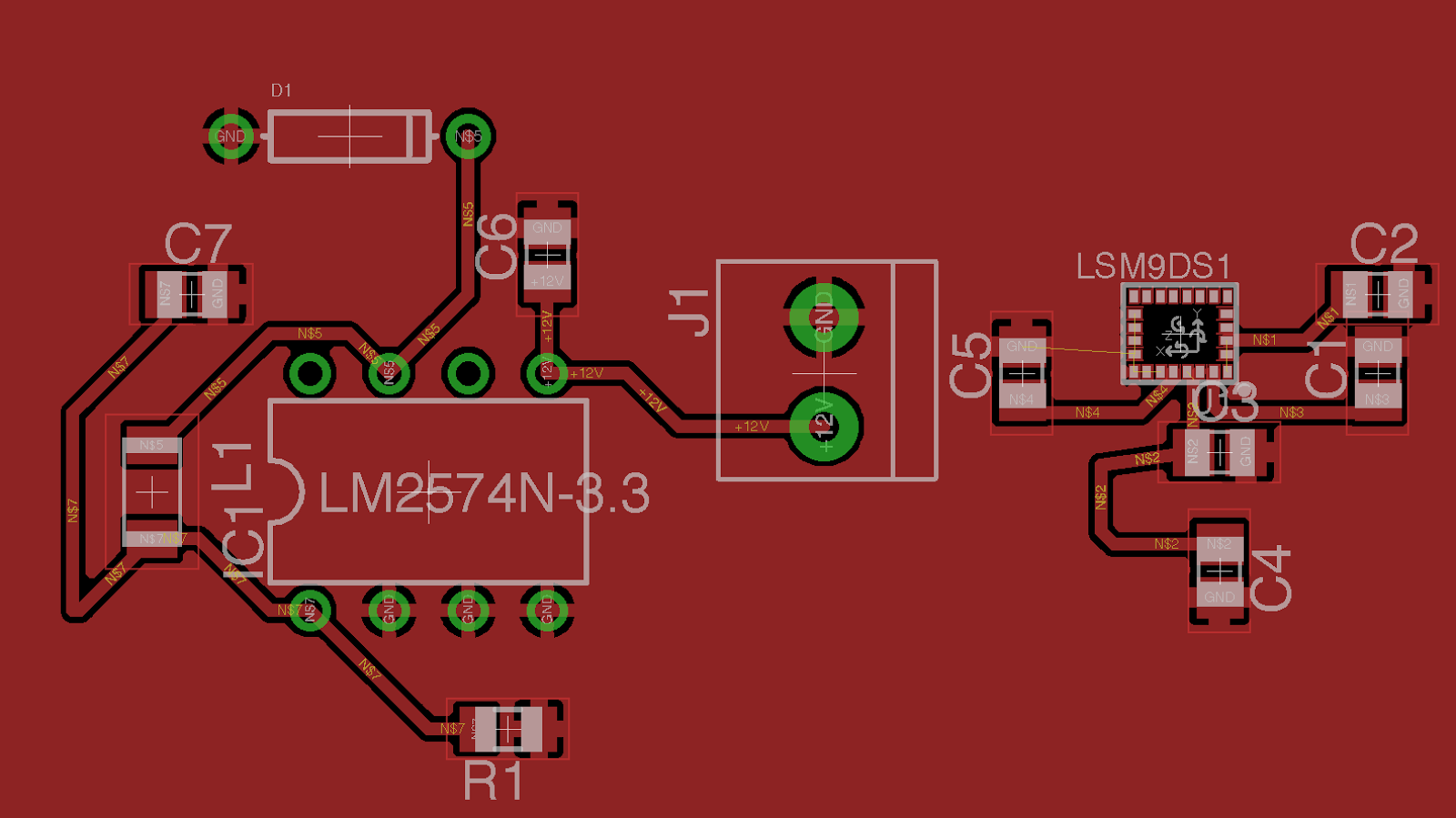


Figure : LM2574N-3.3 PCB Design

# 9. Project Prototypes

In order to prevent the waste of resources and to verify the viability of the project, before sending the PCB design to be fabricated, a basic prototype of the subsystems will be built in order to uncover any mistakes or faults in the design. By testing the functionality of each component in the design process we are better able to understand how they work and how we can get them to work together. To facilitate this the team decided to purchase a majority of the components used in the design, along with their development-edition counterparts. Many of these components were purchased through Mouser Electronics, Digikey, Texas Instruments, and free samples if available. As a breadboard was utilized for prototyping we decided to buy dual inline package (DIP) versions of the components and smaller packages for the final design. If it was not possible to purchase a DIP version of a component the development breakout board was purchased instead. This chapter will detail the major components which were used in the prototyping phase, then show a working variant of the system on a single breadboard.

## 9.1 Prototyping Components

### 9.1.1 CC2650 Launchpad

Though the team will be utilizing the SaBLE-x module in the final design, the underlying microcontroller is the same, the TI CC2640. The Launchpad is using the TI CC2650, however the CC2640 is binary-compatible. This development board seems to be the best way to develop for the CC2640. The Launchpad contains various LEDs, JTAG headers, and an onboard XDS100 JTAG emulator with a USB port for programming. The team ordered one of these boards for development and test BLE communications between a mobile device and itself. Additionally, the board has schematics openly available and TI provides example code which the team can reference when designing the software/hardware.

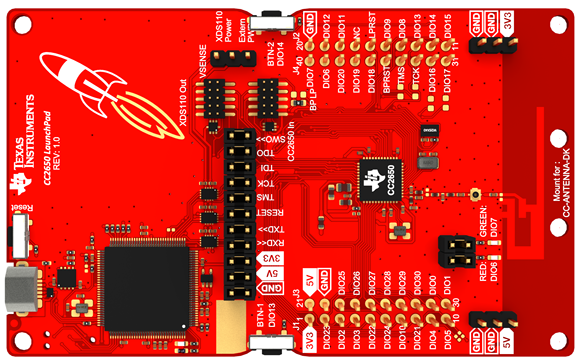


Figure : CC2650 Launchpad

### 9.1.2 LSM9DS0 Accelerometer/Gyroscope/Magnetometer Breakout Board

In order to test the accelerometer we are using in the project we must have a breakout board since it does not come in a DIP package. Adafruit manufactures a breakout board for the LSM9DS0, which is who we purchased this board from. This will allow us to gather accelerometer and gyroscope data from our prototype system. In the final design we will be utilizing the LSM9DS1, but the data is received in the same format. Another difference is that we will purchasing the SPI variant of this IC for our final design, so the software will have to be slightly modified to support that.

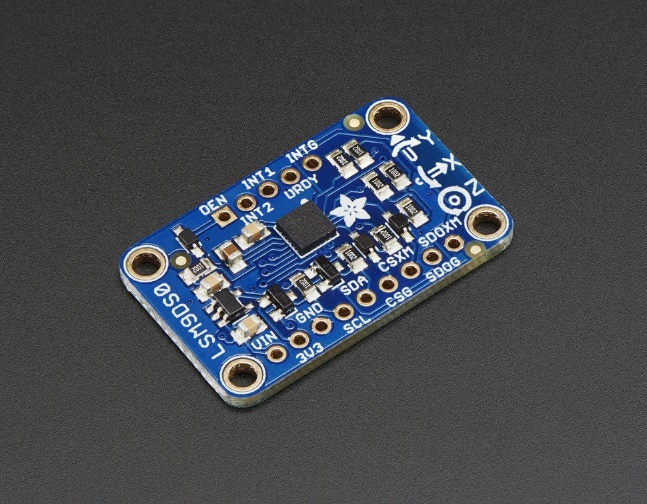
****

Figure : Adafruit LSM9DS0 Breakout Board

### 9.1.3 STN1110 Sparkfun OBD-II Board

In order to communicate with a vehicle we will need to communicate with an IC that is able to communicate with the vehicle over its OBD-II port. The most widely used option is the Sparkfun OBD-II UART board. We used this board in the viability testing of the project in order to gather data from the OBD-II interface. The board is able to be controlled by sending commands over UART to a terminal on the device. The device is ELM327 compatible, so a code list is available online to receive data from the chip. In addition to having the same chip that we’re implementing in our final design there are also schematics openly available that we can use as a reference for our design.

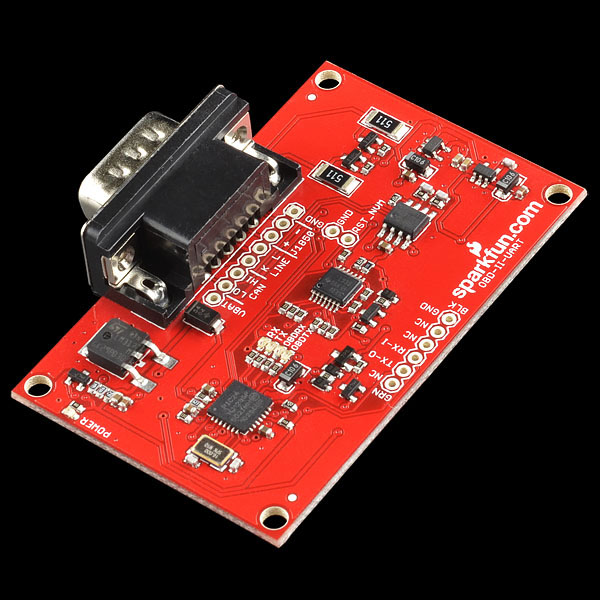


Figure : Sparkfun STN1110 UART Breakout Board

### 9.1.4 SaBLE-x Breakout Board

In an effort to test code on the actual platform we purchased a breakout board for the SaBLE-x IC we’re using in our final design. This is a cost-effective alternative to the development kit, and we can be sure that we don’t have any extra peripherals connected when prototyping. This will serve as a companion to the CC2650 launchpad when developing the software for the main module.

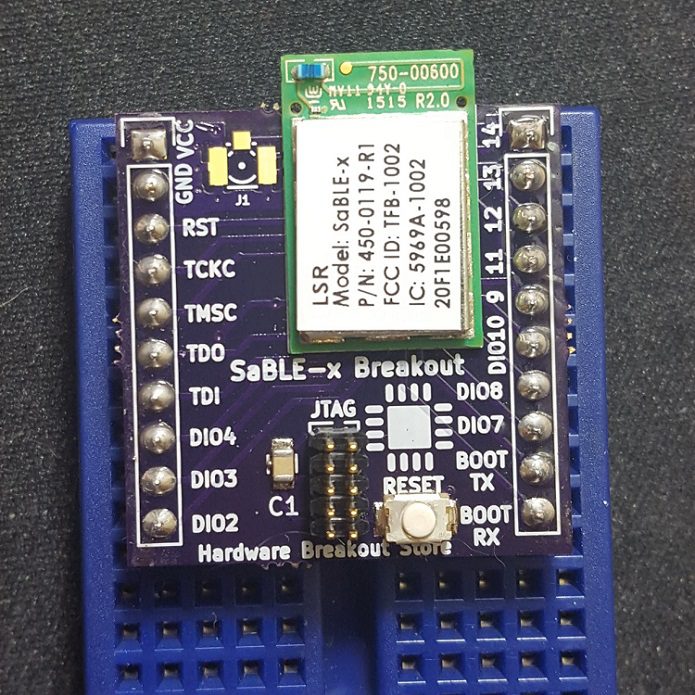


Figure : SaBLE-x Breakout Board

### 9.1.4 LM2574 Buck Switching Regulator

The buck regulator was tested to see if we could get a proper constant voltage with a variable input supply. While prototyping the oscilloscope seemed to be plotting the voltages correctly but unfortunately.

## 9.2 Breadboard

In order to prototype the components for this project the best option is to connect all of the components on a breadboard. This allows for rapid prototyping using jumper cables to connect the components. Once we are completely sure that the desired functionality of the device is realized we will be able to send out a PCB that directly models the circuit we created on the breadboard. Another advantage of using the breadboard for prototyping is that we will be able to replace components in the event that they fail and the rest of the system will be intact. Also, if some components end up not fitting within the specifications we can buy different components and replace the non-useful ones. Figure 49 shows the breadboard with components that we used for prototyping. This figure contains the STN1110, the CC2650 and another microcontroller that was being used to control signals for the regulator.

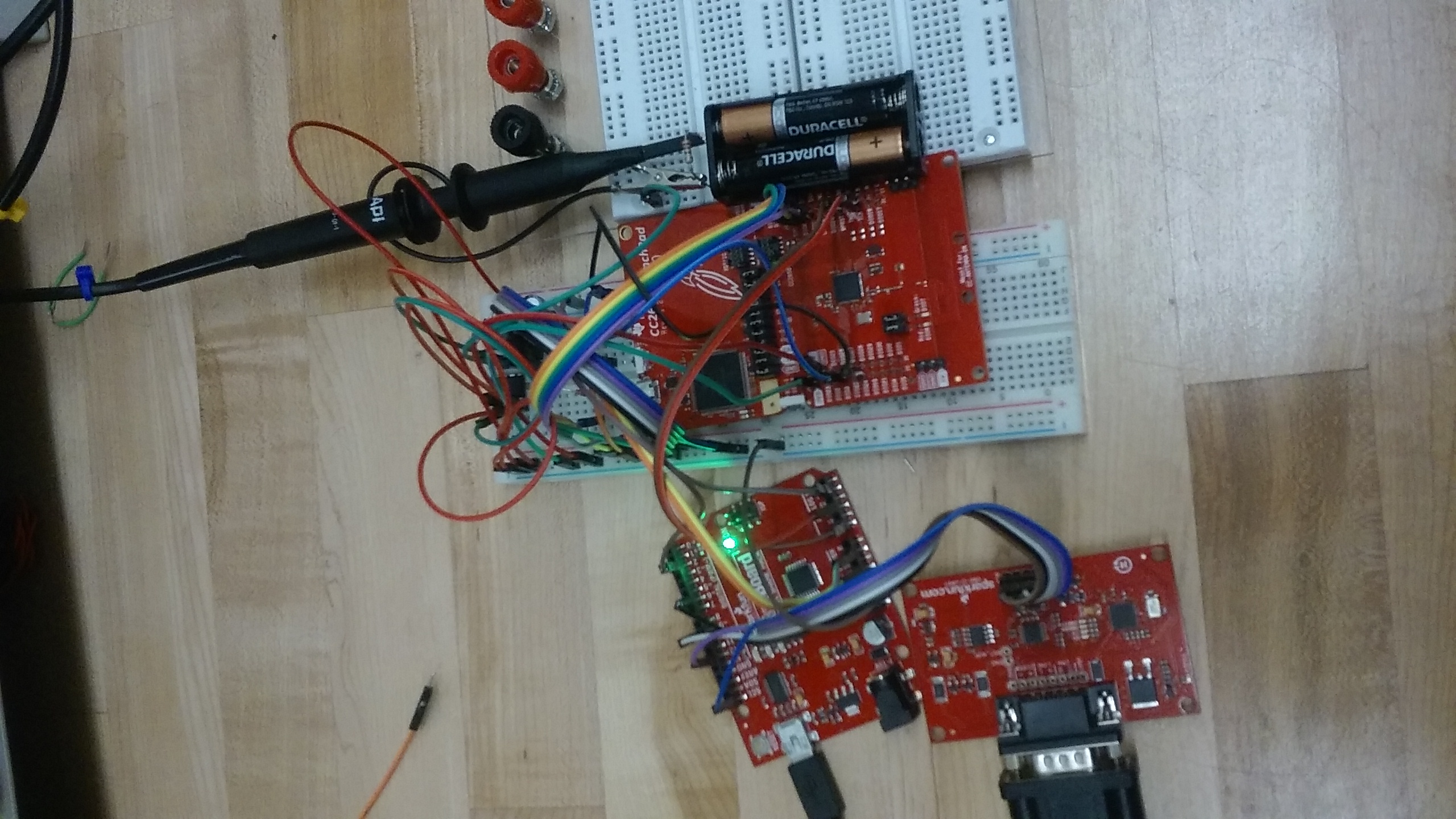


Figure : Breadboard Prototype of H.A.R.A.M.B.E

# 10. Testing

Testing is crucial during and after the development phase of a project. Testing is what allows for errors and mistakes done in the development phase to be discovered, so that they can be fixed. For our project, we are splitting up testing between the hardware and software developments of the project. For example, testing in hardware is required to determine whether the power supply we have chosen is sufficient enough to power the device at the necessary times. Some hardware modules will have connections to software modules, so testing the interaction between both modules is a must. The sensors we will be using, specifically, will require considerable testing in both software and hardware sections of the development phase. The software module must be able to interact with the hardware module without errors at all times. We will be using many different test plans to ensure that the software module does not fail if something wrong happens. Further testing will be necessary in the hardware module as it will be more susceptible to problems in not only the software, but its connection to other hardware modules.

If an error is found in the software, the JTAG will be used to do further testing. The JTAG will make debugging a lot easier as it will help us understand what is exactly happening in the code. Software development will be done to program the OBD-II IC, MCU, the corresponding sensors will be using. Of course, we plan to add other components that will require some sort of programming as the development phase of the project progresses. Initial testing will help us reveal problems in the hardware such as modules not operating correctly and/or connections failing. Once the hardware has been tested properly, the software will be tested on a few set test criteria. The most important software testing we will be doing is to make sure that different modules are able to receive and transfer their data correctly. Our project relies heavily on proper communication between different modules, so it is necessary to allocate more time in testing the software.

## 10.1 Hardware Testing

This section will cover the software and hardware testing for our project. It covers the different ways we will be following to test each chip. This section will also cover the process for each test, as well as a conclusion for each test.

### 10.1.1 Hardware Testing Environment

There will be initial and final tests of the hardware prototypes for our device H.A.R.A.M.B.E.. The initial tests will be tested in the Electrical Engineering lab. The reason for this is due to the quality of the tools and ease of access of this tools. We will need to use these tools to test modules in our System. This testing will include our chosen power supply in the lab to supply the input voltage of switching regulators. For example, we will use these to tools to test the power supply by supplying switching regulators some input voltage.

The initial tests are going to be at the Senior Design Lab, Engineering building 1, Room 456. The equipment that will help us test our project are oscilloscopes, power supply units, function generators, digital precision multimeters, breadboards, and resistors and capacitors. Therefore, when we tested our switching regulators, we used the wires and capacitors to build the circuit for the chip LM2574 and LM2594. Both these chips are switching regulators. We were able to save money by using these tools because we did not need have to purchase any of them. We will using the main battery of the car as the power supply of our unit. This power will come directly from a pin from the OBD-II.

The final tests for our system will be done outside, because our device will be connected to the car, and H.A.R.A.M.B.E. will be receiving power and data from OBD-II in the car. All of the chips will be combined into several breadboards, and we can test the entire project to prove if it works or not. We can also test our device indoor, because we can use the software to input the data, and our device will work as long as it receives the data of speed and acceleration. Based on these data, our device will either stay turning on, or call emergency contact if there is a wired speed or acceleration. Moreover, we do not need to worry about the power supply for our project since equipment are available in the lab.

### 10.1.2 Hardware Chips Testing

The power supply we tested was the car’s battery. We received a 12.51 V when the car engine is off, and we received a 14.23 V when the car engine is on. With the cigarette lighter receptacle, we received a voltage of 4.97 V and a current of 335.2 mA when connected with USB port in on mode. After we tested the output power supply from the car, we can then move on to our next step to test the two switching regulators. In order to test the switching regulators, we decided to build the circuit on the breadboard. By using the power supply in the lab we can change the input voltage from 12V to 15V to test the switching regulator. Furthermore, we can test the efficiency of both regulators by using an oscilloscope and we can use the multimeter to measure the output voltage of the switching regulator.

### 10.1.3 System Testing

We will be using two LEDs on our device. The first LED will be green which will tell us that the device is on. The second LED will be red which will tell us that the device is not working properly. One of the pins on the microcontroller will be used to supply power to both LEDs. This will be represented as a simple circuit on the breadboard. The way we are testing the green LED will be by connecting our device to the OBD-II port and turning on the car. Once the car is on, the green LED should turn on meaning that the system is working properly. On that other hand, if the green LED is off that means that the system is not working properly. The same procedure will be followed to test the red LED.

### 10.1.4 PCB Testing

After PCB is shipped, we will need to be able to test whether it works or not. The first thing to check is whether there any cracks, burns or broken solder joints on the PCB, if there are, then we have to fix them before we connect to the power. The second thing is to use the multimeter to test is there any unwanted shorts between two components, in order to see any shorts, we set multimeter to the current mode, and attach the red cable to each pin and attach to another pin, if there are sound after attached, then it means there is a short between two pins, use the schematic on the EAGLE to verify the short is right or not. If it is unwanted short, then we have to fix it. Next, with the device installed and switched on, attach red cable of multimeter to the OBD-II connector, and attach black cable to the ground, if it does not read 4.9V to 5V, then we need to check the connection at OBD-II, also check the car battery as well. The third thing is to test the ground layer of the PCB, we use the multimeter to connect the ground layer and any positive node, if it gives us a positive voltage, which means the ground layer works properly. The fourth thing is to test the power of the PCB, in order to test the power if is good or not, we plug the power pin from device to the output power pin on the OBD-II, and if the LED green light is on, means our PCB works. Also, attach the red cable of multimeter to the input pin of microcontroller, and attach black cable to the ground pin, it should be read 3.2V to 3.4V, if it is less than 3.2V, we need to check the car battery and connection of the OBD port.  If it is greater than 3.4V, then we need to check if there is any short between two pin.

### 10.1.5 Power Supply

|  |  |
| --- | --- |
| Test | Expected Result |
| Attach red cable of multimeter to the power of regulator, and attach black cable to the ground. | It should give us a 3.3V, if it is not, then there might be a short, open circuit, or improper feedback to the device. |
| Attach red cable of multimeter to the input of the MCU, and attach black cable of multimeter to the ground. | In order to run the MCU, it should read a 3.3V. False reading might caused by short circuit or open circuit. |
| Measure the capacitor between pin 7 and ground on the LM2574 switching regulator | The capacitor value should be 220 uF |
| Measure the capacitor between pin 5 and ground on the LM2574 switching regulator | The capacitor value should be 22 uF  Ensure the ground is not floated. |
| Measure the inductor value between pin 7 and load on the LM2574 switching regulator | The inductor value should be 330 uH |
| Measure the output voltage pin 16 on the OBD-II port | The output voltage should read as 12V - 14V. If not the connection is open. |
| Measure the capacitor between pin 8 and ground on the LM2594 switching regulator | The capacitor value should be 120 uF |
| Measure the capacitor between pin 7 and ground on the LM2594 switching regulator | The capacitor value should be 68 uF |
| Measure the inductor value between pin 8 and load on the LM2594 switching regulator | The inductor value should be 100 uH |

Table : LM2594 test criteria

### 10.1.6 OBD Port

|  |  |
| --- | --- |
| Test | Expected Result |
| Make sure input pin of the switching regulator is connecting to the output pin of the OBD-II, use the multimeter to test the output voltage | input voltage of switching regulator should be 12V - 14V |
| Connect the pin 15 on the microcontroller to the pin 14 on the OBD-II | The LED should be on when the signal is transferring to the microcontroller from OBD-II |
| Connect the ground on the OBD-II to the ground pin on the microcontroller, attach the red cable of multimeter to ground pin of OBD-II, and attach the black cable to the ground pin of the OBD-II | The voltage between two node should be really small, nearly to zero. |

Table : OBD-II port test criteria

### 10.1.7 Microcontroller

This is our microcontroller chip that we are going to use, there are 39 pins. And when we have our PCB ready to test, the table below is the expected result of each pin test.

|  |  |  |
| --- | --- | --- |
| Pin Number | Test | Expected Result |
| 1 | Attach positive cable of multimeter to pin 1, and negative cable to the ground | The resistor value should be 50 ohms. |
| 2 &3 & 33 & 34-39 | Attach positive cable to pin 2,3,33,and 34-39, and negative cable to the ground | The voltage value should be zero |
| 4 & 5 & 9 & 10 &25-28 | Visually check the connection of pin 4 | There should not be any wire connect to it, this pin is not use |
| 6 | Attach positive cable of multimeter to pin 6, and negative cable to the ground | The resistor value should be 100 k ohms. |
| 7 | Visually check the connection of pin 7 and signal pin on the OBD-II, | Both pins should be connected |
| 8 | Visually check the connection of pin 8 | Both pins should be connected |
| 11 & 12 | These two pins are the power supply pin. Attach the positive cable of multimeter to pin 11 & 12 individually, and attach the negative cable of multimeter to the ground | The voltage across two nodes should be 3.3V. If it’s not, check the OBD-II port and car battery. |
| 13 - 19 | Attached the positive cable of multimeter to the pin 13-19 individually, and attach the negative cable to the other side of the wire | The current should be around 100mA to 300mA |
| 20-24 & 29-32 | Attached the positive cable of multimeter to the pin 20-24 & 29-32 individually, attached the negative cable to the other side of the wire | The current should be around 100mA - 300mA |

Table : Microcontroller testing criteria

### 10.1.8 Accelerometer

|  |  |  |
| --- | --- | --- |
| # Pin | Test | Expected Result |
| 1&3 | Attach the positive cable of multimeter to the pin 1, and attach the negative cable to the ground | The voltage should be around 1.9V to 3.6V, if its not, check the connection between MCU and Accelerometer |
| 14 | Attach the positive cable of multimeter to the pin 1, and attach the negative cable to the ground | The voltage between two node is zero. If it is not, there might be some open circuit or connection problems |
| 15 | Attach the positive cable of multimeter to the pin 15, and attach the negative cable to the ground | The voltage between two nodes should be zero |
| 16 | Attach the positive cable of multimeter to the pin 16, and attach the negative cable to the ground | The voltage between two nodes should be zero |
| 17 & 18 | Attach the positive cable of multimeter to the pin 17&18 individually, and attach the negative cable to the ground | The voltage between two nodes should be zero |
| 22 & 23 | Measure the two nodes on the pin 22 and 23 | The voltage between two nodes should be 1.9V 3.6 V |
| 24 | Attach the positive cable of multimeter to the pin 24, and attach the negative cable to other side of the wire | It should read a capacitor value of 100 nF |

Table : Accelerometer testing criteria

### 10.1.9 STN1110 Multiprotocol OBD

STN1110 is our OBD to UART interpreter IC. The packaging of STN1110 that we are going to use is SOIC, and there are 28 pins on the chip.

|  |  |  |
| --- | --- | --- |
| Pin Number | Test | Expected Result |
| 1 | Measure the resistor between pin 1 and input source of 3.3V | The resistor should be 10k ohms, if it’s not, change the resistor value or check the connection of both nodes. |
| 2 | Attach the red cable of multimeter to pin 2 and attach the black cable to the other side | The voltage between two nodes should be zero V. |
| 3 | Visually check the connection at pin 3 | There should not have any wire connected to it. |
| 4 | Measure the resistor between pin 4 and input source of 3.3V | The resistor should be 100k ohms, if it is not, then change the resistor value or check the connection of both nodes. |
| 5 | It’s PWM\_RX, attach the red cable of multimeter to pin 5 and attach the black cable to the other side | The voltage between two nodes should be zero V |
| 6&7 | Both pins are J1850 Bus, not use | There should not have any wires connected to it |
| 8 & 11 &19 | Use the multimeter to check the voltage between pin 8/11/19 and ground | The voltage should be 0 V |
| 9 | Use the multimeter to check the capacitor between pin 9 and ground | The capacitor value should be 30 pF |
| 10 | Use the multimeter to check the capacitor between pin 10 and ground | The capacitor value should be 30 pF |
| 13 | Use the multimeter to check the voltage at pin 13, also attach red cable to the pin 13 and black cable to the ground | It should read a input voltage of 3.3 V, the capacitor value should be 1 uF |
| 14 | Use the multimeter to check the resistor value between pin 14 and pin 13 | The resistor value should he 100k ohms |
| 15 | Use the multimeter to check the resistor value  between pin 15 and input source 3.3V | The resistor value should be 100k |
| 17 | Use the multimeter to check the resistor value between pin 17 and input source 3.3V | The resistor value should be 1k |
| 18 | Use the multimeter to check the resistor value between pin 18 and pin 19 | The resistor value should be 100k |
| 20 | Use the multimeter to check the capacitor between pin 20 and ground | The capacitor should be 10uF |
| 21 & 22 | Use the multimeter to check the resistor value between pin 21/ 22 and input source 3.3V | The resistor value should be 100k ohms |
| 23 & 24 & 25 & 26 | Not use, visually check theses pins | There should not have any wire connected |
| 27 | Ground node | The voltage should be zero V |
| 28 | Use the multimeter to check the capacitor value between input source 3.3V and ground | The capacitor value should be 1uF |

Table : STN1110 testing criteria

## 10.2 Main Module

The main module will be connecting every part of the system into one, so that they all work together and properly. Dedicated testing will be required to ensure that each piece of hardware connected to the main module works properly. Software pieces of the project such as UART, software serial, Bluetooth, sensors, etc, will require a lot of tedious testing to make sure it works properly. The JTAG will be used to debug most of the software while and after the development phase. Memory constraints will limit the amount of code we will be able to develop, but it should not stop us from employing the required functionality of the modules. The most important part of testing for the main module is to ensure that the communication between each module is working properly. We will be analyzing data from the OBD-II port of the vehicle to determine whether it has gotten into a car crash. We will also be using sensors that will be getting information from the car in terms of position and acceleration. We will need to do careful testing to make sure that their connections work properly and the data sent is correctly.

### 10.2.1 UART and Software Serial

The MCU will be using UART for some of its communication. It is essential that this communication works as intended and there are no problems with the connection and the transfer of data. UART communication testing is based on whether the correct information is transferred and received by the corresponding modules. The software for the UART will be composed of setting up proper communication configurations so that different modules can understand each other. We will be performing adequate testing in the software by using the JTAG and other debugging tools. In testing, if we find out that one of the modules is transmitting wrongful information, a fault in the code is the biggest possibility. It will be important to have appropriate error checking so that problems can be fixed easier. Testing, just like for any other software module, we consist of some trial and error to get the module up and running.

### 10.2.2 Bluetooth profile testing and communication

Proper Bluetooth communication is necessary to ensure that the correct information is received by the Android phone application from the MCU. We will be performing the test criteria listed below in the table to make sure the Bluetooth communication in our project is working properly. Bluetooth communication relies more on proper software configuration than hardware configuration. Due to this, we will focusing more on testing whether the software is working as intended. For the test criteria, we are narrowing it down to making sure the Bluetooth module responds to our commands. We need to make the Bluetooth module turns on properly. The MCU, which hosts the Bluetooth chip, should properly pair with another Bluetooth enabled device, in this case, an Android phone. Once paired, the Bluetooth module should be able to send data to the Android phone. In initial tests, this data does not have to be actual OBD-II information. For later testing, it is necessary to test for actual OBD-II information received by the Android application from the OBD-II. Overall, the Bluetooth module should properly pair with a cellular device, send correct information, and close the connection once the Bluetooth connection has been closed. Our testing criteria covers most of the general testing required to test whether the Bluetooth module works properly. We plan to test other Bluetooth configurations as well.

|  |  |  |
| --- | --- | --- |
| Test Number | Test criteria | Correct Result |
| 1 | Bluetooth module is initialized and initiated. | Bluetooth connection is shown in the Android application’s Bluetooth connection menu. |
| 2 | Bluetooth module connects successfully | Data is sent from the MCU to the Android application |
| 3 | OBD-II data is sent from the MCU to the Android application | The correct OBD-II data is sent from the MCU to the Android application |
| 4 | Bluetooth module disconnects properly | Bluetooth connection disappears from the phone signaling and proper disconnection. |

Table : Testing criteria for the Bluetooth module

### 10.2.3 JTAG debugging

The JTAG will be used to debug for problems in the software. The JTAG is a protocol especially design to program and debug microcontrollers. Things like circuit emulation and state of the program in the microcontroller while its running are possible through the use of a JTAG. We are putting a heavy focus in programming and debugging with a JTAG as it will allow for more efficient coding. If a problem arises due to an unknown bug, it will possible to pinpoint exactly why the error is occurring. This is because the JTAG is an in-system debugging tool which allows you to manipulate and examine status while a circuit is running the system. This execution can be stopped at any time, so debugging with the JTAG gives the user a lot of flexibility when testing the system.

### 10.2.4 Memory Constraints

Memory constraints are of real concern when building a device. While building the software for the hardware, it is necessary to always be wary of how much space you have left to make sure you do not run out of space. It is also important to make sure that code is written so that it does not waste memory. We plan to test for memory wastage and plan to refactor code in areas where the code can be improved. The software will be built following the MISRA (Motor Industry Software Reliability Association) standard. This standard aims to facilitate code safety, security, portability, and reliability in the context of embedded systems. By following this standard, we aim to reduce code usage and stray away from back coding practices. For example, some of the MISRA rules are as follows: initialize variables to avoid compiler differences, avoid using functions and constructs that are prone to failure and correct naming conventions and commenting. Overall, good coding practices will allows us to avoid memory constraints in the MCU and other software related development in the project.

### 10.2.5 Extended use effects

The device we are creating is intended to be of use completely during a trip. This trip could last from a simple ten minutes to more than a day of constant driving. During this time, our device should perform without any problems. It will constantly be asking the OBD-II for vehicle information at all times. A backup power supply will be installed to our device to ensure, that if the main battery supply (vehicle’s battery pack) fails during a car crash, the device will still be able to send a signal to emergency services. We will evaluating the test criteria by employing real and simulated conditions. For example, we will test whether the device is working properly after a one hour trip. Information should still be properly sent from the OBD-II port to the MCU and the Android application. For longer durations of travel, we will be performing simulated trips by sending data from the OBD-II to the Android application for various amounts of time. Although it might not be completely accurate, it will help us test how well the device performs.

|  |  |  |
| --- | --- | --- |
| Test number | Test criteria | Correct test result |
| 1 | Device during a 1 hour trip( MCU, sensors, etc). | Data from the OBD-II is properly received by the phone application. |
| 2 | Device during a 6 hour trip( MCU, sensors, etc). | Data from the OBD-II is properly received by the phone application. |
| 3 | Device during a 12 hour trip( MCU, sensors, etc). | Data from the OBD-II is properly received by the phone application. |
| 4 | Device during a 24 hour trip( MCU, sensors, etc). | Data from the OBD-II is properly received by the phone application. |

Table : Testing criteria for extended use effects

## 10.3 Sensor Module

The sensor module will be tasked in receiving information from the environment. We intend to receive a vehicle’s acceleration and position data with the sensor module. Testing criteria for the sensor module will require both testing in the software and in the hardware. Testing the hardware for the sensor module involves testing whether it’s getting the correct amount of voltage/current. The sensor module needs to also be hooked up properly to the MCU, so that they can communicate with each other properly. The software for the sensor module will configure the sensors to receive information from the environment and relay it somewhere else. We intend to use the data obtained from the sensor module to determine whether a vehicle has crashed.

## 10.4 OBD-II IC

The OBD-II IC will be used to send PID codes to the vehicle’s OBD-II port. These codes are in the form of hex numbers representing the type of information the OBD-II should return back. Since the OBD has many different modes of operation, further testing will be done to determine exactly what information we need from the OBD-II. For example, it will be important to get the vehicle’s speed from the OBD-II, but getting the RPM might not be useful at all. We plan on retrieving information, if any, of current problems a vehicle may have from the OBD-II. Getting this information will be important because it will help in determining if a car has gotten into a car crash. As of yet, we are not completely certain the retrieval of this information will help us in identifying whether a car has gotten into a car crash. Further testing will be necessary to determine whether current vehicle problems will help us in evaluating a car crash in real time.

|  |  |  |
| --- | --- | --- |
| Test Number | Test criteria | Correct test result |
| 1 | MCU communicates with the OBD-II IC | A request for OBD information from the MCU is received by the OBD-II IC |
| 2 | OBD-II IC sends a PID code to the OBD | Information is received by the OBD-II from the OBD |
| 3 | OBD-II IC sends the information back to the MCU | Information is properly received by the MCU |

Table : Testing criteria for the OBD-II IC

### 10.4.1 Command Execution

Testing will be done to ensure that PID codes sent to the OBD-II during command execution returns the proper information. Command execution is very important as it ensures that the PID codes get sent to the OBD-II. We intend to manually perform command execution of the PID codes at first to determine whether the information is properly received. Once we are certain that it is working properly, our intentions is to automate the process of command execution at real time speeds. The OBD-II IC will constantly be sending and receiving information from the OBD-II, so the correct speed at which we can prompt the OBD-II for information will require careful testing.

### 10.4.2 Data validation

The data received from the OBD-II IC will require manual validation at first. Initial testing will determine that the information received from the OBD-II IC is correct and valid. Once initial testing confirmed that manual validation of the data is correct, we intend to automate this process. The software will check whether the data received from the OBD-II is correct. If the wrong information is received by the MCU, a reboot of the communication software will be necessary to restart the module.

## 10.5 Android to Main Module communication

The MCU and Android application will always be communicating with each other. The data from the OBD-II will be sent to Android application. Although the Android application will mainly be used as a communication device to contact friends/family/emergency services in the event of a car crash. The Android application will also help us in debugging some of the data arriving from the OBD-II. Once we have set up the whole system, we won’t be able to discern the data as it travels through the intermediaries. Our only choice in debugging the information on whether it is correct or not is by sending the data from the MCU to the Android application and vice versa. There will also be hooks in the software to detect whether wrongful OBD-II information has been received. To initiate the connection between the MCU and the Android application, the Bluetooth module in the MCU will be used. Testing will be necessary in not just the information obtained, but in the Bluetooth connection between the MCU and Android application.

### 10.5.1 Bluetooth listener

The main connection for Android to main module communication will be a Bluetooth connection. The Bluetooth module is located in the MCU and will be configured to pair with an Android application. In this case, we will be creating an Android application specifically for contacting emergency services in case of a car crash. We will also have a feature whether the user will have the opportunity to add custom contacts to the list of contacts the application should contact during a car crash. Since the Android application will be communicating to the MCU with a Bluetooth connection, we will not be using the Android application to do data analysis. There might be a small delay in the Bluetooth connection between the Android application and the Bluetooth connection, so it might not be beneficial to do data analysis in the phone application. The phone application will be beneficial for debugging purposes as it will help us determine whether the OBD-II IC is working properly. If the data from the OBD-II sent from the OBD-II IC from the MCU arrives properly to the Android application then we can say for sure that each module is working properly. We will still need to do a lot of testing to make sure that the data is correct and not wrong and that the data arrives at almost real time speeds. The Bluetooth module will help us a lot in getting help in the event of a car crash, as well as using it for debugging purposes. Overall, the Android application will be the face of the project that users will interact with, so it must be tested a lot to make sure everything is working as intended.

# 11. Project Management

In this section we will cover the structure of our project H.A.R.A.M.B.E. We will look at how we managed the project and how we managed role distribution in our group. Also we will look at individual milestones for both semesters of Senior Design. Moreover, we will explain labor division and the finances and budget for our project.

## 11.1 Management Plan

It is important for any group working together on a project to have a clear understanding of project management. Proper project management includes, but is not limited to, the finances of a project, job responsibilities, time constraints, and the teamwork involved. For a team to be successful, they must be determined to follow proper project management from the beginning. . For starters, the finances in a group needs to be handled properly. A budget needs to be created, so that teams know how much money they are able to spend individually. If the budget is not followed closely, the group will see itself broke in no time. It can be quite a difficult ordeal to have group members not break the budget. In the instance where a group continues to grow, it makes it much harder to track how each person is spending money. This is why, in a group, usually one or a few responsible members of the group tend to handle the money. This method ensures that the money in a group is spent fairly and the budget is not destroyed in a single day. Another crucial aspect of project management that a team must understand is being able to split up job responsibilities of each member fairly. By perfectly splitting up the project amongst the members of the group, you can be confident that the project will be completed on time. The hard part of job responsibilities is finding the proper job for a group member. Sometimes individual skills from one member to another member might be similar making it harder to split jobs based on their strengths. Usually, if the group member has the option, they may decide to choose a few different things in the project they would not mind doing. This will help spread the job responsibilities for each member, so that each group member has the opportunity to work on something they enjoy. If the group does not have the option to choose, they will usually be forced to work on a random part of the project, not based on either their strengths or interest in the part, but because the project requires work done in that area. Time constraints is another big issue that often plagues a group. Similarly to the finances, the group members need to devise a plan on how to tackle the project it is undertaking. If the group waits too long to start working on the project, they might be unable to complete the project. It is important to stay on task and make sure to understand the deadlines, so that the project can be finished on time. An aspect of project management that is sometimes overlooked is the teamwork involved. In an ideal group scenario, a group needs to be able to cover each other weaknesses. Teamwork is critical, in terms of communication and getting along. In general, if a group understands the basic requirements of project management and understands how to work together in group, they will have no problem in successfully completing the project.

## 11.2 Milestones

In the beginning of the semester each group member created their own milestone table for both Fall and Spring semesters. These milestone tables detailed our achievements as we completed them. Although the table does not mention any individual work for each member.

The first group is the software team. These group did most of the software parts that were related to our project, such as the Bluetooth module, GPS module, microcontroller module, OBD-II module and so on. For the software part, we purchased the CC2640 chip, accelerometer. We started preliminary testing each module. For the OBD-II, we were able to receive some data from the car, such as acceleration, velocity, and so on. The second group is the hardware team. The hardware team worked mostly on the hardware parts of the project. For example, some of the hardware parts includes the power supply of the microcontroller from the OBD-II. It will also be the power supply for the LED. The hardware team also worked on the backup battery for the PCB. In addition, the hardware team will be working mostly on the PCB. For testing, we tested the switching regulator for our DC-DC buck regulator. We set up our circuit on the board and by using the power supply in the electrical lab, we were able to get 3.3V. This was done by using a power supply of 12V - 14V as the input voltage for the switching regulators. For the LED, we set up a simple LED light with a 3.3V from the power supply. This power will come from the output pin of the microcontroller. After further research was conducted on power sources, the hardware team purchased the necessary backup battery for the project.

Even though it seems like we did a lot of work, there is still a lot of work left to be done. Therefore, we have decided to keep on working on our project during winter break. Even though winter break is less than a month, we believe we can get a lot of work done during this period. When we start Senior Design 2, we will be ahead, so we can start working on our PCB layout as soon as possible.

We have made some preliminary plans of what we hope to achieve for Senior Design 2. We have created milestone tables for each individual detailing some of things they have accomplished. We have also included some details of what we plan to do for Senior Design 2. At the beginning of the semester, we will meet up to discuss the next necessary steps in our project. This will greatly depend on how much work we can get down during winter break. Since we have purchased all of our parts for our project already, we can get started on working with the PCB immediately. For prototyping, we plan to split it into two groups. The two groups will be the software and hardware groups. During prototyping, each group will continue to work on their respective part. If either group runs to a problem, a discussion will be necessary to come up with a solution the problem. Similarly, prototyping will highly depend on how much work we can get done over winter break. Once we have finished prototyping, we plan to start creating the PCB layout. Creating the PCB layout might take a lot of time. After we have our PCB layout done, then we can get the PCB board created at a manufacturer. We plan to start creating the PCB layout while we are prototyping to save some time.

As a group, important milestones were considered and kept a track of as soon as they were accomplished. We have listed individuals milestones below:

### Tommy Goris

|  |  |
| --- | --- |
| Date of achievement | Achievement |
| September 9, 2016 | Initial Document (group work) |
| September 15, 2016 | Research OBD-II |
| September 21, 2016 | Initial chip research |
| September30, 2016 | Initial OBD-II testing |
| October 4, 2016 | Accelerometer, gyroscope ,and magnetometer research (LSM9DS0) |
| October 12, 2016 | Initial wireless research |
| October 17, 2016 | Bluetooth research |
| October 23, 2016 | Obtained Bluetooth research(CC2640) |
| October 30, 2016 | Continue initial documentation |
| November 4, 2016 | Continue initial documentation |
| November 11, 2016 | Continue initial documentation and research coding for Android/Code Composer Studios |
| November 18, 2016 | Continue initial documentation |
| November 26, 2016 | Continue initial documentation and start preliminary testing |
| December 1, 2016 | Continue initial documentation and continue preliminary testing |
| December 6, 2016 | Finish initial documentation and finish testing |

Table : Tommy Goris Initial Milestones

### Jacob Wurm

|  |  |
| --- | --- |
| Date of achievement | Achievement |
| September 9, 2016 | Initial Document (group work) |
| September 15, 2016 | Research Microcontroller |
| September 21, 2016 | Initial chip research |
| September 30, 2016 | Initial software testing |
| October 4, 2016 | RTOS research |
| October 12, 2016 | Initial wireless research |
| October 17, 2016 | Bluetooth research |
| October 23, 2016 | Obtained CC2650 Launchpad for testing |
| October 30, 2016 | Continue initial documentation |
| November 4, 2016 | Continue initial documentation |
| November 11, 2016 | Continue initial documentation with respect to TI RTOS |
| November 18, 2016 | Continue initial documentation |
| November 26, 2016 | Continue initial documentation and start preliminary testing |
| December 1, 2016 | Continue initial documentation and continue preliminary testing |
| December 6, 2016 | Finish initial documentation and finish testing |

Table : Jacob Wurm Initial Milestones

### Ismael Rivera

|  |  |
| --- | --- |
| September 9, 2016 | Initial Document (group work) |
| September 15, 2016 | Research OBD-II |
| September 22, 2016 | Begin Accelerometer, gyroscope and magnetometer research (IMU) |
| September 30, 2016 | Initial OBD-II testing |
| October 4, 2016 | Accelerometer,gyroscope,and magnetometer research (LSM9DS0) |
| October 14, 2016 | OBD-II Power test (car battery) |
| October 23, 2016 | Begin Power regulation Research |
| October 30, 2016 | Continue initial documentation |
| November 4, 2016 | Continue initial documentation |
| November 10,2016 | Begin backup power Research |
| November 18, 2016 | Continue initial documentation |
| November 22, 2016 | Begin comparator research |
| November 26, 2016 | Continue initial documentation and start preliminary testing |
| November 29, 2016 | Begin testing Buck regulator |
| December 1, 2016 | Continue initial documentation and continue preliminary testing |
| December 4, 2016 | Begin testing Boost regulator |
| December 6, 2016 | Finish initial documentation and finish testing |

Table : Ismael Rivera Initial Milestones

### Jihang (Bruce) Li

|  |  |
| --- | --- |
| September 10, 2016 | Initial Document (group work) |
| September 16, 2016 | Research OBD-II and Power Regulator |
| September 23, 2016 | Begin Accelerometer, gyroscope and magnetometer research (IMU) |
| September 29, 2016 | Initial OBD-II testing |
| October 5, 2016 | Accelerometer,gyroscope,and magnetometer research (LSM9DS0) |
| October 16, 2016 | OBD-II Power test (car battery)  Cigarette Lighter Receptacle (DC outlet) |
| October 25, 2016 | Begin Power regulation Research |
| October 30, 2016 | Continue initial documentation |
| November 5, 2016 | Continue initial documentation |
| November 12,2016 | Begin Power Regulator Research |
| November 20, 2016 | Continue initial documentation |
| November 25, 2016 | Continue initial documentation |
| November 28, 2016 | Continue initial documentation and start preliminary testing |
| November 30, 2016 | Begin testing LM2574 and LM2594  regulators |
| December 2, 2016 | Continue initial documentation and continue preliminary testing |
| December 4, 2016 | Continue buck regulator |
| December 6, 2016 | Finish initial documentation and finish testing |

Table : Jihang (Bruce) Li Initial Milestones

The milestones that we plan to reach while in Senior Design 2 at listed in Table 23.

|  |  |  |  |
| --- | --- | --- | --- |
| Tasks | Start Date | Complete Date | Person in Charge |
| Group Meeting | 1/8/2017 | 1/8/2017 | Group |
| Ordering Extra Parts | 1/8/2017 | 1/15/2017 | Group |
| Software Prototype | 1/20/2017 | 1/24/2017 | Software members |
| Hardware Prototype | 1/25/2017 | 2/01/2017 | Hardware members |
| PCB Layout | 1/20/2017 | 2/10/2017 | Hardware members |
| PCB Testing | 2/10/2017 | 2/15/2017 | Group |
| Paper I Draft | 2/10/2017 | 2/25/2017 | Group |
| Paper II Draft | 2/26/2017 | 03/12/2017 | Group |
| Conference Paper | 03/15/2017 | 03/22/2017 | Group |
| Final Paper | 03/22/2017 | 04/15/2017 | Group |

Table : Senior Design II Fall 2015 - Milestones

## 12.3 Division of Labor

We decided to split our project into two main groups. A software group and a hardware group. The hardware group will be working on sensors, power and other hardware configuration tasks. The software group will be working on communication, RTOS task development, and other software specific tasks. There is only one sensor module in our project, which will be used to receive a vehicle’s acceleration and position data in real time. The power sources for our main module will utilize a car’s battery as a power supply through the OBD-II connection. In case the car battery fails, we will have a backup battery installed into the main module. Other hardware configuration includes creating a PCB layout. This will require the hardware team to understand how the components they are using work together. As for software, this will involve communicating with a Bluetooth module, a microcontroller, an OBD-II IC, and external sensors. In addition, the main module will have to communicate with a mobile device through a Bluetooth connection configured in the main module. Each team member has taken responsibility for one or two different jobs. In the event that another aspect of the project requires more work, we plan for members to take on greater responsibility and share some of the extra work between groups. The list of responsibilities categorized by each member is detailed below.

Tommy Goris is in charge of the OBD-II, IMU, and Bluetooth modules. These components are important to our project because they will be used to measure data from the vehicle. Ultimately, this data will be used to determine if the user has gotten into an accident or not. In addition, the Bluetooth module will allow us to establish a connection with the user’s mobile device which will allow them to contact emergency services.

Jihang Li is in charge of regulation topology research and LED configuration research. The grand majority of the PCB research and implementation will be done by Bruce as well. The car battery is one of ways we will be supplying power to our device through the power pin on the OBD-II. The power section of this project, includes the buck regulators, which regulates the input voltage that goes into the microcontroller. The PCB layout is key to our project. After prototyping, Bruce and Ismael will start working on the PCB layout.

Jacob Wurm is in charge of phone communication, microcontroller programming, and the OBD-II module. As we discussed in the previous section, two of the team members are working on the software section together. This is why Jacob is also working on the OBD-II and Bluetooth for our project. A mobile phone application will be used to receive a signal from the main module. This signal will prompt the user on whether the application should contact emergency services as well as the user’s emergency contacts. The microcontroller is the main system for our project, because it will control each chip to do the work.

Ismael Rivera is in charge of almost all the power supply designs in our project. Some of the sections include but are not limited to, main power regulation, backup power characteristics, backup power configurations and power testing. He will also provide any supplementary PCB work that will be needed. He will be responsible for ensuring that the system will be able to operate at all times when hooked up to the OBD-II port. For a brief period of time if the system loses power from the OBD-II it should also be able to properly function. This will ensure that the user is aware that the system is not working at optimal conditions. He is responsible for providing a concept by the end of Senior Design 1 that works. Any possible upgrades to the power supply may be considered by the beginning of Senior Design 2 but the original power supply prototype shall be kept to allow us to continue working on the programmable devices integrated into the system.

## 12.4 Finance

The finances for the project were personally handled by each member of the group. Because of this we decided to split up the costs based on the different components our system was composed of. For example, if one group member was in charge of the power subsystem they needed to obtain the power components for the system. The same goes for the other components of the system. Each item purchased by the members was carefully tracked and detailed to ensure that an accurate calculation of the money spent was available.

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Unit price ($) | Quantity | Total price ($) |
| LM2574 | 2.76 | 3 | 8.28 |
| LM2594 | 3.18 | 3 | 9.54 |
| CC2650  Development Kit | 30.99 | 2 | 61.98 |
| STN1110 Development Board | 39.96 | 1 | 39.96 |
| Sparkfun RedBoard Arduino Clone | 15.96 | 1 | 15.96 |
| Breadboard | 8.99 | 1 | 8.99 |
| Jumper Wires (400) | 8.00 | 1 | 8.00 |
| SaBLE-x Breakout Board | 44.99 | 1 | 44.99 |

Table : Bill of Materials

## 12.5 Team Organization

### 12.5.1 Overview

As with any engineering project a process must be established, followed, and maintained to ensure the optimal level of work is being done. Organization and a complete understanding of the end goal are essential to having a team be successful. Clearly defining roles among members in a team is essential, but can be challenging in a smaller team as individuals will likely take on many roles depending on their skillsets.

Our team will employ industry standards to optimize our workflow, and ensure that the proper channels of communication are open and available as development on the project continues. As two of the members in our team are well versed in software development, we decided to utilize the agile design methodology. We chose this option because it allows for a flexible process of reacting to changes in engineering requirements while also allowing for a structure where design goals can be reached.

### 12.5.2 Communication

In order to a team to effectively complete their tasks there must be a system in place for effective communication. In addition, when time is limited scheduling can be a difficult task to manage. In order to facilitate both communication and scheduling we have decided to utilize Slack (<https://www.slack.com>). This is a messaging and file sharing application that provides an effortless means to facilitate communication between team members. One major advantage of this application is that it is cross-platform, so it can be used on computers and mobile devices. Another advantage is that there is application integration with Google Drive and other applications which allow for scheduling of events and task distribution.

Aside from communicating on Slack, the team meets every Tuesday and Thursday for a minimum of four hours from 7:00PM - 11:00PM. This time is optimal as our team members do not have classes during this time. In person meetings provide a platform for members to discuss issues they are having and express their thoughts on the project. This also fosters conversation between the members which will likely lead to problems being solved. In addition, this time can be used to complete administrative tasks such as stand-up meetings, issues, and concerns.

### 12.5.3 Information Sharing

In order to the team to communicate their ideas we will need a way to facilitate the sharing of information, preferably one which provides members with real-time updates on all files that are shared. For the purposes of this project all documentation and reading material created or acquired by the team will be centralized on Google Drive, an application which allows for the sharing of documents, pictures, and data in the cloud. A subsystem of Google Drive, Docs, gives the team a place to concurrently edit and collaborate on documentation.

### 12.5.4 Version Control

In order for the team to be able to collaborate on the software portion of the project (i.e. Android and main module) they need to be able to have their code located in a central location. In addition, they need to be able to keep track of the changes that they made, which is where version control becomes extremely useful. The version control system, “git”, will be used in this project for software collaboration. We will be utilizing “git” servers that are located online at GitHub (<https://www.github.com>). By using an online “git” repository we will be able to showcase the software we have created as well as revert back to previous iterations if a problem occurs. This allows us to keep track of progress, and since it is accessible from anywhere the team members will be able to work from any location.

# Appendices

## Appendix A - Abbreviations

|  |  |
| --- | --- |
| **Abbreviation** | **Definition** |
| AC | Alternating Current |
| ADC | Analog-to-Digital Converter |
| AES | Advanced Encryption Standard |
| ARM | Advanced RISC Machine |
| BLE | Bluetooth Low Energy |
| BOM | Bill of Materials |
| DC | Direct Current |
| CAN | Controller Area Network |
| CCS | Code Composer Studios |
| CDMA | Code division multiple access |
| DIP | Dual-Inline Package |
| DLC | Data Link Connector |
| EAGLE | Easy Applicable Graphical Layout Editor |
| EMF | Electric and Magnetic Fields |
| FCC | Federal Communications Commission |
| GM | General Motors |
| GPIO | General-Purpose Input / Output |
| GPS | Global Positioning System |
| HOQ | House of Quality |
| IC | Integrated Circuit |
| IMU | Inertial Measurement Unit |
| ISM | Industrial, Scientific and Medical |
| ISP | In-System Programmer |
| LED | Light-Emitting Diode |
| MCB | Miniature Circuit Breaker |
| MCU | Microcontroller Unit |
| NFC | Near Field Communication |
| NHTSA | The National Highway Traffic Safety Administration |
| OBD | On-Board Diagnostics |
| OS | Operating System |
| PCB | Printed Circuit Board |
| PID | Parameter Identification |
| RTOS | Real Time Operating System |
| RTC | Real Time Clock |
| SAE | Society of Automotive Engineers |
| SDK | Software Development Kit |
| SMD | Surface Mount Device |
| SOIC | Small Outline Integrated Circuit |
| SPI | Serial Peripheral Interface |
| TI | Texas Instruments |
| Tx | Transmit |
| UART | Universal Asynchronous Receiver / Transmitter |
| UCF | University of Central Florida |
| UI | User Interface |
| USB | Universal Serial Bus |

## Appendix B - List of Tables

Table 1: Embedded Development Environment Comparison 27

Table 2: Technical Comparison between Classic Bluetooth and Bluetooth Low Energy 28

Table 3: STN1110 and ELM327 Comparison 41

Table 4: OBD-II Voltage and Current Readings 56

Table 5: Differences in USB voltage between a Honda Accord and a Toyota Camry 57

Table 6: Electrical Characteristic Comparison for Switching Regulators 64

Table 7: Equations for linear and switching regulator 65

Table 8: Boost Switching Regulator Equations 69

Table 9: Component Consideration Table 79

Table 10: Hardware Component Cost Table 82

Table 11: LM2594 test criteria 102

Table 12: OBD-II port test criteria 102

Table 13: Microcontroller testing criteria 103

Table 14: Accelerometer testing criteria 104

Table 15: STN1110 testing criteria 106

Table 16: Testing criteria for the Bluetooth module 107

Table 17: Testing criteria for extended use effects 109

Table 18: Testing criteria for the OBD-II IC 110

Table 19: Tommy Goris Initial Milestones 114

Table 20: Jacob Wurm Initial Milestones 115

Table 21: Ismael Rivera Initial Milestones 116

Table 22: Jihang (Bruce) Li Initial Milestones 117

Table 23: Senior Design II Fall 2015 - Milestones 117

Table 24: Bill of Materials 119

## Appendix C - List of Figures

Figure 1: Overview of Project Functionality 3

Figure 2: CC26XX Functional Block Diagram (Courtesy of Texas Instruments) 24

Figure 3: Photo of Code Composer Studio Version 6 27

Figure 4: Graphical Representation of the Diffie-Hellman Key Exchange Process 31

Figure 5: Android Studio’s Development Environment 32

Figure 6: HARAMBE Mobile Application Use Case Diagram 34

Figure 7: User interface mockup for selecting emergency contacts 35

Figure 8: User interface mockup for user agreement 36

Figure 9: User interface mockup for tutorial 37

Figure 10: Sample emergency alert screen 38

Figure 11: MCP2551 CAN Transceiver: 42

Figure 12: ISO Transceiver 42

Figure 13: J1850 Transceiver 43

Figure 14: Initial setup for MEMS accelerometer and a gyroscope 46

Figure 15: Ending setup for MEMS accelerometer and a gyroscope 46

Figure 16: Closer picture of MEMS 46

Figure 17: High-Level Software Functionality Overview 51

Figure 18: Visual representation of software flow for main module of HARAMBE 54

Figure 19: Ideal linear regulator 58

Figure 20: Non ideal linear regulator 58

Figure 21: Buck Topology 60

Figure 22: Buck Regulator Simulation 60

Figure 23: Buck Regulator Simulation 61

Figure 24: LM2574 Block Diagram 63

Figure 25: LM2954 Block Diagram 64

Figure 26: TPS6020X Family Block Diagram 68

Figure 27: Boost Regulator Common Topology 69

Figure 28: Boost Waveform During Power On 70

Figure 29: TLV61225 Block Diagram 71

Figure 30: TPS61260 Reference Schematic Diagram 72

Figure 31: Buck-Boost Common Topology 72

Figure 32: CD74HC85E Comparator Pinout 74

Figure 33: Schematic for LED 76

Figure 34: Buck Converter Simulation 77

Figure 35: Boost Converter Simulation 78

Figure 36: Power Test of Buck to Boost Conversion 79

Figure 38: Power System Overview for the Main Module: 80

Figure 39: SaBLE-X Sample Schematic 89

Figure 40: STN1110 chip schematic 90

Figure 41: The lMS9DS1 chip configuration. Next to the chip is the buck regulator power supply. 91

Figure 42: Overall PCB Design 92

Figure 43: STN1110 PCB Design 92

Figure 44: LM2574N-3.3 PCB Design 93

Figure 45: CC2650 Launchpad 94

Figure 46: Adafruit LSM9DS0 Breakout Board 95

Figure 47: Sparkfun STN1110 UART Breakout Board 96

Figure 48: SaBLE-x Breakout Board 97

Figure 49: Breadboard Prototype of H.A.R.A.M.B.E 98

## Appendix D - Datasheets

|  |  |
| --- | --- |
| **Component** | **Part Number** |
| BLE Module | CC2650 |
| BLE Module | CC2640 |
| BLE Module | SaBLE-x 450-0119-R2 |
| Buck Regulator | LM2574x |
| Boost Regulator | TLV61225 |
| Comparator | CD74HC85E |
| Charge Pump | TPS60204 |
| OBD TO UART Interpreter IC | STN1110 |
| Accelerometer | LSM9DS1 |
| Switching Regulator | LM2574 |
| Switching Regulator | LM2594 |
| Switching Regulator | TPS62172 |
| Switching Regulator | TPS62177 |
| Linear Regulator & Switching Regulator Fundamental | LM2940, LM2952, LM2953, LM2954 |

## Appendix E - References

* Texas Instruments. (2016, June). SYS/BIOS (TI-RTOS Kernel) User’s Guide (v6.46) [Online]. Available: http://www.ti.com/lit/ug/spruex3q/spruex3q.pdf
* Arduino - SPI. (n.d.). Retrieved December 05, 2016, from https://www.arduino.cc/en/Reference/SPI
* Texas Instruments. (2016, June). TI-RTOS 2.20 User’s Guide (Rev. M) [Online]. Available: http://www.ti.com/lit/ug/spruhd4m/spruhd4m.pdf
* Texas Instruments. (2016, June). TI-RTOS 2.20 for CC13xx/CC26xx SimpleLink Wireless MCUs Getting Started Guide (Rev. C) [Online]. Available: http://www.ti.com/lit/ug/spruhu7d/spruhu7d.pdf
* Texas Instruments. (2015, May). Intro to the TI-RTOS Kernel Workshop Student Guide (v4.00) [Online]. Available: https://training.ti.com/sites/default/files/docs/TI\_RTOS\_Kernel\_Workshop\_Student\_Guide\_rev4.00\_1.pdf
* Texas Instruments. (2016, June). CC13xx, CC26xx SimpleLink Wireless MCU Technical Reference Manual (Rev. F) [Online]. Available: http://www.ti.com.cn/cn/lit/ug/swcu117f/swcu117f.pdf
* LSR. (2015, Feb 5). SaBLE-x Development Board User Guide (Rev. 1.4) [Online]. Available: https://www.lsr.com/downloads/products/330-0168.pdf
* Texas Instruments. (2016, June 27). TI-Simplelink [Repository]. Available: https://github.com/ti-simplelink/
* Arduino - SPI. (n.d.). Retrieved December 05, 2016, from https://www.arduino.cc/en/Reference/SPI
* Texas Instruments. (2010 November). KeyStone Architecture Literature Number: SPRUGP1 November 2010 Universal Asynchronous Receiver/Transmitter (UART)[Online]. Available: <http://www.ti.com/lit/ug/sprugp1/sprugp1.pdf>
* Durda, F. (1996, January 13). Serial and UART Tutorial. Retrieved December 5, 2016, from https://www.freebsd.org/doc/en\_US.ISO8859-1/articles/serial-uart/index.html#uart https://learn.sparkfun.com/tutorials/serial-communication/uarts
* Arduino - SPI. (n.d.). Retrieved December 05, 2016, from https://www.arduino.cc/en/Reference/SPI
* On-board diagnostics. (n.d.). Retrieved December 05, 2016, from https://en.wikipedia.org/wiki/On-board\_diagnostics
* E2E members. (2016, April 23). BLE Wiki [Online]. Available: http://processors.wiki.ti.com/index.php/Category:BluetoothLE
* Texas Instruments. (2016, July). TI Webench [Online]. Available: http://www.ti.com/lsds/ti/analog/webench/overview.page?DCMP=analog\_power\_mr&HQS=webench-pr
* Texas Instruments. (2015, June). Understanding the I2C Bus[Online]. Available: http://www.ti.com/lit/an/slva704/slva704.pdf
* University of Central Florida (2016, November) EEL 5245C Laboratory Information
* Batarseh, Issa. Power Electronic Circuits. Hoboken, NJ: John Wiley, 2004. Print.
* Federal Communications Commission (2016, September). What you need to know about Text-to-911 [Online]. Available: [h](https://www.fcc.gov/consumers/guides/what-you-need-know-about-text-911)ttps://www.fcc.gov/consumers/guides/what-you-need-know-about-text-911
* eeWiki (2016, October). Introduction to BLE security [online]. Available: https://eewiki.net/display/Wireless/A+Basic+Introduction+to+BLE+Security
* Pew Research (2016, September). US Smartphone Use in 2015 [online]. Available: http://www.pewinternet.org/2015/04/01/us-smartphone-use-in-2015/
* Bluetooth Core Specification. (n.d.). Retrieved December 05, 2016, from https://www.bluetooth.com/specifications/bluetooth-core-specification
* IDC (2016, October). Smartphone OS Market Share 2016Q2 [online]. Available: https://www.idc.com/prodserv/smartphone-os-market-share.jsp
* Argenox (2016. September). A Guide to Selecting a Bluetooth Chipset [online]. Available: http://www.argenox.com/bluetooth-low-energy-ble-v4-0-development/library/a-guide-to-selecting-a-bluetooth-chipset/
* ASIRT (2016, September). Road Crash Statistics [online]. Available: https://asirt.org/initiatives/informing-road-users/road-safety-facts/road-crash-statistics

## Appendix F - Copyright Permission Requests

