Huddy Buddy

The Smart Heads-Up Display for Increased Driver Awareness and Safety

University of Central Florida

College of Electrical Engineering and Computer Science
Dr. Samuel Richie, Dr. Lei Wei

College of Optics and Photonics
Dr. David Hagan

Senior Design II Final Report Submission:

Group 6
Aaron Majdali – Optics and Photonics Engineering
(aaron.majdali@knights.ucf.edu)

Evan Hall - Electrical Engineering (ehall17@knights.ucf.edu)

Logan Glowth - Computer Engineering (logan.glowth@knights.ucf.edu)

Pedrhom Nafisi - Computer Engineering (pedrhomnafisi@knights.ucf)

Table of Contents

Table of Figures

Table of Tables

1.0 Executive Summary

In today’s world, where mobile technology has become an essential part of our lives, it is often difficult to disconnect and put these mobile devices away for much longer than a few minutes. This becomes a major hazard when getting behind the wheel of a vehicle. There is a myriad of distractions to take into account when driving a vehicle. According to the National Highway Traffic Safety Administration, “distracted driving is dangerous, claiming 3,166 lives in 2017 alone”. Distracted driving not only puts the driver’s life in danger, but also the lives of the other drivers and passengers on the road, creating dangers such as speeding.

Through this Senior Design project, we designed a device that provides a driver with enhanced situational awareness by displaying pertinent information in the driver’s field of view. To accomplish this, we created our own device using knowledge of hardware and software gained throughout our college careers. This device, called the Huddy Buddy, contains a Bluetooth module for wireless connectivity, an OLED display module, a collimating display unit, a power delivery system, an emergency crash detection and response system, and a custom Android application for turn by turn navigation. The device reads information sent by the phone and uses a display module to project an image onto the windshield of the vehicle, containing information such as speed limit data or navigational aids.

There are vehicle manufacturers that have special packages that contain integrated heads-up displays, but these packages are often associated with premium prices. Additionally, these heads-up displays are not always integrated with advanced navigational awareness features such as turn-by-turn directions and emergency cr. Our goal was to create a low-cost solution that contains advanced functionality beyond what is provided with pre-existing heads-up display systems. To differentiate our device from other products, our device has a minimal power consumption, with a reduced footprint on the dashboard, and an easily readable and functional display.

We believe that this project has pushed us to learn more about the advanced systems required to make this device a reality. As seniors in Computer Engineering, Electrical Engineering, and Optics and Photonics Engineering, we combined our fields of study to effectively and efficiently produce a product that can be used to reduce the amount of distractions and increase situational awareness while driving a vehicle.

2.0 Project Description

2.1 Motivations

The motivation for this project was to demonstrate our knowledge of optics, electrical design, and programming that we have accumulated while studying at the University of Central Florida. Classes such as Electronics, Computer Sciences, Embedded Systems, and Optics have given us in depth knowledge about the processes of engineering in our respective fields of study. It's one thing to take classes that discuss these topics, but it's incredibly beneficial to synthesize a physical project using this knowledge.

Upon initial group formation, we set out to determine the best project for our interests and skills. With a team consisting of two computer engineering students, one electrical engineering student, and one optics and photonics engineering student, there were a plethora of options that we could choose from that would prove to be challenging and exciting. Initially, we thought of a plan to create something that would combine topics covered by our three engineering disciplines to work in tandem.

The idea of creating a Heads-Up Display for a vehicle stuck out as one that could be well designed and implemented in the timeline of Senior Design. The Huddy Buddy required many hardware and software designs that were implemented by Logan and Pedrhom, our Computer Engineering majors. The Huddy Buddy required power and electrical design that were primarily designed by Evan, our Electrical Engineering major. The lens and mirror system needed to collimate and focus our OLED display's image were designed by Aaron, our Optics and Photonics Engineering major. We believe that there was enough depth in each field to provide equal opportunities for all of us to learn and contribute to creating a great Senior Design Capstone project.

One of the great things about having a Senior Design lecture was that it exposed us to a lot of strategies to making a successful design team. Events such as the Senior Design Bootcamp allowed us to come together and determine how each member of the group can contribute to the project. Referenced further in this document is a list of project milestones that we determined to be important to the success of this project. Throughout the lifetime of our project, we stuck to our schedule to the best of our ability given unforeseen events that followed the COVID-19 outbreak.

2.2 Goals and Objectives

The overarching goal for our senior design project was to create a device that could display content relevant to someone driving a vehicle in a manner that is safe, efficient, and user friendly. The device was created to fit into any vehicle, regardless of what design the interior has. This makes it as accessible as possible, enabling a wider range of consumers to use the final product. Each module of the device is capable of being tested in its own environment, enabling easier troubleshooting and development. The various features of the device provide the user with an enhanced sense of situational awareness, as well as a feeling of increased safety. The device aides the user in navigating from a starting point to a destination with minimal input by the driver.

Another goal for this project was to make it as cost efficient as possible. We did not obtain any sponsorships, and as such we have self-funded everything. As college students, bearing the cost of additional materials can be a burden. We sourced materials to make the most effective product at the lowest price possible, reducing the financial strain on each member of the team. The College of Engineering and Computer Science at the University of Central Florida provided resources such as the Texas Instruments Innovation Lab and the Senior Design Lab that provides students with tools, parts, and other useful implements that reduce the need for purchasing components like oscilloscopes and digital multimeters. We used as many of these resources as we can throughout the last two semesters.

During the course of Senior Design I and Senior Design II, many new and challenging obstacles were presented as the design of the heads-up display came together. From an electrical standpoint building a power supply subsystem that supplied power to each of the individual parts was a learning curve in that many topics covered over the years came together. Additionally, creating an application that interfaced with the heads-up display created an interesting learning opportunity to learn and test new skills. Content learned in Embedded Systems became an integral part of the project when pulling data from the OBDII adapter over a UART connection. Finally, combining all of the data obtained by the subsystems and showing them on the display was the hardest goal to reach because everything must work as designed for the user to be able to see the screen and have the right code that clearly marks where the user will go according to the directions given by the navigational service. An overarching objective to our education was to actively learn how to put these parts together and work together as a team to debug the problems that were presented.

Following the end of Senior Design, we would like to continue to support of Huddy Buddy’s development into a product that can be used into the future. Creating our own HUD system gave us a platform with advanced features beyond what is included in the vehicle models we currently own. Many cars today include the convenience of HUDs built-in to the dashboard. Our design allows us to modernize our vehicles without the premium price associated with newer vehicles.

2.2 Design Constraints:

The following constraints were placed on the HUD device due to the factors that exist when building a self-funded design in a collegiate setting. The team adjusted our design to match what is consistent with our expected budget, timeline and restrictions set by the University of Central Florida College of Engineering and Computer Science. Other constraints exist due to the knowledge and background of each of our team members

2.2.1 General Constraints

When we started to think about how to implement the HUD Device, we set some general constraints to give us some boundaries for how it should be built. These constraints apply to both the HUD device and the mobile application. Tables 1 and 2 describe the constraints of both the HUD device and mobile application, respectively.

Constraint	The HUD Device shall:
GC.H.1	Include a custom Printed Circuit Board (PCB)
GC.H.2	Not include pre-built components such as Development Boards
GC.H.3	Be designed by December 10, 2019
GC.H.4	Be user friendly
GC.H.5	Be built by April 2020
GC.H.6	Maximize energy efficiency
GC.H.7	Increase driver safety and awareness
Constraint	**The HUD Device shall:**
GC.H.8	Not interfere with the driver's view of the road
GC.H.9	Not distract the driver in any way
GC.H.10	Be reasonably designed
GC.H.11	Be funded by the students or sponsorship where applicable

Table 1: General Constraints for HUD Device

Constraint	The Mobile Application shall:
GC.A.1	Be developed for Android devices
GC.A.2	Developed using Android Studio
GC.A.3	Use the Google Cloud Platform API for navigational information
GC.A.4	Be able to connect to the internet from the mobile device's carrier network
GC.A.5	Have a minimalistic design to reduce distraction
GC.A.6	Be user friendly
GC.A.7	Be fully functional by April 2020
GC.A.8	Use the mobile device's Bluetooth capabilities to pair with the HUD Device
GC.A.9	Send data to the HUD Device from Bluetooth
GC.A.10	Not distract the driver in any way
GC.A.11	Be reasonably designed

Table 2: General Constraints for Mobile Application

2.2.2 Economic Constraints

Economic constraints are extremely important when considering how the project is designed and implemented. Economies are dynamically changing and can be different depending on what country or market the project is created and sold in. Our project was targeted at the United States market; however, parts were sourced from different countries and could have been subject to certain taxes, importing fees, and tariffs. All these factors need to be considered when determining the value of the final product. Table 3 describes the economic constraints that were placed on this project.

Constraint	Economic Constraint
EC.1	The project shall cost no more than $500 USD
EC.2	The project shall be funded by members of the group
EC.3	The mobile application must not add additional costs to the project
EC.4	The device and application must be created with US Market in mind

Table 3: Economic Constraints

2.2.3 Environmental Constraints

Environmental constraints are limiting factors due to the impact of used materials such as production, disposal, energy consumption, and emissions of the products involved. Our environmental constraints were pertaining to the energy efficiency of our HUD Device, the energy efficiency of the mobile device running our custom application, and the emissions of the vehicle that is being used for testing of the HUD Device. Table 4 references the environmental constraints placed on this project.

Constraint	Environmental Constraint
ENVC.1	The project shall be energy efficient
ENVC.2	The HUD Device must be powered by the vehicle containing it
ENVC.3	The vehicle used must be compliant with economic policies in place by the United States Government.
ENVC.4	The project shall not use hazardous materials

Table 4: Environmental Constraints

2.2.4 Social Constraints

Social Constraints are due to societal norms, traditions, and other factors that affect how humans interact and view the project. Our project needed to take special consideration for social constraints due to the nature of human use. Table 5 describes each social constraint placed on our project.

Constraint	Social Constraint
SC.1	The HUD Device must be user friendly
SC.2	The mobile application must be user friendly
SC.3	The project must not violate social norms
SC.4	The project and its accompanying documentation must contain language that can be understood across different cultures.

Table 5: Social Constraints

2.2.5 Legal Constraints

Throughout the timeline of Senior Design I and II, there were no federal laws that limit or control the use of a heads-up display in a vehicle. Similarly, there were no laws in any of the 50 states that regulate the use of heads up displays. Because of this, there are many car manufacturers that integrate heads up displays into their vehicles, all of which are 50 state legal. Heads up displays are also not subject to laws in Canada [4]. In fact, heads up displays seem to really have no legal restrictions in North America or Europe. As this device was designed with the United States market in mind, the GPS heads up display should face few to no legal issues. One possible point of concern could be regulations on flashing lights that are visible from vehicles. This ruled out the use of red or blue LED’s to illuminate the heads up display.

2.2.6 Political Constraints

Political constraints are due to the implications of integrity and motivations by parties affiliated with a certain subject. There are no political intentions that surround this project, the members of the team, nor the potential users of the project.

2.2.7 Health and Safety Constraints

Health and safety constraints are extremely important to keeping those who interact with the project and its components from sustaining serious injury or harm. Table 6 describes the health and safety constraints that are placed on our project.

Constraint	Health and Safety Constraint
HSC.1	Any electrical component shall be used within the designed specifications by the original manufacturer.
HSC.2	The HUD Device must not add any distractions to the driver
HSC.3	The HUD Device must not hinder the driver’s view of the road
HSC.4	The mobile application must not add any distractions to the driver
HSC.5	The vehicle used for testing must be road-worthy and properly insured
HSC.6	Any electrical connections must be properly grounded

Table 6: Health and Safety Constraints

2.2.8 Manufacturability Constraints

Manufacturability constraints are due to the components required to realize the HUD Device, software for the application, or any other physical or virtual part of the project. To ensure that the project can be manufactured within our budget and timeline, each component of the project was created or implemented using parts that were readily available or easily obtained. The College of Engineering and Computer Science at the University of Central Florida had a myriad of resources for students to use, enabling the project to be manufactured at a minimal cost. Table 7 describes the manufacturability constraints of this project.

Constraint	Manufacturability Constraint
MC.1	The project shall use parts that are readily available or easily obtained
MC.2	The project shall take advantage of labs and tools provided by the College of Engineering and Computer Science of the University of Central Florida

Table 7: Manufacturability Constraints

2.2.9 Sustainability Constraints

Sustainability constraints are due to the ability for the project to be supported and maintained after being produced and completed. Once a product has launched, it must be maintained to ensure that it remains in a useable state for the lifetime of the product. Unforeseen issues can arise long after the development stage has completed. Table 8 describes the sustainability constraints of the project.

Constraint	Sustainability Constraint
SUSC.1	The project shall be supported and updated after completion
SUSC.2	The project shall allow for further development as needed

Table 8: Sustainability Constraints

2.3 Engineering Requirement Specifications:

The following requirements in Table 9 set a defined scope for how the project was designed and built. Each of the requirements gave the team a guideline for the end goal. Referencing this table was useful for staying within the bounds of what we are going to create.

Requirement	The HUD Device shall:
R.1	Weigh no more than 1 lb.
R.2	Not exceed 5x3x2 in. in size
R.3	Run off a USB Port
R.3.1	From this port there must be a voltage step down to run about 5 Volts or lower for low power consumption
R.4	Interface with a mobile phone via a Bluetooth connection
R.5	This will be compatible with android devices
R.6	Be able to display GPS data onto a windshield or dedicated screen
R.7	Have good resolution for easy viewing
R.8	The integration of software will be done using APIs provided by Google Cloud Platform
R.9	If the data displayed, such as google maps, requires sound, there will be a speaker on the side of the device
R.10	Be able to operate within a temperature range of -20 C to 50 C
R.11	Be able to operate in and be stored in direct sunlight
R.12	Be capable of adjusting to be visible with different windshield designs
R.13	Produce an image that is visible when viewed through polarized sunglasses
R.14	Be able to automatically adjust its brightness level according to the amount of ambient light

Table 9: Engineering Specification Requirements

2.4 Related Standards

As with any quality engineering project, there are standards that must be adhered to in order to ensure that a product is safe, reliable, and compatible with other systems. Such standards can involve communication, data storage, or even legal considerations. Here, we list relevant standards and laws and state how we ensured that our heads-up display conforms to them.

2.4.1 IEEE 802.15.1: WPAN / Bluetooth:

The IEEE 802.15.1 standard applies to wireless personal area networks (WPAN) and the construction of them using Bluetooth technology for small, low power devices [1]. The standard contains a wide variety of clauses, message types, data-formats, and structured formats. Within the clauses state specifications for how the physical layer as well as the Medium Access Control (MAC) must operate in order to meet this standard.

2.4.2 IEEE 802.15.4: LR-WPANs

The IEEE 802.15.4 is a technical standard which defines the operation of low-rate wireless personal area networks (LR-WPANs) used by our BLE module. It specifies the physical layer and media access control for LR-WPANs, and is maintained by the IEEE 802.15 working group, which defined the standard in 2003. The standard specifies the architecture and topology of the wireless protocol.

2.4.3 IEEE 802.11i: WPA2 and CCM

The IEEE 802.11i standard specifies security requirements and procedures for wireless networks, replacing the short authentication and privacy clause of the legacy standard with a detailed security clause. This standard also deprecated the privacy and security algorithm WEP in favor of the new and improved WPA2 algorithm which uses the CCM mode that is implemented in our Bluetooth mechanism.

2.4.4 NMEA 0183: Data Sequencing

The NMEA 0183 standard is a technical standard governed by the National Marine Electronics Association, designed to standardize the format in which data is transmitted between transmitting and receiving devices. The data contains 8-bits synchronized to a 4800 Baud rate. There is one stop bit, and no parity or handshake bits. The NEO-6M GPS Module, SIM808 2G GPS/GSM Module, and the SIMCom 5230 3G/GPS Module may have used provides output data formatted to the NMEA 0183 Standard. This means that we needed to set up our programming, hardware design, and implementation of the heads-up display and accompanying applications to support data processing according to this standard.

2.4.5 RoHS Compliance

RoHS stands for Restriction of Hazardous Substances. It is a standard that dictates whether electrical or electronics components have complied with the restrictions on using hazardous material in their products. The hazardous materials include lead, mercury, cadmium, hexavalent chromium, polybrominated biphenyls, polybrominated diphenyl ethers, and four different phthalates. Because there are a lot of components made every day and disposed of eventually, this helps stop the pollution of landfills, and when manufactured, helps keep workers safe, in addition to keeping pollutants out of the recycling process.

Most of the parts that were ordered for the heads-up display project were RoHS compliant. This was important because it could potentially help with marketing, showing that our product does not hurt the environment and can be recycled. At the minimum it will not hurt having RoHS compliant parts, in the sense that it is safe for our consumer base and that using our product will not endanger them from potentially harmful substances.

2.4.6 3GPP TS 23.040: SMS Cell Broadcast

The Third Generation Partnership Project (3GPP) is a standards organization that develops and maintains protocols for mobile cellular networks. The SIMCom SIM808 GSM module that we used to send SMS text messages follows the standards set forth by the 3GPP including the services and elements, network architecture, SMS router functionality, and router protocols. Additionally, the 3GPP maintains the AT Commands that modern cellular devices use to communicate with the wireless networks. Researching the AT commands in-depth provided us with the ability for our HUD Device to send text messages to mobile phones. This technology is available worldwide, allowing our HUD device to be accessible from anywhere. These standards make it possible for fast and reliable communication between networks, carriers, and countries.

2.4.7 SAE J1962: Onboard Diagnostics Standards

The SAE J1962 standard set forth by the Society of Automotive Engineers satisfies the regulations of the United States Onboard Diagnostics regulations. These standards regulate the physical implementation of Onboard Diagnostics ports in vehicles, primarily the OBD connection, access and design. Additionally, this standard regulates the electrical requirements for the OBD connector. We used this standard to attach our HUD Device to the vehicle’s OBD port to obtain data that can be displayed to the user.

2.4.8 SAE J1979: Onboard Diagnostics Parameter Identification

The SAE J1979 standard provides car manufacturers with a set list of parameters that are associated with specific systems for vehicles. This allows the creation of tools that can find data about a vehicle’s current state. We used this data to provide the HUD device with information about the vehicle’s speed, instantaneous fuel consumption, and other information that is relevant that can be pulled from the OBD port.

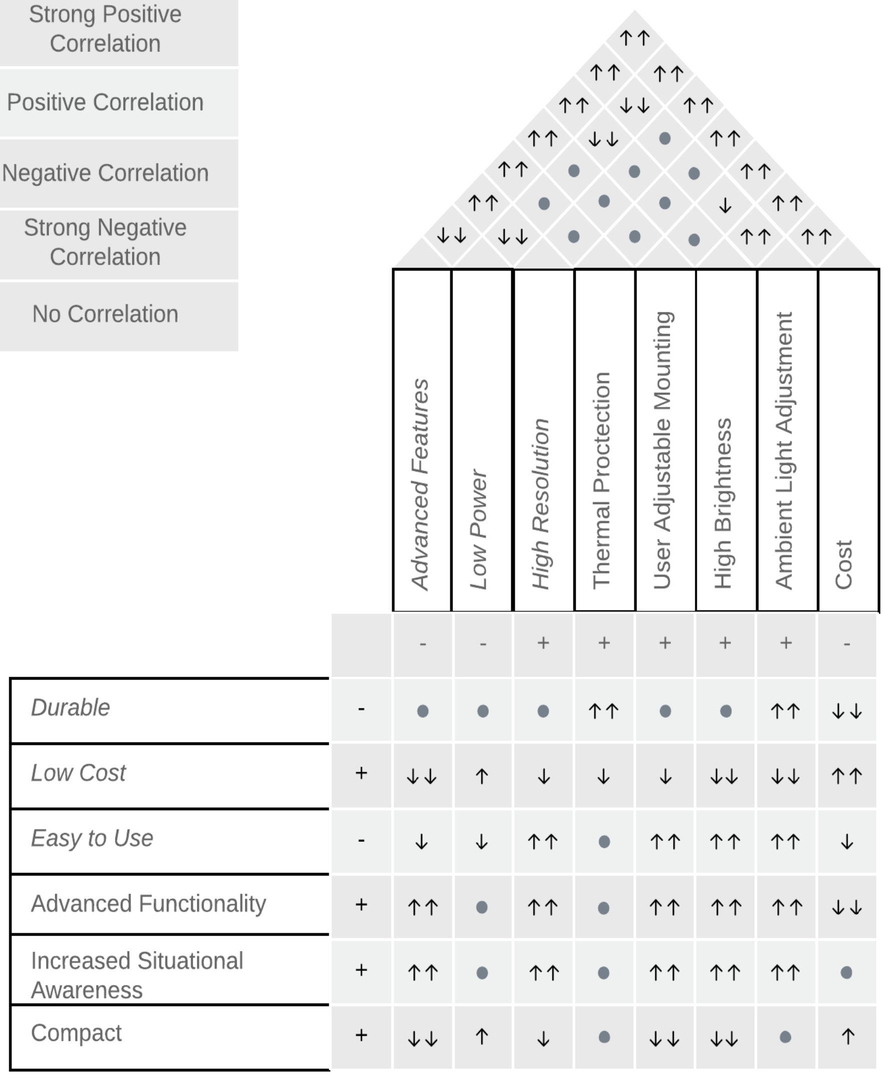
2.4.9 SAE J2012

The SAE J2012 standard contains definitions for diagnostic trouble codes (DTCs) thrown by diagnostics equipment that can be connected to the OBD2 port. This requires manufacturers to associate errors with a specific set of codes. Manufacturers and mechanics can use this standard when diagnosing issues with vehicles by replicating errors associated with a part without having to disassemble the entire car to find the issue. Standardized error codes make troubleshooting systems that span multiple countries much simpler and more efficient.

2.5 House of Quality:

There are many factors that need to be analyzed and discussed when designing our Heads-Up Display Device. Each of these factors has an impact on the implementation of our project. The House of Quality chart for our project, located in Figure 1, weighs the tradeoffs and effects of each factor on the outcome of our project.

Legend	
+	Positive Polarity
-	Negative Polarity
↑↑	Strong Positive Correlation
↑	Positive Correlation
↓	Negative Correlation
↓↓	Strong Negative Correlation
●	No Correlation



		Advanced Features	*Low Power*	*High Resolution*	Thermal Proctection	User Adjustable Mounting	High Brightness	Ambient Light Adjustment	Cost
		-	-	+	+	+	+	+	-
Durable	-	●	●	●	↑↑	●	●	↑↑	↓↓
Low Cost	+	↓↓	↑	↓	↓	↓	↓↓	↓↓	↑↑
Easy to Use	-	↓	↓	↑↑	●	↑↑	↑↑	↑↑	↓
Advanced Functionality	+	↑↑	●	↑↑	●	↑↑	↑↑	↑↑	↓↓
Increased Situational Awareness	+	↑↑	●	↑↑	●	↑↑	↑↑	↑↑	●
Compact	+	↓↓	↑	↓	●	↓↓	↓↓	●	↑

Figure 1: House of Quality

The house of quality diagram above was useful when making critical design decisions during the duration of our design process. We used it to analyze which systems were dependent on other factors within the project. Changing one factor could have a positive or negative effect, depending on what the polarity or correlation.

2.6 Project Milestones:

At the beginning of Senior Design I, we spent some time discussing projects that be both challenging and exciting. Once we had narrowed down what we wanted to do, we needed to create a timeline for us to design and implement the project. The senior design project spans across our final two semesters at the University of Central Florida. Projects of this scale do not happen without proper planning and time management. Table 10 outlines our timeline for certain stages of the project that were completed throughout Senior Design I.

Task	Start Date	End Date	Status
Senior Design I			
Create Groups	8/30/19	8/30/19	Completed
Project Ideas	8/31/19	9/6/19	Completed
Role Designation	9/7/19	9/8/19	Completed
Initial Project Documentation - Divide and Conquer	9/13/19	9/20/19	Completed
Start Design Documentation	9/23/19	9/23/19	Completed
Table of Contents	9/23/19	12/2/19	Completed
Research Individual Parts	9/23/19	12/2/19	Completed
Schematic Design	9/23/19	12/2/19	Completed
60 Page Rough Draft	9/23/19	11/1/19	Completed
100 Page Submission	11/1/19	11/15/19	Completed
Final 120 Page Submission	11/15/19	12/4/19	Completed
Parts Acquisition	11/30/19	1/1/20	Completed

Table 10: Senior Design I Milestones

Upon reaching the second semester of Senior Design, the team had already designed and planned out all the necessary milestones to achieve success for our project. We had already ordered and received our initial batch of parts and components. Our initially testing will have been done and completed. We aimed to design and finalize our PCB implementation by the fifth week of Senior Design II. The testing process we used throughout Senior Design was crucial to completing our design with minimal conflicts. Table 11 describes our expected timeline for Senior Design II.

Task	Start Date	End Date	Status
Senior Design II			
All Parts Must Have Arrived	1/1/20	1/15/20	Completed
Schematic Implementation and Testing	1/16/20	2/28/20	Completed
Testing Design	3/1/20	3/30/20	Completed
Final Prototype	4/1/20	4/11/20	Completed
Miscellaneous Time for Further Troubleshooting	4/11/20	413/20	Completed
Panel Presentation	4/15/20	4/15/20	In Progress

Table 11: Senior Design II Milestones

We believe that the timelines in the tables notated above gave us ample time to complete the project successfully. It required every member of the group to stay focused throughout the duration of Senior Design I and II. Something that we did not expect to occur was the offset due to the global Coronavirus pandemic. Throughout Senior Design II, we experienced many issues around the logistics of ordering parts from foreign countries that had ceased production and shipments to slow the spread of the virus. This affected acquiring parts for our custom PCB, as well as for our development environment. To mitigate this issue, we ordered twice the amount of parts needed for our custom PCB to ensure availability throughout the semester. When we began the Huddy Buddy project. We understood that there could be adverse situations that might arise throughout the design process. Throughout our college careers, we have learned how to deal with unexpected issues that can push deadlines back by days or even weeks. We are confident that this experience allows us to handle any situations moving forward.

3.0 Research Related to Project Description

3.1 Existing Projects and Products

Parallax-free sighting systems have been in use by the military since before World War II in fighter planes. Indeed, military aircraft have been the primary use case for a heads-up display for much of the technology's life. More recently, heads up displays have made their way into consumer vehicles. What's more, there are available third-party heads up display options that read out car diagnostic data and can even hold a cell phone to act as a heads-up GPS.

3.1.1 Display Design

There are different approaches to creating a heads-up display. From these, we know that brightness is a common concern with third-party models. This section will cover existing variations of heads up displays.

Many third-party solutions utilize a standard smartphone as a display. The phone is held in place via a mount that also attached to a small transparent screen. The image from the phone's screen is reflected off the mount's screen and is visible to the driver. The phone requires a special app to display GPS data and speed that can be seen by the driver i.e. the image is reversed so that the reflection is readable to the driver. Note that when in use the heads-up display wholly monopolizes the phone. If the driver wishes to use the phone in any way, the phone must be removed from the mount and the app must be closed.

Another possible design for a car heads up display uses a module with a dedicated LED-lit instrument cluster. While this implementation is unable to display GPS data, it is able to read information from a vehicle's OBD2 port. This allows the device to display fuel efficiency, speed, and tachometer data read from the vehicle itself. Note how the device uses the vehicle's windshield as the screen. This simplifies the use of the device but introduces the problem of possibly having a display that is too dim to see in direct sunlight. This product tries to alleviate that by including a reflective film to place on the window.

One more example is one that sits above the driver's head. The device is attached to the sun visor and displays its own image onto a screen that is attached to the system. The phone is connected via USB and uses a specialized app for displaying the appropriate data. The phone can be used to play music while the GPS heads up display is used. The device is battery powered and therefore has a finite run time before it needs to be recharged. Also, the sun visor is unusable while the device is attached.

3.2 Mounting the HUD

The main theme of our project is to reduce distractions around the driver of a vehicle. Every part of the project must be designed with this theme as the priority. As such, the device needs to be mounted in such a way that it does not impede with the driver's field of view of the road. Additionally, our design will need to be universal, allowing it to be used in vehicles of many different makes and models. Designing a platform that contains all our components from the custom PCB and the Heads-Up Display poses a unique challenge for us to tackle. This section will discuss various way we could mount the HUD Device in the vehicle.

3.2.1 Visor Mounted Device

An option for mounting the HUD Device would be to take advantage of a vehicle's sun visor. Mounting the display in this location would provide the driver with information that is still within the driver's field of view. This allows the driver to maintain focus on the road while glancing at the information in the display. Examples of this mounting location can be found in aviation, such as in the Boeing 787 Dreamliner. A pilot can bring a transparent display down to view information in the HUD while maintaining visual contact with what lies ahead of the aircraft. If the pilot no longer wishes to view the HUD, he or she can simply fold up the display out of view.

Mounting our HUD device in this fashion would be great for the display but could pose an issue depending on how bulky our final design will be. Additionally, routing the cables required to power the device and provide data would need to be routed to be out of sight of the driver.

3.2.2 Windshield Mounted Device

Another option would be to mount the device to the vehicle's windshield. Suction cups would stick the device to the windshield and adjustable arms could be used to place the display in the optimal position depending on the driver's height and position within the vehicle. Designing a mount in this way poses a potential risk of interfering with the driver's field of view.

3.2.3 Dashboard Mounted Device

A third option would be to mount the HUD device to the dashboard behind the steering wheel. Our preliminary research showed that projects that contain a similar focus have a strong bias toward this mounting method. The display unit would be seated in a base that can be stuck to the dash using a light adhesive or suction cup, preventing the device from moving due to forces of acceleration. A candidate would be the Tesa Powerstrip double-sided tape. It can hold up to 2 kg (4.40 lbs) which would be strong enough to hold the HUD in place. Another good thing about this product is that the tape can be pulled away easily, leaving no damage to the interior if the driver wishes to remove the device. A translucent pane of glass with a dark tint could be attached to the base.

3.3 Display

The display is a challenge that requires a lot of thought and care. One of the primary challenges of the display will be acquiring one that can adequately show GPS and other information, all the while in bright sunlight. Consumer solutions involve a sort of reflective film that can be directly applied to a piece of glass, usually the windshield. Even if we choose to go with a separate glass screen that is attached to the device, it may be worthwhile to also include a reflective film.

Because the device is to receive its power from a USB port, an otherwise adequate display may consume too much of the power budget. Another consideration is whether we want to use some sort of projector, a simple LCD screen, or a small LCD screen with a collimating lens. The advantages of a projector are that it can be collimated easily to show an image that focuses at an infinite distance. The downsides are its higher power consumption, larger size, and larger price tag. The advantages of using a small LCD with a collimating lens are lower power consumption, smaller footprint, and being able to be focused at infinity. The disadvantages are the difficulty of finding a smaller screen with adequate brightness, lower resolution, and issues with using a collimating lens, such as distortion. The advantages of using a larger LCD are the ability to simply reflect the screen without any extra lenses, high resolution, and ease of finding one with adequate brightness. The disadvantages are higher power consumption, inability to be focused at infinity, larger size, and possibly inadequate brightness within a reasonable financial and size budget.

Another topic that needs to be considered when choosing a display is contrast and color. One of the group’s biggest concerns regarding a transparent display is that the driver will need to see the text against what is in front of the car. If the transparent display that is chosen can only output black text, there may be an issue with discerning text against a dark colored road in both bright sunlight and at night. Typically, automotive companies will use a display with bright white text that contrasts heavily to the road ahead. Additionally, aviation companies will often use green displays to contrast with the sky.

3.3.1 Projector

The P1+ Mini Projector is very compact, which is extremely valuable in a space-constrained environment. However, the device itself is only capable of 30 lumens output. This isn’t very bright and would easily be washed out in bright sunlight. The unit itself can be found for roughly $100, which is more than we are willing to spend at this time. It is also a self-contained OEM solution and is therefore unsuitable for use in our project. In fact, upon further research it was found that there are few to no bare projectors to be had.

The P1+ Mini Projector was the only projector that was seriously considered. All other projectors that were reviewed were inadequate in multiple ways. Because projectors of suitable brightness and size are too expensive, too power-hungry, and too pre-built, it was decided that projectors would not work for our heads-up display.

3.4 Small LCD Screen to be Paired with Collimating Lens

3.4.1 Option 1: Grove OLED Display 0.96”

The Grove OLED Display, shown in Figure 2, is a 0.96” 128x64 black/white OLED screen. Because the screen itself is small, it is a candidate for use with a magnifying lens to be collimated and focused at infinity. On top of that, the screen uses OLED and is monochromatic black/white. Since OLED black is done by switching off individual pixels, the contrast of this screen is excellent. As a result, the only light that would be seen would be from the active pixels and the resulting reflected image would have no extra “black” in the background. The price is not too bad, as the screen can be found for $15 or less. However, the screen will not be bright enough to be seen in direct sunlight. What's more, the Grove uses OLED. OLED pixels have a finite lifespan, and so our screen can be subject to burn-in or general loss of brightness in as little as 1000 hours! However, the ease of implementation, as well as cost, will be valuable as the Grove OLED allows us to show that the HUD works in practice. While other display technologies may have other benefits, the Grove OLED will allow us to demonstrate our project with ease.

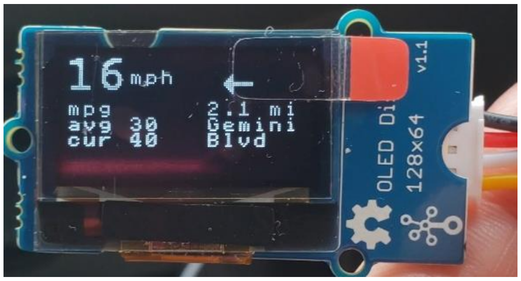


Figure 2: Grove 0.96” 128x64 OLED Display

3.4.2 Option 2: ENH-DG128064-66 Transparent LCD

The ENH-DG128064-66 shown in Figure 3 is another small screen. Unlike the ADA938, this screen uses standard LCD technology and is transparent. This gives us the benefit of a less expensive display that will not suffer from burn-in and gives us the ability to use a separate LED back light of our choosing. This screen, like the ADA938, has a resolution of 128x64, which is not a high resolution but will be readable when used. The key feature is the ability to be paired with a high-intensity LED of our choosing. This gives us the ability to pick an LED that meets our brightness and power consumption requirements while fitting within our size and financial constraints. This screen can be found for as little as $3 online and can be ordered at that price from multiple sources. The low price gives us the option of ordering multiple screens to be combined in a 2x1 or 2x2 setup to achieve a higher resolution that can be used to create higher quality images for the driver. While that would involve extra cost and complexity, these screens are so inexpensive that it could be viable.

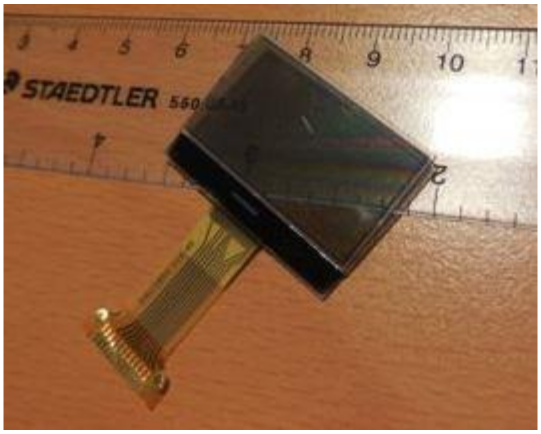


Figure 3: ENH-DG128064-66 transparent LCD display

3.4.3 Option 3: TP241MC01G transparent OLED screen

The TP241MC01G is another small screen that was considered. Unlike the previous two screens, this model has a higher resolution of 128x160. This would give us a much higher resolution that would allow for more detailed graphics to be displayed. This screen is also capable of showing color images, which would give us the ability to use multiple colors in our display. One massive benefit of this screen is that, like the ENH-DG128064-66, this screen is transparent. This would allow us to choose an LED light source that would produce a satisfactory amount of screen brightness. This screen would be roughly twice as large as our other two options, at 2.4" across. The larger size would require the use of a larger positive collimator lens. This screen is also using OLED technology, which would result in a noticeably finite lifespan of our heads-up display. The price of this display dwarfs that of the others we researched, as the TP241MC01G is difficult to find for less than $80. Because of its high price, larger size, and limited lifespan, this screen will likely be passed over in favor of a more compact and affordable option.

Because a heads-up display works best with a collimated image that is focused at infinity, using a small screen with a magnifying lens will likely be the solution we choose for this project. Also, due to brightness considerations, we know that a transparent LCD-type screen will be ideal because we have the freedom to choose an adequately bright LED to illuminate the screen.

3.5 Large LED Screen

3.5.1 Option 1: W050P40PH01 LCD Screen

The LCD screen is a 5" diagonal LED backlit LCD display with RGB capability. The screen is affordable and, more importantly, bright. With a brightness of 1000 cd/m^2, the screen should be bright enough to be seen in daylight. However, this display brightness may be at the lower limits of what is visible on a bright sunny day. What's more, this arrangement does not allow for a collimating lens. Instead, the screen will simply reflect off of a glass surface and its reflection will be focused at a finite distance. While this is common for aftermarket heads up solutions, this does not allow the image to be in focus at all distances the driver may be looking.

Our group found that the best design was to use an OLED display with a set of lenses to collimate the image from the display. This array was chosen for its cost, ease of use, and power efficiency. This setup is also flexible enough that we can optimize the distances between the optical elements, and even include a mirror to redirect where the image appears.

3.6 LED Backlighting

If we are to use a transparent LCD or OLED screen, then we will need to use very bright LED’s as the light source. In fact, the ability to choose bright LED’s is one of the primary reasons for choosing a transparent LCD in the first place. By choosing bright LED’s, we gain the ability to tailor the brightness requirements to our needs. We even have the option of adjusting brightness independent of the LCD display. If we so choose, we can pair the LED with a photodiode or other light-detecting device in order to automatically adjust LED brightness based on the amount of ambient light.

Any LED that we use will have to be paired with both a diffuser and a reflector. The purpose of the reflector is to capture and focus as much of the light as possible. This will allow us to use much of the light emitted by the LED’s, maximizing their potential in this application. Reflectors are commonly used in devices like headlights or flashlights, and reflectors are readily available on the market today.

Because we are using a transparent LCD screen, any light source we use will require the use of a diffuser. A diffuser gives a “frosted” appearance to the emitted light. This will help to soften, and diffuse bright spots caused by the LED. Diffusers are typically applied to light sources used in displays such as televisions and computer monitors. Because our display does not come with its own backlight, we need to use our own diffuser to achieve the same effect.

Fortunately for us, diffusers and reflectors are easily obtained from a number of sources. Because we are looking at small, bright LED’s as our light source, commercial reflectors intended for flashlights would be an effective and inexpensive solution to focus more light into the display. Diffusers are also easy to obtain but are usually separate optical elements that would have to be mounted. However, reflectors exist for flashlights that come with a built-in diffuser that sits at the top of the reflector. Such a setup is usually billed as a reflector that reflects light into a certain shape or pattern with a frosted look. Again, these parts are easily obtained and implemented. A reflector with a diffuser built into it would allow us to solve both LED issues with one part that can easily be mounted. As we are considering small LED’s such a part will be easy to find.

3.6.1 Luxdrive Endor Star 07007-PW740-N

The Luxdrive 07007-PW740-N in Figure 4 is a small PCB with 3 neutral white (4000K) LED’s mounted on it. The setup is intended and marketed towards those who wish to upgrade their flashlights to something brighter. The LED setup is capable of over 500 lumens with a concentrated beam. The setup is rated at 9.0V and up to 700mA, giving it a max power consumption of 6.3W. Unfortunately, that is close to our max power budget of 10W. While it may not be necessary for us to power the device at the full 700mA all the time, having a device that sucks up over half of our power budget would be a large constraint. If we were to use this option, it may be necessary to redesign our input power and switch from using a USB port to using the cigarette lighter.

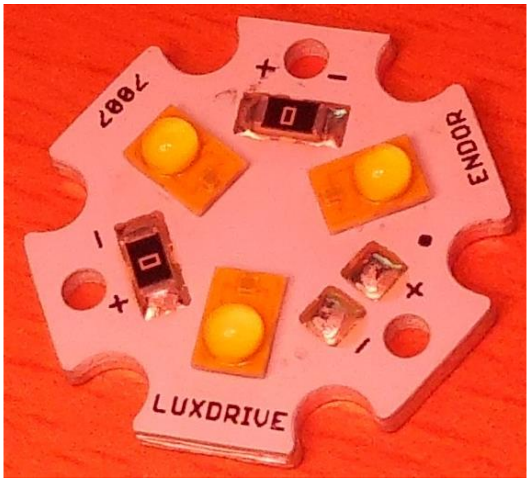


Figure 4: The Luxdrive Endor Star 07007-OW740-N

3.6.2 XC-10W-C

The XC-10W-C is a single LED that is listed as consuming 10W, or the entire output of a USB port. This comes from using a 10V source at up to 1000mA. However, at full power this LED is capable of emitting 1000 lumens. This would require a heat sink, as the LED circuitry would try to protect itself and shut down due to heat after a short matter of time. This can be avoided by using a heat sink on the LED, as well as limiting the input current of the device. It is unlikely that we would need the full 1000 lumens, but having the capability is nice. This LED has a color temperature of 6500K, but there is a version with a color temperature of 3500K.

3.6.3 GT-P25W

The Getian GT-P25W is another white LED, with a max forward current of 1200mA and a max forward voltage of 11V, for a total of 13.2W max power. In real operation, it runs closer to 10W. The LED is warm white, with a color temperature of 3200K. It would need a heat sink for proper operation. The advantage to this LED is its color temperature and its wide operating power range. Depending on our needs, we may not need the full power the LED provides to get a good projected image. Its color temperature means it will be easier on the eyes after prolonged use, compared to cool white.

3.7 Reflective Film

The key to getting a viewable image with a heads-up display is to properly reflect the image off of an optical combiner. The optical combiner acts as a sort of beam splitter, which will both reflect and transmit light that is emitted towards it. In our case, the displayed image will be reflected off such a combiner, whether we choose to use the windshield or a separate sheet of glass. In order to ensure that the driver can see the displayed image, we must ensure that as much of the image is reflected as possible.

One possible way to ensure that we can reflect as much light as possible is to orient the display such that the image is reflected at Brewster's Angle. Brewster's Angle is the point where all the light of a certain polarization will be reflected. However, this approach is not without its setbacks. For one, our display itself is already polarized. This is unavoidable, as all current LCD technologies feature some sort of polarization. Less advanced displays, such as ours, utilize polarization to the extent that the image is invisible when oriented at a certain angle and observed through polarized sunglasses! For testing purposes, we acquired two of the same display. It was noted that the two displays we ordered did not have perfectly matching polarizations, which means that one display may be better suited to utilizing Brewster's Angle. However, testing will be required to determine whether Brewster's Angle will be usable with the polarization that our displays already have.

One other method, one used by readily available heads up displays, is to attach a reflective film to the combiner (usually the windshield) that will reflect more of the light emitted by the heads-up display. Such films are usually reflected on both sides of the film, and the film will appear as a sort of silvery rectangle when viewed from outside to the vehicle. Despite this, the films add a higher reflectivity that is not as affected by the polarization of the light from the heads-up display. As such, the reflective films may be useful if we are not able to utilize Brewster's Angle to reflect the image from the heads-up display. One potential problem introduced using a film is the appearance that something is obstructing the driver's view. This is especially apparent when the film is placed directly on the windshield of the vehicle. This could be a problem, as it is generally illegal to have an object obstructing the driver's view. While the driver's view of the road is not actually obstructed, a passing police officer may still pull the driver over. Such a scenario is not ideal, but further testing will be needed.

Reflective films are readily available and have been used not only for heads up displays, but also in roles such as limiting views inside homes from the outside.

Because augmented reality technology is always advancing, films that perform a similar role are becoming more advanced. Such films may employ such devices as Lucius prisms, which are a collection of rows of triangular prisms with space in between the rows. The space between the rows allows light to pass through the film. The angle of the prisms allows light to be reflected from certain angles. These prism films have higher transmittance than normal reflective films and have higher reflectivity. However, the prism films are sold in larger sheets and are not readily available for small orders. Were we to find Lucius prism film in small quantities for a reasonable price, we would be able to solve our reflectance problem in one fell swoop. Until that time, we are limited to current commercial offerings.

3.7.1 Xbes CA0019

The Xbes CA0019 is a simple reflective film that sticks to glass and reflects light on both sides. While this will reduce the amount of light that is visible from the view of the road, the reduction in light will not impact the driver’s ability to see the road. Such a solution is inexpensive and simple to use. However, reviews on popular retailer Amazon seem to indicate that the film has difficulty staying adhered to glass. While this solution could potentially be overcome through use of adhesives, we are not ruling out the possibility of using a different product.

3.7.2 Red Shield HUD-FILM2PK

Another commercial solution is the Red Shield HUD-FILM2PK. The film, like the Xbes CA0019, simply sticks to glass. It is reflective on both sides and appears as a silvery area on the other side of the glass it’s applied to. Also, like the Xbes CA0019, the Red Shield solution will darken the image of the road as seen through the glass. This will not fully block the view of the road, but further testing will be required to see if the darkened view of the road is an acceptable trade-off for better visibility of the image from the heads-up display.

3.7.3 KOBWA 6025779505680

Like the two products above it, the KOBWA 6025779505680 is another stick-on reflective film. The KOBWA 6025779505680 follows the same trend in that it is reflective on both sides, will darken the view of the roadway where it is placed, and may peel off after months of use in direct sunlight. While it does not hurt to have options, all of our options at the moment are essentially the same.

3.8 Heads Up Display

The main feature of this device is that it is to display GPS information via a heads-up display. Heads up displays have been in use, in some form or another, since World War II, where they found use in fighter planes. In 1942, the British Royal Air Force experimented with projecting information from the radar onto a flat screen that also displayed the plane’s gyroscopic gunsight [6]. The inclusion of the radar readout onto the screen allowed pilots to more quickly engage targets while flying at night. The heads-up display emerged again in the Royal Air Force, who coined the term “heads up display” in the late 1950’s [7]. Heads up displays then went on to be included in different NATO and Warsaw Pact jet fighters.

In more modern times, heads up displays can be found in commercial aircraft and even consumer cars and trucks. In commercial aircraft, heads-up displays serve as a means for increased situational awareness to the pilots. This becomes especially critical when the visibility outside an aircraft is diminished. Leveraging a heads-up display, pilots can have an overlay of the runway ahead of them projected into their field of view. This allows them to verify that the airplane is headed in the correct direction, even if the runway is not physically visible. It is also common to see the aircrafts, speed, altitude, heading, and rate of decent displayed on the heads-up display. All these factors provide enhanced safety, situational awareness, and convenience. It is for these reasons that we would like to implement a heads-up display into our Senior Design Project.

Heads up displays work by using three main components. These components are the projector, the combiner, and the video generation computer [8]. The projector unit uses optical components to collimate the image. To collimate an image, a screen or other display is placed at the focal point of a positive lens or negative mirror. When an object is placed at the focal point of a positive lens or negative mirror, the resulting image is focused at infinity. Instead of the resulting rays converging at a single point somewhere past the optical component, the rays instead stay entirely parallel to each other to an infinite distance. The result of using a collimator in a heads-up display is that the image is always in focus, whether the viewer is looking at an object 5 feet away or 5 miles away.

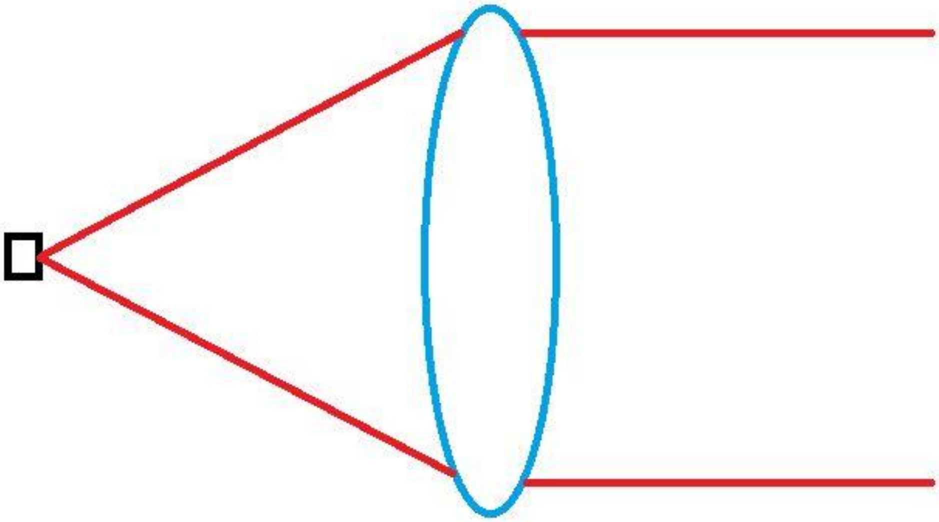


Figure 5: A laser diode and a collimating positive lens.

The combiner in Figure 5 is simply the medium used to overlay the collimated image with some other image, usually to overlay the heads-up display image onto a view of the outside world. For this, all that is needed is a piece of transparent glass. In many heads-up display applications, the combiner is a vertical or near-vertical sheet of glass that is placed within a few feet of the user. In most jet fighter cockpits and in some commercial HUD solutions, the combiner is a vertical or near-vertical glass sheet. In some consumer cars that feature heads up displays, the windshield itself is used as the combiner.

It is possible to create a display without a collimated lens. However, displays without collimated lenses pose some issues with viewing angles. If the driver moves around within the vehicle, the displayed text would not be as readable due to focus distance, lighting, and reflected viewing angle. Collimated displays are the best application for our design. The driver will be able to see a clear and focused image of the display regardless of their position relative to the reflected display. Many projects use collimated displays to achieve this effect. For example, flight simulators will often use a large collimated display to mimic the viewing angles in a real airplane’s flight deck. This allows both the pilot and copilot to see an accurate representation of what is viewable from the respective seats in the flight deck.

The final component of a standard heads up display is the video generation computer. The video generation computer is all of the necessary hardware and software to process data as input and give data as output that can be transformed by the projector into an image. As computers have become more advanced, video generation computers have become faster and more powerful while also using less power and taking up less space. Because of this, video generation computers can be used the heads up displays of a wider variety of vehicles, and even in fields such as augmented reality. Modern-day systems such as Arduino or Raspberry Pi have enough power to be used as video generation computers and in many cases even have pinouts for display devices built into the board itself. Back before the transistor was in common use, such computers would use vacuum tubes that requires much more space and power. This limited the use of video generation computers to vehicles with enough space, like larger aircraft. Some of these systems are specified for different nations' militaries or corporate designs, so even otherwise modern aircraft may still be using vacuum-tube-based video generation computers.

3.8.1 Displayed Content

We would like to use this display to provide the driver with relevant information at a glance, without taking his or her eyes off the road to refocus on the image. The content that we choose to display must not distract the driver in any way. We aim to implement the HUD display with the following features:

Speeding is a common occurrence on our roads today. Speeding sometimes occurs because the driver does not look at the speedometer to check his or her current speed often enough. We would like to make the driver's current speed visible on the display to make the driver more aware of their speed. This could reduce the risk of speeding tickets and increase the safety of both the driver and the other drivers on the road.

As discussed later in Section 7.0, we will be adding advanced navigational features into the HUD device. We would like to implement turn by turn navigational data by displaying an image of the driver's upcoming turn. Additionally, we would like to display a countdown timer with the distance from the driver to the upcoming turn. This will allow the driver to be aware that a turn is coming and make the proper accommodations to make that turn safely.

We would also like to add the vehicle's current fuel consumption to the display. Some models of vehicles provide data about its current fuel consumption including instantaneous miles per gallon and average miles per gallon. With our project's goal being increased driver awareness and making an impact on society, we think that providing information about current fuel consumption could make the driver perform more economically and save fuel. In the long run, this could aid in reducing emissions and fuel consumption further.

As we move forward in the project, we will take into account which data is the most relevant to display to the driver. This will also depend on what make and model we use as our test vehicle, as older vehicles do not provide as much data about current conditions as compared to newer vehicles.

4.0 Onboard Diagnostics Data (OBD)

Before the digital explosion of the 1990’s, vehicles did not have accessible information regarding the current state of its systems. If a vehicle began to develop a mechanical issue, the vehicle’s owner would have a difficult time troubleshooting the issue without the knowledge of a professional mechanic. Even then, the mechanic would have to do extensive manual diagnostics to find what the issue could be.

In 1991, the California Air Resources Board (CARB) began to push vehicle manufacturers to make new vehicles that included Onboard Diagnostics for emission control. The Onboard Diagnostics would provide information about how much fuel the vehicle was using, and the types of emissions were being produced as a result of driving. In 1994, CARB made the OBD standard mandatory for all vehicles created after 1996. This standard forced vehicle manufacturers to provide a standardized method for error codes, allowing for easier diagnostics and transparency about vehicular systems.

In 2020, all modern vehicles are equipped with Onboard Diagnostics tools that provide information about the current state of the vehicle. This information is comprised of Digital Trouble Codes (DTCs) that contain data about certain parameter of the vehicles systems. Primarily, this allows mechanics to diagnose issues that arise without taking the entire vehicle apart. Modern vehicles are equipped with OBD2 ports that are accessible near the driver’s seat, into which a mechanic can plug an OBD2 scanner into the port to analyze the current state of any part that adheres to the standard.

4.1 Digital Trouble Codes (DTC)

Accessing data from the OBD requires us to locate the parameter identifier (PID) listed in SAE J1979 for each desired value. The following is a list of DTCs we used to access the data we wished to display from section 3.7.1:

4.1.1 Speed Readout

The vehicle’s speed can be read by accessing the PID of value 0x0D. The returned value is 1 Byte, containing the exact value of the vehicles speed between 0 and 255 kilometers per hour. Considering our main market is the United States that uses imperial units, we converted that value into miles per hour before pushing to the display.

4.1.2 Fuel Consumption Rate

The vehicle's fuel consumption was calculated using a compound equation containing values from the vehicle's speed reading and the mass airflow sensor. A vehicle's mass air flow is a ration of the amount of air to fuel. The ratio is ideal when 14.7 grams of air are present for every gram of gasoline. The mass air flow sensor can be accessed from the PID of value of 0x10. The returned value will be 2 Bytes and needs to be converted to grams/sec by using the following formula $\frac{256A+B}{100}$, where A is the first Byte and B is the second Byte. The following equations can be used to obtain the miles per gallon.

$$GPH_i = \frac{MAF\ gram/sec}{14.7\ gram/sec} \times \frac{1\ lb}{454\ gram} \times \frac{1\ gal}{6.701\ lb} \times 3600\ sec$$

$$MPG_i = \frac{Speed}{GPH}$$

As we further developed the OBD2 system, we looked into other parameters that could be useful for displaying information about the vehicle's current status to the driver.

4.2 Freematics OBD-II UART Adapter V2 (for Arduino)

The Freematics OBD-II UART Adapter is a product that is compatible with many Arduino products, including the Arduino Mega 2560 that we used in our project. It features access to all standard OBD-II PIDs with an extended ELM327 AT command set. The main purpose was to read and clear diagnostic trouble codes from the engine and powertrain. The main interest for us was to use readouts that include vehicle speed, engine RPM, throttle position, calculated engine load, intake pressure and temperature, and fuel pressure just to name a few. In addition to supplying data, this adapter was capable of supplying power to the Huddy Buddy, along with a backup battery in the event that power supplied by the OBDII port was suddenly cut. The OBD-II plug has two separate pins for its own V_{CC} and ground. It can supply a regulated 5 volts and up to 2.1 amps. If the device is turned off it also has a low power mode drawing only 5 mA.

The interface for this connector used a serial UART data interface with both 3.3-volt and 5-volt micro-controllers like the one that we used. The adapter plugs into the OBD port usually located under the steering column of the car. All modern cars past 1996 are OBD-II certified and depending on the car's model year will have more supported features because of additional sensors in the car. To check a car's certification under the car's hood on the vehicle emission control sticker it will have a note that it is OBD-II certified. A list below shows the following countries and years of manufacture that it supports:

- United States (Gas) 1996+
- United States (Diesel) 2004+
- Canada (Gas) 1998+
- Europe + UK (Gas) 2001+
- Europe + UK (Diesel) 2004+
- Australia + NZ (Gas/Diesel) 2006+

For using the OBD-II adapter cable, the cable itself splits into 2 two-pin connectors. One set is for the power which will be the red for V_{CC} and black for ground. The other set is for data; the white cable is for R_X which connects to the Arduino's serial T_X and the green cable is used for R_X which then connects to the Arduino's serial R_X. The adapter has a dedicated Arduino library that is maintained regularly and has easy-to-use API's to retrieve real-time data from the vehicle.

This adapter comes from arduinodev.com and will cost $ 25.00 for an individual unit.

5.0 Components Selection

5.1 Power Supply:

There are several USB types from the plug inside to the connector. The most prominent are the USB 2.0 type A and USB 3.0 type. These plug versions both provide 5 volts nominal. The difference between the USB 2.0 and USB 3.0 type A is that the USB 2.0 type A provides 500 mA compared to the USB 3.0 which can provide 1.5 to 3 A over the 5-volt bus. The other big difference was that the USB 2.0 has a throughput 480 Mb/s and the USB 3.0 has a throughput of up to 5 Gb/s. For power needs it is not as important. The next decision is whether to use a USB micro B or USB type C connection ends. The micro USB is more simplistic with only four pins, two for power and two for data transfer. The USB type C has an additional five pins, in terms of current three of those wires are for standard downstream port, a charging downstream port, and a dedicated charging port. The charging downstream port and dedicated downstream charging port supports up to 1.5 A.

To supply power to the HUD the USB 3.0 A to micro type B connector and cable will be used. The reason why is because the micro type B is cheaper and more readily available with breakout boards that allow for easy access to the two pins that are needed for power. They may have a lower rating but the good thing about USB is that you can plug any USB device into any USB cable and into any USB port. The initial idea was to use the cigarette port, but more and more cars are solely using the USB connectors in cars and doing away with the cigarette ports. If that is not an option, a typical car charger that has a USB connector will still work. Below are three choices that use a female connector USB board mount. The factors that go into this decision are cost, accountability, and time to ship.

5.1.1 Option 1: USB Micro-B Breakout Board Product ID: 1833

This first option comes from the online supplier Adafruit. They offer a breakout board with the female USB type B connector and 5 pins broken out to easily attach wires to supply power to a device. In addition to easy mounting and use they even supply a small stick of 0.1” header so it can be soldered on and plugged into a breadboard. It has through hole shielding pads for a strong connection. It costs $1.50 and can ship in less than a week. Also, there was a video demonstration that shows how robust this connector is. The datasheet also shows that this item can withstand temperatures from -30 to 80 degrees Celsius, which helps to comply with our need for higher temperature resistance.

5.1.2 Option 2: BOB-12035

This is another breakout board from DigiKey. It seemingly offers the same as the breakout board from Adafruit, with a female USB type B connector with 5 pins broken out for easy connection. Looking at the datasheet did not give as much information. It could be assumed that it can withstand higher temperatures of a car, but it is not known for sure. This board costs $2.50 and can also be shipped in a week. This board does not seem as robust, but it is still cheap and fits the functional requirements.

5.1.3 Option 3: 2174507-2

This third option is a little less friendly but looks more professional. This USB type B female connector is just the connector part. It does not offer the easy access to the pins as the other two options provided. This part is more prone to come off as it does not have the reliable through hole mounts rather this would just be soldered onto the PCB and could prove a problem with repeated connects and disconnects. This part is $1.87 and can also ship within a week. Looking at the datasheet also gave less than desired information, still assuming it could resist temperatures in a car it is not explicitly given.

5.2 Power Regulators:

The next item on the list is something that can convert the 5 volts to something smaller for the smaller components. The simplest item to use for this would be a linear regulator but that can only be tuned using the adjustment pin with resistors, if that would be adequate. Otherwise a buck converter might have to be used a tunable potentiometer for precise voltage regulation.

5.2.1 Option 1: LM317T

This is a basic linear regulator which can take in 5 volts and make it into something smaller. The drawback of using linear regulators is that it produces a good amount of heat. That means getting a heat sink, which means more space taken up inside the unit. The LM317T can operate between 0 and 125 degrees Celsius. If in the event the part overheats it has over current and over temperature protection. The regulator can take in a maximum of 40 volts and have a minimum of 1.2 volts minimum with tuning on the output of the adjustment pin. The output current is up to 1.5 amps. But they're very cost effective, this part is only $0.64, and through Digi-key can ship out on the day of purchase.

5.2.2 Option 2: LM2596 DC-DC Adjustable Buck Converter 3.2-46V to 1.25-35V Step Down Power Supply High Efficiency Voltage Regulator Module

The next option is a buck converter. The only downside to a buck converter is that they are marginally more expensive, this is a four pack so individually they would be about $2. With that being said, the buck converter can take in 3.2 volts to 46 volts, having an output of 1.25 volts to 35 volts with a maximum of 3 amps output. The trick here is that the input must be 1.5 volts higher than the output, so not that we would need it, but it cannot be used as a boost converter. To set this up, connect the input to the input terminals and output to the output terminals. Then tune the potentiometer to the desired voltage level. Another thing is that if this buck converter isn't used for very long periods of time, it is heat efficient and will not require a heat sink.

5.2.3 Option 3: TPS61222DCKR

This is a tiny boost converter that would take in 5 volts and convert it to anything between 1.8 to 6 volts. It has a high efficiency above 90 percent for 5-volt input to lower output, but with higher output current. This is a good choice because it operates at -40 to 85 degrees Celsius and will not require a heatsink to dissipate power loss. In addition, it has output overvoltage, overtemperature, and input undervoltage lockout protection. This comes with its own schematic and set of equations for making the output voltage adjustable. Ideally this will provide power to most of the HUD, except the screen display. This boost converter comes from Texas Instruments and sells for one dollar and ships in five days.

The simple equation for determining output voltage is $R_1 = R_2((V_{out}/V_{FB})\text{-}1)$. V_{FB} should be at 500 mV so choosing R_1 and R_2 becomes easy. Figure 6 below shows the schematic of the TPS61222DCKR and how it is to be used as a voltage regulator.

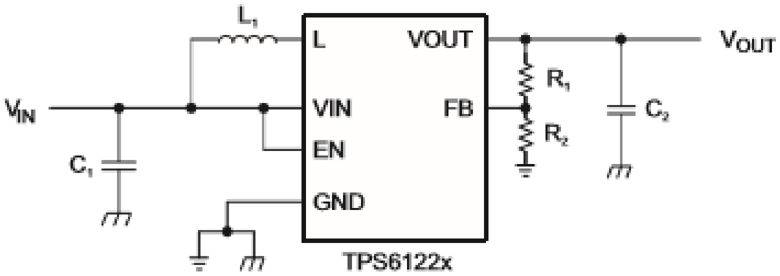


Figure 6: Schematic for TPS61222DCKR
Courtesy of Texas Instruments Inc.

5.3 Boost Converters

Boost converters are an easy way to create higher voltage than the input voltage. The tradeoff is that the higher voltage reduces the amount of current provided from the input. The basic setup of a boost converter is an inductor in series with a voltage source, with a switch to ground and then a diode and capacitor in series which are in parallel to the switch. The way it works is when the inductor is connected to ground via the switch current flows through it for a very small amount of time that allows magnetic energy to be stored inside of it, and when the switch is opened the polarity of the inductor changes so current flows through the diode and two sources are now in series which charge the capacitor. The switch that allows this to happen must cycle on and off fast enough to not ruin the inductor, usually performed by a switching device.

Initial estimates, ranging from 10 to 12 Watts, show that the power consumption of the device will be a little higher than what a single USB port alone can provide. So, to combat this a boost converter will be used to generate a higher output voltage source primarily for the bright screen display that is required. Boost converters are cheap and can either be bought or made. The downside of using them is they take up room and generate heat.

5.3.1 Option 1: Super XL6009 DC-DC Adjustable Step-up Boost Power Converter

This step-up booster is on par with what is required. Being able to take in 5 volts as the input and being able to output about 12 volts at 0.8 amps, this provides 9.6 Watts of power. This is at the lower end of the estimate but should be able to handle the power requirements of the display. In addition, the operating temperature of this device is between -40 to 85 degrees Celsius. This has a very simple implementation design where the input voltage is connected to the input terminals and the output has output terminals. A plus side too, it that the output can be tuned with the adjustable potentiometer on the boost converter for precise output conditions. This product comes in a set of two for $6.93 from amazon.

5.3.2 Option 2: LT1613CS5#TRMPBF

This option is a build it yourself option. The LT1613 is the integrated circuit only. To make it a boost converter the rest of the circuit is made with other elements attached to the LT1613. There is a schematic provided already to boost 5 volts to 12 volts at 130 milliamps. This gives a smaller power output of around 1.56 Watts, which is on the much lower end of the power requirement that was needed. The problem with this option is that there is a very long manufacturing lead time to get this piece, it would take about 8 weeks to ship. The single component is cheaper at $4.28. Figure 7 includes the schematic for the 5 volts to 12 volts conversion.

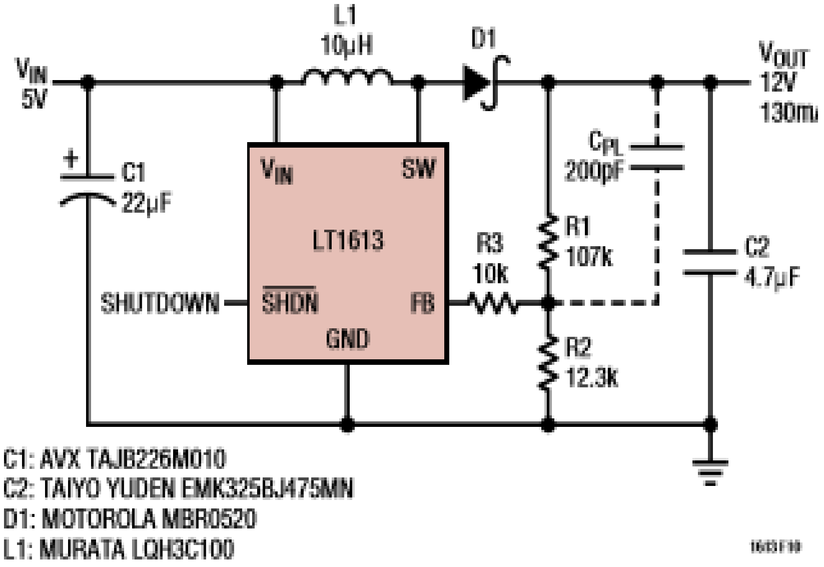


Figure 7: Schematic for 5 Volt to 12 Volt using LT1613
Courtesy of Texas Instruments Inc.

5.3.3 Option 3: LM2577-ADJ

The LM2577-ADJ is an adjustable boost DC-DC switching regulator. This is also a build it yourself option. This component has a wide input voltage range from 3.5 to 40 volts. With simple schematic design, the LM2577 can be made to boost 5 volts to 12 volts at 800 milliamps. This results in a power output of 9.6 Watts. This is still on the lower side of the anticipated power requirement but should be enough to power the bright display. The LM2577 will operate at around 25 degrees Celsius when at 12 volts. In addition, this part is readily available to be shipped with an asking price of $8.24**,** which is more expensive but worth the delivery time. Figure 8 shows the schematic for the LM2577-ADJ.

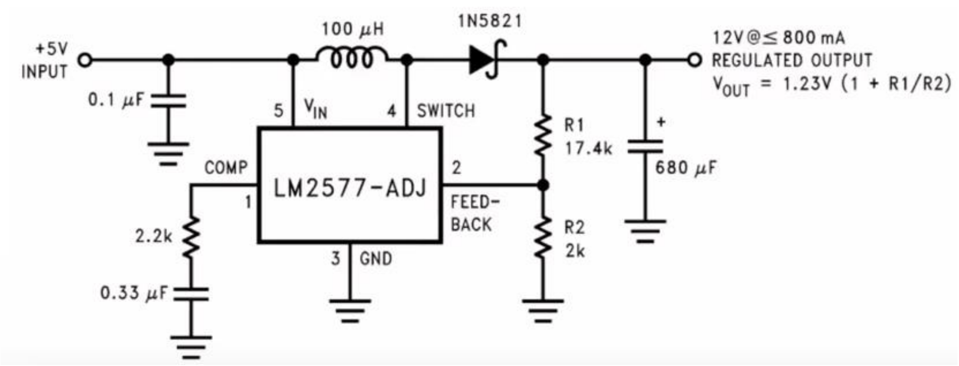


Figure 8: Schematic for the LM2577-ADJ for 5V to 12V

Courtesy of Texas Instruments Inc.

5.5 Light Sensor:

The reason for a light sensor is so that when the HUD is in use it knows when to dim and brighten the display. This will be useful for the transition between day and night. This does not need to be a complex light sensor rather something that's close to the human eye.

5.5.1 Option 1: Adafruit ALS-PT19 Analog Light Sensor Breakout

This is another breakout board from Adafruit. It's very simple with a power need of around 2.5 – 5.5 volts and once it is on the only thing to do is read the analog voltage on the OUT pin. As light increases the voltage increases. A bonus is that it's RoHS compliant. Due to the high rejection ratio of infrared radiation, the spectral response of the ambient light sensor is close to that of human eyes. This goes for about $2.50 and can be shipped out in less than a week. Also, it meets the temperature performance of being able to operate between -40 and 85 degrees Celsius according to the datasheet.

5.5.2 Option 2: OPT3007YMFR

This product is made by Texas Instruments. This is a good fit because it's a super thin light-sensor with a fixed I^{2}C address. This device matches the human eye with rejecting more than 99% of infrared light. It's able to measure between 0.01 to 83k Lux, this will allow for fine tuning when dimming the display. It has a very low operating current at 1.8 μA which is what we're looking for in terms of the limited power that is available. The dimension of this chip is 0.856-mm × 0.946-mm × 0.226-mm which is space efficient but will require a PCB since it's a surface mount device. The OPT3007 will operate between -40 to 85 degrees Celsius. One last upside is that this is a smaller version of the OPT3001 that was used in the embedded systems class, so this isn't a totally new component to figure out.

5.5.3 Option 3: Adafruit 161

The Adafruit 161 is a very simple photodiode. This CdS cell respond to light between 400nm and 600nm wavelengths, peaking at about 520nm. Basically, all they can detect is if there is light or if there isn't light. For this reason, they shouldn't be used to try to determine precise light levels in lux. So, the upside is that they are very cheap, this one is $0.95, and they are very robust so no worries of it giving out. To read this our main CPU will determine what the voltage off the photodiode is. The higher the voltage the brighter it is.

5.6 Making Arduino Talk:

A key feature to include is to have output voice commands to the user that tells where they are going in their route. Arduino has a library that will allow the Arduino Uno to take WAV file types from a SD card reader and transmit it to a speaker connected to the Arduino Uno.

The reason the WAV file type was chosen is because it is a standard PC audio file format. These files are mainly used in professional music recording industry to maintain the best audio quality. With being the best though these files usually start at around 10 MB per minute that is recorded and can easily go up to 4 GB. This is not going to be a limiting factor because each voice command is going to be a matter of seconds only and the storage on a Micro SD card is extensive and inexpensive. The reason for such large files is because the files are uncompressed and lossless whereas an mp3 file for example is compressed and lossy.

The addition of speakers allows for visual cues as well as audio cues to be given. The idea is that if there is a turn to be taken, the user can hear that a turn will come up in so many feet, or if there is an alternative route to be taken that the option is there and can be selected when prompted. The goal in choosing a speaker is that it must use low power. With low power though, it must also be loud enough to be heard over normal traffic conditions. The next few sections will talk about how it will be implemented.

5.6.1 Option 1: Breadboard-Friendly PCB Mount Mini Speaker - 8 Ohm 0.2W

This little speaker has it all. It's small and robust being only about 30 mm in diameter. The pins can fit perfectly into a perfboard too. The speaker is an 8 Ohm and uses 0.2 W or less of power. The optimal temperature range is within -10 to 40 degrees Celsius. The frequency range is between 600 Hz to 11 kHz. There is a class D amplifier that will work with the speaker if that is something that is needed down the line. This speaker is $1.85 and can be delivered in about a week.

5.6.2 Option 2: Mini Metal Speaker w/ Wires - 8 ohm 0.5W

This tiny 1-inch diameter speaker cone has an 8 Ω impedance and will be using 0.5W or less of power. This particular speaker is very simple, and its metal body is extremely lightweight. The rated frequency range is like the speaker above, operating between 600 Hz to 10 kHz. Also, it has a good temperature resistance being able to operate between -20 to 55 degrees Celsius. Again, this speaker can work well with class D amplifier if it is something that is needed. This speaker is $1.95 and can be delivered in about a week.

5.6.3 Micro TF Card Memory Shield Module SPI Micro Storage Card Adapter for Arduino

This SD card reader, as per the name, is compatible with the Arduino ATMega2560. It is the ideal choice for easy integration into the project. This supports either the regular Micro SD card, or the Micro SDHC card. The power supply is a 4.5 to 5.5-volt source and pulls about 80 mA on average which is within the available power limits of the project. It has six pins, two for power and ground, then four others for MISO, MOSI, SCK, and CS (chip select). The data on the SD card would contain WAV file types that are used for the audio outputs for giving directions. The communication interface is a standard SPI interface. This part is only $ 2.09 from the online source banggood.com.

55.6.4 SASTFOE Green Edition 8GB U1 Class 10 TF Micro Memory Card for Digital Camera MP3 TV Box Smartphone

This Micro SD card has 8 GB of storage so that translates to about 200 songs averaging 4.1 MB, but it this SD card won't be used for songs, but rather short audio commands so the average space goes way down, which will ensure that a robust library can be created for a wide range of commands. This is a U1 class 10 SD card; U1 means that it has a theoretical maximum bus speed of 104 MB/s and the class 10 denotes that it can write at 10 MB/s. These are a standard norm these days among micro SD cards, so they are widely available and cheap. This part will come from banggood.com as well which costs $ 3.99.

5.6.5 Working Together

The SD card module V$_{CC}$ connects to the 5-volt of the Arduino board and the ground is connected to the ground of the Arduino. Next connect the chip select (CS) pin to digital 4 pin of the Arduino. SCK connects to digital pin 13 of Arduino. MOSI then connects to digital pin 11 of the Arduino. MISO pin connects to the digital pin 12 of the Arduino. Then one wire of the speaker will connect to ground of the Arduino and the other wire will connect to digital pin 9 of the Arduino. The reason why this connection is so specific is because the library that enables this speaker function uses these specific pins. The Micro SD card uploads through the computer that has audio files that are converted to WAV files through online conversion websites, there are plenty. Wherever the file is converted it must have a bit resolution of 8 bits, sampling rate of 16000 Hz, and select mono for audio channels. From there the last thing to do is create the code that uploads to the Arduino that allows turn directions to be given.

5.7 Dimming Circuit

The photoresistor helps tell when the outside light gets brighter or dimmer, but now an actual dimmer needs to be created for the LED once it is known what level of light is needed to clearly see the display of the heads-up display. Looking at Table 12 below from the datasheet for the Luxdrive Endor Star 07007-OW740-N, the white LED has a rated 2.5 forward voltage and a maximum of 3.51 volts. The individual LED’s are the LXML-PW51 which when combined have a collective voltage drop of about 9 volts.

Color	Part Number	Forward Voltage (V$_f$)		
		Min.	Typ.	Max.
Warm White	LXML-PW71	2.55	3.00	3.51
Neutral White	LXML-PW61	2.55	3.00	3.51
Warm White	LXML-PW51	2.55	3.00	3.51

Table 12: Characteristics of the LXML-PW51

To produce the maximum number of lumens, it can draw up to 700 mA. The challenge is to control the current that flows through the LED. One way to accomplish this is have a power supply of 9 volts that is supplied using a step-up voltage regulator. From the voltage supply the Luxdrive Endor Star 07007-OW740-N is connected to the drain of the N-channel logic-level FET, FQP50N06L. Then the source connects to the base of the NPN transistor, 2N5088BU followed by a 2-watt rated resistor. A pulse width modulation output from the Arduino Mega 2560 supplies pulses of 5 volts through a resistor to the gate of the NFET and to the collector of the NPN transistor. The emitter of the NPN transistor connects to ground.

The resistor that is connected to ground will be a power resistor. The reason being that so much current and heat will be created during the different diming levels of the circuit that it would melt a regular resistor. This should be about two times the power calculated that will drop across the resistor. Ideally it should handle up to about half a Watt of rated power. This also handles the set current for the LED.

The idea behind this is that the power NFET will act as a variable resistor which is turned on through the resistor connected to the PWM. Then the NPN transistor will act as an over current switch that is triggered through the resistor connected to the base of Q1. The reason the FQP50N06L was chosen was for its ability to handle large amounts of power. Alone it can handle about 0.66 Watts before a heatsink is needed. Then the microcontroller does the rest with the PWM signal. To make the LED dim, the PWM will turn on and off rapidly. In addition, the ratio of on-time to off-time will need to be adjusted. This works in conjunction with the analog input from the photoresistor circuit that determines how bight it is outside. The brighter it is outside the higher the level of intensity the LED needs to be and vice versa when it gets dark it does not need to be as bright, so it is not blinding, and too distracting, driving in the dark. Figure 9 below is a schematic of how the dimmer circuit works. As a disclaimer not all the parts are supported in Multisim so a symbol with the appropriate name was created, but unable to be simulated.

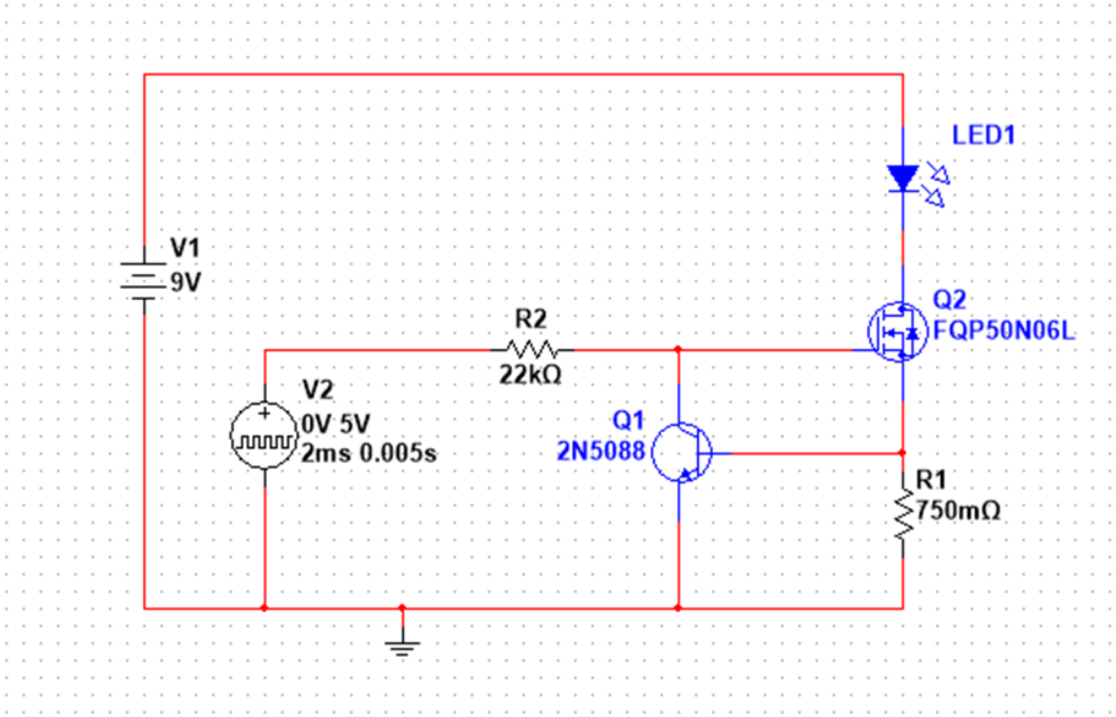


Figure 9: Schematic of Dimmer Circuit

5.7.1 2N5088

This is an NPN general purpose amplifier. This is needed as a feedback loop in the dimmer circuit. The reason this part was selected was based off the need for its ability to handle higher voltages and current. According to the datasheet it was given the following specs below:

Absolute Maximum Ratings:

- V_{CEO} = 30 V
- V_{CBO} = 35 V
- V_{EBO} = 4.5 V
- I_C (continuous) = 50 mA

On Characteristics:

- $V_{CE(sat)}$ = 0.5 V (with IC = 10 mA, IB = 1.0 mA)
- $V_{BE(on)}$ = 0.8 V (with IC = 10 mA, VCE = 5.0 V)
- DC Current Gain = 300

5.7.2 FQP50N06L

This is a N-channel QFET MOSFET designed to handle high voltage and high current well above that is required for the LED that is to be used. It is designed to provide superior switching performance and have low on-state resistance. The purpose of this MOSFET is to act as a resistor to control the current that flows through the LED, and the advantage of using this MOSFET is that any current that is not used by the LED when it is dimmed can be sent through the MOSFET without any trouble handling it. According to the datasheet the following specs were given:

Absolute Maximum Ratings:

- V_{DSS} = 60 V
- I_D = 52.4 A at T_C = 25 °C
- I_D = 37.1 A at T_C = 100 °C
- V_{GSS} = 20 V

On Characteristics:

- V_{GS} = 2.5 V with V_{DS} = V_{GS} and I_D = 250 µA
- Forward Transconductance = 40 S

5.8 Backup Battery

In the event of a crash and no power can be provided to the SMS module, there still needs to be power to allow for the emergency text message or email to be sent out to the respective emergency contacts. In order to make sure that, as long as the module isn't broken, a notification can be sent a backup battery circuit will be implemented. The normal DC power supply is connected to a blocking diode rated at a current higher than the power supply. Then the rechargeable battery is connected to the circuit with a resistor and another diode. The value of the resistor is very important because while the battery isn't being used its going to be charged. Overcharging is a factor to consider since damaging the battery is not desired. To determine the value of the resistor the difference from the voltage source and the battery is determined. Ideally the source is 5 volts and the voltage of the battery is about 3.6 volts. This gives a difference of 1.4 volts. To be safe the current to charge the battery should be 8 mA or less. Dividing 1.4 volts by 8 mA this gives a resistor value of 175 ohms. Staying on the side of caution the value of the resistor will be around 500 ohms. Even though this takes the charging current way down this is not an issue since the backup battery is not used very much. If need be a voltage regulator can be put at the output. The backup system was simulated in Multisim under both conditions, notated in Figures 10 and 11.

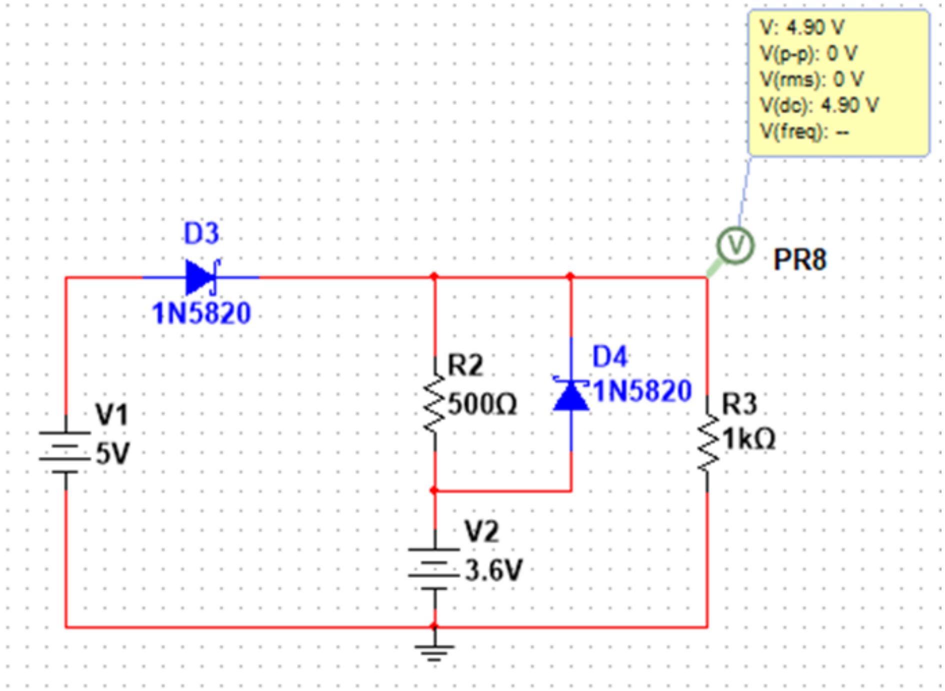


Figure 10: Battery Backup with Main Power Supply

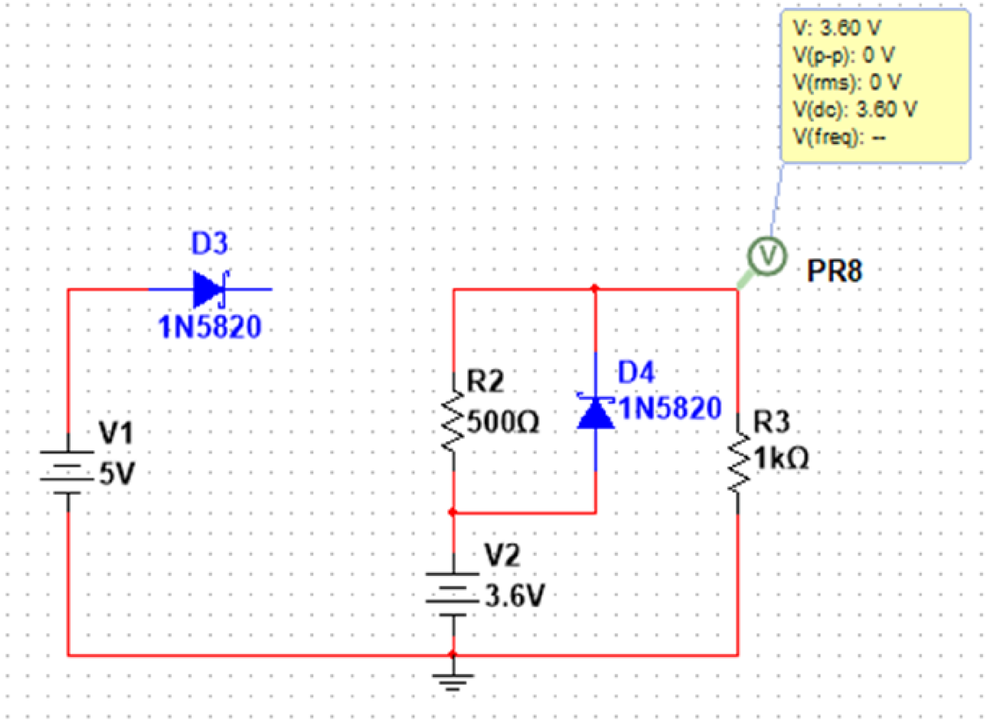


Figure 11: Battery Backup with Battery as Power Supply

As seen in the simulations this system works just fine. The diodes being used are the 1N5820 as shown in the schematic. The battery used is the Samsung 25R 18650 2500mAh 20A Battery. It outputs 3.6 volts and is rechargeable as is required. The battery is held in place with a single AA battery 18650 battery holder with wire leads are used for the placement of the rechargeable battery.

5.8.1 Overcharge Protection

Taking into consideration of the battery in the backup battery system, an overcharge protection circuit could be implemented if need be. Although it would take a very long time to charge the battery it is still a good idea to have. This overcharge protection circuit would be placed in parallel with the battery. The best option for this single cell AA battery is the PCB-S1A6TH. This is a protection circuit board (PCB) rated at the following specs below:

- Over-charge protection voltage：4.300 V ± 0.050 V
- Over-discharge protection voltage： 2.40 V ± 0.100 V
- Over-current protection：7 ~ 9 A
- Maximal continuous discharging current：5 A
- Maximal current consumption： 10 µA
- Short circuit protection： Automatic Recovery
- Protection circuitry resistance: ≤ 40 mΩ

5.9 Parts Chosen

The main power supply ultimately became the output from the OBD-II port and cable. Instead of opting for USB female connectors, 2.54 mm pitch standard male pin headers were chosen to accommodate the female ends of the OBD-II connectors. In the end the LM2577 was chosen for its ability to convert 5 volts to 9 volts with a current rating high enough to supply the whole device with the LED, the ATMega2560, and SIM808 all running off it. Later it was discovered that a part that was described in the schematic of the ATMega2560 could not be found, the voltage regulator NCP1117ST50T3G, so a different part, the LD1117S50TR, was used instead that has the same parameters as the original part with better current handling and voltage dropout ratings. The Adafruit 161 light sensor was an easy choice with the thought in mind that a hole for the sensor can be made, and without it needing to be placed directly on the PCB, it has the freedom to be placed where maximum light could hit it. Finally, the Samsung 25R 18650 2500mAh 20A Battery was used as the backup battery.

Unfortunately, the section, Making the Arduino Talk, could not be implemented with time restraints. Another section, Dimming Circuit, was tested as described with all said parts, but was ditched once testing the original transparent LCD wasn't going to work. The light sensor was still to be used with the new OLED screen obtained from seeedstudio.com to change the contrast lighting on the display. Another nixed idea was the overcharge protection since the project was moved to more of a development environment after COVID-19 placed time restrictions on part ordering.

6.0 Navigation

One of the main features we wanted to implement into the heads-up display was displaying navigational information to the driver. Looking at a phone for navigation can be dangerous and distracting as it removes the driver's eyes from the road ahead. It is possible to eliminate this risk by displaying pertinent navigational information right in the driver's field of view. For example, if a driver's desired route has a series of turns, a counter with how far the driver is from the turn will appear, along with street names and the direction in which the driver has to turn. This increases the driver's awareness and reduces the possibility of making a wrong turn or getting lost. To accomplish this, we designed a process to obtain the path the user will take from starting point to destination. There were a variety of ways we could create a path from beginning to end.

6.1 Predetermined Routing

Predetermining the route ahead of traveling means we could gather coordinates for each turn along the way. The waypoints could be determined by mapping out the user's desired route using an application such as Google Earth. The route would most likely contain a series of turns that the driver will have to make in order to reach the destination. At each turn, coordinates would be created to mark the waypoints that the driver must drive through to stay on the route. These waypoints could be entered into the device and stored for processing. Using a GPS chip, we would be able to keep track of where the driver is in relation to those stored waypoints. When the driver begins his or her route, a distance variable would keep track of how far the driver is from the first point. This information would be displayed to the driver on the Heads-Up display in the form of a visible distance counter. As the driver gets closer, the distance would be shown counting down accurately. Once the driver gets to specified distances from the next waypoint i.e. 1 mile, the system would announce a turn is ahead using the built-in speaker and sound recorded onto the device. Additionally, the HUD would display an arrow in the direction of the next turn as well as the street name that the driver will be turning on to. This would allow the driver enough time to make note of his or her surroundings and prepare to make the turn. Once the driver reaches the waypoint, the system would move to the next waypoint and begin counting down the distance to it.

This process of predetermining the route the driver would take and inputting the turn coordinates manually has advantages and disadvantages alike. For advantages, it drastically reduces the number of components needed to implement the device. The driver would predetermine the waypoints of each turn, enter and store them into the device as coordinates, and then have the GPS chip determine the distances between each of the coordinates as the driver is on the route. Being able to reduce the components required to implement the device means that the cost to both manufacturer and consumer can be minimized, as well as complexity to build would decrease. This decreased complexity would allow the device to require less power, making it more efficient electronically. As for disadvantages, predetermining and storing waypoints is an extremely rigid system. It does not allow the driver a quick way to modify the route once it has been programmed into the device and started. If the driver wishes to change the route from what is already active on the device, the driver would need to manually reprogram each waypoint for the GPS to track. We anticipate that this would be a time-consuming process and would require the driver have access to a computer and the interface required to program the device. Due to the inflexibility of this approach, it is unlikely that we would implement our device in this fashion.

6.2 Google Cloud Platform API

As mentioned above, manually entering waypoints into the device is not the most efficient way to implement navigation for our HUD. A much more efficient implementation would leverage the power of pre-existing and well-established navigational platforms to assist in obtaining route information. Google Maps a mapping application for mobile devices that uses the device’s internal radios, sensors, and GPS to stream information about the user’s current location to Google’s infrastructure, determining the user’s exact location along a route down to great precision. Maps is extremely useful for determining the most efficient route between the starting point and destination. Engineers from Google have spent years developing algorithms that analyze real-time traffic data, road closures, and other variables to give the driver the safest and most reliable route. The application’s user can add custom filters to the route, such as avoiding tolls or prioritizing distance against time.

Leveraging information from Google Maps would be the ideal scenario for implementing a navigational system into our HUD device. In order to do this, we would need to create a custom mobile application that would implement API calls to the Google Maps platform and obtain the data for a given route. For example, the driver would enter his or her desired destination into our application. The Maps API would return the most efficient route to the driver’s destination at the time of computation. This information could be streamed from our mobile device to the

HUD device over wireless standards like Bluetooth. The HUD device would then show the route information just as in the previous implementation.

The ability to use Google Maps API would enable us to build a deeper and more advanced implementation for navigation. It would allow us to provide real-time updates to the driver's route. This opens the door for advanced features to be implemented, such as speed limit monitoring. Google Maps has information regarding the speed limit for the road the driver is currently driving on. The speed limit is displayed on the application, notifying the driver and potentially preventing them from speeding. We would like to implement this feature into our HUD by accessing the API for the speed limit of the road the driver is currently on and displaying it in the driver's field of view. We can combine this information with the live speed read by the onboard OBD2 port of the vehicle. If the driver's current speed goes beyond the speed limit retrieved from Maps, we would like to advise the driver to slow down, ensuring the driver is within safe driving conditions. The use of APIs would make route modification much easier, as it would only require the driver to enter the new location into our application and recalculate the route. This process is much easier than having to remap every coordinate by hand like the previous implementation. Hazards along routes such as traffic jams and accidents are often spontaneous and unplannable. We could implement a function to ping the API every few minutes to check for any hazards along the current route. If the API returns that a hazard lies ahead, we could have the API recalculate the route to avoid the hazard and keep the driver on the most efficient route.
Ultimately, implementing a custom mobile application paired with Google Maps APIs and streaming the information to our HUD would provide us with the most flexibility and advanced feature set. This is the implementation we would like to strive for when building the device.

One of the biggest drawbacks about the Google Maps Cloud Platform is that they have strictly disallowed the use of the platform in any embedded devices, according to Section 3.2.3 (h) of the Google Maps Platform Terms of Service. Due to this restriction, we had to look at other mapping services that would allow the use of the service in embedded applications.

6.3 Mapbox Routing API

Mapbox is an online mapping tool similar to Google Maps. A user can view maps, enter locations and get turn by turn directions to a desired destination. Powerful API calls from the Mapbox SDK, Mapbox Geojson, Mapbox API Core, and essential Android API allowed us to ascertain our goal via phone application of having an easy to use mapping interface that gets a user's destination and creates a turn by turn list that becomes packaged and sent off to the Huddy Buddy's screen for display. In order to use Mapbox APIs, an access token was required which was acquired by signing up for an account on their website and listing the features we would be pulling for this application. Then, within the Huddy Buddy application's code, the first API call includes the Mapbox Mapview object creation with the access token as one of the parameters. Mapbox routing works by inputting two points that include latitude and longitude coordinates for the origin and destination. Then, using a callback response allows us to access the details of the route from the overall length to a specific turn direction at a specific leg.

7.0 Application

To support advanced navigational features noted in the previous section, we needed to access the APIs provided from Mapbox API Platform. There are already applications that exist that can access information from the Mapbox API and the standalone Mapbox application, however we do not have access to the source code of these applications and cannot ensure that the data we need for this project would be available. Additionally, the data would need to be streamed over our Bluetooth module once the destination is chosen by the user.

7.1 Application Design

The best method for ensuring that all the required data is being received is to build a mobile application from scratch. We will be creating an application for Android phones using the Android Studio, since the development kits are readily available and easily implemented. Cost to implement this application will be minimal as we already have access to Android devices. The application's design language is going to be simplistic and minimal, enabling the user to locate the desired destination and send it to the HUD with ease. The mobile device will communicate with the HUD by pairing the two devices over a Bluetooth connection.

The application design features a search bar at the top for the driver to enter his or her desired destination. As they are entering the destination, the Mapbox API will be suggesting destinations based on the text input from the driver. Once the desired destination appears, the driver will select it from the list. The Mapbox Routing API will then be called to find the most efficient route from the mobile device's current location to the destination.

Once the route is returned, it will populate in an embedded map in the center of the screen. The driver will press the Send to HUD button when ready to proceed on the route. The Routes API will return each turn on the route as a coordinate, which can be used to create waypoints along the route. These waypoints will enable turn by turn navigation to be implemented in our HUD. Once the coordinates are obtained, the application will push them to the HUD from the Bluetooth connection. From this point, the responsibility for navigation is passed to the HUD device, where it will start comparing distances between the HUD's onboard GPS chip and the route’s first turn coordinate. The application is not needed for navigation until a new route is desired.

The application will provide the driver with a convenient way to interface with the HUD. Since most people already own a smartphone, there will not be any additional cost to implementing this design.

7.2 Application Implementation

To develop a phone app, there are many factors that first must be considered. What operating system is it going to be for, what program will we use to test code, what kinds of API calls and file structure will it have, etc.

The phone app for the HUD was decided to be designed for use on Android devices. Android is the most popular phone operating system on the market right now. Developing applications for it includes a massive amount of support and guides due to its large userbase. With all the resources available and its accessibility, we chose Android over all other operating systems. The other operating system considered was Apple’s iOS. One problem with iOS is the ability to load apps on to an iPhone for development purposes. Apple is very strict with what can be installed on their devices and applications require certifications and regulation passing in order to be put on the AppStore. Another issue is the members working on this project do not have Mac computers which are used to develop iOS applications. To avoid all the rules and restrictions, it made sense to just go with Android.

To develop an android application, an IDE must be used. There are many to choose from, but the most common and supported one is Android Studio. Google themselves advocate this IDE and have official guides that use Android Studio as its IDE. The IDE that was used before was Eclipse and some guides and documentation that are older use it. Eclipse is now deprecated and the company that releases Eclipse has told its user to move to Android Studio as well.

When it comes to choosing a programming language to use within Android Studio, it came down to Java and Kotlin. Java has a huge history within the programming world and is used practically everywhere, and Android application development has been mostly Java up until recently. Kotlin is a newer programming language that is fully compatible with Java and JVM (Java virtual machine) but has more concise syntax and implementations of crucial components of code design. Ultimately the language chosen was Java due to the programmers on the team having years of prior experience with the language. It is possible for the future of Android application implementation to be exclusively Kotlin, but for the time being Java is more than capable of meeting the needs of this project.

7.3 Application Features

The application's main purpose is to keep a constant connection between the HUD and the Mapbox. The most important information that is being sent back and forth is the GPS coordinates of the HUD's current location, and what the location of the next checkpoint along your route is. This information is all being sent in the background and the user does not see much of the process of what is being sent. There are some features that are required to be displayed on the app, especially important first time run settings.

When the application is running for the first time there are settings that must be set from the main screen. The first thing that must occur is permissions for contacts, cellular antenna, and Bluetooth. This is done very simply in Android studio by doing

```
<uses-permission android:name="android.permission.BLUETOOTH" />
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
```

Both permissions must be checked for. This is because BLUETOOTH_ADMIN allows the app to discover Bluetooth devices and pair with them while BLUETOOTH allows to connect with already paired. Other permissions are set in a similar matter. If permissions are not activated, the app will close.

Contacts and Cellular Antenna permissions are required for emergency contact purposes. The HUD will have an accelerometer that keeps track of the vehicle's speed and direction. The purpose of this is if the car gets in an accident and the change in speed is extremely abrupt, an emergency contact is immediately messaged to inform them of what happened.

Data permissions must also be approved of. In order for the HUD to receive GPS coordinates and route information, the Mapbox API must be used in order to grab such information.

The interface upon startup will display the logo of the HUD while running startup code. Once all the startup code is done running, the app will load the route screen where there is four tabs at the bottom. The four will be Route, Status, Settings, and About. On the Route tab, the user will be able to enter an address that is desired as the destination. When an address is chosen, the Mapbox API is called, and GPS coordinates are given to the HUD. Based on the coordinates of the HUD itself, the HUD will operate using these pieces of information. The Status tab will show the connection's health between the HUD and the app. If any sorts of connection issues occur whether it be Internet sided, Bluetooth sided, or some other error, it will be shown here. The settings page will include places where you can input who your emergency contact is, and small other features such as dark mode and text size. The About tab will include information about the creators. Figure 12 below demonstrates the design our application implemented on the Android device.

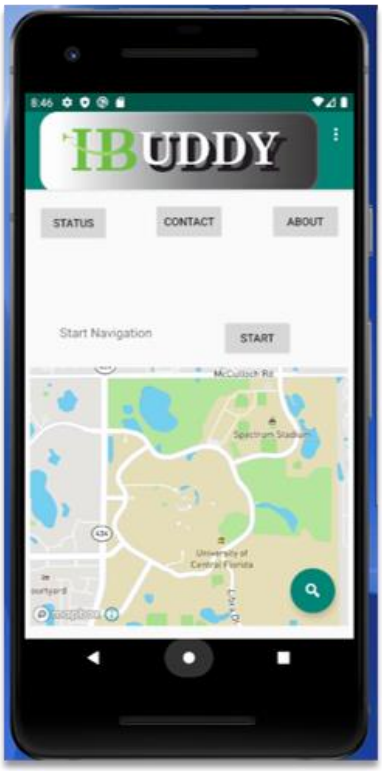


Figure 12: Application Interface Example Look

8.0 Location Tracking:

As mentioned in Section 6, our project implemented navigational features to give the driver enhanced awareness when driving to a destination. Implementing these advanced navigational features required us to ensure the location of the device is always known. When we initially began brainstorming how this could be accomplished, we had some ideas about ways we could go about tracking the HUD Device. The following is a list services that will provide the HUD Device with its current geographical location.

8.1 Google Cloud Platform Location

The simplest and least complex method of tracking the device's location would be to use the information taken directly from the Google Maps API running in the application on the mobile device. Since the mobile application will already be reaching out to the Google API, it would be minimal to add in the ability for the application to read the device's current location and update this to the HUD. The

Google Cloud Platform API would provide longitude and latitude for the exact location of the device. The GPS data could be stored in the HUD device to control the contextual information displayed on the HUD.

8.2 Mobile Device GPS Location

Our second option would be analyzing data provided by the onboard GPS of the paired mobile device. API Libraries provided in the Android Studio application would allow us to build GPS tracking directly into the application. This option as well as the Google API option would limit the need for extra components, lowering the overall cost and power required to implement the HUD. Once the data from the API has been received, it would be sent over to the HUD device via Bluetooth, limiting the amount of computation needed to measure distances between waypoints along the route. Additionally, using this method would provide

8.3 Standalone GPS Module

The third option for GPS capabilities was to add a standalone GPS unit into the HUD’s design. The GPS module works by detecting multiple satellites in geosynchronous orbit and using their relative positions to triangulate its exact position on Earth. As our HUD device was mounted around the windshield of the vehicle, the GPS module would be able to sync with the satellites without interference. Implementing a standalone GPS chip would add power draw to the system and increase the cost of the overall design. Additionally, it would create a scenario where the distance given is not accurate to the driven distance, as it would be a direct distance between too coordinates, rather than the distance that the driver will have to travel on a given road. However, it would allow us to gain experience coding for extra components that we do not have experience with, as well as add complexity to our final PCB. Standalone GPS modules are readily available on the market.

8.3.1 Ublox NEO-6M GPS Module

The NEO-6 module interfaces directly with the Arduino’s UART over the TX/RX pins located on the module itself. The module is available for purchase on Amazon from DIYMall with an integrated GPS antenna to synchronize with the signal from the GPS satellites. Since the NEO-6M module comes implemented on its own breakout, we will need to source all of the components required to operate and implement them on our own custom PCB.

8.3.2 GPS from FONA 3G Module

In contrast to the Ublox NEO-6M, the GPS module located within the FONA 3G module would be a much better implementation for tracking location by connecting to GPS satellites. The GPS data can be read directly from the 3G Module without any additional chips. The only additional accessory that will be required is an Active External GPS Antenna, as the FONA 3G Module does not come with one integrated on the board. Implementing the FONA 3G Module will have some other benefits as well. The total cost of implementation of advanced navigational features will decrease, since we are combining the use of an already purchased device. Additionally, we will save on power consumption if we only need to implement the 3G module without needing to add a separate GPS module.

8.3.3 GPS from SIMCom SIM808 2G GSM/GPS Module

Later in Section 10.2, we will be discussing the SIM808 Module in depth. Like the FONA 3G Module, the SIMCom SIM808 2G Cellular GPS Module implements a GPS signal from an external GPS antenna. The SIMCom SIM808 Module is less expensive that the FONA 3G Module, due to the older standard of the 2G Cellular protocol. Since the GPS capabilities are like the FONA 3G module at a lower cost, it would be much more efficient to use this module.

8.4 Implemented Method – Mapbox API

After researching the above methods of GPS tracking, it became apparent that the most accurate method of tracking the Huddy Buddy's current location for navigation would be to use the local GPS on the mobile Android device. We were able use our custom Android application to upload the Android device's current location to the Mapbox API to determine the distance from the device and the next waypoint along the route. This would ensure that the distance returned between the waypoints was attributed to the length of the roads that would be travelled, rather than a straight line over the Earth. The Mapbox GPS API is called every half second in order to provide proper routing. The destination coordinates remain the same, but the origin point is updated with the current location of the device. Any changes to routing would be quickly updated and presented to the user on the Huddy Buddy screen.

9.0 Wireless Communication:

The HUD must have a feasible way to communicate with a user's cell phone. This will allow crucial data to be transferred to and from the devices in order to maintain contact. Since the communication is to be with a user's cellphone, the protocol needs to be wireless. This means communication types such as serial and Ethernet are out of the picture. These methods, although very stable and well established in the tech development industry, will not work as running a wire through a car would be a safety hazard as well as unfeasible as most phones only have one port which is used for charging. If the device forced users to lose their one port to the HUD then it would not be a popular item in the slightest.

Fortunately, there are a lot of different types of wireless protocols available for different requirements. Popular protocols include Wi-Fi and Bluetooth, with lesser known ones such as ANT and ZigBee used for low power devices. The distance between master and slave, which is the user's phone and our device respectively, is going to be very small as it will be no larger than within a car. This allows us to choose a low power wireless protocol which fits our design best/

Based off specifications listed below, this project will be optimal under the use of Bluetooth low energy.

9.1 Wireless Protocol Comparison

Below is a list of popular wireless protocols that are used for electronics around the world for different purposes. Each will be given a brief description along with its typical uses followed by Table 13 which concisely compares the features of each type.

- Bluetooth Asynchronous Connection-Less Highspeed (ACL/HS) is the standard type of radio link used for general data packet transmission. ACL packets are retransmitted automatically if unacknowledged, allowing for correction of a radio link that is subject to interference. There is another feature dubbed forward error correction that will automatically reduce data rate in favor of reliability in order to keep data packets aligned without having to retransmit data. ACL has a packet formatted as access code(72bit) + packet header(54bit) + payload + CRC (16bit) which allows high bandwidth data to be sent through using additional time slots of x1, x3, or x5 depending on payload size. It has high performance but requires higher power than other wireless protocols.

- Bluetooth Synchronous Connection-Oriented (SCO) is a type of radio link that allows voice data to be sent along side of small data packets. This technology is used for Bluetooth headsets and microphones.

Retransmission is not a feature in this type, but forward error correction is still available. This would be very important if the HUD were to have features such as voice commands in order to complete tasks rather than using touch.

- Bluetooth Low Energy (BLE) is the low power radio link that cuts a lot of features of ACL and SCO Bluetooth in order to provide the fastest most efficient packet transfers while using a fraction of the power. A BLE device remains in sleep mode constantly except when a connection is initiated, and connect times are only a few milliseconds compared to ACL and SCO where it can take up to 100 milliseconds.
- Wi-Fi and Wi-Fi Direct are super high frequency high bandwidth high range wireless protocols that are the most powerful of all the protocols researched. Wi-Fi is extremely popular, and most people use it every day to get internet on their phones, laptops, tablets, and other wireless devices. Although powerful, this protocol would not work with our HUD as Bluetooth ACL is more power than desired and Wi-Fi is more powerful than ACL.
- ZigBee and ANT are two different wireless protocols that are comparable to Bluetooth Low Energy. They are used in many devices that are low power such as wireless sensors and heartrate monitors. The range of these protocols are larger than BLE and they use less power overall as well

Type	Voice	Data	Audio	Video	Low-Power
Bluetooth ACL/HS	X	Y	Y	X	X
Bluetooth SCO/eSCO	Y	X	X	X	X
Bluetooth Low Energy	X	X	X	X	Y
Wi-Fi	Y	Y	Y	Y	X
Wi-Fi Direct	Y	Y	Y	X	X
ZigBee	X	X	X	X	Y
ANT	X	X	X	X	Y

Table 13: Wireless Protocol Comparison

Since the device will only need minimal communication between itself and the phone, characteristics such as audio and video streaming are not needed. The data column represents the ability to file transfer between one device and the other. Because the only real communication needed is packets to be sent over rather than whole multi megabyte files, the data column is not needed either.

ZigBee and ANT are good options with their own perks. ZigBee has a larger range than Bluetooth Low Energy (BLE), almost three times as much. ZigBee also supports multi node connections and is much more stable with multiple connections coming into it. Since this device only needs one node as a connection and distance is not an issue, BLE has 1/10 the power consumption and most importantly is compatible with Android and Apple operating systems which makes it perfect for device to smartphone connectivity.

ANT is ultra-low power, so low that it can run off a coin cell battery for years. The problem with ANT is that it is too low power with not enough features available for this project. ANT does not support operating systems ran on phones similarly to ZigBee and so the most logical choice is to use BLE.

BLE is the fastest and cheapest 2.4 GHz wireless protocol in the market. It’s the only wireless protocol that you can use with iOS without needing special certification, and it's supported by all modern smart phones.

9.2 Bluetooth Low Energy Specifications and Factsheet

BLE follows IEEE standards and so there is a set performance rate of the protocol across all BLE devices. Bluetooth and other wireless protocols have become very popular in modern Internet-of-Things (IoT) and embedded devices. In order to maintain interoperability between each of these concepts, the IEEE laid out a set of rules that electronics should abide by when implementing Bluetooth Protocols.

Specifically, to be qualify for a Bluetooth Low Energy device, certain thresholds must be met to ensure the power consumption does not exceed a certain value. Below are specifications of BLE that are important to be aware of, as it includes performance, security, and stability. Specifications given to meet these requirements are located in Tables 14 and 15 below.

Technical specification	**Bluetooth Low Energy technology**
Modulation	GFSK @ 2.4 GHz
Latency (from a non-connected state)	6 ms
Minimum total time to send data (det. battery life)	3 ms (3)
Voice capable	No
Network topology	Scatternet
Power consumption	0.01–0.50 W (depending on use case)
Max current consumption	<15 mA
Service discovery	Yes
Profile concept	Yes
Modes	Broadcast, Connection, Event Data Models, Reads, Writes
Primary use cases	Mobile phones, gaming, smart homes, wearables, automotive, PCs, security, proximity, healthcare, sports & fitness, Industrial, etc.

Table 14: BLE Specifications and Fact Sheet

Technical specification	Bluetooth Low Energy technology
Distance/range (theoretical max.)	>100 m (>330 ft)
Over the air data rate	125 kbit/s – 1 Mbit/s – 2 Mbit/s
Application throughput	0.27-1.37 Mbit/s
Active slaves	Not defined; implementation dependent
Security	128-bit AES in CCM mode and application layer user defined (2)
Robustness	Adaptive frequency hopping, Lazy Acknowledgement, 24-bit CRC, 32-bit Message Integrity Check
Connections	> 2 billion

Table 15: BLE Specifications and Fact Sheet Continued

9.2.1 Bluetooth Low Energy Specifications Points of Interest

Distance fits our need perfectly. Considering the distance between HUD and phone will be no more than a couple of meters, 100 meters is quite overkill. This however potentially allows us to add some functionality such as the HUD beginning to turn on once a phone connects to it. Since most people allow Bluetooth devices to automatically connect once it is within range and powered on, the option to add functionality involving greeting the user and automatically turning on within a range is added.

Security is very important, especially with wireless communication. BLE uses AES-CCM which is a mode of operating block ciphers which use a deterministic algorithm. This means that given a specific input, the encrypted output will always be the same as the key does encryption key does not change. This allows encrypting of data to be efficient and predictable to the developer to make sure it is working as intended, while also keeping data safe if the key is hidden. Figure 13 is a schematic behind how AES-CCM works within the context of 128bit packets.

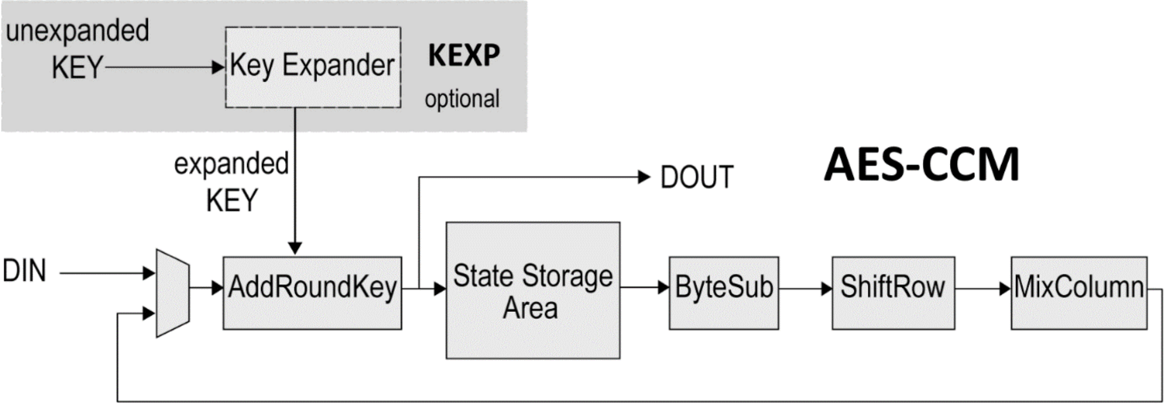


Figure 13: AES-CCM Algorithm

Speed and power of this BLE protocol fits our requirements as well. With a latency of 6ms, information gets sent at the blink of an eye. Also, the power requirements will be low as the nRF52840 does not need much power to begin with. This is listed under 8.5 Bluetooth Module Specification.

9.3 Bluetooth Low Energy Architecture

The physical layer (PHY) refers to the physical radio used for communication and for modulating/demodulating the data. It operates in the ISM band (2.4 GHz spectrum). This clock is standard for most wireless connectivity, even other protocols such as Wi-Fi use the 2.4 GHz spectrum.

The Link Layer is the layer that interfaces with the Physical Layer (Radio) and provides the higher levels an abstraction and a way to interact with the radio (through an intermediary level called the HCI layer which we'll discuss shortly). It is responsible for managing the state of the radio as well as the timing requirements for adhering to the Bluetooth Low Energy specification.

Direct Test Mode: the purpose of this mode is to test the operation of the radio at the physical level (such as transmission power, receiver sensitivity, etc.).

The Host Controller Interface (HCI) layer is a standard protocol defined by the Bluetooth specification that allows the Host layer to communicate with the

Controller layer. These layers could exist on separate chips, or they could exist on the same chip.

The Logical Link Control and Adaptation Protocol (L2CAP) layer acts as a protocol multiplexing layer. It takes multiple protocols from the upper layers and places them in standard BLE packets that are passed down to the lower layers beneath it. Figure 14 gives an overview of how each layer is defined in the BLE stack.

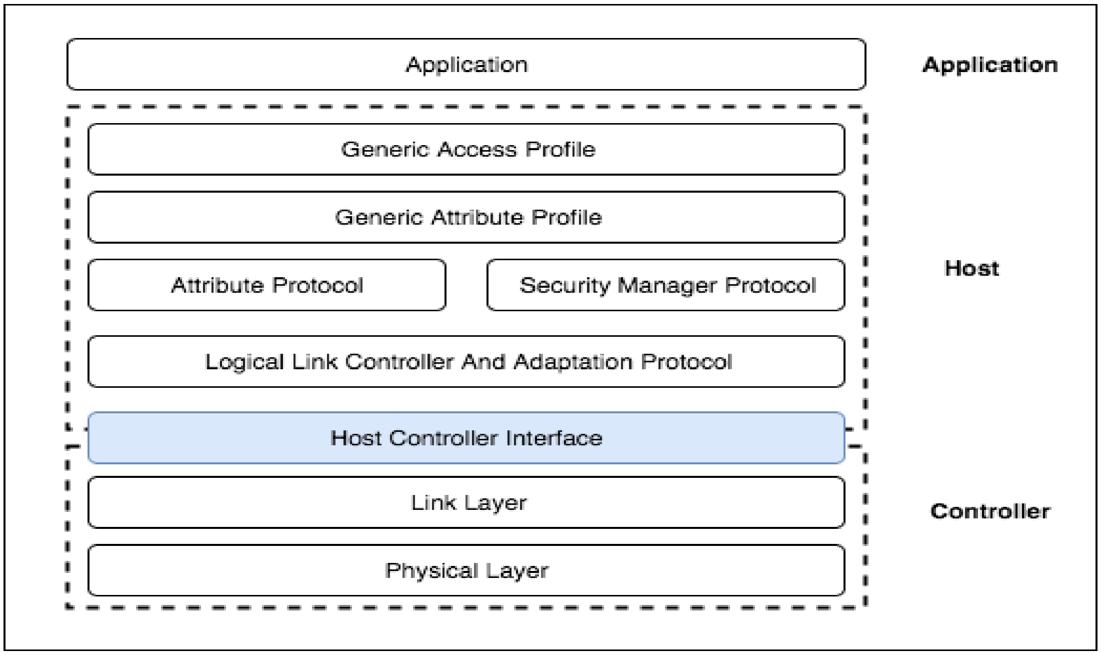

Application
Application
Generic Access Profile
Generic Attribute Profile
Attribute Protocol
Security Manager Protocol
Host
Logical Link Controller And Adaptation Protocol
Host Controller Interface
Link Layer
Controller
Physical Layer

Figure 14: BLE Architecture

9.4 Bluetooth Packet Format

Bluetooth is a networking protocol that follows similar structure to other wireless networking protocols. Data is sent between transmitters and receivers using a packet to package the data. These Bluetooth packets are packaged in a specific format, described in detail below and in Figure 15 and Table 16.

Preamble: It is used by the BLE module for synchronization of time and frequency. The preamble also performs AGC (Automatic Gain Control). It is a predefined pattern of size 1 byte which is known to the receiver. Advertising packet use "10101010" in binary. Data packet use either "10101010" (if LSB of access address is 0) or "01010101" (if LSB of access address is 1) in binary form.

Access Address: For all advertising packet is uses fixed pattern "0x8E89BED6" in hexadecimal form with size of 4 octets. or 32 bits. This address is nicknamed “bed six” and is consistently the access address across every BLE module as part of a standard. For data packets it consists of a 32-bit random value generated by BLE device in "initiating state". The same value is used in a "connection request (CONNECT_REQ)" message.

PDU: It consists of either "advertising channel PDU" or "data channel PDU" as defined in the figure. The advertising channel 0000 (ADV_IND) will be the primary mode used for this project.

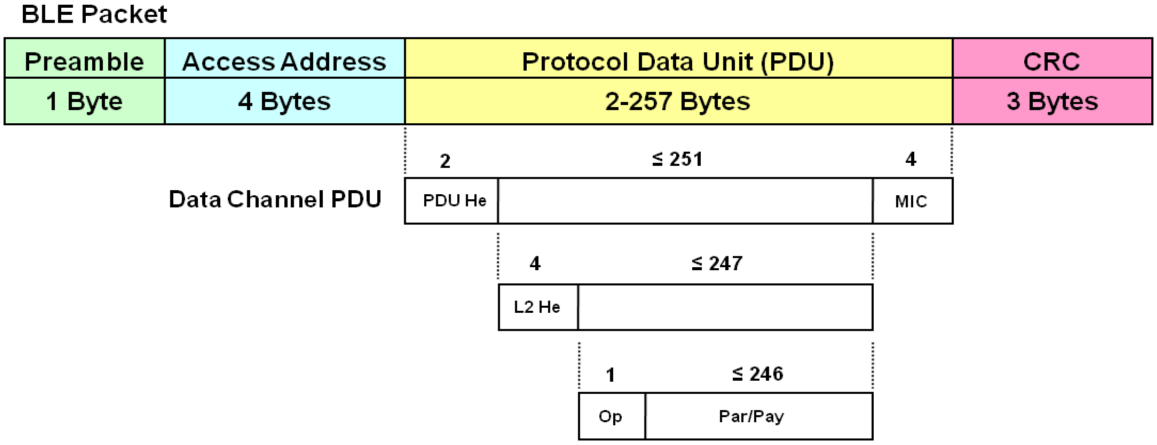


Figure 15: BLE Packet Format

PDU Type	Packet Name	Description
0000	ADV_IND	Connectable undirected advertising event
0010	ADV_NONCONN_IND	Non-connectable undirected advertising event
0110	ADV_SCAN_IND	Scannable undirected advertising event

Table 16: PDU Packets

CRC: It is 24 bit in size. It is calculated over PDU. It is used for error detection of the packet. CRC is calculated using polynomial of the form $x^{24} + x^{10} + x^9 + x^6 + x^4 + x^3 + x + 1$.

This in total allows a packet to have a size of 265 bytes. The fastest a packet can be sent from a BLE module is 7.5 milliseconds. This allows an extremely fast transfer rate, however for consistency sake and for timing, we will send packets at a slower rate.

9.5 Bluetooth Module Specifications

Below in Tables 17 and 18 are the technical specifications of the nRF52840 and HC-05. For the HUD, we needed to make sure that this module would fit power requirements and have a speed that was ample to keep up with the packets needed to be sent. The processor must support Bluetooth 5 at the minimum, meet minimum power requirements, and have a strong built in antenna radio. Ultimately, the HC-05 was chosen over

Technical specification	**nRF52840**
Processor	32-bit ARM® Cortex™-M4 CPU with floating point unit running at 64 MHz.
Radio	Bluetooth 5, IEEE 802.15.4, 2.4 GHz transceiver
RF Sensitivity	-95 dBm sensitivity in 1 Mbps Bluetooth low energy mode -103 dBm sensitivity in 125 kbps Bluetooth low energy mode (long range)
RF Power	-20 to +8 dBm TX power, configurable in 4 dB steps
Data Rate	Bluetooth 5: 2Mbps, 1Mbps, 500kbps, 125kbps IEEE 802.15.4-2006: 250 kbps Proprietary 2.4 GHz: 2 Mbps, 1 Mbps
Power	1.7 V to 5.5 V supply voltage range
USB	USB 2.0 full speed (12Mbps) controller
Other Serial	QSPI 32MHz interface, SPI 32 MHz, Up to 4 x SPI masters / 3 x SPI slaves with EasyDMA
NFC	Type 2 near field communication (NFC-A) tag with wake-on field･Programmable peripheral interconnect (PPI)
Support	I2C, UART, QDEC
Temperature	-40°C to +85°C

Table 17: nRF52840 Specifications

Technical specification	**HC-05**
Bluetooth protocol	Bluetooth Specification v2.0+EDR
Frequency	2.4GHz ISM band
Modulation	GFSK(Gaussian Frequency Shift Keying)
Emission	≤4dBm, Class 2
Sensitivity	≤-84dBm at 0.1% BER
Speed	Asynchronous: 2.1Mbps(Max) / 160 kbps, Synchronous: 1Mbps/1Mbps
Security	Authentication and encryption
Profiles	Bluetooth serial port
Power	+3.3VDC 50mA
Working temperature	-20 ~ +75Centigrade
Dimension	26.9mm x 13mm x 2.2 mm

Table 18: HC-05 Specifications

9.5.1 nRF52840 Schematic Configurations:

The nRF52840 has different features that can be activated through different circuit layouts. Below in Figures 16, 17, and 18 are the two that were considered for this project with each having different components on and off. Schematics were taken directly from Nordic’s website, who designed the chip.

9.5.2 VDDH vs VDD

VDDH is the high voltage configuration while VDD is the regular configuration. VDDH is to be used when the voltage provided is more than 3.6V. The highest the voltage can be for VDDH is 5.5V. The cap at 3.6V is due to internal regulators not being able to sustain more than that voltage when in that mode.

9.5.3 EXTSUPPLY

An external supply is only available when running in VDDH mode. This is because external supply only needs to exist if you need certain features that need that 3.6V voltage where 5.5V is too high. For example, if the chip was powered by USB, the chip must be in VDDH mode and the VDD pins can now act as an external supply to external circuitry.

9.5.4 DCDCEN0 and DCDCEN1

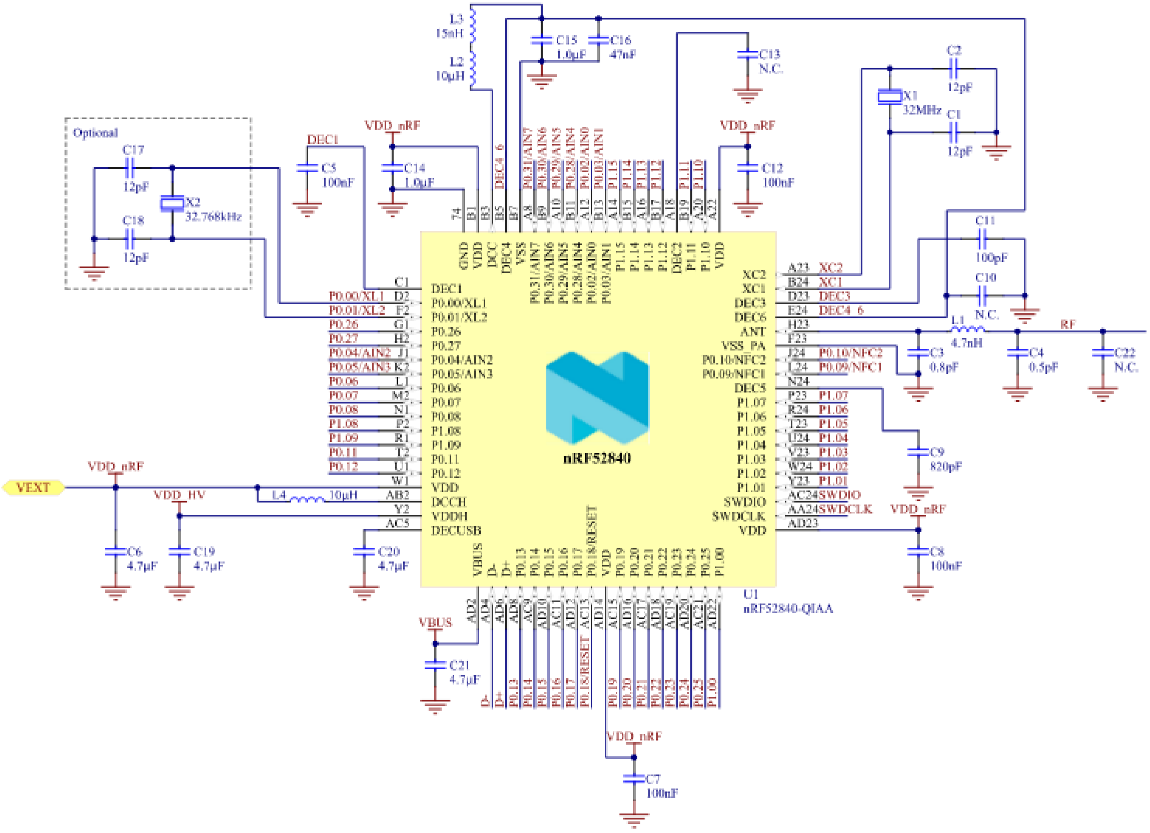
When DCDCEN1 is set, it allows low current DC/DC conversion regulator to be activated for REG1 and DCDCEN0 for REG0. A figure below shows the schematic of the DC/DC and LDO interact.

9.5.5 USB

Allows the chip to be powered through the Universal Serial Bus without an external power source, note that this forces you to run in VDDH mode if this design choice is taken. USB can be turned on for either voltage mode, but VDD pins must be used in this case. For the development board, a USB will be used to power the chip but in the final custom PCB version, the chip will get a direct battery source.

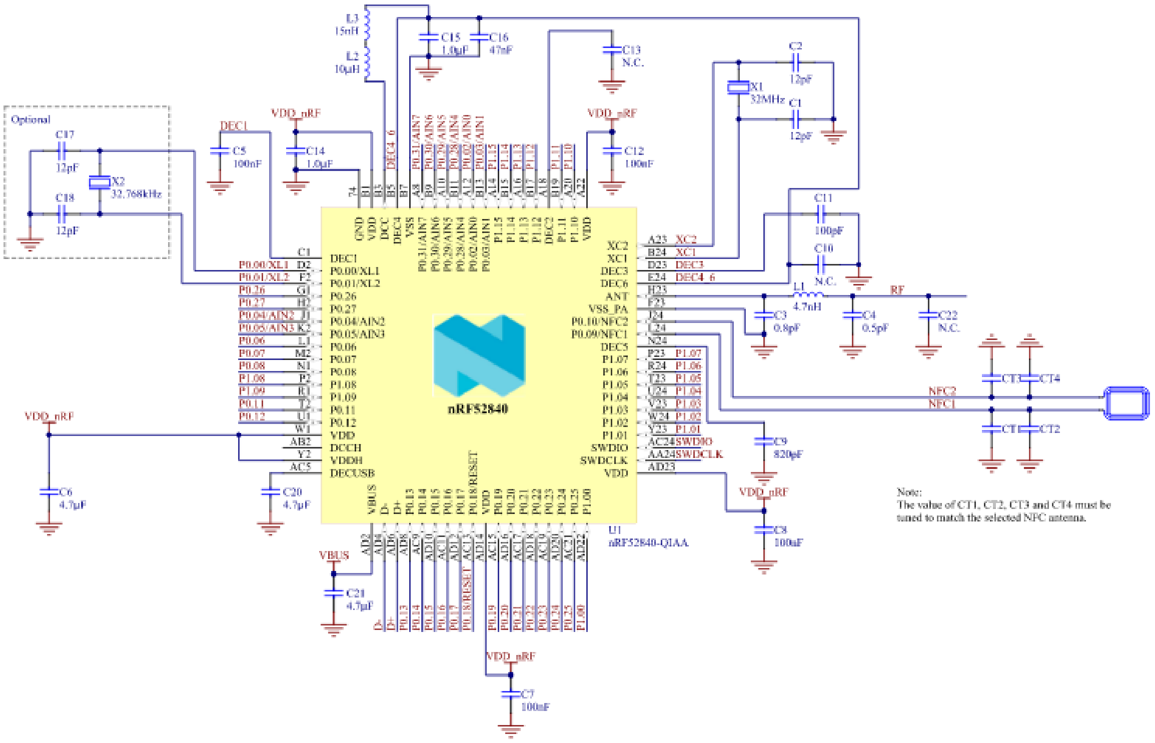
9.5.6 NFC

Near field communication allows small data packets to be sent from a distance of a few inches. This is the technology that allows features such as Apple Pay to work when all you do is put your phone near a scanner to pay for an item. This feature is not needed for this project so there is no need for it to be on in our configuration.



Config no.	Supply configuration		Enabled features				
	VDDH	VDD	EXTSUPPLY	DCDCEN0	DCDCEN1	USB	NFC
Config 1	Battery/Ext. regulator	N/A	Yes	Yes	Yes	Yes	No

Figure 16: Schematic of nRF52840 Configuration 1



Note:
The value of CT1, CT2, CT3 and CT4 must be tuned to match the selected NFC antenna.

Config no.	Supply configuration		Enabled features				
	VDDH	VDD	EXTSUPPLY	DCDCEN0	DCDCEN1	USB	NFC
Config 2	N/A	Battery/Ext. regulator	No	No	Yes	Yes	Yes

Figure 17: Schematic of nRF52840 Configuration 2

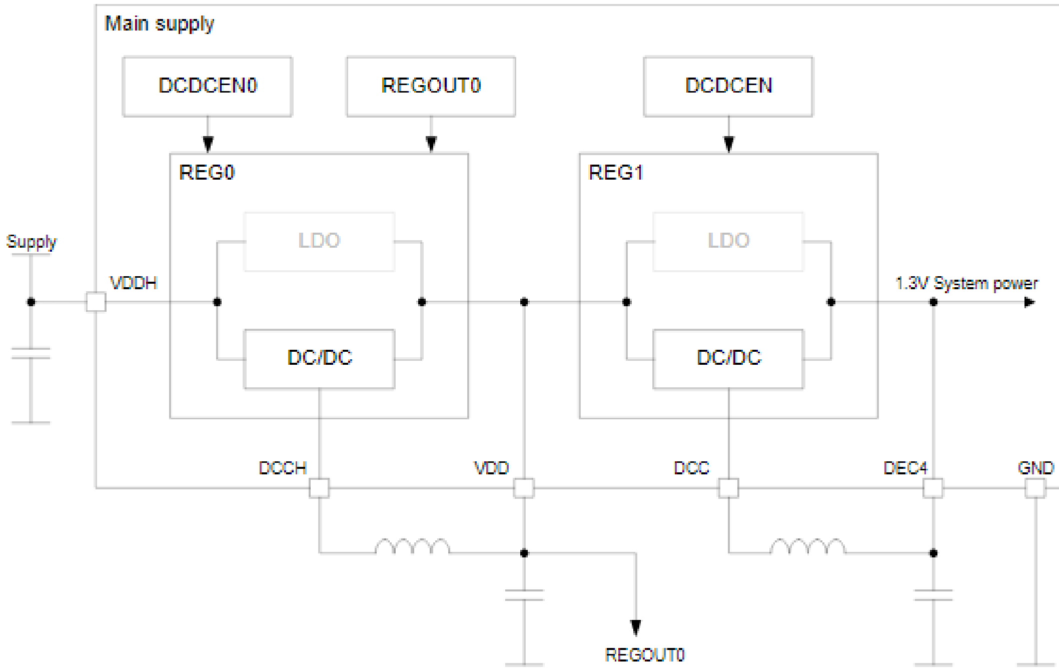


Figure 18: Block Diagram of nRF52840

The main feature of the nRF52840 that is a selling point is the 2.4GHz antenna that is built into the little chip. It follows the IEEE 802.15.4 standard of 1Mbps speed. Below in Table 19 and Figure 19 is a state table of the radio antenna that shows every step that occurs behind sending and receiving packets. A state diagram in Table 18 also shows the order in which these states occur. Understanding the order of the states will become useful when we start to implement the Bluetooth Module.

When we initiate the Bluetooth data transfer from the mobile device to the HUD device, the device will enter the TXRU stage. It will then confirm that the other device is ready to receive information in the RXRU stage. The device then transmits the data to the receiver. The process will repeat for however many times are required to send the data.

After the data transfer is complete, the devices enter the RXDISABLE and TXDISABLE states. Entering these states after transmitting and receiving is no longer necessary is critical for saving energy. It also eliminates the possibility for erroneous information to be transmitted or received.

State	Description
DISABLED	No operations are going on inside the radio and the power consumption is at a minimum
RXRU	The radio is ramping up and preparing for reception
RXIDLE	The radio is ready for reception to start
RX	Reception has been started and the addresses enabled in the RXADDRESSES register are being monitored
TXRU	The radio is ramping up and preparing for transmission
TXIDLE	The radio is ready for transmission to start
TX	The radio is transmitting a packet
RXDISABLE	The radio is disabling the receiver
TXDISABLE	The radio is disabling the transmitter

Table 19: Bluetooth Radio TX/RX States

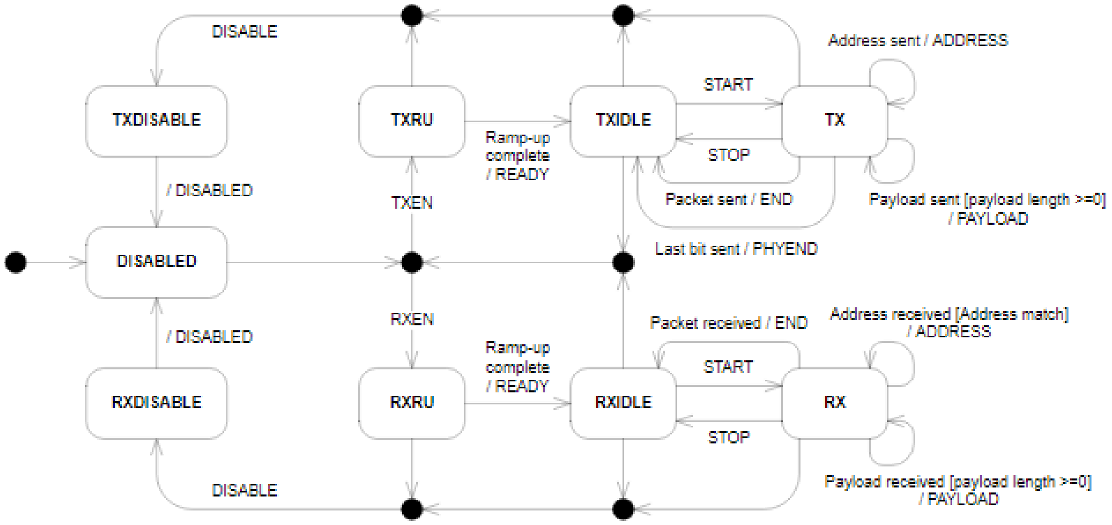


Figure 19: State Diagram of Bluetooth Radio Transmission

9.5.1.1 Bluetooth Module HC-05

Due to COVID-19 causing a huge shift in this project's design phase, the HC-05 was used in place of the nRF52840. This module has breakout pins that allow easy hookup to PCBs and development kits rather than nRF52840 requiring a stencil and oven soldering in order to implement it to a PCB for testing and implementation. The HC-05 uses Bluetooth protocol 2.1+EDR rather than BLE so it is a little more powerful, but because 2.1+EDR is very similar in specifications compared to BLE, the project was able to move forward with the HC-05. An example of a difference is the HC-05 requires 30mW of power compared to the nRF52840's 15mW but due to these differences being very small in terms of overall power, it worked all the same. Another difference is the size of the device, the HC-05 being almost triple the size of the nRF. This also was not a problem as the nRF was very small and was supposed to be soldered directly to the PCB while the HC-05 can be fit wherever as long as the breakout pins were accessible. Due to other components of the Huddy Buddy being significantly larger than either of these modules, there was plenty of room for the HC-05 to be implemented.

10.0 Crash Detection:

When we initially discussed senior design project ideas, making an innovation-packed project was our priority. Given that the project we chose was focused around driver safety and awareness, adding functionality that increased these concepts was important to us. The Huddy Buddy allowed us to reduce distractions and prevent accidents caused by distracted driving. However, accidents can happen at any moment due to factors that we cannot plan for nor have control of. If an accident was to occur, response time is paramount to ensuring those involved are given the care they need. In addition to the reduced distraction provided by our HUD device, we added a crash detection system to our design. Creating a crash detection feature allowed us to gain experience with some systems we were not initially familiar with at the beginning of Senior Design. The following subsections will discuss each feature required to implement safety features if a crash does occur.

10.1 Accelerometer

Vehicular crashes are often paired with impacts that produce strong and abrupt forces. These forces can be detected using an accelerometer to analyze the gravity relative to the position of the sensor. When a change in force occurs, the sensors on the accelerometer report the differences as data points and the microprocessor can determine if the impact is enough to constitute a crash.

10.1.1 Adafruit ADXL335

The Adafruit ADXL335 Accelerometer is a triple-axis accelerometer that comes with pinouts for x, y, and z axis measurements. It has an onboard voltage regulator that steps down 5V to the 3.3V required to power the accelerometer. The ADXL335's datasheet states that the chip can withstand forces up to 10,000g's, which is well within the impact forces we would expect from the typical car crash. Once the accelerometer detects an impact, the HUD device sends a command to an SMS module located on the chip. The SMS chip sends text message containing the device's current location and details about the crash to an emergency contact programmed by the user.

Adafruit sells the ADXL335 on a breakout board with the required components required to implement it. We will be utilizing the schematics provided by Analog Devices in Figure 20 to implement the ADXL335 onto our custom PCB.

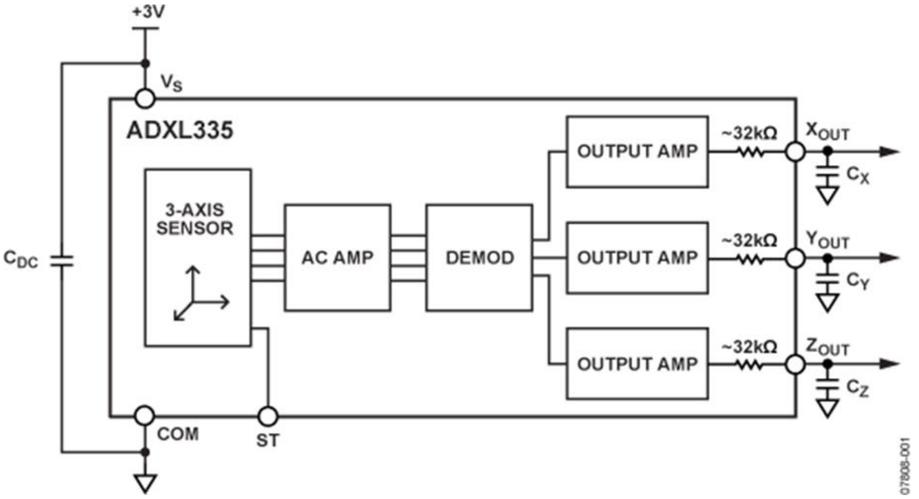


Figure 20: Schematic for ADXL335

10.1.2 Adafruit ADXL337

The Adafruit ADXL337 is a High-G triple-axis accelerometer. It operates in a similar fashion to the ADXL335 but can detect a greater range of G forces. According to the datasheet, the ADXL337 can detect up to positive or negative 200 g's of force.

When this is compared to the ADXL335's positive and negative 3 g's of force, we believe that the finer granularity of the ADXL337 will allow us to more accurately identify an impact strong enough to constitute a crash.

10.2 2G GSM/GPS Module

After the accelerometer detects an impact strong enough to constitute a crash, communicating that a crash has occurred to an emergency contact with minimal delay is paramount to providing with the driver the greatest amount of safety features. Initially, we considered implementing both cellular networking and GPS components to the final design. We thought that these components would be separate and require their own space on the final PCB. Upon researching cellular networking components, we discovered the SIMCom SIM808 module, which implements cellular communication and gps communication on the same chip. The SIM808 connects to a wireless cellular network over a GSM connection with a SIM card. The SIM808 takes advantage of a 2G network, providing the ability to send data, SMS, MMS, voice, and other data from the cellular connection.

To use the wireless cellular network, the team acquired a SIM card with an active plan. In the United States, there are regulations that control many aspects of cellular network operations. Adafruit has partnered with Ting, a mobile virtual network operator that leases network space from established telecom companies. Adafruit provided a package that includes a free Ting SIM card with activated 2G service when purchasing the SIM808 module. This was ideal for our project, as it minimized the total cost of implementation. However, there was the addition of having a monthly subscription model to implement a cellular connection service. Ting is a 'pay-as-you-go' provider. Given the rarity of accidents and the amount of text messages we needed to be sending over the network, the cost to implement the Ting subscription service is minimal at about 10 to 15 dollars per month.
Finding the exact location of the cellular module can be done via triangulation of the cellular connection. The cellular module locates nearby cellular towers and find the strongest connection between them. This data can be analyzed and measured against locations of cellular towers to determine the exact location of the module. However, this process is arduous and requires access to locations of network cellular tower locations for which the module is connected to. Fortunately, the SIM808 implemented built-in GPS capabilities onto the chip. This allowed the team to access the location of the device by connecting to geosynchronous satellites that provide location data with great precision. We used this data to analyze the driver's current location during an accident to send to an emergency contact.

We were able to locate the schematic for the SIM808 to implement its design into our custom PCB. All the components required to implement the SIM808 were readily available. This was the ideal choice for us to implement active GPS and Cellular capabilities.

10.3 Effects of Impact on Power Delivery and Electronics

Large impacts during a car crash are often strong enough to reduce the vehicle's ability to continue under its own power. Once the engine stops running, the vehicle is no longer able to use the alternator to produce electricity. For our device to work, we relied on the vehicle's alternator to provide enough power for the device and accompanying components.

Since it is critical to have the driver's location via SMS text at the time of impact, we needed to have the device stay active long enough for that signal to be sent. To ensure reliability of our electronics during and after the event of a crash, we researched how to add a secondary power source into our design. The amount of time needed for this secondary power source to be active would be minimal, just short enough duration to complete the SMS message containing the driver's current position to his or her emergency contact. Small lithium polymer batteries are cheap and would provide enough power to allow the microcontroller and accompanying electronics for a few seconds after the main car power source becomes unavailable.

We were successfully able to create a backup battery system by using two 18650 lithium-ion batteries in series to produce 7.3V, enough to power the microcontroller and the SIM808 module long enough to complete the emergency text message. When the OBD is supplying power to the components, a blocking diode prevents current from being drawn by the batteries as they are not needed. As soon as the OBD power is cut, the blocking diode is turned off and allows current to flow from the batteries to the components.

As previously mentioned in this section, crashes are typically accompanied by strong forces that can sustain serious damage to the vehicle. The Huddy Buddy was designed in such a way that it will survive the impact and operate normally. We ensured that all components were secure and would not be disconnected in a crash.

11.0 Microcontroller:

The center of any electronics project is typically the microcontroller. It serves as the brains of the operation, manipulating data and running certain programs to achieve the end goal. Choosing the correct microcontroller for our HUD device was extremely important to creating an effective and efficient design.

The HUD device needed to receive data from the GPS module multiple times per second and calculate the distance between the received data and the waypoints along the route. This required processing power and storage to manipulate and store the calculations. Once these calculations were complete, the display needed to be updated to reflect the most current information to the driver.

As we began to research microcontrollers needed to handle all the information from the GPS module, APIs, SMS module, and drive the display unit, it became apparent that a powerful microcontroller would be needed to provide the driver with the best user experience. The microcontroller needed to have UART communication to interface with all of our modules. There was a common thread in projects that implement similar designs, most of them use Arduino or TI MSP based microcontrollers. Upon researching Arduino and TI microprocessors, there were two that stood out. A comparison of these microcontrollers is listed in Table 20 below.

Microcontroller	**MSP430F447**	**Arduino ATMega 2560**
Price ($)	8.50	38.50
Processor Speed	16 MHz	16 MHz
Data Bus Bandwidth	16-bit	8-bit
RAM	1 KB	8 KB
Flash Memory	32KB + 256B	256 KB
UART Channels	3	4
I/O Pins	48	54
Operating Temperature	-45 C to 85 C	-40 C to 85 C
Operating Voltage	1.8 V to 3.6 V	5 V
Special Features	IntegratedLCD Controller	Open Source

Table 20: MSP430 and Arduino Comparison

11.1 Texas Instruments MSP430F447

The Texas Instruments MSP430F447 was unique as it has its own integrated LCD controller. This would prove useful when driving the Heads-Up Display unit. A segment of the microprocessor would solely be devoted to this display, eliminating the need for the whole processor to calculate the updates to the display a few times per second. This would free up used memory, power, and allow the microprocessor to do calculations on other components and tasks.

11.2 Atmel ATMega 2560

The ATMega 2560 was the perfect microcontroller to implement into our project. The hardware for the ATMega 2560 is open source which allowed us to create our own implementation of the microprocessor without infringing on copyrighted or patented designs. Additionally, this microprocessor is well known for its flexibility and portability for projects containing embedded design like our project narrative, typically utilizing something like an Arduino based system. Arduino provided a development kit that includes the ATMega 2560 fully implemented. The development kit included custom PCB files and other documentation for creating custom components using the technology they provide. Our custom PCB was designed to contain the Mega 2560 and all its components required to operate correctly and efficiently. Figures 21 and 22 below are the schematics needed to implement the ATMega 2560.

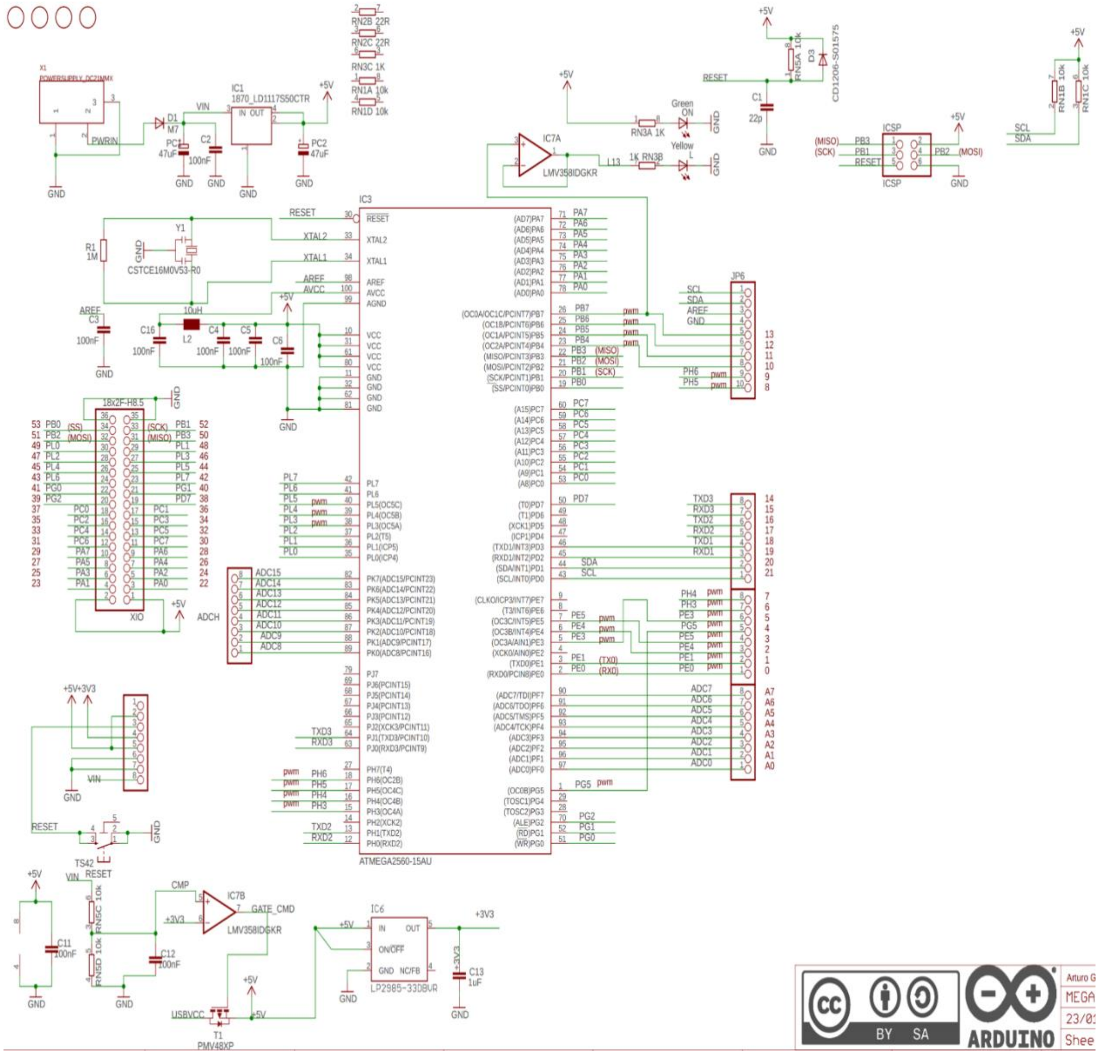


Figure 21: ATMega 2560 Schematic

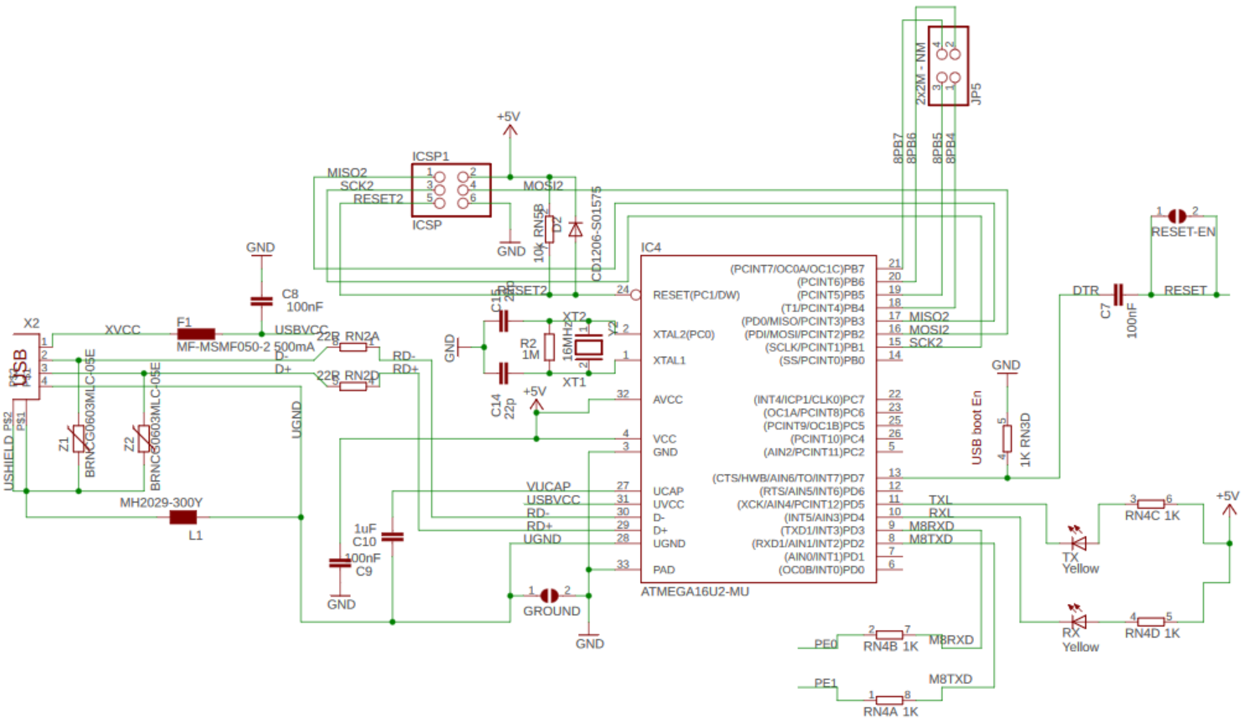


Figure 22: ATMega 2560 Schematic Continued

11.2.1 Custom Bootloader

One of the biproducts of using a standalone ATMega 2560 microcontroller instead of the full development board is that we needed to implement our own Bootloader. A benefit of a development board is that the bootloader comes built in. Building our own bootloader required us to purchase an Arduino UNO to flash the ATMega 2560. Once the microcontroller is seated in the custom PCB, we connected it to the Arduino UNO using the pin mapping located in Table 21. Then, using the Arduino IDE, we pushed code to the microcontroller to enable its functionality.

Arduino UNO pins	ATMega 2560 pins
10	30
11	21
12	22
13	20
5V	VCC
GND	GND

Table 21: Bootloader pin mapping between UNO and ATMega 2560

11.2.2 Concurrency

Microcontrollers that are used to handle communication between devices use concurrency to ensure that the processor can read all of the needed data without waiting for other tasks to complete. This is often accomplished by creating multiple threads for the application to run on, allowing the processor to split the processes into segregated lanes for each task to be completed. For a project such as the Huddy Buddy, it is critical that the microcontroller can handle communication between the OBDII Adapter, OLED, Accelerometer, and SIM808 module without having to wait for one communication to complete. One of the major drawbacks to using a microcontroller like the ATMega 2560 is that it only has one compute core. This means that typical concurrency using concepts of multithreading or multicore processing is not possible.

One successful solution that we were able to implement was currency through scheduled events. Each component would be assigned a specific time for the microcontroller to read or write data. For the OBDII and OLED display data, we scheduled the microcontroller to update these components every 50 milliseconds. We chose this value because we felt that it was important to give the user an accurate reading of speed that is updated rapidly. Many of-the-shelf HUD units that were researched had a very poor refresh rate for speed, reducing the effectiveness of the data and giving the user an unenjoyable experience. The accelerometer would be checked every 200 milliseconds to check for large forces that would constitute a crash and trigger the emergency response system.

Concurrency though scheduling is not the most accurate method, as it requires the clock for the microcontroller to fire at that exact time for the action to be ran. However, with the 16Mhz clock rate of the ATMega 2560, it is rare for a timed event to be skipped due to a mismatch of clock timing.

12.0 Testing Plan

Any successful project needs to go through various testing phases to ensure each component is working properly and the intended outcome is achieved. As mentioned in the above sections, there were numerous components that contribute to the operation of the Huddy Buddy. Each of these components needed to be thoroughly inspected, tested, and verified to guarantee that the HUD device could be completed on time and avoid last minute changes. The following sections will describe the test plan for each one of the components and how we were able to test them.

12.1 Microcontroller Testing

The ATMega 2560 Microcontroller was the heart of the operation of the HUD Device. Without proper functionality of the microcontroller, we would not have been able to operate the display that provides the driver with information required to reduce distractions while driving. We would not have been able to transfer data between the mobile application and the HUD device over Bluetooth protocols. Data from the accelerometer would not have been processed if a crash were to occur, reducing the ability for our project to help those impacted by the crash.

For these reasons, we needed to be certain without doubt that our microcontroller was working properly. One of the benefits of choosing a microcontroller as popular as the ATMega 2560 is that there was a myriad of online resources that discussed how to properly develop and maintain the platform. Documentation about implementing the microcontroller was readily available in detail from the manufacturer. This information became paramount when implementing and troubleshooting the microcontroller and its functionality.

To test the implementation of the ATMega 2560, the team initially purchased an Arduino Mega 2560 Development Kit that includes a fully integrated PCB. Development kits provided resources such as easy-to-use IDE’s and verified components to ensure proper functionality. This allowed us to verify the use of the other components using methods discussed later in this section. Once we were able to verify that the components we selected worked with the ATMega 2560, we then looked to implement the microcontroller into our own custom PCB.

Upon completion of the verification phase, we entered the testing phase for our own microcontroller implementation. The ATMega 2560 is an open-source hardware platform, which means we had access to the documents the manufacturer uses to create the final product. Adding the ATMega 2560 to our custom PCB required us to add in all the required features from the development board ourselves. This included power delivery, firmware flashing, pushing code to the microcontroller, and other important tasks to operate the microcontroller.

12.2 Mobile Application Testing

The Android app was written in Android Studio using Java as its programming language. In order to examine how initial interfacing looked such as buttons and banner information, Android Studio's built in Android emulator was used for quick examinations. The emulator was a Google Pixel 2 and was used for testing of the basic tools of the app as well as the Mapbox portion. For features that required a physical device for testing, a Samsung Galaxy S8 was used.

Android Studio had a ton of debugging tools and quality of life features to help keep all files in sync while providing explicit information on what is going on as the app runs. Android Studio provided instant updating of Gradle files upon the inclusion of a new API import, as well as manifest updates for when code written required the new permission to be accepted from the phone.

Logcat is a built-in logger for Android Studio that provided all system level logging from debugging mode filter console mode to high level fatal errors. With its search and filter features, debugging code and receiving logs straight from the phone/emulator was very efficient. Logs were set with a tag so that the search feature found the line in the log quickly, and logs can pull variable information in real time in order to test and check that the correct information was being pulled rather than null for example.

A second application was created purely for testing purposes. This second app which was dubbed BTChat was used in order to test Bluetooth functionality from the phone in coordination with the computer. For this portion, a physical Android device was used as Bluetooth functionality can not be tested from an emulated environment. BTChat allowed the app to choose a paired Bluetooth device and send a string to it. Using Android libraries such as IOStream, testing was done so that a connection could be made with the Bluetooth module and send whatever message I want inputted into the BTChat application. Once data packets were successfully sent from the phone to the Bluetooth module, the code within this application was merged with the main application.

12.3 Accelerometer Testing

The accelerometer's accuracy was extremely important considering we are using it as a safety mechanism. Any erroneous measurements or failures would have prevented the project from enabling greater safety for the driver. Once the accelerometer components were received, we tested its accuracy to ensure it is measuring the expected outputs. According to omnicalculator.com, a person that weighs 165 pounds travelling at 45 miles per hour will experience roughly 10g's of force when given a stopping time of 0.2 seconds. Typically, a car and its driver or passengers do not exceed more than about 1.5g's of force at any given time.

We used this information to determine that the given threshold to initiate the emergency response sequence would be 1.5g's. To test how much force is being observed by the ADXL335 Accelerometer, we set up a sketch that was provided by Adafruit to enable the functionality of the microcontroller. The accelerometer plugs into the 5v pin, ground pin, analog reference pin, and 3 analog I/O pins on the microcontroller to enable its functionality. The sketch reads the analog data every 200 milliseconds and performs calculations based on the raw data to extract what the current g force is.

To ensure that the accelerometer is reading accurate information, it had to be calibrated to set the bounds on the force for each axis given its orientation. During our initial calibration where the ADXL335 was flat on a breadboard, the following raw results in Table 22 were achieved:

Axis	Minimum Value (Raw Data)	Maximum Value (Raw Data)
X-Axis	409	613
Y-Axis	403	606
Z-Axis	425	629

Table 22: Raw Max/Min sensor data from ADXL335 Accelerometer

Since our accelerometer would be in the same orientation, we could use these values to determine the values of gravity at any given time. If this value goes over 1.5g's the system will enter the emergency response subroutine.

12.4 SIM808 2G GSM/GPS Module Testing

After we received our SIM808 Module, we began to look into the documentation for how the AT Command Set tells the module what to do. Upon further research, we discovered a program created by M2MSupport.net called AT Command Tester.

This program is a convenient tool to streamline the sequence of AT commands needed to implement specific functions by providing the user with a GUI based application. Within the application, the user specifies which function they would like to use, and the program will create automated AT commands to complete the task. There is a console that displays every command sent and received by the SIM808 module. There are specific tabs within AT Command Tester that pertain to the SMS and GPS capabilities.

To connect the SIM808 to the Arduino Mega, we attached the SIM808’s Tx and Rx pins to two of the Mega’s I/O pins. We then built a sketch that uses the AT Commands provided from AT Command Tester. We used these AT Commands to successfully check the SIM808’s current network status, send a text message, and grab current GPS coordinates. This proved that the SIM808 module was going to provide us with the exact information we needed to implement the emergency response system discussed in Section 10.2.

Further research into adding AT Commands into our Arduino Mega 2560’s program sketch led us to the discovery of the Adafruit FONA library. This library provides functions and files that enable chips such as the SIM808 to do tasks like make a call, send a text, get current GPS coordinates, check for GPS synchronization, etc. while needing minimal code to be implemented. We were able to use the library to simplify our code down to only 5 lines to retrieve the current GPS coordinates, convert it to a string, and send an SMS text.

12.5 Grove OLED Testing

Testing the Grove OLED display with our development environment initially seemed to be a daunting task. Lessons in Embedded Systems taught us that each frame of the OLED had to be built manually by sending data from the microcontroller’s I2C bus to provide the information that is requested. Fortunately, an open source Arduino library called U8glib created by GitHub user olikraus provides needed functions and files to realize the display unit.

There are a few modes that the U8glib provides, consisting of the u8x8 library for text displays and the u8g2 library for graphical displays. Throughout the design process of our Heads-Up display, we understood that there was a possible need for the display to provide information that is mirrored, allowing the driver of the vehicle to read the text in the correct orientation. Upon reading the documentation provided in the U8glib libraries, it was apparent that the easiest way to implement this was to use the u8g2 graphical library. Once the library is implemented, mirroring the display is possible by changing one argument in the display initialization code attached to the sketch.

To test the display, we began by connecting the OLED’s 5v, ground, SCL, and SDA pins to the Arduino Mega 2560. We then loaded a sketch onto the Arduino module, consisting of the u8g2 library using the full-buffer method. In the full buffer method, each frame is built by storing the information in the frame in a buffer. Once the frame is completely built, the microcontroller sends this frame to the display. This can be done in rapid succession to implement a display that is constantly updating, just as other televisions and monitors create moving images.

Once the full frame buffer sketch was completed, we were successfully able to send data over the microcontroller’s I2C bus to the display. The information we wished to provide to the driver included the vehicle’s current speed, fuel economy, and navigation information. This was built by sending each of those variables to be printed to the display. Each frame needed to be updated rapidly to ensure a good user experience and accurately displayed information. We found that updating the display every 50 milliseconds was enough to provide the user with a fluid transition as speed and fuel economy changed. May other off the shelf units that were researched had a slow refresh rate, which takes away from the user experience. The image in Figure 23 is an example of how we were able to notate the displayed information in a way that was readable and informative to the driver.

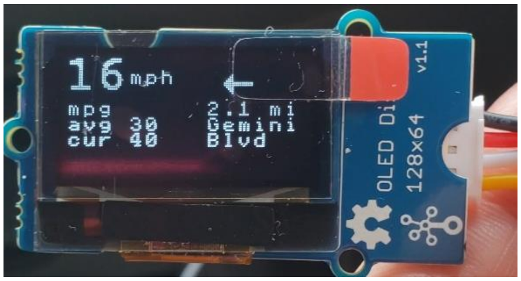


Figure 23: Layout of information on OLED DIsplay

The top left of the display includes the driver’s current speed in miles per hour. Since this can change between a 1,2, or 3-digit number, we biased the data to the right to ensure each digit remained in the same position no matter how many digits were displayed. The same is true for the fuel economy, located at the bottom left. The right side of the display includes information about the driver’s next turn along their desired path.

One downside to the Grove OLED that we discovered during testing is that it is not sufficiently bright enough to be used in direct sunlight when reflecting off the windshield of a vehicle. The white text on the screen does not appear as it is competing with the brightness of the sun. However, if the OLED is pointed directly at the driver in direct sunlight, it is indeed readable. In conditions where the display is not in direct sunlight, viewing information on the OLED reflecting off of the windshield is possible.

12.6 Freematics OBD-II UART Adapter Testing

As referenced in the research section of this paper, every vehicle after 1996 was required to have an OBD-II port to enable more in-depth troubleshooting of vehicular systems. Since all group members have vehicles manufactured after 1996, each one of us was capable of testing the OBD-II Adapter.

We began by downloading the Freematics ArduinoOBD Library from GitHub. This library provides functions and files needed to send and receive data over the Arduino Mega's serial UART pins. The documentation provided with the library along with forum posts on the Freematics website explained how to use the OBD-II UART adapter to extract the data. Noted in Section 12.5, we wanted to grab the current vehicle's speed and current fuel economy from the vehicle. This is done by sending the PID code to the adapter, which it will return the value at that parameter. Once the value was retrieved, we were able to print it onto the OLED screen.

Further testing of the values returned by the OBD-II Adapter and measuring them against values that were given by the vehicles instruments showed that the values for speed and fuel economy were exactly as expected and accurate to what the vehicle was outputting.

12.7 Collimating Lens Testing

The means by which we will produce an image that is focused at infinity is by using a lens array. Any object placed at the focal point of such a lens will be collimated, which means that all rays of light reflected (or emitted) by the object into the lens will be parallel out to an infinite distance. However, this is assuming a perfect lens and a small object. Our lens will most likely not be perfect, and our object is an OLED with a diagonal size of roughly 1 inch. Therefore, our image will not be perfectly focused at infinity. However, we were able to test our system with a positive and a negative lens, as shown in Figure 24.

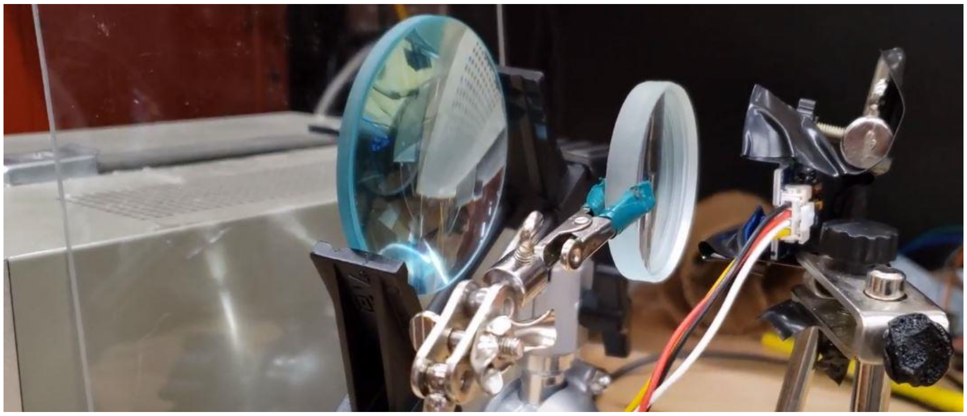


Figure 24: A positive and negative lens placed in front of the OLED display

The issue of having a larger object, in conjunction with possible lens imperfections, may lead to some level of distortion of the displayed image. Such distortion, however, is not easy to overcome. For one, distortion can be lowered by using a longer focal length. However, given that our device has size constraints, building a sufficiently large system to accommodate a longer focal length is not feasible. In our initial testing with a single positive lens, there was considerable distortion of the image if the image was viewed off-axis in any way. However, our solution to this problem was to include a smaller negative lens into the system. This negative lens was equal but opposite in power to the positive lens. With the negative lens a certain distance away from the OLED display, a virtual image of the OLED display would be seen from the other side of the lens. This virtual image would be smaller than the OLED and simple calculations could determine whether this virtual image would be at the focal point of the positive lens.

In our testing, we were able to orient our lenses and OLED display a certain distance from each other. Our overall goal was to see what distances produced an image with the least distortion while still producing a collimated image. Regardless of lens separation, we were able to calculate the proper distance from OLED to the lenses to maintain a collimated image. Our testing found that placing the negative lens within 2 inches of the positive lens produced the image with the least distortion. However, with the mirror we chose to include in our projector, this distance was not entirely feasible. With a separation of 2.25 inches to accommodate the mirror we found that we still had an acceptable image quality. Thus, 2.25 inches was the goal for the lens spacing.

12.7.1 Reflectivity

One factor that affects how we use our LED's, and even our display system, is reflectivity. Reflectivity is the ratio of light that, instead of being transmitted through a medium, is reflected away from the medium. In our case, the light is passing through air into glass. In order to make the most out of our display, it would be wise for us to attempt to maximize our reflectivity. Doing so could potentially lower the amount of light needed to make our display visible to the driver. This would have positive effects on our power consumption and heat output, as well as visibility.

One factor that affects our reflectivity is the angle at which the light source is placed, according to Fresnel's equations. Fresnel's equations give the reflectance and transmittance of optical power based off the incident angle and the indices of refraction. Normally, we would be limited to using glass (whether the windshield or a vertical glass screen) and air as our media the light must pass through. Our reflectivity would be worst at a 90-degree angle to the glass and would be highest at what is called the Brewster's Angle, as shown in Figure 25. This angle is the angle at which all of the light would be reflected. This is most apparent when looking towards the surface while underwater; at some point, the image of the sky will be hidden and only a clear reflection of the seafloor would be visible. Shining the display at Brewster's Angle would have its own set of challenges. For most types of glass, the index of refraction is roughly 1.5 (it is 1 for air). Brewster's Angle with these parameters would be roughly 50 to 60 degrees. While it would be possible to orient the device to project light at that angle, we would run into space concerns. The device itself would have to be oriented in such a way that it would achieve Brewster's Angle, which could potentially lead to the device blocking the driver's view of the road. Because the device is intended to solve that problem in particular, we would either have to modify the device's enclosure to better fit our needs, or we may need to use a vertical sheet of glass that would allow us to more easily orient our device without obstructing the driver's view.

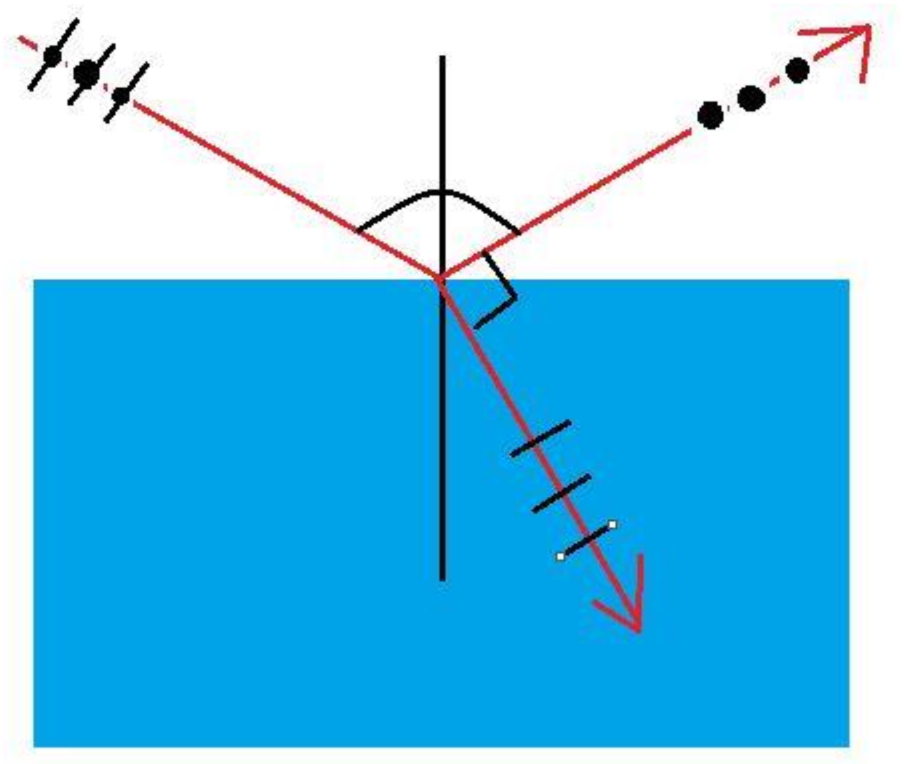


Figure 25: Brewster’s Angle, which shows how unpolarized light is transmitted and reflected.

One other effect of using Brewster’s Angle to reflect light is that, in the process, the light will be polarized. The light emitted that doesn’t match this polarization will simply be transmitted through the windshield or other optical device, while the light with matching polarization will be almost 100% transmitted. This becomes a problem on two fronts. The first problem would be that the light emitted from the display itself would be randomly polarized. This is because the OLED display itself uses small organic LED’s, which have no polarization, to produce light. Because much of the polarization of the transmitted light is not like the polarization brought on by the Brewster’s Angle, then up to 50 percent of the light will not be reflected off the glass. Another issue will be that the polarization brought on by using Brewster’s Angle will be like the polarization of light reflected off the roadway, especially when the road is covered in water or snow. This is the same polarization that is meant to be blocked by polarized sunglasses. Even if we can overcome the reflection polarization issue, the displayed image would be very difficult to see when the driver is wearing polarized eyewear. The simple answer to this problem is to avoid wearing polarized sunglasses or other eyewear while the heads-up display is in use.

One possible solution to the reflection problem is a simple method used by commercial heads up displays: a reflective film. Such films are readily available and are even sold with heads up displays in mind. The purpose of such a film is to increase the reflected power of the displayed image. This solution is not without its own faults, however. In use, the film is reflective on both sides. This leads to less light being transmitted through the windshield, and it would be directly where the image would appear.

With our current use of a randomly polarized OLED, we find that the simplest solution for ensuring maximum reflection is to use a reflective film on the windshield. This allows for more light to reach the driver and ensures that less light from the image will be filtered out by polarized sunglasses. Due to the weak brightness of OLED displays, the image will still be very faint, if not invisible, in daytime. However, other technologies like micro LED or liquid-crystal-on-silicon (LCOS) would allow for sunlight-visible displays.

Our testing found that our current OLED display is not bright enough to produce a sunlight-visible image, but we were able to show that our projector does produce an adequately visible image indoors. Moreover, we were able to ensure that our projected image was focused at infinity by placing an object on the opposite side of the room from our projector and combiner. Once the observer was able to focus on the distant object, they would find that the image was also in focus. This was verified by all members of the group.

12.8 BOB-12035 Testing

This test is to prove that the main power supply of our project is working properly and to our standards. This test should be conducted with a wall charger that has the same specifications of the car charger we will be using, which is a 5-volt, 2.1-amp fast charger. In order to make sure our connector works the wall mount will be inserted with a USB type B cable connected to the connector. A multimeter should be used to test the output at the pins. The only way this part passes inspection is if it outputs 5 volts and 2.1 amps anymore or any less and the test is a fail and a new part will be ordered on the assumption it was damaged on arrival.

12.9 LM2596 (Step down voltage regulator) Testing

The point of this test it to configure the voltage regulator needed to step down the 5-volt input for the smaller parts of the system. This works with a combination of diodes, capacitors, and inductors, with the main part being the LM2596. There will be strict tolerances for each individual part. The goal is to output 3.3 volts at around 2 amps. In order to do this, the datasheet gives specific parts that should be used following the schematic from Figure 27 and Tables 23 and 24 below for part selection.

NOTE: Blank table fields left in this section will be filled as testing each part is completed

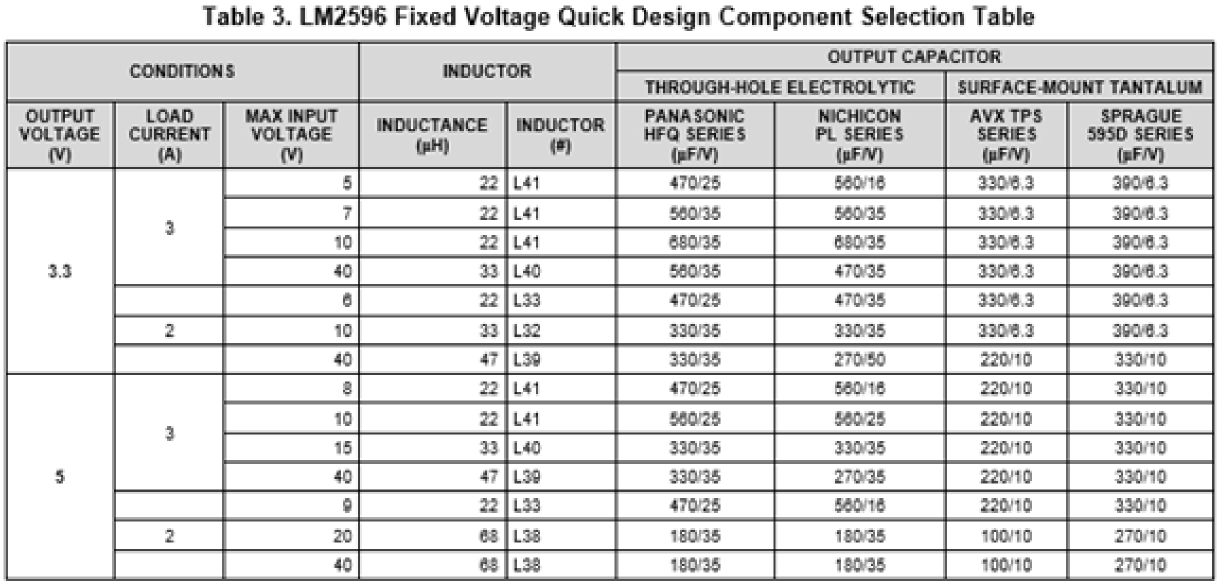
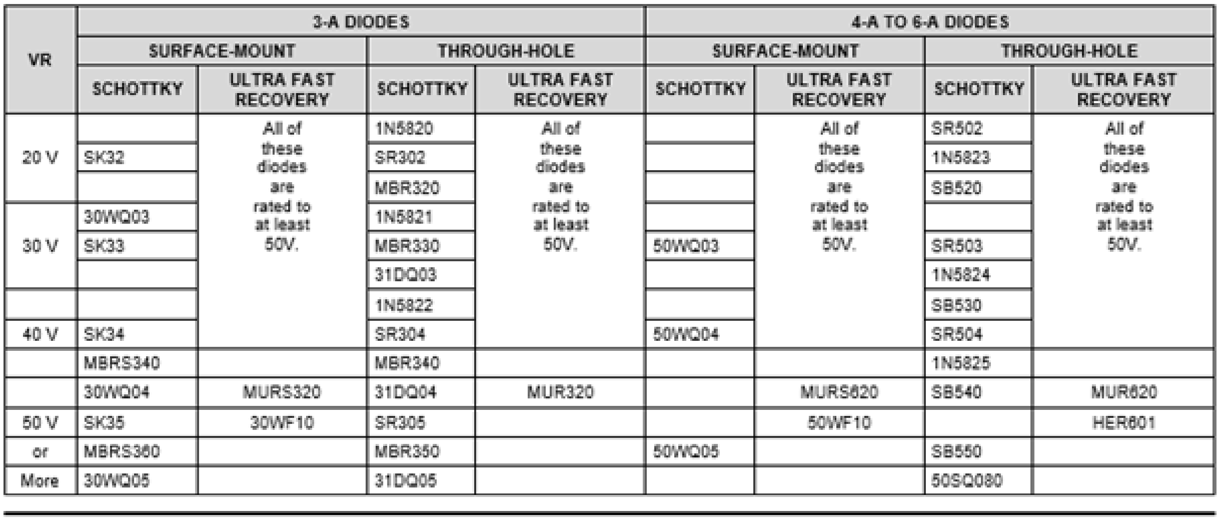


Table 3. LM2596 Fixed Voltage Quick Design Component Selection Table

CONDITIONS			INDUCTOR		OUTPUT CAPACITOR			
					THROUGH-HOLE ELECTROLYTIC		SURFACE-MOUNT TANTALUM	
OUTPUT VOLTAGE (V)	LOAD CURRENT (A)	MAX INPUT VOLTAGE (V)	INDUCTANCE (µH)	INDUCTOR (#)	PANASONIC HFQ SERIES (µF/V)	NICHICON PL SERIES (µF/V)	AVX TPS SERIES (µF/V)	SPRAGUE 595D SERIES (µF/V)
3.3	3	5	22	L41	470/25	560/16	330/6.3	390/6.3
		7	22	L41	560/35	560/35	330/6.3	390/6.3
		10	22	L41	680/35	680/35	330/6.3	390/6.3
		40	33	L40	560/35	470/35	330/6.3	390/6.3
		6	22	L33	470/25	470/35	330/6.3	390/6.3
	2	10	33	L32	330/35	330/35	330/6.3	390/6.3
		40	47	L39	330/35	270/50	220/10	330/10
5	3	8	22	L41	470/25	560/16	220/10	330/10
		10	22	L41	560/25	560/25	220/10	330/10
		15	33	L40	330/35	330/35	220/10	330/10
		40	47	L39	330/35	270/35	220/10	330/10
		9	22	L33	470/25	560/16	220/10	330/10
	2	20	68	L38	180/35	180/35	100/10	270/10
		40	68	L38	180/35	180/35	100/10	270/10

Table 23: Choosing an Inductor and Capacitor for LM2596

Courtesy of Texas Instruments Inc.



VR	3-A DIODES				4-A TO 6-A DIODES			
	SURFACE-MOUNT		THROUGH-HOLE		SURFACE-MOUNT		THROUGH-HOLE	
	SCHOTTKY	ULTRA FAST RECOVERY	SCHOTTKY	ULTRA FAST RECOVERY	SCHOTTKY	ULTRA FAST RECOVERY	SCHOTTKY	ULTRA FAST RECOVERY
20 V		All of these diodes are rated to at least 50V.	1N5820	All of these diodes are rated to at least 50V.		All of these diodes are rated to at least 50V.	SR502	All of these diodes are rated to at least 50V.
	SK32		SR302				1N5823	
			MBR320				SB520	
30 V	30WQ03		1N5821					
	SK33		MBR330		50WQ03		SR503	
			31DQ03				1N5824	
			1N5822				SB530	
40 V	SK34		SR304		50WQ04		SR504	
	MBRS340		MBR340				1N5825	
	30WQ04	MURS320	31DQ04	MUR320		MURS620	SB540	MUR620
50 V	SK35	30WF10	SR305			50WF10		HER601
or	MBRS360		MBR350		50WQ05		SB550	
More	30WQ05		31DQ05				50SQ080	

Table 24: Choosing a Diode for LM2596

Courtesy of Texas Instruments Inc.

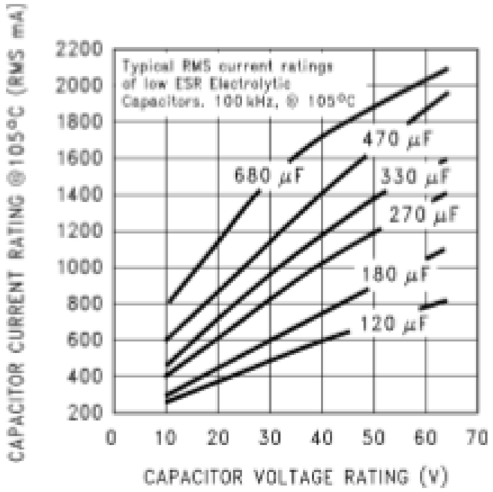


Figure 26: Choosing an Input Capacitor for LM2596

Courtesy of Texas Instruments Inc.

After appropriate part selection is made then each part must pass its individual tolerance test to make sure the part will perform as desired. In order to achieve the 5-volt, 2.1-amp configuration the following parts will be used; a 22 uH inductor, with an output capacitor of 470 uF / 25 V. The diode must be selected so that the current rating is 1.3 times higher than the output current which is 1.3 times 2.1 which is 2.73. Knowing that, the 1N5820 Schottky diode will be used because of its maximum repetitive reverse voltage of 40 volts and average rectified forward current of 3 amps. Finally, the input capacitor must be selected such that its voltage rating is 1.25 times greater than the input voltage with a ripple current rating of about half the load current; this means a 680 uF capacitor with 100 voltage rating and ripple current rated at 1280 mA will be used.

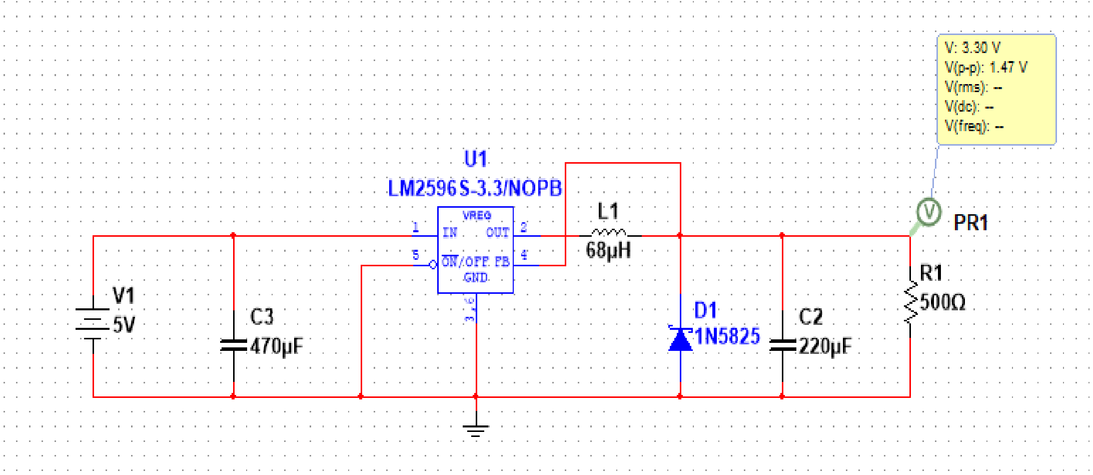


Figure 27: Fixed Output of 3.3 Volts with LM2596

From the simulation above in figure 34, the step-down voltage regulator has no problem taking 5 volts to 3.3 volts. The only concern would be the current drawn from the load, which is simulated by the 500-ohm resistor.

Because the output is across the capacitor the use of an oscilloscope is able to show the output voltage. Then the current will be shown using a multimeter. Table 25 below will be used to determine if the parts are within tolerance and if the circuit passes the testing phase.

Part	Required Value	Actual Value	Pass/Fail
Input Capacitor	680 uF	676 uF	Pass
Output Capacitor	470 uF	469 uF	Pass
Inductor	22 mH	21.96 mH	Pass
Circuit Testing			
Output Voltage	3.3 Volts	3.29 Volts	Pass
Output Current	2.1 Amps	2.07 Amps	Pass

Table 25: Part Testing for LM2596

12.10 LM317T (Step down voltage regulator) Testing

The LM317T is another step-down voltage regulator that was tested to see if it is a part to be used or not. The LM317T is a linear regulator and is simple in design. Below is a schematic of how it would look.

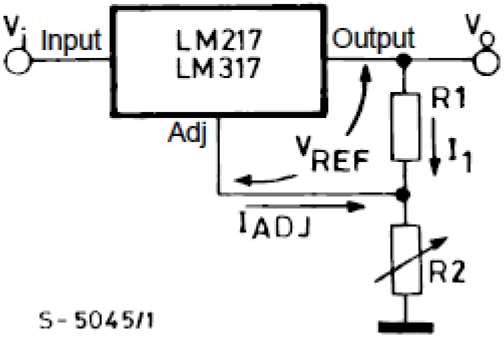


Figure 28: Schematic for LM317T
Courtesy of Texas Instruments Inc.

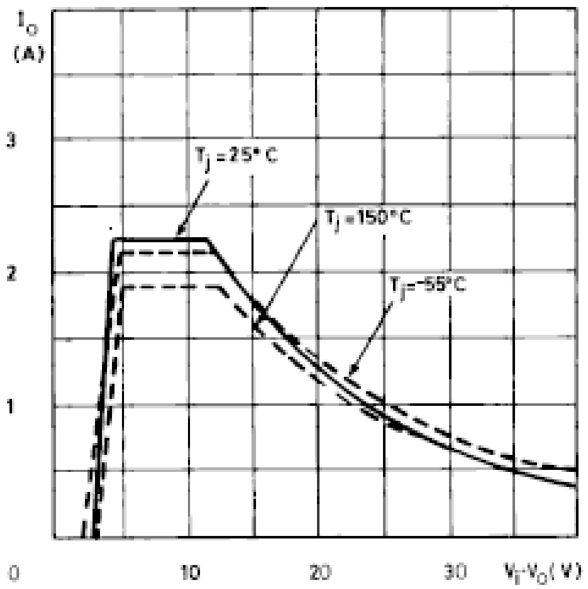


Figure 29: Graph for Output Current for LM317T

Courtesy of Texas Instruments Inc.

The equation for output voltage is $V_o = 1.25(1+(R_2/R_1)) + I_{ADJ}R_2$. Since at the most I_{ADJ} is 100 µA the $I_{ADJ}R_2$ term can be ignored. The desired output is 3.3 volts. Solving for R_1 and R_2 gives values of 1 kΩ and 1.640 kΩ respectively. This would have an output current of about 2 amps. In order to test this, the resistors are checked to confirm they are the correct values. Then the output will be tested to ensure the desired results. Table 26 below shows whether if the parts pass or fail.

Part Testing			
Parts	Required Value	Actual Value	Pass/Fail
R_1	1000 Ω	1002 Ω	Pass
R_2	1640 Ω	1000 Ω + 680 Ω	Pass
Circuit Testing			
Voltage Output	3.3 Volts	3.28 Volts	Pass

Table 26: Results for LM317T

12.11 TPS61222DCKR (Step down voltage regulator) Testing

The TPS61222DCKR is the last step-down voltage regulator to be tested. It has a simple schematic layout that was provided in the datasheet shown in Figure 36. The input is 5 volts and the output should be about 3.3 volts. The output is determined by a voltage divider and has the equation below.

$$R_1 = R_2 \times \left(\frac{V_{OUT}}{V_{FB}} - 1\right)$$

V_{FB} is about 500 mV so that means that the resistor values of R_1 and R_2 must be 5.6 kΩ and 1 kΩ respectively. The test will be conducted to ensure that each part is the correct value that is needed and then to test whether the desired operation has occurred or not. Because the output is across the capacitor the use of an oscilloscope will be able to show the output voltage. Then the current will be shown using a multimeter. Table 27 discusses the results of testing.

Part Testing			
Part	Desired Value	Actual Value	Pass/Fail
R_1	5.6 kΩ	5.59 kΩ	Pass
R_2	1 kΩ	1.01 kΩ	Pass
Inductor	4.7 µH	4.68 µH	Pass
Capacitor	10 µF	10 µF	Pass
Circuit Testing			
Output Voltage	3.3 Volts	3.3 Volts	Pass
Output Current	1.5 Amps	1.51 Amps	Pass

Table 27: Results for TPS61222DCKR

12.11 LM2577 (Boost converter) Testing

This part assists in providing higher voltage to the LCD screen and the backlight. It needs to take 5 volts and boost it to 12 volts. Looking at the datasheet for the LM2577 a typical application of 5 volts to 12 volts is already provided as shown in figure 36 from earlier. Listed in the Table 26 is one 0.1 µF, one 0.3 µF, and one 680 µF capacitor. There is one 100 µH inductor. The three resistors have values of 17.4 kΩ, 2 kΩ, and 2.2 kΩ. All these parts must be tested individually for accuracy purposes. Then the circuit will be tested for the desired voltage and current output. Table 28 below will be used to document the testing parameters.

In addition, this boost converter has to be tested in conjunction with the LED and dimmer circuit as this is the primary usage of the booster.

Parts Testing			
Part	Desired Value	Actual Value	Pass/Fail
R$_1$	17.4 kΩ	17.2 kΩ	Pass
R$_2$	2 kΩ	2 kΩ	Pass
R$_3$	2.2 kΩ	2.18 kΩ	Pass
C$_1$	0.1 µF	0.1 µF	Pass
C$_2$	0.33 µF	0.32 µF	Pass
C$_3$	680 µF	679.4 µF	Pass
L	100 µH	99.08 µF	Pass
Circuit Testing			
Voltage Output	12 Volts	11.98 Volts	Pass
Current Output	800 mA	799 mA	Pass

Table 28: Results for LM2577

12.12 LT1613 (Boost converter) Testing

The purpose of this boost converter is to take 5 volts to 12 volts. It needs to provide a higher voltage to the LCD screen and the backlight. Looking at the datasheet it already comes with a schematic for 5 volts to 12 volts boost converter. The figure 15 from earlier shows how. In the schematic all resistors, capacitors, and inductors will need to be tested to ensure that they perform at the rated value. In addition, the circuit will need to be tested to ensure that the output voltage and current are at the desired values. Table 29 below will record the data for testing.

Part Testing			
Part	Desired Value	Actual Value	Pass/Fail
R_1	107 kΩ	106.2 kΩ	Pass
R_2	12.3 kΩ	12 kΩ	Pass
R_3	10 kΩ	9.97 kΩ	Pass
C_1	22 µF	22 µF	Pass
C_2	4.7 µF	4.7 µF	Pass
C_{PL}	200 pF	198 pF	Pass
Circuit Testing			
Output Voltage	12 Volts	11.99 Volts	Pass
Output Current	120 mA	119 mA	Pass

Table 29: Results for LT1613

12.13 Adafruit ALS-PT19 Analog Light Sensor Breakout Testing

The light sensor’s goal is to determine if when the backlight dulls or intensifies. The way it works is by connecting the light sensor to a voltage source of around 2.5 to 5.5 volts. Then on the analog output you can either measure the voltage or the current that is measured. The increase or decrease in either voltage or current shows if it is getting duller or brighter. Figure 30 below shows a typical setup for getting a readout. Figure 31 below that shows how the values are to be interpreted off the analog output.

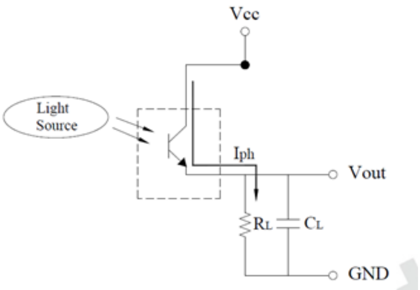


Figure 30: Schematic for ALS-PT19 Analog Light Sensor

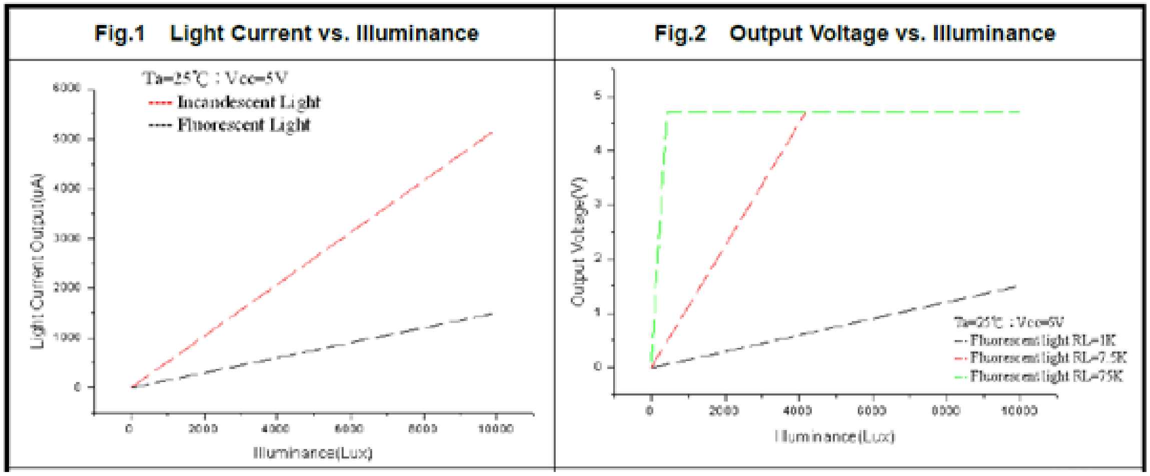


Figure 31: Current and Voltage of Analog Light Sensor based on Illuminance

12.14 Adafruit 161 Photodiode Testing

The photodiode is a simple component that shows the level of lux by changing the resistance like a variable resistor. The higher the resistance the lower the level of light and the lower the resistance the brighter the level of light. It creates a resistance from 200 kΩ to 10 kΩ. The voltage input is up to 100 volts and uses less than 1 mA of current on average. To test if the photodiode works all that needs to be done is connect each end to a multimeter set to resistance. Then to test it in a circuit a simply connect one side of the photodiode to a voltage source connected to a pull-down resistor with a value of 1 kΩ to 10 kΩ. If you need to get better readings to differentiate between bright and really bright use the 1 kΩ. Using the 10 kΩ resistor give different light level ranges. In Figure 32 it can be seen how the photoresistor changes its resistivity with the amount of light that shines on it. The following equation shows the relation between the combined resistance of the resistor and the photocell regarding voltage output. Vo = Vcc(R / (R + Photocell))

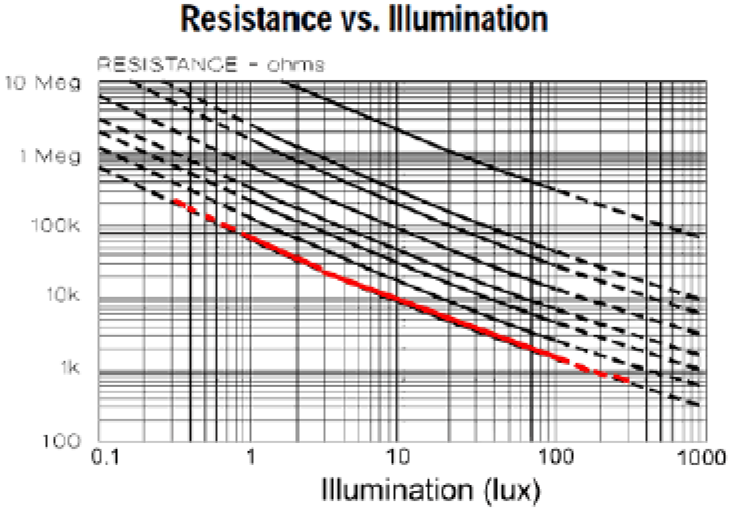
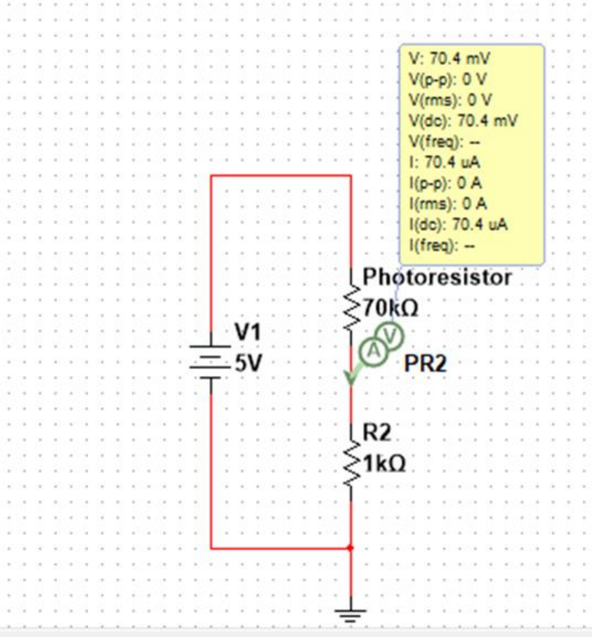


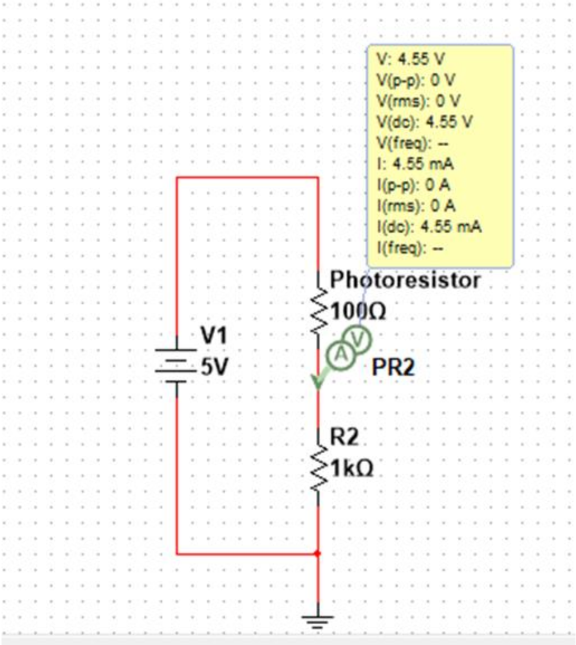
Figure 32: Light vs Illumination

For the photoresistor, even though Multisim does not have an actual photoresistor, a resistor will give a good rough estimate for the photoresistor since that is the basic principle of how it works. The dimmer the light the higher the resistivity, and according to the datasheet it will act like a 70 kΩ resistor. Then when light does shine on it the resistivity drops to about 100 Ω. Very basically this setup works like a voltage divider and the measurement comes from the pull down 1 kΩ resistor. The idea is to use a 5-volt input even though it can take an input of greater or lesser voltage. This gives a good range of values that can be measured across the 100 Ω resistor. The Table 30 below shows a good estimate at how the photoresistor will act at different light levels so that way code is able to be developed to take these values and accordingly dim the display. Simulations of the photodiode can be found in Figures 33 and 34.



```
V: 70.4 mV
V(p-p): 0 V
V(rms): 0 V
V(dc): 70.4 mV
V(freq): --
I: 70.4 uA
I(p-p): 0 A
I(rms): 0 A
I(dc): 70.4 uA
I(freq): --
```

Figure 33: Simulation for Photoresistor in Low Level of Light



```
V: 4.55 V
V(p-p): 0 V
V(rms): 0 V
V(dc): 4.55 V
V(freq): --
I: 4.55 mA
I(p-p): 0 A
I(rms): 0 A
I(dc): 4.55 mA
I(freq): --
```

Figure 34: Simulation for Photoresistor in High Level of Light

Light is Like	Lux Value	Approximate Photoresistor Value	Photoresistor + Pulldown Resistor	Current (mA)	Voltage (V)
Moonlight	1	70 kΩ	71 kΩ	0.0704	0.07
Dark Room	10	10 kΩ	11 kΩ	0.455	0.445
Bright Room	100	1.5 kΩ	2.5 kΩ	2	2
Overcast Day	1000	300 Ω	1.3 kΩ	3.85	3.85
Daylight	10000	100 Ω	1.1 kΩ	4.55	4.55

Table 30: Typical Output Reading from Photoresistor

Tables 31 and 32 below will show how the photodiode responds using a 10 kΩ resistor and a 1 kΩ resistor. There is a disclaimer saying that not all the photodiodes are made the same having around 50% in discrepancy even within the same batch. The same type of testing will be performed with a halogen light like in the ALS-PT19 Light Sensor. A schematic for the light sensor can be found in Figure 35 below.

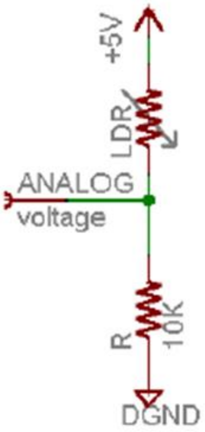


Figure 35: Schematic for Testing Photoresistor

10 kΩ Pull-Down Resistor			
Level of Light	Inches away from light sensor	Voltage Output (V)	Current Output (mA)
Dark (Completely covered)		0.625	0.625
High Intensity	1	4.89	0.498
Medium to High Intensity	2 to 5	4.81	0.48
Medium Intensity	6	4.25	0.426
Low Intensity	6-10+	2.32	0.23

Table 31: Photodiode with 10 kΩ pull-down resistor

1 kΩ Pull-Down Resistor			
Level of Light	Inches away from light sensor	Voltage Output (V)	Current Output (mA)
Dark (Completely covered)		0.023	0.024
High Intensity	1	4.62	4.61
Medium to High Intensity	2 to 5	3.54	3.55
Medium Intensity	6	2.16	2.15
Low Intensity	6-10+	0.62	0.62

Table 32: Photodiode with 1 kΩ pull-down resistor

From this data it will be determined if 1 kΩ or a 10 kΩ resistor will be used and if these are not adequate results maybe another resistor will be used.

12.15 Speaker System Testing

The sections below will conduct the entirety of the testing for individual parts and then the system.

12.15.1 Breadboard-Friendly PCB Mount Mini Speaker - 8 Ohm 0.2W

Testing speakers is a simple enough task. This can be accomplished without an amplifier or source of sound. To test this the only tools that will be needed is a dc power source and a multimeter. Since this is an 8-ohm speaker if the multimeter is in resistance setting, placing the probes on the speaker input and output should produce about 8 ohms. Then to check that the phase is good simply take a dc source around 1 volt since these are tiny speakers and apply a voltage to the input and output terminals. Having positive to positive and negative to negative will make the cone stick out and then reversing the polarity will make the cone stink inwards. Then a continuity check would be the last thing to test to make sure the speaker is intact.

12.15.2 Mini Metal Speaker w/ Wires - 8 ohm 0.5W

The same type of test is performed for the metal 8-ohm 0.5-Watt mini speakers. To test this the only tools that will be needed is a dc power source and a multimeter. Since this is an 8-ohm speaker if the multimeter is in resistance setting, placing the probes on the speaker input and output should produce about 8 ohms. Then to check that the phase is good simply take a dc source around 1 volt since these are tiny speakers and apply a voltage to the input and output terminals. Having positive to positive and negative to negative will make the cone stick out and then reversing the polarity will make the cone stink inwards. Then a continuity check would be the last thing to test to make sure the speaker is intact.

12.15.3 Micro SD Card and Micro SD Adapter for Arduino

The test includes making sure the SD card works and that the SD card reader is able to pull the desired information from the SD card and then play the audio file to the speaker. First, an audio file needs to be created using an online file conversion from mp3 to WAV. Multiple files will be made for the various simple commands that are planned to be utilized for making directions along a route. The online website called https://audio.online-convert.com/convert-to-wav is a great tool for this purpose. Once the audio file is created download it to the Micro SD card and insert it into the Micro SD adapter. From there configure in the manner described in section 4.6.5 Working Together. This is important for the Arduino library because it sends and receives certain signals and commands to specific ports. With speaker connected as well, code will be developed for off-road testing conditions in which debugging the audio output doesn't have to be done in real time. Then the speaker wires are connected to an oscilloscope to measure the speaker's actual waveform output if sound is not produced so further debugging can continue. Further actual testing will have to be done on the road to ensure that the proper commands deliver the correct directions to the user. In the table is a list of possible commands that will be recorded to be coded into the directions part of the navigation.

12.16 Dimmer Circuit Testing

Using the figure 19 as the schematic for this test the purpose is to test if the dimmer circuit can dim the LED according to different pulses provided by the PWM that comes from the microcontroller. To run this test, it has to independent from the light just for testing purposes. Because we need to find out the best timing for how the LED will respond to the modulation of the PWM, the test is about the illumination of the LED that corresponds to the best on-time off-time that gives the best result for dimming. For this test to be meaningful, multiple levels of brightness need to be determined for when the microcontroller interprets how bright it is outside. Ideally about five levels should be determined for a good range of levels of light.

12.16.1 Backup Battery Testing

Making sure that the backup battery system works is important for the emergency SMS and email and making sure that they can make contact in the case of a wreck. Testing the backup battery is done in two parts. First make sure the original circuit works and supplies the regulated 5 volts to the load. Then disconnect the power supply so the battery takes over to be the power supply and make sure that the battery can handle the load. The power supplies will travel through a 3.3-volt regulator that works for the SMS module. Then make sure that the battery can last long enough to send out the emergency SMS and email to their respective contacts.

12.17 PCB Testing

A huge thanks is needed to be given to Quality Manufacturing Services, as they helped put the parts onto the PCB. A lot of practice went into learning how to solder, but they helped put the smaller, and harder pieces on the PCB. Once assembled testing began on the board and several issues arose. The biggest one of all was the 16 MHz crystal that the ATMega2560 uses. In order to place code onto the ATMega2560 via the In Circuit Serial Programming (ICSP), two 22 pF capacitors were needed to be placed on the ends of the crystal. To test if programming was going to be an issue an initial test, a simple one second incremental counter was used, was displayed onto the OLED screen. While running the code each individual second took approximately 20 seconds between increments showing that the crystal that is used as a clock signal was giving an inaccurate signal. It was soon found that there is an internal clock in the ATMega2560 that can be used as a clock signal which solved the timing issue. Upon deeper investigation it was shown that instructions on how to push code onto the ATMega2560 was slightly flawed and once fixed worked perfectly. Another problem was within the PCB itself. With the first prototype there were to footprints that were wrong such as the pins for the ICSP, and the footprint for the MIC29302WU had wrong pin sizes so wires were needed to properly connect the part on. Another lesson learned was to not always trust the schematic because new parts are needed to replace old ones and they have different attributes. An example is the status LEDs that were used would not turn on simply because of a resistor value didn't allow enough current to flow through to the LEDs so a new value was calculated. Another revision of the PCB fixed those problems. Figures 36 and 37 show the final product of our design.

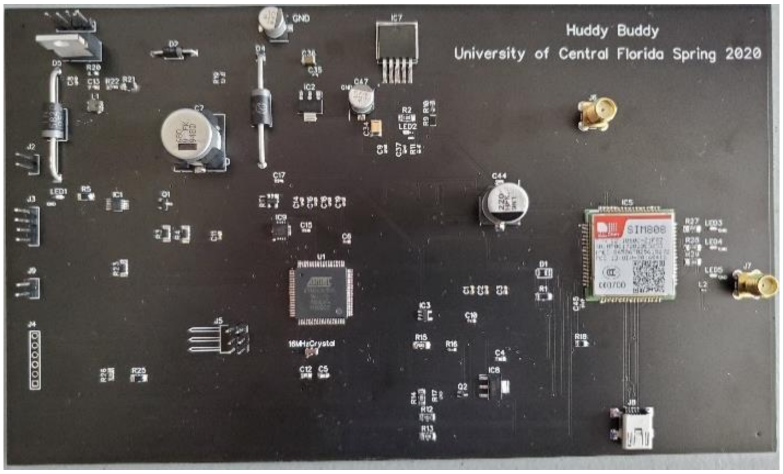


Figure 36: Front of PCB

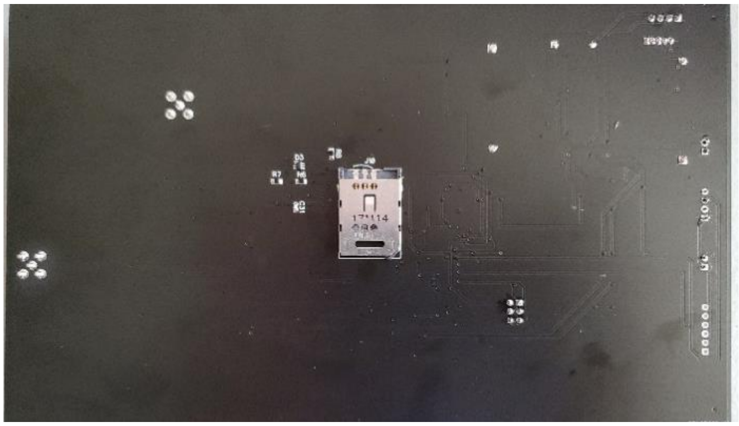


Figure 37: Back Of PCB

12.18 Bluetooth Prototyping and Testing

The Bluetooth communication is tested using RSSI readings and serial monitor.

Using a free smartphone application, we can determine RSSI commands that will verify an established Bluetooth connection. A signal strength of -30dBm or more is considered according metageek documentation and will be used as a reference to determine the HUD's communication signal strength.

To check the validity of data, a serial monitor on Arduino IDE along with a Command Line prompt can be used. The data bytes transmitted are ASCII characters and because

ASCII is a global standard; it makes deciphering the code easier. A good software test will be to run a loop that transmits packets continuously and the receiving end sends an acknowledgement in return to validate that the data has been received.
A counter should be incremented every time a packet is received and sent to track the number of packets exchanged and test if any packets were lost during transmission. If a packet is lost, debugging begins to see if the issues are due to RF or digital portions.

Oscilloscopes that display Bluetooth RF Frequencies are useful when troubleshooting and analyzing Bluetooth compliant devices. Although this hardware is not available to us, a regular oscilloscope will be more than sufficient for testing as the specification sheet provides us all the expected values needed to be read.

12.18.1 Bluetooth Module Bootloader

The nRF52840 module is a blank chip that must be flashed with code before use. To do this, we use a combination of a SWD (Serial Wire Debugger) Programmer with its hardware setup similar to the figure below, as well as DFU Bootloader tools in order to configure the kernel and other important aspects of startup code.

Nordic's SDK for boot loading uses a Python based tool for updating via serial called adafruit-nrfutil. This requires Python 3 to be installed on a computer and installed via PyPI through Command Line. Once all the necessary programming tools are installed and hooked up, we will use Nordic's Secure DFU bootloader mode in order to update firmware on the device and make it secure.

The way the DFU is secure is using cryptography keys. Two keys are used, a public key and a private key. This is generated through the nrfutil library installed prior. The commands to do this are as follows.

```
nrfutil.exe keys generate private.key
```

This creates the private.key file that will be used for decryption. After this key is created, a public key is made through command line using the private.key file.

```
nrfutil keys display --key pk --format code private.key --out_file public_key.c
```

Now the respective public key that is used for encryption is created based off of the private key. With the two keys generated begins the compiling and building of the bootloader itself. An external open source library called uECC on github. The uECC library implements a small and fast Elliptic-curve Diffie–Hellman (ECDH) and Elliptic Curve Digital Signature Algorithm (ECDSA) algorithm that is made for specifically public key-based cryptography, which is what we made earlier. This library is compatible with our 32-bit ARM processor within the BLE module and is written in C so everything is compatible. This library is then cloned into a SDK folder and compiled. Once this is compiled, the command "make" will build the bootloader based on the public key.

A DFU .zip packet is required for the DFU master to send new image file to the DFU target. With the hex file generated prior, the zip file is made with the following command

nrfutil pkg generate --hw-version 52 --application-version 1 --application nrf52840_xxaa.hex --sd-req 0x98 --key-file private.key app_dfu_package.zip

--hw-version: By default, this should match with the chip. Since we are using a nRF52xxx chip, we use "52".

--application-version: By default, the start number for application version is 0. To be able to update new application, the application version should equal or greater than the one the bootloader stored. This case we use 1.

--sd-req: In this case the application runs with Softdevice S132 v4.0.2. The code number for this softdevice version is 0x98. This was found by typing *nrfutil pkg generate –help.*

--application: Tells nrfutil that you are going to update the application and the application image is provided.

With the final zip file created, the zip file is sent to the IDE used and update via serial.

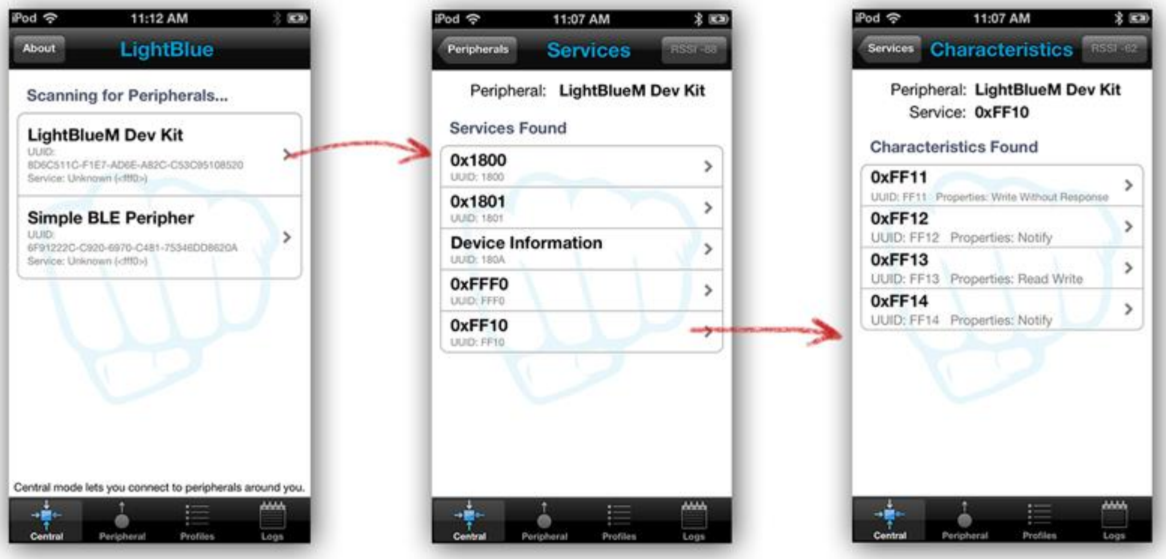
With the bootloader applied to the device, any application code sent to the device must take into consideration that the boot loader exists in memory. Table 33 below is a memory map of addresses of the nRF52840.

Description	**Start Address**	**End Address**	**Size (KB)**
Bootloader Settings	0x000FF000	0x000FFFFF	4
Master Boot Record Params	0x000FE000	0x000FEFFF	4
Bootloader	0x000F4000	0x000FDFFF	40
Application Code	0x00026000	0x000F3FFF	824
SoftDevice	0x00001000	0x00025FFF	148
Master Boot Record	0x00000000	0x00000FFF	4

Table 33: Bluetooth Addresses

12.19 Communication tests

When packets are sent over from the device to the phone and vice versa, we must confirm that the data being sent over are not garbage values and that packets do not overlap each other. This is done through multiple ways. First, we confirm the validity through RSSI, or “Received Signal Strength Indicator,” which is one of two ways to confirm the packet. The second is through measurement of the RF signal. Because Bluetooth is a radio frequency-based technology, an oscilloscope is used to measure the signal is dBm.



Scanning for Peripherals...

LightBlueM Dev Kit
UUID:
8D6C511C-F1E7-AD6E-A82C-C53C95108520
Service: Unknown (<fff0>)

Simple BLE Peripher
UUID:
6F91222C-C920-6970-C481-75346DD8620A
Service: Unknown (<fff0>)

Central mode lets you connect to peripherals around you.

Peripheral: LightBlueM Dev Kit

Services Found

0x1800
UUID: 1800

0x1801
UUID: 1801

Device Information
UUID: 180A

0xFFF0
UUID: FFF0

0xFF10
UUID: FF10

Peripheral: LightBlueM Dev Kit
Service: 0xFF10

Characteristics Found

0xFF11
UUID: FF11 Properties: Write Without Response

0xFF12
UUID: FF12 Properties: Notify

0xFF13
UUID: FF13 Properties: Read Write

0xFF14
UUID: FF14 Properties: Notify

Figure 38: LightBlue App Functionality

There exists free applications on the Apple AppStore that allows testing of Bluetooth devices in order to gain knowledge of what hex values are being sent. One of these applications is called LightBlue, demonstrated in Figure 38.

The HUD’s BLE will be configured as a peripheral that has data the phone, the central, needs. With this application, we can send packets with a delay as low as 50 milliseconds and as long as 2 seconds. The data on Peripherals are organized according to their respective Profile. This profile contains a variety of services and characteristics. Characteristics are the holders of data, and can be accessed in 3 ways- Read, Write, and Notify. With these three types of data manipulation, we can fully test out the BLE module on a high level and make sure that distance is not a problem as well as any kind of interference.

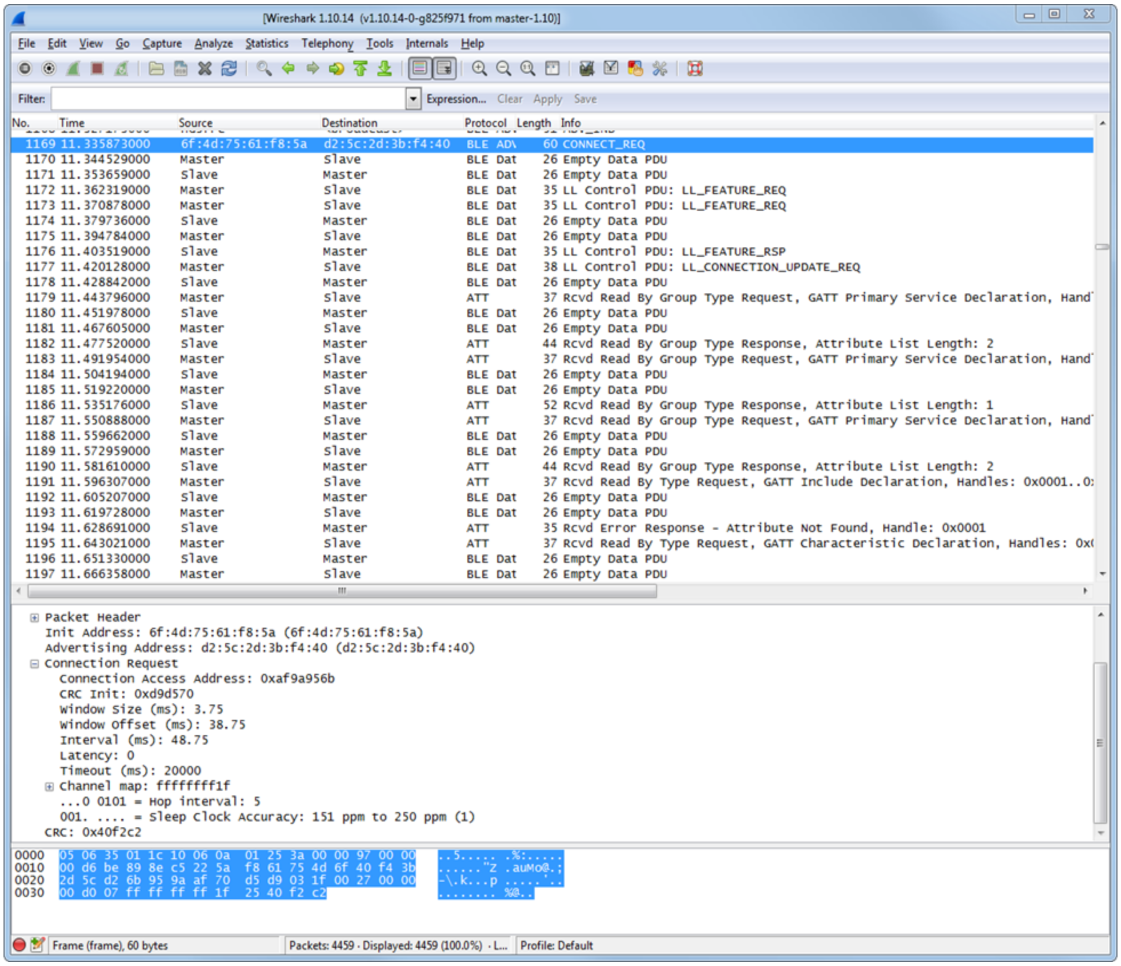
Within the application we can find addresses to read and write to. In the figure above we can see their example has an address that starts at 0xFF10, and from there they can look at nearby addresses and determine what type of address it is. This way we can access data values such as RAM.

12.20 Packet Sniffing for Testing Both Nodes

This sniffer uses a Nordic evaluation board with special firmware programmed to the device. The nRF52840 captures the BLE packets, adds time, RSSI, and other metadata, and forwards them to Wireshark. The advantage here is that Wireshark is an industry standard tool that allows anyone to view your information. Wireshark is a free and open-source packet analyzer available for Android devices. A few of the members on the team have used Wireshark in Computer Communication Networks to analyze packet formatting between a computer and the network. We are already familiar with how Wireshark works, thus being a great tool for us to take advantage of.

Having software to check packets on both Android and iOS operating systems allows us to test on two of the most common smartphone operating systems on the market today. LightBlue and Wireshark will provide all the Bluetooth packet testing needed once the RF portions are confirmed to be operational and stable. We anticipate that data being sent between Bluetooth devices may initially contain erroneous data. LightBlue and Wireshark will aid us in troubleshooting any errors that might occur. Additionally, we can verify that Bluetooth packets are actually being sent between the two devices and are arriving in the correct format.

Once we have confirmed that the packets are being formatted correctly, we can analyze the data on the received end. They should arrive in the exact same way that Wireshark viewed the data. Figure 39 demonstrates how packets look when they are analyzed by WireShark.


[Wireshark 1.10.14 (v1.10.14-0-g825f971 from master-1.10)]
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter:
Expression... Clear Apply Save

```
No.  Time         Source             Destination        Protocol Length Info
1169 11.335873000 6f:4d:75:61:f8:5a  d2:5c:2d:3b:f4:40  BLE ADV    60 CONNECT_REQ
1170 11.344529000 Master             Slave              BLE Dat    26 Empty Data PDU
1171 11.353659000 Slave              Master             BLE Dat    26 Empty Data PDU
1172 11.362319000 Master             Slave              BLE Dat    35 LL Control PDU: LL_FEATURE_REQ
1173 11.370878000 Master             Slave              BLE Dat    35 LL Control PDU: LL_FEATURE_REQ
1174 11.379736000 Slave              Master             BLE Dat    26 Empty Data PDU
1175 11.394784000 Master             Slave              BLE Dat    26 Empty Data PDU
1176 11.403519000 Slave              Master             BLE Dat    35 LL Control PDU: LL_FEATURE_RSP
1177 11.420128000 Master             Slave              BLE Dat    38 LL Control PDU: LL_CONNECTION_UPDATE_REQ
1178 11.428842000 Slave              Master             BLE Dat    26 Empty Data PDU
1179 11.443796000 Master             Slave              ATT        37 Rcvd Read By Group Type Request, GATT Primary Service Declaration, Hand
1180 11.451978000 Slave              Master             BLE Dat    26 Empty Data PDU
1181 11.467605000 Master             Slave              BLE Dat    26 Empty Data PDU
1182 11.477520000 Slave              Master             ATT        44 Rcvd Read By Group Type Response, Attribute List Length: 2
1183 11.491954000 Master             Slave              ATT        37 Rcvd Read By Group Type Request, GATT Primary Service Declaration, Hand
1184 11.504194000 Slave              Master             BLE Dat    26 Empty Data PDU
1185 11.519220000 Master             Slave              BLE Dat    26 Empty Data PDU
1186 11.535176000 Slave              Master             ATT        52 Rcvd Read By Group Type Response, Attribute List Length: 1
1187 11.550888000 Master             Slave              ATT        37 Rcvd Read By Group Type Request, GATT Primary Service Declaration, Hand
1188 11.559662000 Slave              Master             BLE Dat    26 Empty Data PDU
1189 11.572959000 Master             Slave              BLE Dat    26 Empty Data PDU
1190 11.581610000 Slave              Master             ATT        44 Rcvd Read By Group Type Response, Attribute List Length: 2
1191 11.596307000 Master             Slave              ATT        37 Rcvd Read By Type Request, GATT Include Declaration, Handles: 0x0001..0:
1192 11.605207000 Slave              Master             BLE Dat    26 Empty Data PDU
1193 11.619728000 Master             Slave              BLE Dat    26 Empty Data PDU
1194 11.628691000 Slave              Master             ATT        35 Rcvd Error Response - Attribute Not Found, Handle: 0x0001
1195 11.643021000 Master             Slave              ATT        37 Rcvd Read By Type Request, GATT Characteristic Declaration, Handles: 0x(
1196 11.651330000 Slave              Master             BLE Dat    26 Empty Data PDU
1197 11.666358000 Master             Slave              BLE Dat    26 Empty Data PDU
```

```
⊞ Packet Header
  Init Address: 6f:4d:75:61:f8:5a (6f:4d:75:61:f8:5a)
  Advertising Address: d2:5c:2d:3b:f4:40 (d2:5c:2d:3b:f4:40)
⊟ Connection Request
    Connection Access Address: 0xaf9a956b
    CRC Init: 0xd9d570
    Window Size (ms): 3.75
    Window Offset (ms): 38.75
    Interval (ms): 48.75
    Latency: 0
    Timeout (ms): 20000
  ⊞ Channel map: ffffffff1f
    ...0 0101 = Hop interval: 5
    001. .... = Sleep Clock Accuracy: 151 ppm to 250 ppm (1)
  CRC: 0x40f2c2
```

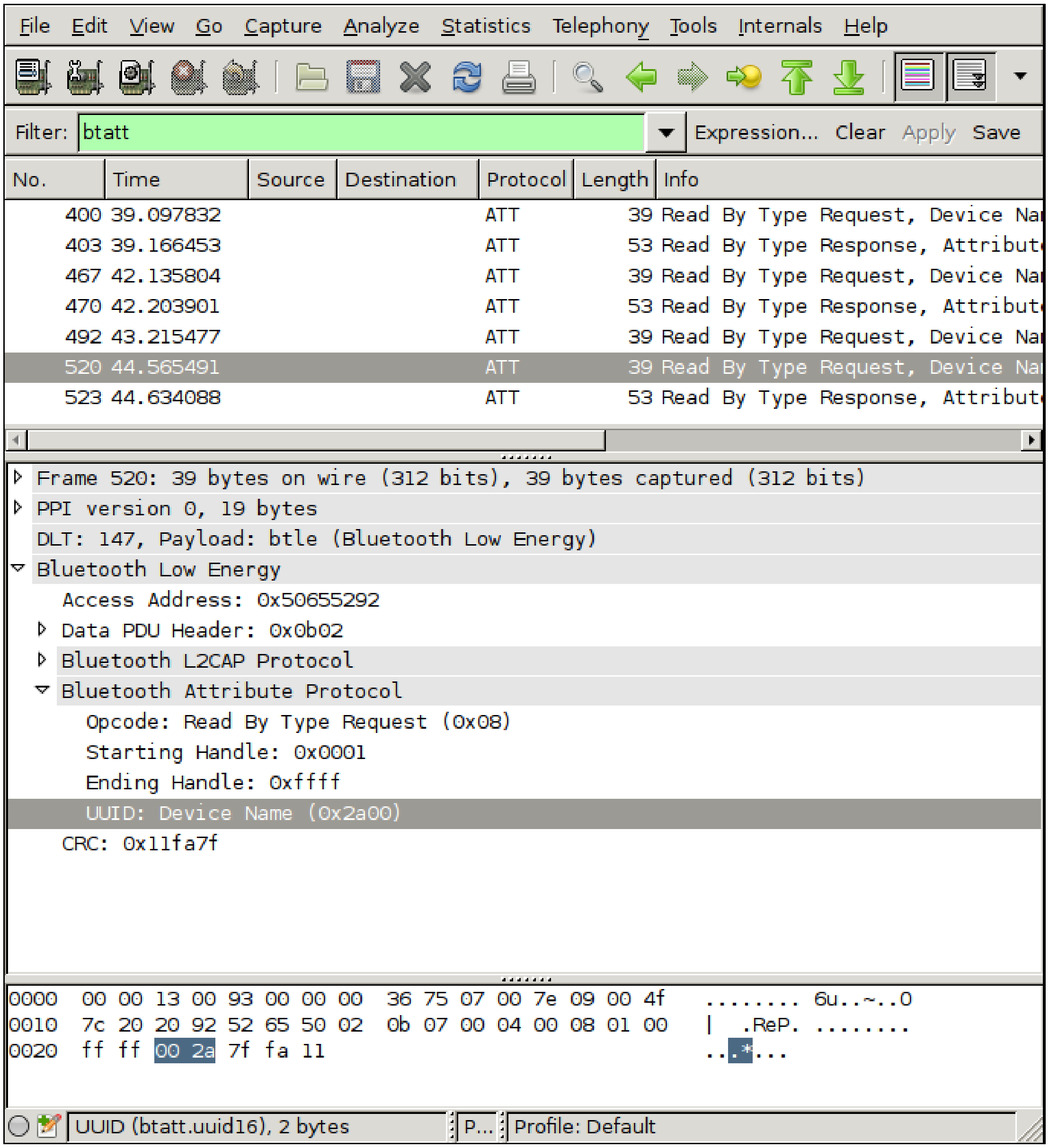
```
0000  05 06 35 01 1c 10 06 0a  01 25 3a 00 00 97 00 00   ..5..... .%:.....
0010  00 d6 be 89 8e c5 22 5a  f8 61 75 4d 6f 40 f4 3b   ......"Z .auMo@.;
0020  2d 5c d2 6b 95 9a af 70  d5 d9 03 1f 00 27 00 00   -\.k...p .....'..
0030  00 d0 07 ff ff ff ff 1f  25 40 f2 c2               ........ %@..
```

```
Frame (frame), 60 bytes | Packets: 4459 · Displayed: 4459 (100.0%) · L... | Profile: Default
```

Figure 39: WireShark Interface

12.20.1 Packet Sniffing with WireShark

This is an example of what Bluetooth traffic is formatted like when testing packets for sending and receiving. This excerpt is taken from the command prompt portion of WireShark. Important information to take note of is the access address value, the attribute protocol that tells what the packet's characteristics are, as well as Opcode that says whether it is a packet being sent or received. The example above is a request from the computer to the BLE module asking for what the device name is. In the bottom where "00 2a" is highlighted, this is the portion that is being sent and specifically asks for the device name. When translated out of hex, it represents ".*" which does not mean anything to us but to the device, its recognized at requesting the name, demonstrated in Figure 40 and 41.



Filter: btatt

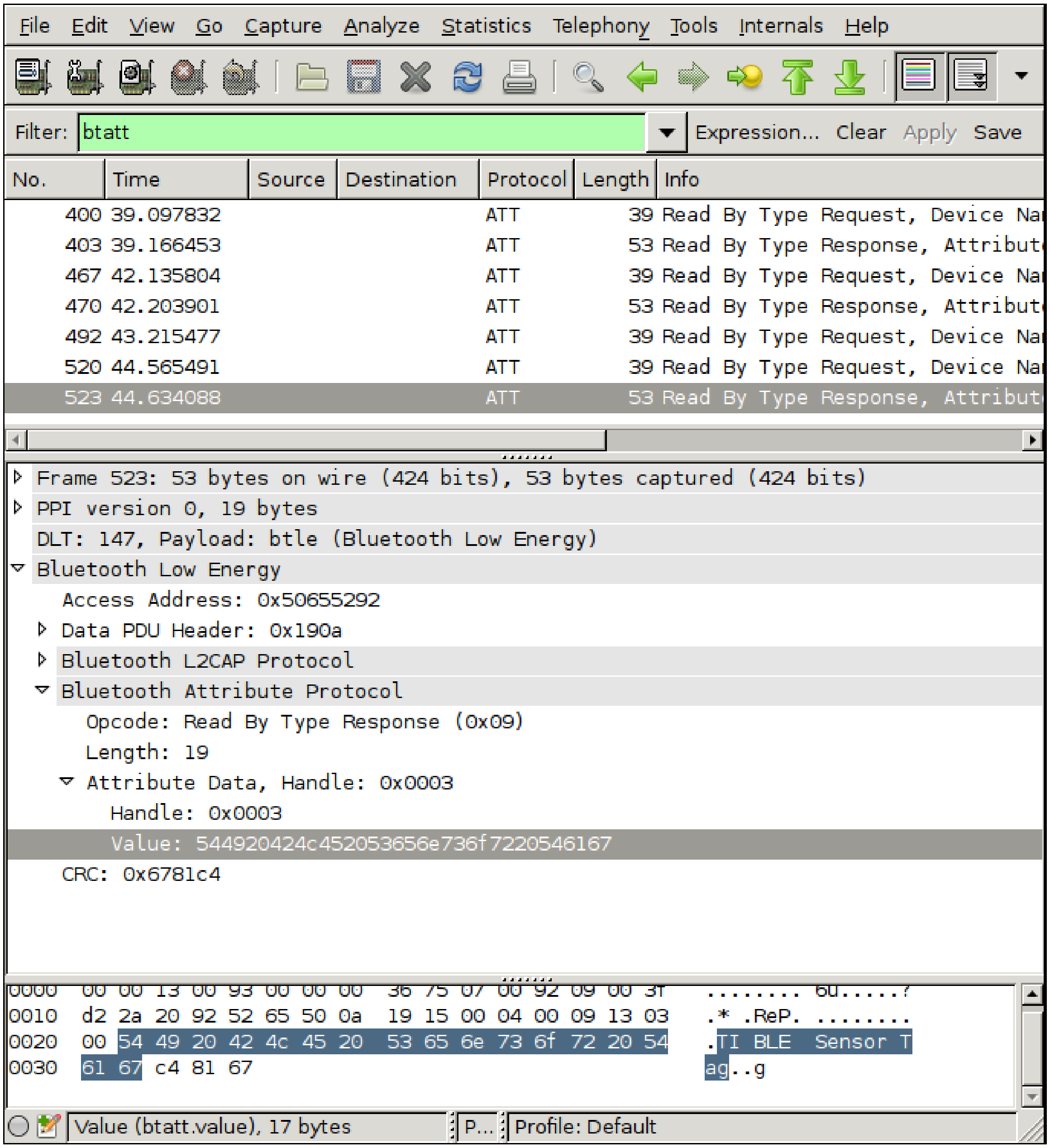
```
No.     Time        Source  Destination  Protocol  Length  Info
    400 39.097832                        ATT           39 Read By Type Request, Device Na
    403 39.166453                        ATT           53 Read By Type Response, Attribut
    467 42.135804                        ATT           39 Read By Type Request, Device Na
    470 42.203901                        ATT           53 Read By Type Response, Attribut
    492 43.215477                        ATT           39 Read By Type Request, Device Na
    520 44.565491                        ATT           39 Read By Type Request, Device Na
    523 44.634088                        ATT           53 Read By Type Response, Attribut
```

```
▷ Frame 520: 39 bytes on wire (312 bits), 39 bytes captured (312 bits)
▷ PPI version 0, 19 bytes
  DLT: 147, Payload: btle (Bluetooth Low Energy)
▽ Bluetooth Low Energy
    Access Address: 0x50655292
  ▷ Data PDU Header: 0x0b02
  ▷ Bluetooth L2CAP Protocol
  ▽ Bluetooth Attribute Protocol
      Opcode: Read By Type Request (0x08)
      Starting Handle: 0x0001
      Ending Handle: 0xffff
      UUID: Device Name (0x2a00)
    CRC: 0x11fa7f
```

```
0000  00 00 13 00 93 00 00 00  36 75 07 00 7e 09 00 4f   ........ 6u..~..O
0010  7c 20 20 92 52 65 50 02  0b 07 00 04 00 08 01 00   |  .ReP. ........
0020  ff ff 00 2a 7f fa 11                               ..*....
```

UUID (btatt.uuid16), 2 bytes | P... | Profile: Default

Figure 40: WireShark Address 00 2a being sent to the HUD


File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter:
Expression... Clear Apply Save

Filter: btatt

```
No.     Time        Source  Destination  Protocol  Length  Info
   400 39.097832                          ATT          39 Read By Type Request, Device Na
   403 39.166453                          ATT          53 Read By Type Response, Attribut
   467 42.135804                          ATT          39 Read By Type Request, Device Na
   470 42.203901                          ATT          53 Read By Type Response, Attribut
   492 43.215477                          ATT          39 Read By Type Request, Device Na
   520 44.565491                          ATT          39 Read By Type Request, Device Na
   523 44.634088                          ATT          53 Read By Type Response, Attribut
```

```
▷ Frame 523: 53 bytes on wire (424 bits), 53 bytes captured (424 bits)
▷ PPI version 0, 19 bytes
  DLT: 147, Payload: btle (Bluetooth Low Energy)
▽ Bluetooth Low Energy
    Access Address: 0x50655292
  ▷ Data PDU Header: 0x190a
  ▷ Bluetooth L2CAP Protocol
  ▽ Bluetooth Attribute Protocol
      Opcode: Read By Type Response (0x09)
      Length: 19
    ▽ Attribute Data, Handle: 0x0003
        Handle: 0x0003
        Value: 544920424c452053656e736f7220546167
    CRC: 0x6781c4
```

```
0000  00 00 13 00 93 00 00 00  36 75 07 00 92 09 00 3f   ........ 6u.....?
0010  d2 2a 20 92 52 65 50 0a  19 15 00 04 00 09 13 03   .* .ReP. ........
0020  00 54 49 20 42 4c 45 20  53 65 6e 73 6f 72 20 54   .TI BLE  Sensor T
0030  61 67 c4 81 67                                     ag..g
```

Value (btatt.value), 17 bytes | P... | Profile: Default

Figure 41: WireShark Bluetooth Name Returned

This next excerpt shows a packet received from the Bluetooth module in response to the packet sent above. The access address of the BLE is not surprisingly the same as all communication is based off that address. The response is in hexadecimal which is highlighted above. Wireshark lists that the total length of the packet as 19. When the hexadecimal string is translated, it reads “TI BLE Sensor Tag” which is the name of the TI CC2540 that is used in this example. These two screenshots show that Wireshark is very powerful, handy, and fits our needs perfectly.

12.21 HC-05 Testing

Testing and implementation of the HC-05 was done through an Arduino environment with it being hooked up to an Arduino Nano to pass over sketches as well as using Arduino’s Serial monitor to see messages received over Bluetooth and sent to the monitor via USB serial. The LightBlue app on an Android device allowed a simple message to be sent to the HC-05 once a connection was made, and simple text strings sent from the phone would appear in the serial monitor once the HC-05 was properly configured. From there, custom Bluetooth code was tested and implemented into the Huddy Buddy app in order to have the same results as the LightBlue app. Testing was considered done once the custom Bluetooth code was able to send data from the Huddy Buddy app to the HC-05 and that data be properly received and shown in the Arduino Serial Monitor.

13.0 Project Hardware Design:

The hardware mentioned in this section are the physical components that will be used to power and control our heads-up display. This section will provide information concerning each section of hardware design.

13.1 Block Diagram

Our project requires a broad scope of knowledge to implement each subsystem to the final product. The group broke down each task by topic and system, allocating our knowledge and resources efficiently. The block diagram in Figure 36 describes each task with how it fits into the final product

The left side of the block diagram, highlighted in yellow, contains all the electrical subsystems required to power the HUD device. Evan Hall, our Electrical Engineering student, was tasked with designing the power delivery and PCB aspect of the final design. Evan designed a system to deliver power from the OBD-II port in the vehicle to the HUD device. As mentioned in Section 4.0, we implemented different electrical components to power each component of the design.

The middle section of the block diagram, highlighted in blue, contains all the display focused components of the HUD device. Aaron Majdali, our Optics and Photonics Engineering student, designed and implemented the display unit, optical focusing system, and the mounting of the display within the vehicle.

The right section of the block diagram, highlighted in green, contains all the programming and microcontroller hardware required to implement the HUD device. Logan Glowth and Pedrhom Nafisi, our Computer Engineering students were tasked with designing and building the mobile application, computing components, and data transferring methods

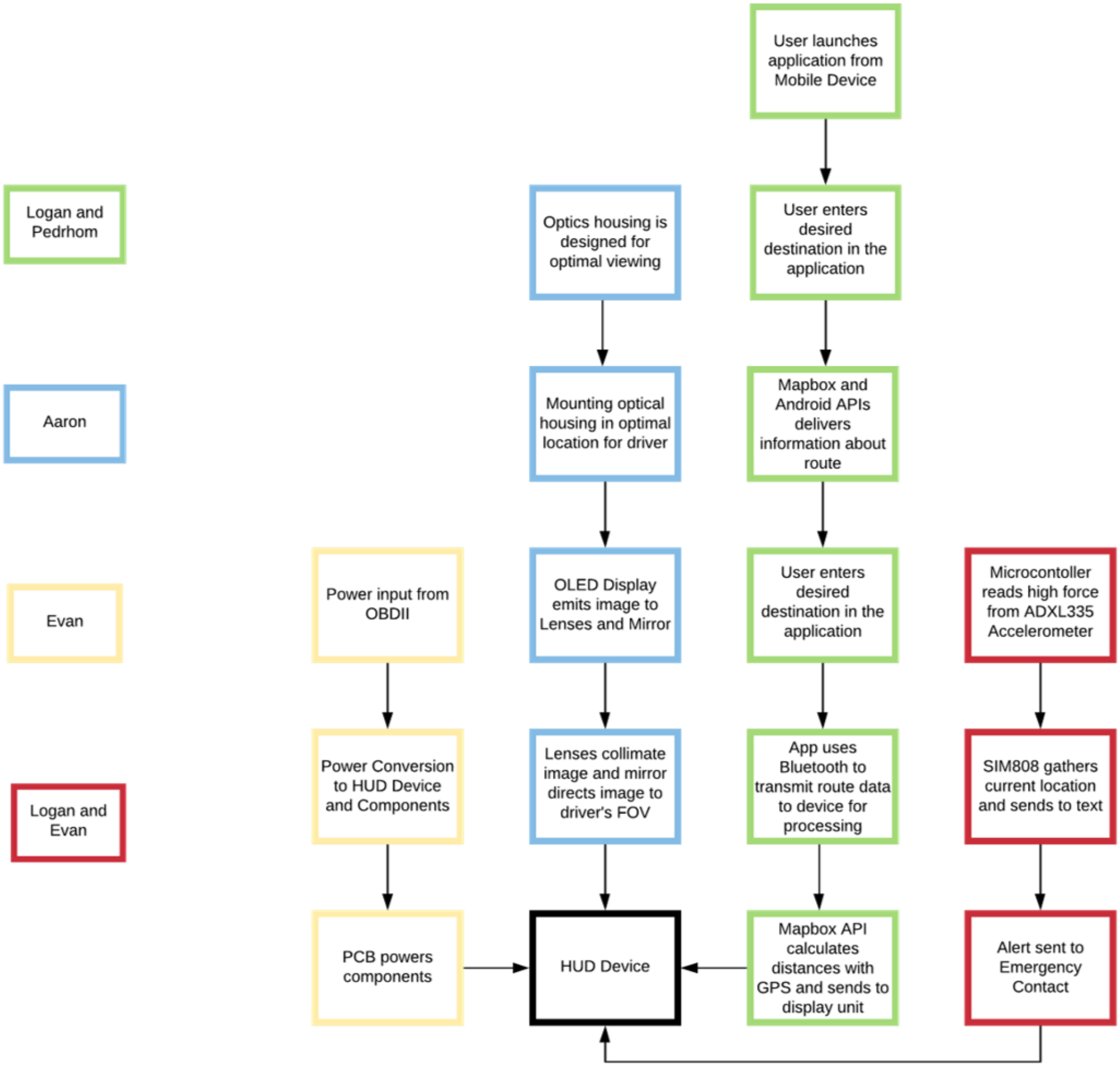

Logan and Pedrhom
Aaron
Evan
Logan and Evan
User launches application from Mobile Device
Optics housing is designed for optimal viewing
User enters desired destination in the application
Mounting optical housing in optimal location for driver
Mapbox and Android APIs delivers information about route
Power input from OBDII
OLED Display emits image to Lenses and Mirror
User enters desired destination in the application
Microcontoller reads high force from ADXL335 Accelerometer
Power Conversion to HUD Device and Components
Lenses collimate image and mirror directs image to driver's FOV
App uses Bluetooth to transmit route data to device for processing
SIM808 gathers current location and sends to text
PCB powers components
HUD Device
Mapbox API calculates distances with GPS and sends to display unit
Alert sent to Emergency Contact

Figure 42: Block Diagram

Figure 43: Front of PCBFigure 44: Block Diagram

13.2 Display System

The primary optical elements of a heads-up display are a projector and a combiner. The projector is anything that can project an image onto the combiner, which is a transparent surface that will show the user both the outside world and the projected image. The first step in determining what components to use for the heads-up display was to determine how we wanted to display the data. At first, it was thought that the display could merely be a bright screen reflecting off of a windshield or a smaller vertical piece of glass. Upon consultation with several experts, it was decided that our heads-up display would feature a collimated image, generated via a smaller screen and a lens array.

13.2.1 Projector Design

Our heads-up display will feature a small monochromatic OLED display. The OLED, a Grove OLED 0.96”, will be placed at the focal length of a positive lens after a negative lens is inserted between the two other elements. The positive lens will have a diameter of 3 inches and a focal length of 4 inches, while the negative lens will have a diameter of 2 inches and a focal length of 4 inches. With the negative lens installed, the OLED will be inside the negative lens’ focal length and a smaller virtual image of the OLED will be formed. This virtual image will be 4 inches away from the positive lens, thus producing a collimated image that will have less distortion than a system with a single positive lens. Between the two lenses is a mirror, angled at 45 degrees, that will redirect the image up through the positive lens and onto the driver’s windshield. Current design for the projector housing uses 3-D printed parts due to their availability and ease of controlling final dimensions. The housing will be mounted to the vehicle using adhesive velcro strips, commonly available from many stores and non-permanent. Because the display used is an OLED display, power consumption is minimal at 1.5 Watts. This system will produce an image with adequate resolution and high contrast. Due to the brightness constraints of OLED technology, the system will not be visible in bright sunlight. However, other technologies like micro-LED, liquid crystal on silicon (LCOS), and others may produce an acceptably bright image. Moreover, these alternate display technologies would simply substitute the OLED display currently in the projector.

13.2.2 Combiner Design

The intended combiner for the display system will be the windshield of the vehicle. This reduces cost and complexity, as the projector now only needs to fit under the windshield without worrying about a separate combiner. The projector will project its image vertically, but the projector enclosure can be oriented to accommodate different windshield slopes and other constraints. If need be, readily available aftermarket reflective films can be applied to the windshield to produce a more easily seen image.

14.0 Budget:

To implement this project, we will need to research and obtain materials that will meet our specifications and requirements in order to produce an effective heads-up display device. Many of the parts purchased in this project will be sourced from online retailers, brick-and-mortar retailers, and from other vendors as necessary. Due to the nature of electronics production, most parts that are ordered will come directly from overseas manufacturers. This will require factoring in cost of shipping, lead times, and backup plans to each part purchased. We will need a PCB, a power delivery system, microprocessors, LEDs, soldering equipment, a Bluetooth module, and a speaker. We will not be obtaining funding or sponsorships for anything involved with this project. As such, the group will be self-funded and will split the cost required to complete the device.

All parts obtained throughout the project will be noted with the price of purchase, not including shipping, to ensure we are staying within our allotted budget. If staying within our budget is not possible, we will discuss the ramifications of increasing our budget or making cuts to minimize over-expenditures. We have determined that a rough estimate of $500 will be enough to completely implement the project.

Table 38 discusses the budget that we have allotted for the project. As we begin ordering parts for the HUD Device, we anticipate that parts that were not originally discussed will need to be ordered to fully implement the discussed items. This will increase our total expenditures as we move forward.

Item	Description	Price ($)
PCB	Implement all hardware needed onto PCB	~35
Display Unit	Display information to driver on windshield	100
Power Delivery System	Provides power to system	~45
Microprocessor	Processes data and sends to display unit	50
LEDs	Show power states and pertinent information	15
Soldering Equipment	Needed to implement electrical components	Free if borrowed from school
Bluetooth Module	Receive data from phone over Bluetooth connection	20
Bluetooth Testing Development Board	Allows debugging and initial exposure to the nRF52840 through Arduino and Nordic SDK	25
J-Link OB ARM Programmer	Flashes firmware and sends code to blank Bluetooth module.	5
Oscilloscope	Allows testing of RF portions as well as Voltage differences	Free if borrowed from school
Speaker	Play recorded sounds in specific situations	~2
Mobile Smartphone	Needed to host custom Application	N/A
GPS/3G Module	Feed location data to system	90
3G Antenna	Antenna to connect to mobile network	15
Battery	Used as backup power	20

Table 34: Budget

14.1 Cost Analysis of the Circuity Involved in Production

The items listed in Tables 35 and 36 are an initial analysis of what would be required when purchasing the components that will be applied to our PCB. Throughout the next few months, we will be adding to this table as we add more components to the final PCB design.

Parts for Power Supply			
Part	**Cost of Single Unit**	**Cost of 1k units**	**Supplier**
Micro USB Type B	$ 1.50	$ 1.20 ($ 1200)	Adafruit
LM317	$ 2.03	$ 0.93 ($ 930.15)	Digi-Key
LM2577	$ 5.62	$ 2.92 ($ 2920)	Texas Instruments
LM2596	$ 4.16	$ 2.16 ($ 2160)	Texas Instruments
LT1613	$ 4.28	$ 2.28 ($ 2280)	Digi-Key
TPS6122	$ 1.14	$ 0.48163 ($ 481.63)	Digi-Key
Peripheral Parts			
Light Sensor (Photodiode)	$ 0.95	$ 0.76 ($ 760)	Adafruit
2N5088	$ 0.46	$0.17424 ($ 174.24)	Digi-Key
FQP50N06L	$ 0.921	$ 0.779 ($ 779)	Newark

Table 35: Cost of Power Supply Parts

Parts for Speaker Implementation			
Part	Cost of Single Unit	Cost of 1k Units	Supplier
8 Ohm Metal Speaker	$ 1.95	$ 1.56 ($ 1560)	Adafruit
SD Card Reader	$ 2.09	$ 1.41 ($ 1410)	Banggood
Micro SD Card	$ 3.99	$ 3.49 ($ 3490)	Banggood

Table 36: Cost of Speaker Implementation

Buying the parts for one unit for the heads-up display from the power delivery point of view and minor peripheral parts will cost $ 21.06. Making at least one thousand units will reduce this cost to $ 11.68, almost a half reduction in price. In addition, the cost of a single implementation of the speaker is going to be $8.03, but for one thousand units it drops to $5.36, which makes more room for profit margins when the product goes to market.

As with buying anything in bulk it is always cheaper and would prove to be beneficial when selling our product on the market making it easier to turn a profit. In addition, what was not listed was the price of shipping. In obtaining these parts the cost of shipping for an individual unit was well over twice to three times the cost of the actual part and buying in bulk at the one thousand units level will reduce the effect of the cost of shipping per unit. The companies listed above are well established and would be able to provide a reliable source for the parts.

14.2 Bill of Materials

The items in Tables 37 and 38 are what the team has purchased up to the end of Senior Design I. We will continue adding to this table throughout Senior Design II, keeping track of all parts purchased and prices associated with these parts.

Part	**Cost of Single Unit (USD)**	**Quantity**	**Supplier**
Adafruit FONA 3G Cellular Breakout	$ 79.95	1	Adafruit
Slim GSM/Cellular Antenna	$ 2.95	1	Adafruit
External Active GPS Antenna	$ 14.95	1	Adafruit
ADXL377 High-G Triple-Axis Accelerometer	$ 24.95	1	Adafruit
SMA to uFL RF Adapter Cable	$ 3.95	1	Adafruit
1200mAh Lithium Ion Polymer Battery	$ 9.95	1	Adafruit
Breadboard	$ 5.00	1	Adafruit
Arduino Mega 2560 Rev3	$ 28.50	1	Amazon

Table 37: Bill of Materials

Part	**Cost of Single Unit** (USD)	Quantity	**Supplier**
Arduino UNO R3	$ 18.50	1	Amazon
3" Positive Lens	$ 9.39	1	Amazon
3" Square Mirror	$ 4.88	5	Amazon
2" Negative Lens	$ 9.49	1	Amazon
nRF52840 Bluetooth Module	$ 12.99	1	Adafruit
nRF52840 Feather Development Kit	$ 24.99	1	Adafruit
J-Link OB ARM Programmer	$ 5.99	1	Alibaba
Arduino Nano	$ 19.88	1	Amazon
HC-05 Bluetooth Module	$ 8.49	1	Amazon
ADXL335 Triple-Axis Accelerometer	$ 14.95	1	Adafruit
Dupont Wire Kit	$ 6.98	1	Amazon
Freematics OBD-II UART Adapter for Arduino	$ 49.90	1	Freematics
SeeedStudio Grove 0.96" OLED Display	$ 27.71	1	SeeedStudio
Ting Mobile Cellular Service	$ 44.59	1	Ting
Peripheral and Passive Parts for PCB	~$ 33.36	1	Mouser
TOTAL	$ 462.29		

Table 38: Bill of Materials Continued

15.0 Tools

Implementing a project of this scale becomes much easier and more efficient with the aid of tools at our disposal. The College of Engineering and Computer Science provides a myriad of resources to assist with the design and implementation of the Senior Design project. The following section is devoted to the tools our team will take advantage of throughout the course of Senior Design.

15.1 Communication

Clear channels of communication play a pivotal role in staying organized and maintaining the set schedule. The team will be utilizing a few different communication tools to ensure the project stays on track and is completed within the given timeline.

15.1.1 Discord:

Discord is a free VoIP application that provides the ability for users to talk, video call, chat, share files, screen share, and much more. Many of us already use Discord as a primary communication platform. Discord contains the functionality needed to archive group discussions if referencing them is needed. This will be the preferred communication platform moving forward

15.1.2 Text Messaging:

Text messaging has been the group’s initial form of communication since the beginning of Senior Design I. Sharing content across MMS is difficult as it compresses images and video to the point that it is not very usable. We will be shifting to utilize Discord moving forward.

15.2 File Preservation and Collaboration

Technology is not perfect, and accidents often happen when they are least expected. In an event where a computer crashes and data are lost, we want to ensure all our hard work is backed up and protected. We will be utilizing Google

Drive to store all the documents and related materials to the project. Google Drive is free and provides us with enough storage to preserve all our data.

Additionally, we will be using Microsoft OneDrive and SharePoint as tools to collaborate on documentation relating to the project. These programs utilize cloud stored data, so every file we create will be backed up and accessible across platforms. OneDrive and SharePoint includes features for multiple users editing the same document at the same time. This document was created using SharePoint as the host of the file. Each member could access, read, and modify the file concurrently.

15.3 Other Software

15.3.1 LucidChart:

LucidChart is a web-based application that allows users to collaborate in creating charts, diagrams, and other related tools. Some of the figures in this document were created using LucidChart. LucidChart is a paid service but provides student licenses to users that apply with their University’s domain email address.

15.3.2 Multisim

The main circuit simulation that was used was a program called Multisim. It is a schematic capture and simulation program. What makes using this ideal is that it is an industry-standard SPICE simulation with an interactive schematic environment to help debug and run basic testing on circuitry. This saved money by being able to accurately simulate circuits and not have to waste parts by building and debugging incorrect circuits.

15.3.3 DipTrace

DipTrace is a software tool that allows the creation of PCB Schematics to be designed efficiently. We were originally planning to use Eagle PCB Design but DipTrace seems to be more feature-rich. DipTrace includes a 3D module that will produce a mockup of what the final design will look like fully implemented. We have acquired a lite version of DipTrace for free that includes 300 pins and 2 signal layers.

15.0 Results

Though a trite saying, every project should be planned and executed to expect the unexpected. This was certainly true for our Senior Design Project. When we set out to order our PCB components toward the end of January, China had just begun the process of reducing product output and availability due to the COVID-19 Coronavirus. The order for our first produced PCB was delayed by a few days due to the reduced production and shipping capabilities of JLCPCB. After this delay, we decided to order an excess of materials to reduce the downtime if product availability continued to be an issue.

Relatively unaffected by the Coronavirus was our development environment. We were able to source all of the parts discussed in the Research portion of this paper, including the Arduino Mega 2560, SIM808, HC-05, ADXL335, Freematics OBDII UART Adapter, and the Grove OLED Display. Once we obtained all of these components, we began to build an environment mimicked what would be capable from our custom PCB implementation.

As the semester continued, it became more apparent that the Coronavirus was going to make an impact on more countries than just China. By the time we ordered our second revision of the PCB, the World Health Organization had declared the Coronavirus a global pandemic, which led to countries, schools, and other organizations closing to mitigate the impact throughout the world. Included in this closure was the University of Central Florida, moving to a more online mode of instruction. This reduced the amount of time that was allocated in the labs for us to work on our PCB. As a result, the previous requirement of a significant custom PCB was dropped by the College of Engineering. We shifted our design to focus solely on the development environment to continue working to implement the Huddy Buddy with the materials we already had acquired.

The Arduino Mega 2560 that we had purchased during the initial research stage became the hub of operations around the Huddy Buddy. We used it to combine all of the other components that we had designed to implement the functions of the Huddy Buddy. One of the benefits of having all of our components on the PCB is that it minimizes the footprint that each component takes. We wanted to keep this same design philosophy even in the development environment. To accomplish this, we attached our Arduino Mega 2560, SIM808, breadboard, and backup battery to a small sheet of plexiglass using the mounting holes located on the development boards. Once the components were affixed to the plexiglass, we routed wires between the components and the breadboard to enable the required functionality. After our development environment was secured, we tested it in a team member’s vehicle and proved that the system could remain portable enough to transfer to any vehicle regardless of make or model. An image of our development environment is given in Figure 43 below.

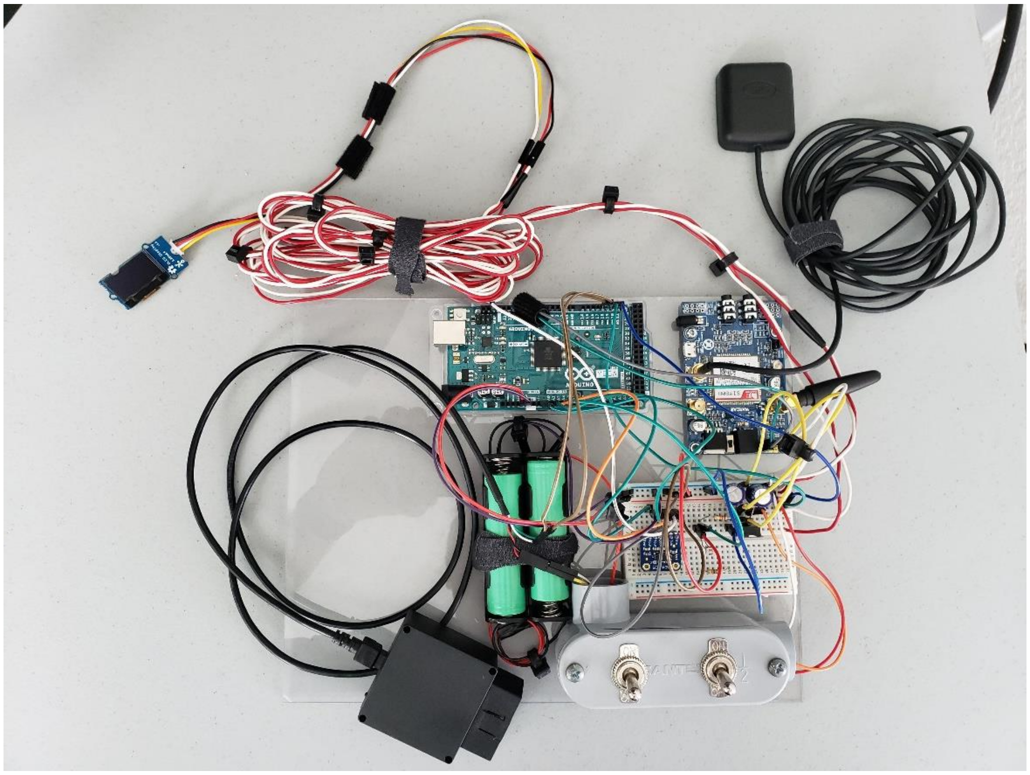


Figure 45: Completed Development Environment for the Huddy Buddy

At the center of the plexiglass is the Arduino Mega 2560 which connects to the SIM808 in the top right, breadboard in the bottom right, OBDII Adapter in the bottom left, and Grove OLED Display in the top left. The breadboard houses the ADXL335 Accelerometer as well as the converter needed to take the 7.3 volts provided by the batteries down to 5 volts. We wanted to allow the user to turn the device off when it is not needed, since the batteries and OBDII port provide continuous power. We added two toggle switches to our design, the left switch controls power from the OBDII port and the right switch controls power from the batteries.

An unexpected result we discovered during our testing phases was that the SIM808 module would suddenly turn off when being powered by the Arduino Mega 2560's 5V power jumper. Further research into the SIM808's documentation led us realize that the SIM808 can instantaneously pull a burst of 3A of current for a very short period. If this current target is not met, the SIM808 will unexpectedly shut off. This led us to power the SIM808 directly from the backup battery unit rather than by the OBDII port or the Arduino module. Further testing confirmed that having the backup batteries powering the SIM808 was much more stable than previous implementations.

We put the development environment to the test, plugging into a vehicle and flipping the power switches on. After all the components were powered, the system booted up, initialized with the vehicle's OBD data, configured the SIM808 and calibrated the current GPS location, each step being announced on the OLED Display. Once the GPS location was calibrated, the system shifted the OLED to display current vehicle speed and fuel economy data. The data that was received was very accurate to what the vehicle was actually reading on its own gauges. To demonstrate the ability of the crash recovery system without actually putting our vehicles or persons in jeopardy, we coded it to trigger with a slight but sudden change in force. Shaking the development environment triggered the emergency response system, indicated by a "CRASH" announcement on the OLED display. Shortly after the emergency recovery system was initiated, a text message was received containing an alert and a link to Google Maps with the Huddy Buddy's GPS coordinates appended to the link. Clicking the link opened Google Maps and placed a pin directly on the coordinates in the text message. The coordinates that the SIM808 provided were within 5 meters of the actual location of the Huddy Buddy. We believe that in the event of an actual crash, the Huddy Buddy would alert an emergency contact of the location of the crash with great accuracy, increasing the likelihood of those affected by the crash to receive needed medical care.

Similar tests were conducted with our Custom Android application. The app's results included two key parts that the Huddy Buddy needed, one being the emergency contact number for the SIM808, and the other being routing information for the OLED display. Since the OLED screen reads information via serial, the app sends data over Bluetooth to the HC-05, and then it is read in via serial by the OLED display. The resulting data sent over always has three lines, the first being a direction the user must go to follow the route, the next street that direction pertains to, as well as the distance between the user's current location and the next point of interest. These three lines are sent on a half second loop, where the main value that changes is the distance variable. In the case the user makes a wrong turn or is presented with an unexpected detour, the fact that the routing information is called every half second allows the information sent to the display to be updated quickly.

Finally, we tested our Display Unit that houses the OLED display, the two lenses, and the mirror. Because the COVID-19 pandemic did not allow for us to have the

Grove OLED display and the display unit in the same location, we instead partially disassembled the display unit to fit a group member's phone in to use as the "display". With this display unit setup, as well as a sheet of plexiglass acting as the "windshield", we were able to demonstrate that the display unit does indeed allow for an image to be clearly visible and focused at infinity. Because the phone used does not have a particularly bright display, this demonstration was done in a dimly lit room. While focused on a distant object, the image from the Display Unit was indeed visible, as shown in Figure 44 below.

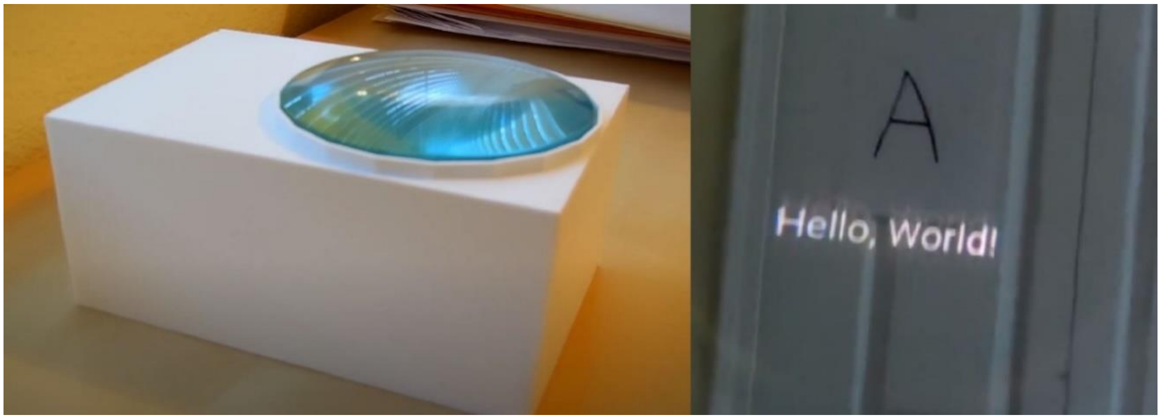


Figure 46: Display Enclosure and Resultant Image

16.0 Project Operation

The following documentation is a guide for operating the project. Please follow each step in succession as the system must follow the correct order to work properly.

1. Place OLED into the Display Unit by feeding the OLED's wires through the hole on the left side of the Unit, and then securing the display with the connector.
2. Place a non-slip mat onto the dashboard of the vehicle, such that it does not impede the view of the road but still maintains the display area in the driver's field of view.
3. Place the development environment in the vehicle in a place that is suitable to fit, such as under the driver's seat.
4. Plug OBD-II Adapter into the vehicle's OBDII port, typically located underneath the steering column of the vehicle. A blue light will turn on, verifying the adapter is receiving power.

    a. If the blue light does not turn on when the adapter is plugged into the OBD-II port, the vehicle may not provide power to the OBD-II when it is turned off. Turning the car on should begin to power the OBD-II port and the light should turn on.

5. Flip the power switches to ON, starting with the left main power switch and then the right SIM power switch.
    a. Confirm that power is on by locating LEDs on the Arduino Mega and SIM808 boards, as well as information displayed on the HUD. If LEDs are not on, check all power connections and try again.
6. Press the ON button located next to the SMS Antenna on the SIM808 Board. The SIM808 board will begin to initialize, notated by quick flashing LEDs. **NOTE:** Please ensure you are in an area with adequate cellular service to ensure proper operation
    a. There are 4 Status LEDs on the SIM808 Module.
        i. Bottom by GSM Antenna – Confirms board is powered
        ii. Top Left – Cellular Status LED
            1. If the Cellular Status LED is blinking quickly, the SIM Module is initialized and looking for a cellular network connection. When the cellular connection is received, the Cellular Status LED will slow to blink once every two seconds. If this does not happen, ensure that there is adequate cellular service in the area and reboot the module by holding the power button on the sim module twice, each for 1 second.
        iii. Top Center – Bluetooth Status LED – Not used in this project
        iv. Top Right – GPS Calibration Status LED
            1. See step X for information about this LED
7. At this stage, the HUD should display an image stating, "**Powering on… Initializing OBD"**. If this image appears for more than 30 seconds, do the following:
    a. Power cycle the SIM808 Module, IMMEDIATELY followed by pressing the RESET button on the bottom of the Arduino Mega module. This sometimes occurs when the SIM and OBDII data are conflicting and preventing the system from advancing.
8. Once the OBDII data is initialized, the HUD will print each system being initialized, moving to SIM808, Accelerometer, and then the GPS calibration.
9. Ensure the Active GPS Antenna is located in an area that is unobstructed from the sky to provide the fastest calibration.
10. Once the system has calibrated the current GPS location, the HUD will display a screen containing the following elements:

   a. Top Left – Current Vehicle Speed in miles per hour
   b. Bottom Left – Current Vehicle Fuel Economy in miles per gallon
   c. Right Side of HUD – Current leg of navigation containing miles to next turn, direction of next turn, and street name of next turn

11. At this stage, the system is fully calibrated and ready to go. The vehicle may be driven as normal.
12. The Android Application will continue to track the distance to the next waypoint and update this information on the right side of the HUD. Once the current waypoint has been reached, the system will update to show the distance, direction, and street name of the next waypoint. This will occur at each waypoint to the destination waypoint.
13. If the system is calibrated correctly and there was to be a large force to the vehicle, such as a car accident, an emergency response subroutine will be executed.
    a. The Arduino Mega 2560 will grab the current GPS location of the Huddy Buddy using the calibrated GPS signal.
    b. The Mega 2560 will use the SIM808 to send a text message to the designated emergency contact, containing the current GPS coordinates appended to a Google Maps link which will pinpoint the exact location to the vehicle within 5 meters.
    c. The emergency contact can use this link to be guided directly to the location and assist where needed.

17.0 Conclusion

Over the last two semesters, we have researched, designed, and built a Heads-Up display for a vehicle. The Heads-Up Display was designed to increase driver awareness and safety when driving. Information is displayed to the driver in a fashion that does not impede with the driver’s view. The application we created allows the driver to choose a destination. The device will contact the Mapbox API and provide an efficient route to the destination. The application then calculates the driver’s current location using the API and compares this location to waypoints along the route. The display unit provides the user with a visual representation of the route and guide them to the destination with turn-by-turn directions.

The device was designed to reduce the likelihood for a driver to be distracted while driving. However, there is always the possibility for other drivers to be distracted. If a crash occurs, we have an impact detection system that will trigger when a strong or sudden impact is registered. This system sends an automated SMS text message to the user’s designated emergency contact with information regarding the user’s last reported location at the time of the impact. We hope that this system facilitates the ability for those impacted by the crash to get the help they need as soon as possible.

We are confident that this Senior Design Project has taught us countless lessons about design, implementation, project management, and time management. Creating an Android application gave us extensive experience with mobile application software development. Designing our own PCB taught us about schematic design and hardware optimization. Implementing the display onto the windshield of the vehicle taught us about light reflectivity and principles of modern Heads-Up Displays. Using a cellular module gave us the opportunity to learn about the formatting that mobile phone companies use to send SMS text messages, phone calls, and data usage. Implementing the GPS tracking features into the HUD device gave us the chance to learn about how the National Marine Electronics Association formats GPS data, enabling us to use tracking methods for other applications. Building our own microcontroller platform gave us experience with developing an embedded design with specific requirements. Accessing navigational information from the Mapbox API's gave us insight into how mapping platforms are able to route vehicles on the most efficient path taking traffic, road closures, and other hazards into account. The team has learned an extensive amount about Bluetooth devices to transfer data between the HUD Device and the mobile phone. Using an OBD-II adapter taught us about getting sensor data that is pertinent to the user such as speed, RPM, and ambient temperature to name a few. This will teach basic car integration and how to read sensor data in automobiles. These actions are accessed through ELM327 AT command-set All the topics discussed in this paper gave the team a wealth of knowledge that will be valuable for our careers ahead.

The device mentioned above was designed and realized utilizing the knowledge that each member of the team has accumulated throughout their time at the University of Central Florida. This device has provided us with challenging and exciting design decisions, while promoting a positive effect on our community by producing a project focused on increased safety and awareness. Additionally, we hope to continue use this project well after we have all graduated. We would like to continue to support its development and improve upon its design as technology becomes more advanced. The skills we learned from this project will carry with us for many years to come.

18.0 References

"Digital Micromirror Device." *Wikipedia*, Wikimedia Foundation, 29 July 2019, https://en.wikipedia.org/wiki/Digital_micromirror_device

"DLP Products – Getting Started." *DLP Getting Started* | *DLP Products* | *TI.com*, http://www.ti.com/dlp-chip/getting-started.html.

"nRF52840 Info Center" https://infocenter.nordicsemi.com/index.jsp?topic=%2Fstruct_nrf52%2Fstruct%2Fnrf52840.html&cp=4_0

"olikraus/u8g2." https://github.com/olikraus/u8g2

"stanelyhuangyc/ArduinoOBD" https://github.com/stanleyhuangyc/ArduinoOBD

"U Drive. U Text. U Pay." *NHTSA*, 8 May 2019, https://www.nhtsa.gov/risky-driving/distracted-driving.

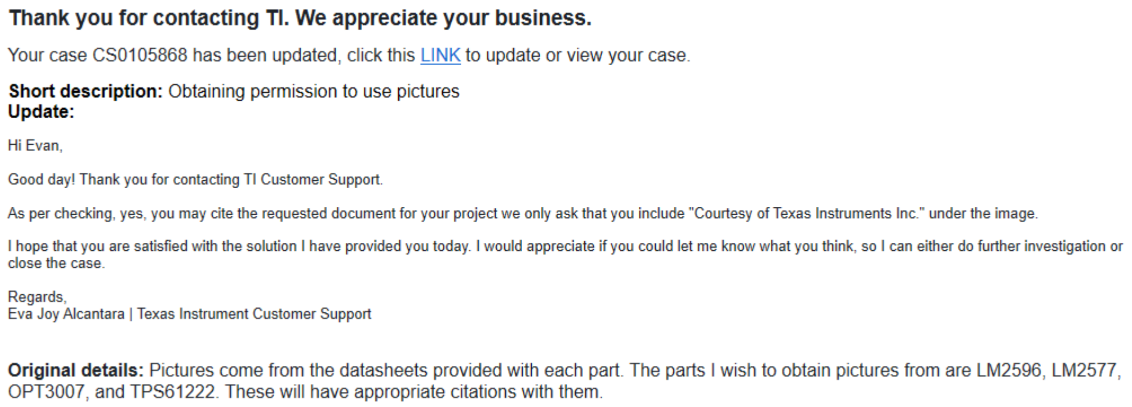
Czernia, Dominik. "Car Crash Calculator." *Omni*, Omni Calculator, 22 Feb. 2019, https://www.omnicalculator.com/physics/car-crash-force.

[4] "Transport Canada Guidelines to Limit Distraction from Visual Displays in Vehicles." *Transport Canada*, Government of Canada, 27 February 2019, https://www.tc.gc.ca/en/services/road/stay-safe-when-driving/guidelines-limit-distraction-visual-displays-vehicles.html

[9] https://www.adafruit.com/product/4078

https://www.windmill.co.uk/fuel.html

The following is an email approval for use of schematics by Texas Instruments, Inc.



Thank you for contacting TI. We appreciate your business.

Your case CS0105868 has been updated, click this LINK to update or view your case.

Short description: Obtaining permission to use pictures
Update:

Hi Evan,

Good day! Thank you for contacting TI Customer Support.

As per checking, yes, you may cite the requested document for your project we only ask that you include "Courtesy of Texas Instruments Inc." under the image.

I hope that you are satisfied with the solution I have provided you today. I would appreciate if you could let me know what you think, so I can either do further investigation or close the case.

Regards,
Eva Joy Alcantara | Texas Instrument Customer Support

Original details: Pictures come from the datasheets provided with each part. The parts I wish to obtain pictures from are LM2596, LM2577, OPT3007, and TPS61222. These will have appropriate citations with them.

19.0 Appendix A – Final Arduino Huddy Buddy Code

```
/*
 * Huddy Buddy
 * Group 6
 * Senior Design II - Spring 2020
 * University of Central Florida
 *
 *
 *
 * This Sketch is to be used with the Huddy Buddy,
 * a Heads Up Display with advanced features to
 * increase driver awareness and safety.
 *
 * *************************************
 * *            INSTRUCTIONS           *
 * *************************************
 *
 * To use, plug in the OBDII adapter into the vehicle.
 * Flip on the switches for Main Power and SIM808 Power,
 * then, press the ON button for the SIM808 Module.
 * The module will start and a boot screen will appear.
 * If "Initializing OBD" is stuck, power cycle the SIM808
 * module by pressing the PWR button, and then IMMEDIATELY
 * press the reset button on the Arduino Mega 2560. Continue
 * this process until the system boots properly.
 *
 *
 *
 */

// Initialize FONA libraries
#include <SoftwareSerial.h>
// Setting up Fona Library
#include "Adafruit_FONA.h"
//SoftwareSerial sim808(10,11);
#define FONA_RX 11
#define FONA_TX 10
#define FONA_RST 4 // not used
SoftwareSerial fonaSS = SoftwareSerial(FONA_TX, FONA_RX);
SoftwareSerial *fonaSerial = &fonaSS;
Adafruit_FONA fona = Adafruit_FONA(FONA_RST);

// Initialize OBD2 Libraries
#include <OBD2UART.h>
#define mySerial Serial1
COBD obd;

// Initialize u8g2 OLED libraries
#include <Arduino.h>
#include <U8g2lib.h>

#ifdef U8X8_HAVE_HW_SPI
```

```
#include <SPI.h>
#endif
#ifdef U8X8_HAVE_HW_I2C
#include <Wire.h>
#endif

// Initialize Huddy Buddy Logo
#define hb_logo_width 48
#define hb_logo_height 48
static unsigned char hb_logo_bits[] = {
  0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
0xFF,
  0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
0xFF,
  0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
0xFF,
  0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0x5F, 0x52, 0xAF, 0x49, 0xF2,
0xFF,
  0x0F, 0x00, 0x07, 0x00, 0x80, 0xFF, 0x2F, 0x80, 0x1F, 0x40, 0x00,
0xFF,
  0x7F, 0xE0, 0x7F, 0xE0, 0x07, 0xFE, 0x7F, 0xE0, 0x7F, 0xF0, 0x0F,
0xFC,
  0x7F, 0xE0, 0x7F, 0xE0, 0x0F, 0xFC, 0x7F, 0xE0, 0x3F, 0xF0, 0x1F,
0xFC,
  0x7F, 0xE0, 0x7F, 0xE0, 0x0F, 0xFC, 0x7F, 0xE0, 0x7F, 0xF0, 0x1F,
0xFC,
  0xFF, 0xE0, 0x7F, 0xE0, 0x1F, 0xFC, 0x7F, 0xE0, 0x3F, 0xF0, 0x0F,
0xFC,
  0x7F, 0xE0, 0x7F, 0xE0, 0x0F, 0xFE, 0x7F, 0xE0, 0x7F, 0xF0, 0x0F,
0xFE,
  0x7F, 0xE0, 0x7F, 0xE0, 0x07, 0xFF, 0x7F, 0x00, 0x04, 0xA0, 0xC1,
0xFF,
  0x7F, 0x00, 0x00, 0x00, 0xE0, 0xFF, 0x7F, 0x00, 0x00, 0x40, 0x80,
0xFF,
  0x7F, 0xE0, 0x7F, 0xE0, 0x07, 0xFE, 0x7F, 0xE0, 0x7F, 0xF0, 0x0F,
0xFC,
  0x7F, 0xE0, 0x7F, 0xE0, 0x1F, 0xF8, 0x7F, 0xE0, 0x7F, 0xF0, 0x3F,
0xF8,
  0x7F, 0xE0, 0x7F, 0xE0, 0x1F, 0xF8, 0x7F, 0xE0, 0x3F, 0xF0, 0x3F,
0xF0,
  0x7F, 0xE0, 0x7F, 0xE0, 0x3F, 0xF0, 0x7F, 0xE0, 0x7F, 0xF0, 0x3F,
0xF8,
  0x7F, 0xE0, 0x7F, 0xE0, 0x3F, 0xF0, 0x7F, 0xE0, 0x7F, 0xF0, 0x1F,
0xF8,
  0x7F, 0xE0, 0x7F, 0xE0, 0x1F, 0xF8, 0x7F, 0xE0, 0x7F, 0xE0, 0x0F,
0xFC,
  0x7F, 0xC0, 0x3F, 0xE0, 0x07, 0xFE, 0x0F, 0x00, 0x07, 0x00, 0x00,
0xFF,
  0x0F, 0x00, 0x0F, 0x00, 0xE0, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
0xFF,
  0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
0xFF,
  0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
0xFF,
  0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
0xFF,
  0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
0xFF,
  };

// Call OLED with the specified chipset. Uncomment second definition
for mirrored display.
U8G2_SSD1306_128X64_NONAME_F_HW_I2C u8g2(U8G2_R0, /* reset=*/
U8X8_PIN_NONE);
//U8G2_SSD1306_128X64_NONAME_F_HW_I2C u8g2(U8G2_MIRROR, /* reset=*/
U8X8_PIN_NONE);

// Initialize Accelerometer Values
// The Raw range was calibrated to be accurate values based on
// device at rest.
const int xInput = A0;
const int yInput = A1;
const int zInput = A2;
const int buttonPin = 2;

//// Raw Ranges:
//// initialize to mid-range and allow calibration to
//// find the minimum and maximum for each axis
//int xRawMin = 512;
//int xRawMax = 512;
//
//int yRawMin = 512;
//int yRawMax = 512;
//
//int zRawMin = 512;
//int zRawMax = 512;

int xRawMin = 409;
int xRawMax = 613;

int yRawMin = 403;
int yRawMax = 606;

int zRawMin = 425;
int zRawMax = 629;

// Take multiple samples to reduce noise
const int sampleSize = 10;

// Init variables for sim text
int time = millis();
int counter = 0;
float latitude, longitude;
char clat[10];
char clong[10];
boolean gps_success;

// OBD variable init
int value, maxSpeed = 0, curSpeed, speedMPH, maf, currentMPG;
float mafVal = 0, curMPG = 0, roundedMPG = 0;
String strMPG;

const int testPin = 2;
```

```
//char phone_no[] = "7724537459";
//String data[5];
//#define DEBUG true
//String state, timegps, latitude = "28.6013", longitude = "-81.1984";
boolean calibrated = false;
bool sentMessage = false, crashDetected = false;
int directionSelector = 0;
float distanceFloat = 2.0;
int photocellPin = 4, photocellReading;
void setup() {

  // Start the OLED
  u8g2.begin();
  u8g2.setPowerSave(0);
  u8g2.clearBuffer();          // clear the internal memory
//  u8g2.setFont(u8g2_font_victoriamedium8_8r);
//  u8g2.setCursor(30,30);
//  u8g2.print("Powering on...");
//  u8g2.setCursor(50,30);
//  u8g2.print("Please wait :)");

  // Draw Huddy Buddy Logo and version number, 5 sec delay
  u8g2.drawXBM(0,0,hb_logo_width,hb_logo_height,hb_logo_bits);
  u8g2.setFont(u8g2_font_t0_12_tf);
  u8g2.setCursor(58,10);
  u8g2.print("Huddy Buddy");
  u8g2.setCursor(58,22);
  u8g2.print("Version 1.0");
  u8g2.sendBuffer();
  delay(5000);

  // Start power on process with status on OLED
  u8g2.clearBuffer();
  u8g2.setFont(u8g2_font_t0_12_tf);
  u8g2.setCursor(20,20);
  u8g2.print("Powering On...");
  u8g2.setCursor(15,32);
  u8g2.print("Initializing OBD");
  u8g2.sendBuffer();

  // Begin connection to OBD with mySerial.
  // do while until obd is initialized
  mySerial.begin(115200);
  obd.begin();
  do
  {
    mySerial.print("Init...");
  }
  while(!obd.init());

  // Begin connection to SIM808 with the FONA Library
  // Used example sketch from FonaTEST
  u8g2.clearBuffer();
  u8g2.setFont(u8g2_font_t0_12_tf);
  u8g2.setCursor(20,20);
  u8g2.print("Powering On...");
  u8g2.setCursor(0,32);
  u8g2.print("Initializing SIM808");
  u8g2.sendBuffer();

  Serial.begin(115200);
  Serial.println(F("Adafruit FONA 808 & 3G GPS demo"));
  Serial.println(F("Initializing FONA... (May take a few seconds)"));
  fonaSerial->begin(4800);
  if (! fona.begin(*fonaSerial)) {
    Serial.println(F("Couldn't find FONA"));
    while(1);
  }
  Serial.println(F("FONA is OK"));
  // Try to enable GPRS

  u8g2.clearBuffer();
  u8g2.setFont(u8g2_font_t0_12_tf);
  u8g2.setCursor(20,20);
  u8g2.print("Powering On...");
  u8g2.setCursor(15,32);
  u8g2.print("Enabling GPS");
  u8g2.sendBuffer();

  Serial.println(F("Enabling GPS..."));
  fona.enableGPS(true);

  // Begin Accelerometer
  u8g2.clearBuffer();
  u8g2.setFont(u8g2_font_t0_12_tf);
  u8g2.setCursor(20,20);
  u8g2.print("Powering On...");
  u8g2.setCursor(15,32);
  u8g2.print("Initizalizing Accelerometer");
  u8g2.sendBuffer();

  analogReference(EXTERNAL);

}

void loop() {

  // Initialize local loop vars
  float latitude, longitude;
  char clat[10];
  char clong[10];
  char message[140];
  char distanceChar[10];
  // We need to be calibrated to the GPS signal before continuing.
  // When calibrated, the system will enter the main program.
  // Else, it will wait until a connection is made.

  if (calibrated) // was if(gps_success)
  {


    // Every 50 milliseconds, get data from OBD
    // and print to OLED.
    if(time % 50 == 0)
    {
      // Set up display frame with Constant text
      // Grab brightness from photoDiode
      photocellReading = analogRead(photocellPin);

      u8g2.clearBuffer();          // clear the internal memory

      // uncomment this section for the photodiode
//      if(photocellReading > 600)
//      {
//        u8g2.setContrast(255);
//      }
//      else if(photocellReading < 600 && photocellReading > 400)
//      {
//        u8g2.setContrast(200);
//      }
//      else if(photocellReading < 400)
//      {
//        u8g2.setContrast(150);
//      }


      u8g2.setFont(u8g2_font_victoriamedium8_8r);
      u8g2.setCursor(33,20);
      u8g2.print("mph");
      u8g2.setFont(u8g2_font_open_iconic_arrow_2x_t);
          u8g2.setCursor(80,20);
          u8g2.write(65);

          u8g2.setFont(u8g2_font_victoriamedium8_8r);
          u8g2.setCursor(70,30);
          u8g2.print("2.1 mi");
          u8g2.setCursor(70,40);
          u8g2.print("Gemini");
          u8g2.setCursor(70,50);
          u8g2.print("Blvd");

      // Get speed from OBD, convert to MPH, and display on OLED.
      // Added logic to make displaying more fluid by taking values
      // less than 10 and making the single digit appear as a right
      // bias.
      if(obd.readPID(PID_SPEED, curSpeed))
      {
        speedMPH = curSpeed * 0.621371; // converting kph to mph

        u8g2.setFont(u8g2_font_courB18_tf); // choose a suitable font
        if(speedMPH < 10)
        {
          u8g2.setCursor(15,20);
          u8g2.print(speedMPH);
        }

        else if (speedMPH > 9)
        {
          u8g2.setCursor(0,20);
          u8g2.print(speedMPH);
        }
      }

      // Get Mass Air Flow value from OBD, which is used for MPG
calculation, then
      // print MPG on OLED.
      if(obd.readPID(PID_MAF_FLOW, maf))
      {
        // currentMPG is derived from the MAF and speed of the vehicle,
as well as conversion values
        // given by
https://github.com/oesmith/obdgpslogger/blob/master/doc/mpg-calculation
        // NOTE: To get a realistic factor, divide again by 100

        currentMPG = ((14.7 * 6.17 * 454 * curSpeed * 0.621371) / (3600
* maf / 100)) / 100;
        u8g2.setFont(u8g2_font_courB14_tf);
        if(currentMPG < 10)
        {
          u8g2.setCursor(20,40);
          u8g2.print(currentMPG);
        }

        else if (currentMPG > 9 && currentMPG < 100)
        {
          u8g2.setCursor(10,40);
          u8g2.print(currentMPG);
        }
        else if (currentMPG > 99)
        {
          u8g2.setCursor(0,40);
          u8g2.print(currentMPG);
        }
        u8g2.setFont(u8g2_font_victoriamedium8_8r);
        u8g2.setCursor(33,40);
        u8g2.print("mpg");
      }

      u8g2.sendBuffer();
    }


    // Read values from Accelerometer
    int xRaw = ReadAxis(xInput);
    int yRaw = ReadAxis(yInput);
    int zRaw = ReadAxis(zInput);

    // Every 200 milliseconds, check accelerometer for high g forces.
    if(time % 200 == 0)
    {

      // Convert raw values to 'milli-Gs"
      long xScaled = map(xRaw, xRawMin, xRawMax, -1000, 1000);
      long yScaled = map(yRaw, yRawMin, yRawMax, -1000, 1000);
      long zScaled = map(zRaw, zRawMin, zRawMax, -1000, 1000);

      // re-scale to fractional Gs
      float xAccel = xScaled / 1000.0;
      float yAccel = yScaled / 1000.0;
      float zAccel = zScaled / 1000.0;

      // If the accelerometer reads value OVER or UNDER 1.5g from any
axis, enter emergency routine
      if(zAccel > 1.5 || zAccel < -1.5 || yAccel > 1.5 || yAccel < -1.5
|| xAccel > 1.5 || xAccel < -1.5 )
      {
        // Get current GPS Coordinates from SIM808
        fona.getGPS(&latitude, &longitude);

        // Convert float lat and long to char array to print into
string
        dtostrf(latitude, 4, 4, clat);
        dtostrf(longitude, 4, 4, clong);

        // Set up full message string with char array coords at the end
of a google maps link.
        sprintf(message, "ALERT: Huddy Buddy detected a crash. Visit
Google Maps link for location: http://maps.google.com/maps?q=%s,%s",
clat, clong);
        // Declare number to send emergency text
        char number[21] = "7724537459";
        // Send the text message to the designated number
        fona.sendSMS(number, message);
        // Print crash on OLED
        u8g2.clearBuffer();
        u8g2.setFont(u8g2_font_courB18_tf);
        u8g2.setCursor(15,20);
        u8g2.print("CRASH");
        u8g2.sendBuffer();
        // Delay 30 seconds
        delay(30000);

      }
    }
  }
  else
  {
    // GPS is not calibrated. Print status on OLED and check for GPS
connection.
    // If no connection is made, delay 5 seconds.
    u8g2.clearBuffer();
    u8g2.setFont(u8g2_font_t0_12_tf);
    u8g2.setCursor(20,20);
    u8g2.print("Powering On...");
    u8g2.setCursor(15,32);
    u8g2.print("Calibrating GPS");
    u8g2.sendBuffer();
    if(gps_success = fona.getGPS(&latitude, &longitude)) // COULD
RETURN INFINITE FALSE
    {
      calibrated = true;
      Serial.print("GPS lat:");
      Serial.println(latitude, 6);
      Serial.print("GPS long:");
      Serial.println(longitude, 6);
    }
    else
    {
      delay(5000);
    }
  }
}

//
// Read "sampleSize" samples and report the average
//
int ReadAxis(int axisPin)
{
  long reading = 0;
  analogRead(axisPin);
  delay(1);
  for (int i = 0; i < sampleSize; i++)
  {
    reading += analogRead(axisPin);
  }
  return reading/sampleSize;
}

//
// Find the extreme raw readings from each axis
//
void AutoCalibrate(int xRaw, int yRaw, int zRaw)
{
  Serial.println("Calibrate");
  if (xRaw < xRawMin)
  {
    xRawMin = xRaw;
  }
  if (xRaw > xRawMax)
  {
    xRawMax = xRaw;
  }

  if (yRaw < yRawMin)
  {
    yRawMin = yRaw;
  }
  if (yRaw > yRawMax)
  {
    yRawMax = yRaw;
  }

  if (zRaw < zRawMin)
  {
    zRawMin = zRaw;
  }
  if (zRaw > zRawMax)
  {
    zRawMax = zRaw;
  }
}
```