BLUETOOTH SPEAKER WITH MOTION-BASED AUTOMATED VOLUME ADJUSTMENT

ECE 445: Senior Design

Chirag Kikkeri
Raj Pulugurtha
Dhruv Vishwanath
TA: Abhisheka Mathur Sekar

Spring 2023

Abstract

The project outlined in this paper addresses the issue of distracted driving, which is a problem that is prevalent across the world. Our product, a Bluetooth speaker with motion-based automated volume adjustment, addresses this problem by reducing the need to constantly change the volume while driving in the car and listening to music. The report documents our project from start to finish, and addresses the design process we went through as we worked towards the project demonstration.

Table of Contents

1. Introduction

1.1 Problem

When driving and listening to music, oftentimes we want to change the volume based on the speed of the vehicle. For example, when moving at higher speeds, drivers will raise the volume to better hear the music, and when stopped at a red light, drivers will lower the volume significantly. This issue is a clear nuisance, but can also present a major safety hazard that takes the user's concentration away from driving to adjusting the volume, especially for drivers who do not use the car sound system.

Furthermore, this problem does not only pertain to drivers. Anyone who is on any type of vehicle, be it a bike, scooter, or skateboard, will suffer from the same problem of not being able to hear their media at high speeds, or the volume being too high at low speeds.

1.2 Solution

Our solution is to create a speaker that will automatically increase and decrease volume based on the speed that the speaker is moving. The speaker will be a portable Bluetooth speaker that the user can take in and out of the vehicle. The speaker system will have two modes: one for when it is moving, and one for when it is stationary. When it is in the stationary mode, the user can increase and decrease volume with buttons. When it is in moving mode, the automatic volume control is turned on, so that the user focuses on driving. The implementation of multiple modes makes the speaker useful for different types of users, making our product relevant for a variety of potential use cases.

2. Design

2.1 Functionality

In order to have a clear goal for what we prioritize in the development of our product, we set three high level requirements for ourselves which were the focus of what we wanted to implement. After the Design Review, we modified our high level requirements to match the feedback that we received from the course staff during this meeting. Based on the modifications, our high level requirements are as follows:

1. Speaker volume changes automatically in driving mode at a relative rate of 15% for an increase/decrease of 5 MPH
2. In stationary mode the speaker is controlled manually using buttons
3. Any bluetooth device should be able to transmit media continuously to the system, resulting in a change of volume in the speaker based on a moving average of the speed at which the speaker is traveling, preventing the volume from constantly changing at minor changes in speed

The first of the three requirements was changed so that rates of change for both the speed and volume were set in a way where we could easily demonstrate our product in a lab environment. Additionally, we made this change so that we could safely record a video of the demonstration of our product while in the car, while still showing that the technology in our product works as designed. The second of our two requirements stayed the same, as we were successfully able to change the speaker between the two modes by using a physical button on the top of the speaker. Lastly, our third requirement was changed slightly as we were told during the Design Review to focus less on the audio quality of our product and more on the technical implementation. This led us to changing the requirement to focus on the moving average element of our software, which made sure that the volume did not spike up and down constantly for sudden changes in speed.

2.2 Subsystem Overview

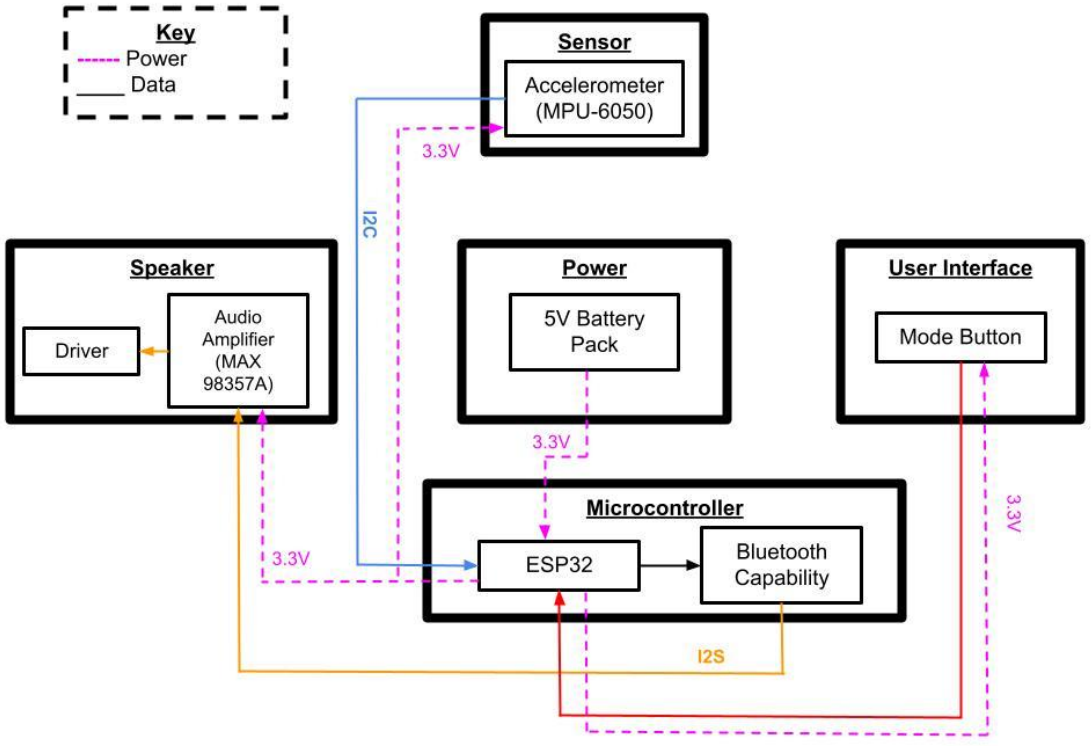

Figure 1: Finalized block diagram for the Bluetooth speaker

As shown in Figure 1 above, the project was broken down into five overall subsystems. The power subsystem contains a 5 Volt battery pack which we attached to our printed circuit board (PCB) in our speaker's enclosure through a Micro-USB port. This allowed our microcontroller to receive power, which is fundamental to the rest of our design. The microcontroller subsystem accounts for both the ESP32 and our Bluetooth connection due to the ESP32 having Bluetooth capabilities, so this subsystem is where the speaker receives the audio from any Bluetooth device that is connected and playing media. The ESP32 also received and interpreted the acceleration data from our accelerometer, as shown by the I2C connection between our sensor subsystem and microcontroller subsystem. This data was then converted into speed values by the code we wrote to the microcontroller using calibration thresholds in addition to the basic physics equation, $V_f = V_0 + at$, where 'V' is velocity (initial and final), 'a' is acceleration, and 't' is change in time. Based on the output of that data in the microcontroller, the volume would be changed and the media received via Bluetooth would be transferred from the microcontroller to the audio amplifier in the speaker subsystem, where it is converted from a digital signal to an analog signal before it is then amplified and sent out through our speaker driver. Lastly, the user interface subsystem has a physical button which can be pressed, through which we can change the mode of the speaker from "stationary" to "moving", and vice versa. A front and rear view of our product with these subsystems is shown below in Figure 2.

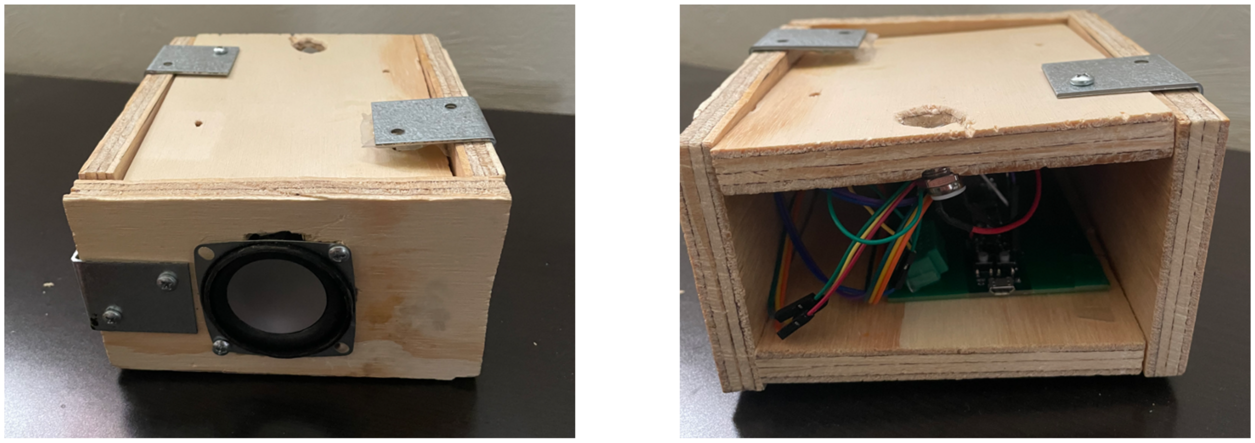
Figure 2: Front and rear facing views of the speaker with the button shown on the top face

2.3 PCB Design

In order to implement the design that we created as shown in Figure 2 above, an essential component was the PCB that we designed. Without the completion of the PCB, we would not have been able to create a compact and portable design as we wanted to, thus making it a valuable part of our project. The schematic for the PCB that we created is shown below in Figure 3.

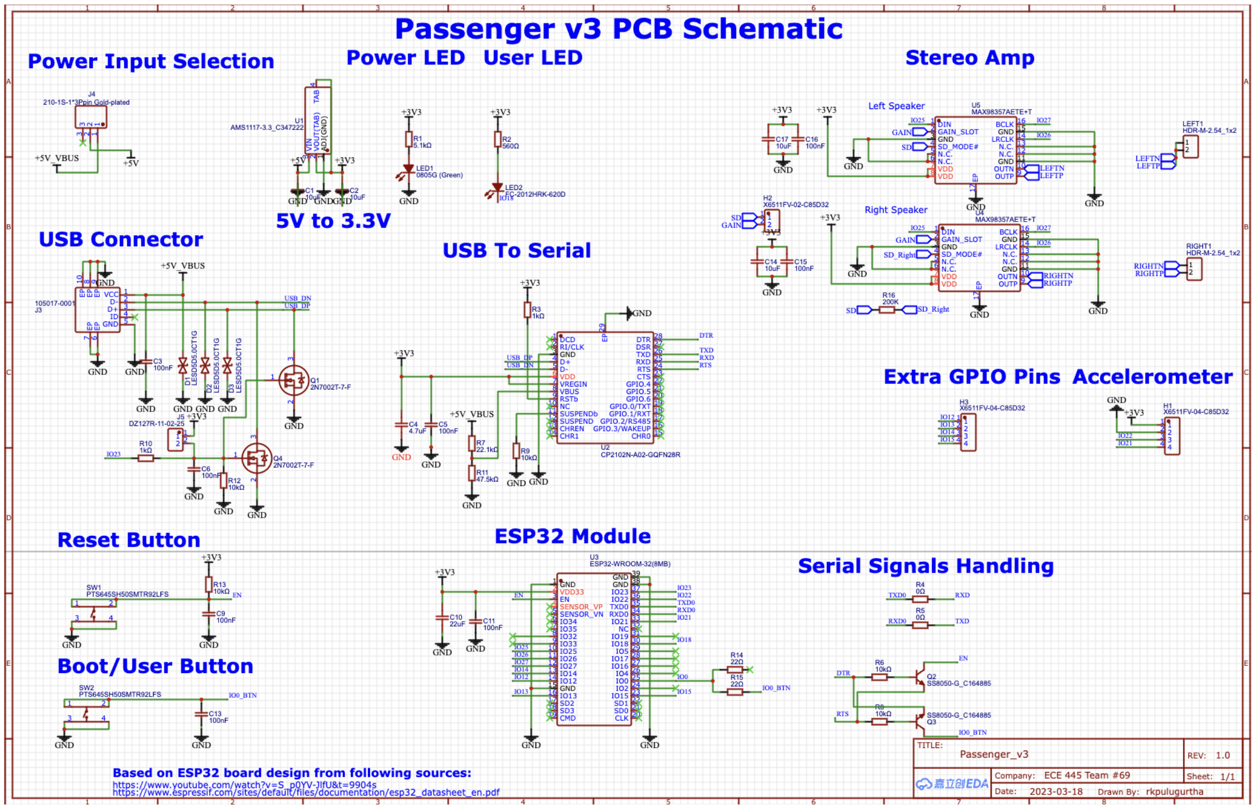

Figure 3: PCB schematic for our Bluetooth speaker

As our group did not have any previous experience with PCB design, taking advantage of external resources was important in the development of the board. Specifically, we relied heavily upon the datasheet for the ESP32 [1] and Youtube videos [2] in order to orient ourselves when designing the board. Though the PCB was able to successfully be integrated into our project, it did not work perfectly as we had planned, and required us to make some dynamic changes to our interfaces in order to produce a working product. Details regarding these modifications are addressed in the following sections discussing specific subsystems.

2.4 Microcontroller and Bluetooth Subsystem Design

In the original design of our project, our goal was to connect a Bluetooth amplifier to our microcontroller and speaker so that audio would be received from our respective device and play back on the speaker. However, we quickly hit a roadblock with this upon realizing that the Bluetooth amplifier we selected, the RN52-I/RM116, did not come with a breakout board. This prevented us from testing our design on the breadboard, and forced us to change how we planned to implement the project. A design alternative we considered was using a potentiometer to vary the volume, but as this complicated our project instead of simplifying it, we searched for other alternatives. Eventually, after doing research we settled upon switching our microcontroller from an ATmega328P to an ESP32, as the latter has built-in Bluetooth functionality, as well as the ability to change volume through the microcontroller itself. Taking advantage of this functionality, we were able to simplify our design and address the problem that came up. Based on existing libraries that corresponded with the ESP32 [3], we were able to write code that allowed volume modification through the microcontroller, based on changes that were made through the data received from the accelerometer. Figure 4 below shows the schematic in our PCB design for the microcontroller.

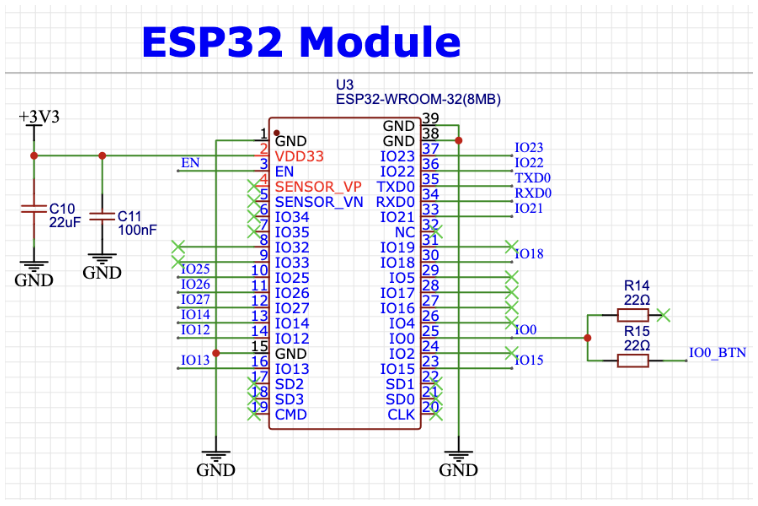

Figure 4: Schematic showing microcontroller connections in PCB

2.5 Sensor Subsystem Design

In order to properly change the volume through our microcontroller, we needed to receive data from the accelerometer so that we could convert it to a velocity. The accelerometer we selected receives data on three axes, and we were able to convert the values to velocity by using the base formula of $V_f = V_0 + at$, as mentioned in section 2.2. However, due to the budget-friendly nature of our accelerometer, we realized when testing it that it did not consistently receive proper values, leading us to add additional thresholds and calibration values to our code in order to receive a more consistent stream of data. Through test-driven development, we were able to verify the accuracy of our acclerometer's values when using it in the car and tracking its data. Additionally, another problem we had with the sensor was its lack of sensitivity when moving at walking speed, which was a hurdle as we would be unable to demonstrate our product during the Final Demo. To address this, we modified the code for the Final Demo so that it took advantage of the Y-axis on the accelerometer, which accounted for the acceleration value of gravity, 9.8 m/s^2. We then changed our formula so that the velocity readings would change based on the tilt of our speaker on the Y-axis, thus allowing us to demonstrate our product during our Final Demo. As shown in Figure 5 below, we attached female connectors to our PCB so that we could plug the accelerometer into it, allowing us to stream its data.

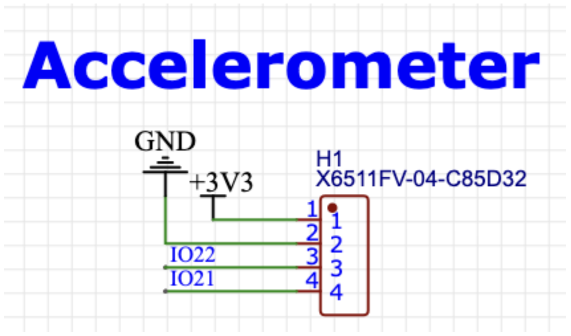

Figure 5: Schematic showing connectors for accelerometer in PCB

2.6 Speaker Subsystem Design

When addressing the speaker design for our PCB, we attempted to implement a stereo sound system with two audio amplifiers so that we could connect two speaker drivers to the PCB. However, upon receiving the PCB, we quickly found out that the stereo functionality failed, as we received no audio output. Our initial thought was that our traces were not wide enough, but after realizing that that should not impact it enough to where the audio output would not be there, we looked for other possible errors. Upon inspecting the design further, we finally realized after already picking an alternative approach that we had swapped two pins in the design for the stereo output. Although we are not sure if switching them back would fix our output, we know for sure that this problem would definitely prevent us from getting an audio output. The alternative approach we picked was to use the extra pins we had for our user interface as pins for a singular speaker driver, as all the pins are General Input Output Pins (GPIO), so we would be able to get output for one speaker driver. This was further simplified by the fact that our audio amplifier has digital to analog converter capabilities, allowing us to use any pins. The drawback to this approach, however, is that we used up pins on the PCB that were for the user interface, thus preventing us from having as many buttons on the user interface as we originally wanted. The design for our stereo output is shown below in Figure 6.

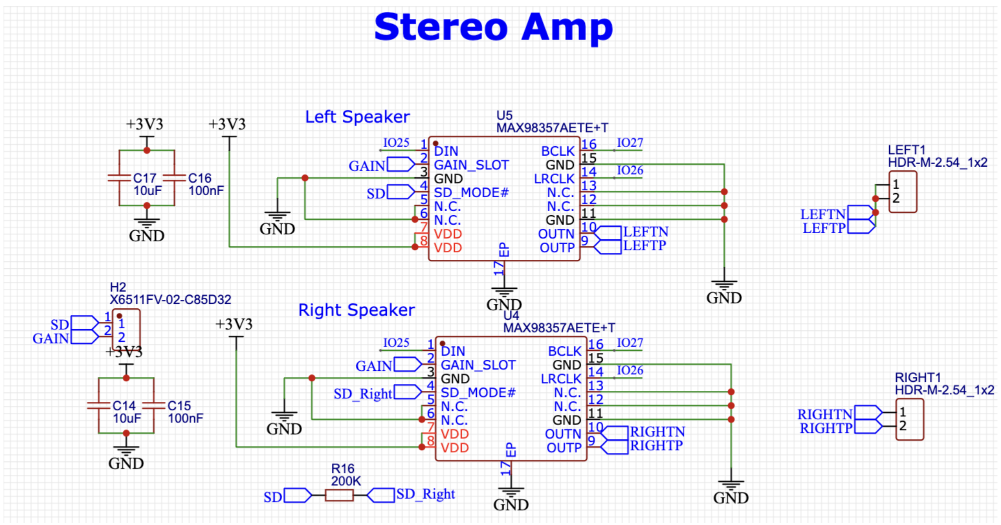

Figure 6: Schematic showing stereo design in PCB

2.7 User Interface Subsystem Design

Our initial plan for this subsystem was to have several buttons and LEDs that would be similar to what you would find on any Bluetooth speaker. However, after needing to produce sound on the PCB and being in a time crunch, we decided to compromise on the buttons and use the extra pins (shown in Figure 7 below) to produce audio output on one speaker driver. In order to do so, we needed three buttons, which left one extra pin available. Since having a mode button was a high level requirement, we decided to use the extra pin to implement that, as we felt it was fundamental to the purpose of our project, and because you can verify what the other buttons could show on the respective Bluetooth device that is connected to the speaker.

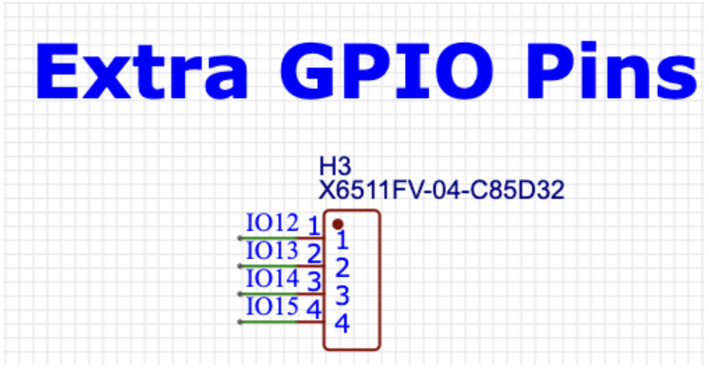

Figure 7: Schematic showing extra pins for UI design in PCB

2.8 Power Subsystem Design

The last remaining subsystem that we needed to address is regarding power, as we needed a way to make our speaker work. To implement this, we added a voltage regulator to our PCB design, as well as a Micro-USB port that we could connect our power source to. The source we chose was a 5 Volt battery pack, which can be recharged via USB-C. Figure 8 below shows the design for our Micro-USB port, which allowed us to receive power from the battery pack.

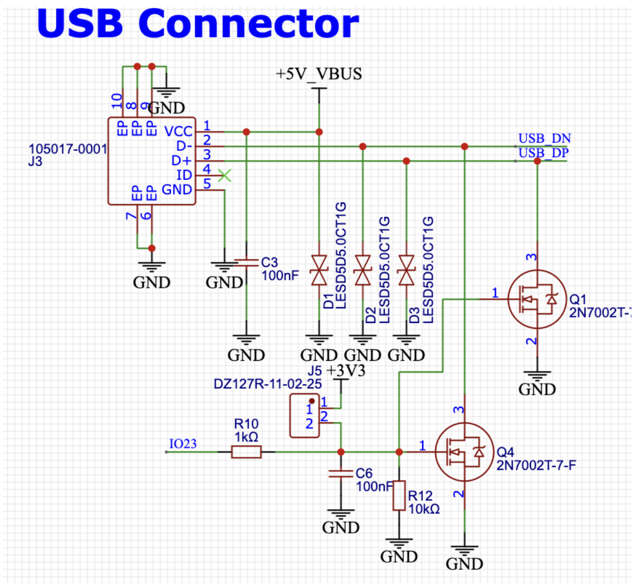

Figure 8: Schematic showing design for USB connector

3. Verifications

3.1 Power Subsystem

Requirements	**Verification**
Power supply needs to be able to steadily supply 3.3V ± 5% to the microcontroller.	Using a voltage regulator, we can make sure that there is a constant 3.3V supplied to the circuit. We can use an oscilloscope and multimeter to measure both current and voltage and check that it's in our desired range.
The speaker should be able to play music at 80 dB for 10 hours, which the specification for the Beats pill speaker [4]	We can test this requirement by connecting our speaker to a Bluetooth device and playing music at an average volume level for an extended period of time.

Table 1: Power Subsystem Requirements and Verifications from Design Document

For the first requirement, we used a voltage regulator in the circuit to make sure that we were getting 5 Volts from the battery pack to the PCB and the ESP32. We used an oscilloscope to confirm that we were getting within the expected 5% tolerance. We changed the requirement from 3.3 Volts to 5 Volts because we changed the microcontroller, and needed 5 Volts for other components in our design. The second requirement we simply tested by using our speaker for an extended period of time on battery power. Although we did not reach ten hours of playback, this was dependent on our battery pack and could be easily addressed by buying a more durable battery.

3.2 Microcontroller and Bluetooth Subsystem

Requirements	**Verification**
Any device with bluetooth functionality should be able to connect to the speaker.	Manually connect any device that has bluetooth capabilities to the speaker to verify it's bluetooth compatibility.
The Bluetooth data transfer speed should be at least 2.1Mbps [5]	Use a bluetooth speed test app between a phone and the speaker to make sure the transfer speed is up to our standards. A bluetooth analyzer program on a computer can also be used to not only measure bluetooth transfer speed, but also interference and signal strength.

Table 2: Bluetooth Subsystem Requirements and Verifications from Design Document

The microcontroller and Bluetooth subsystem changed quite substantially from the time we finalized our Design Document (see section 2.4). We decided to change our microcontroller to the ESP32 which eliminated the need for either an independent Bluetooth module, or a two-in-one bluetooth amplifier. The ESP32 is equipped with BLE (Bluetooth Low Energy) which has transfer speeds of around 1Mbps. This is significantly lower than the goal that we had originally set for ourselves. This was a result of us deciding to prioritize the electronics in our project rather than the audio quality. When we tested and found that not only does the ESP32 satisfy our first requirement, but can also play audio with decent clarity, we were satisfied with our change in the bluetooth subsystem.

3.3 Sensor Subsystem

Requirements	**Verification**
The accelerometer should be able to send data continuously to the microcontroller.	We will connect the accelerometer to the microcontroller and then connect the microcontroller to the computer using the serial port. We will then verify that the accelerometer is continuously sending data and is giving different values if we manually move the accelerometer around.
The microcontroller is able to accurately convert the acceleration readings to velocity.	We will use the microcontroller to compute velocity from acceleration using the equation $V_f = V_0 + at$. We will set t to be 10 ms, so we will convert the acceleration to velocity every 10 ms. We will verify the conversion by connecting the microcontroller to the computer and verifying that the velocity is being updated every t milliseconds and that the velocities are accurate.

Table 3: Sensor Subsystem Requirements and Verifications from Design Document

We were able to connect to the microcontroller using the SDA and SCL pins on the MPU-6050 accelerometer, and after writing some code to fetch values, we were getting continuous acceleration readings along the X, Y, and Z axis. The values would also change if we moved the accelerometer around, showing that the unit worked. To get velocity from acceleration we used the simple physics equation outlined in the table above (see section 2.5). After testing, we decided to sample every 300 milliseconds instead of 10 milliseconds, as our values were not changing as quickly as we wanted them to. After changing the sampling rate, we were able to see good results in the serial monitor. To verify the values we drove around in the car and confirmed that accurate speeds were being recorded. The table below outlines the results that we obtained after testing and calibrating the accelerometer.

Speed in MPH	Relative Volume Level on iPhone
20-25	95%
15-20	90%
10-15	75%
5-10	60%
<5	45%

Table 4: Speed and Volume values as shown in Final Demo Video

3.4 Speaker Subsystem

Requirements	**Verification**
The max volume should be 85 dB. The minimum volume should be 60 dB, similar to the Beats pill speaker [4].	Use a decibel meter to measure the maximum and minimum volume that the speaker is capable of producing. Check that the upper bound is 85 dB and that the lower bound is 60 dB
The speaker should have an SNR of at least 60 dB	Measure the signal level by playing an audio test track and using a decibel meter. Then measure the background noise using the same meter to get the noise level. Then plug the values into the equation below to calculate the SNR. $SNR = 20 \times log(signal\ level \div noise\ lev$
The speaker should be able to play sounds at 10 different volume levels spaced out between 60 and 85 dB.	We will play a single buzzing sound from the speaker and use a decibel meter to ensure that there are discrete volume levels that are equally spaced out between the minimum and maximum volumes.

Table 5: Speaker Subsystem Requirements and Verifications from Design Document

The speaker subsystem was drastically changed as result of feedback after the Design Review. As a result we decided to abandon these initial requirements for our speaker system to focus more on our other subsystems and making sure they worked. As mentioned in section 3.3, we did do testing by driving around, and doing that we ensured that we had 10 different volume levels. These volume level changes were done in code and to do that we had to set values between 0 and 127. To test this we played “Hotel California” by the Eagles and observed that the volume did increase in increments, up to 10, as the car sped up. For the Final Demo Video, in order for these changes to be audible we had to set a minimum volume at around 58, or 45% of the maximum volume, in code.

3.5 User Interface Subsystem

Requirements	**Verification**
Each of the buttons should be debounced, so that the desired output from pressing a button only occurs when the button is fully pressed.	We will ensure debouncing using software. Our code will keep track of if the button is high or low, and keep track time between presses, and then decide whether the button press should be processed. We can verify this by connecting the buttons to an LED that will only turn on once per button press.
The 10 count LED strip should display the correct volume as the volume changes.	We will have code that maps speaker volume into a mapping of which LEDs should be on. We can verify that the LEDs display the correct volume by setting test volumes using the microcontroller and visually checking that the correct LEDs are on.
The LED to display battery level should show the correct color based off battery level (green 30-100%, yellow 10-30%, red 0-10%)	We will have code that maps a battery percentage to either red, green, or blue. We can test this by setting test battery percentages using the microcontroller and checking that the LED is the correct color.

Table 6: UI Subsystem Requirements and Verifications from Design Document

Because of our mishap with the stereo setup on the PCB, we had to settle for only the mode button instead of the three to four that we had originally intended to have. With the mode button we did fulfill the first requirement, and were able to debounce the signal. We used the "ezbutton" library [6] in the Arduino IDE to achieve this. We also tested that the button only turned on an LED one time by breadboarding the design. For the second requirement we decided to not have volume indicating LEDs because of the lack of GPIO pins. Our alternative was that the user could keep track of volume changes by observing their volume bar on their devices. The third requirement also couldn't be completed because of the lack of GPIO pins. Had our PCB not worked out, we did have these buttons and LEDs all working on a breadboard. However, with the heavy emphasis on PCB functionality in the course, we decided to make the best of what we could with our one extra GPIO pin on the PCB and implement the crucial mode button. If we were to have more time to be able to fix our PCB design, we would implement the stereo output and thus be able to have more GPIO pins to add LEDs and buttons (see section 2.7).

4. Costs and Schedule

4.1 Cost Analysis

Cost Analysis Table for Key Parts			
Description	**Manufacturer**	**Quantity**	**Extended Price**
MPU6050 Accelerometer	HiLetGo	3	$9.99
ESP32 microcontroller	Espressif	3	$12.60
Single LEDs	Digikey	10	$8.06
Buttons	Digikey	10	$15.54
MAX98357A Bluetooth Amplifier	Adafruit	4	$23.80
MICRO-USB-SMD_105017-0001	Ubibot	2	$7.84

Table 7: Cost analysis table for essential parts that were purchased

Table 7 highlights the essential parts that our group purchased for building our speaker. In addition to these parts, there were also several types of resistors, capacitors, connectors, and other miscellaneous items that were needed to put the project together, or for the PCB design (see section 2.3). The total cost of the parts that were utilized for the testing and final implementation of the project is close to $150.

To further account for total labor costs, we first start with the average salary of a computer engineering graduate from UIUC [7] and perform some simple math to calculate the hourly rate:

$105,352 annual salary → 50 weeks / year * 40 hours / week = $52.68 per hour

Now we account for how many “employees” we have, number of hours of work per week and number of weeks till the project is completed.

$52.676 / hour 20 * hours / week * 10 weeks * 3 employees = $31,605.60

The total labor cost is the sum of the components and total labor, leading to a total evaluation of $31,755.60 as the cost of our project.

4.2 Schedule

Scheduling Table		
Week	**Task**	**Task Lead**

2/20/23 - 2/26/23	Design Document	All
	Ordering parts	
2/27/23 - 3/5/23	Design Review/Design Document Revisions	All
	PCB Board Design	Raj, Dhruv
3/6/23 - 3/12/23	Develop Bluetooth subsystem	Chirag, Dhruv
	Develop Power subsystem	Raj, Chirag
	Order PCB Board	All
3/13/23 - 3/19/23	**SPRING BREAK**	
3/20/23 - 3/26/23	Develop Sensor subsystem	Chirag, Dhruv
	Run testing requirements on Power and Bluetooth subsystem	All
	PCB Board development	Dhruv, Raj
	Develop UI subsystem and Speaker subsystem	Raj, Chirag
3/27/23 - 4/5/23	Progress Reports	All
	Run testing requirements on UI, Speaker and Sensor subsystems	
	Finalize PCB Design	
4/3/23 - 4/9/23	PCB Board Revision	All
4/10/23 - 4/16/23	Final PCB Board Revision/Order	All
	Testing/Minor bug fixes	Raj, Chirag
4/17/23 - 4/23/23	Testing and PCB Board Assembly	All
	Testing/Minor bug fixes	Raj, Dhruv
4/24/23 - 4/30/23	Final Demo/Mock Presentation	All
	Testing/Minor bug fixes	Dhruv, Chirag
5/1/23 - 5/7/23	Final Presentation/Paper	All

Table 8: Scheduling table which shows an overview of tasks for the project

Table 8 shows the schedule that was followed over the course of the semester in developing the project. As a group, a strength we had was the ability to stay disciplined and consistently work on the project over the course of the semester instead of waiting until the end to make things happen. Although the schedule was occasionally modified because of unanticipated bugs and roadblocks, we were still able to stay within the allotted windows for completing most tasks.

5. Conclusions

5.1 Accomplishments

The main successes of our project revolve around the successful retrieval of data from the accelerometer and a working PCB. Specifically, the PCB was able to connect our microcontroller to our Bluetooth devices, as well as play music through the speaker driver from any application of our choosing. Although it was not perfect, we found much satisfaction in our PCB because of what we were able to accomplish despite none of us having experience in designing or building a PCB before this project.

5.2 Uncertainties and Future Improvements

Despite the successes and accomplishments we had, there were still some areas where we felt we could have improved. One of the biggest challenges we faced was with getting consistent data from our accelerometer, as well as being able to demonstrate its functionality at lower speeds. These were both issues we faced because of the quality of the accelerometer we selected, which led to sensitivity issues in certain cases. Another challenge we faced was implementing stereo sound in our product, which was something our group wanted to do at the beginning, but we were not able to properly implement it due to the errors in our pin assignments in our PCB design. This problem also led to our user interface not being as robust as we wanted it to be, as we were forced to use those pins to produce sound from our speaker driver.

If we were to continue working on this project in the future, we would want to improve our project by first fixing these issues that we were not able to address in the project. By investing in higher quality parts, we would be able to improve data collection with our accelerometer, as well as improve the sound quality from our speaker drivers. An alternative design we could develop would include an improved stereo amplifier on our PCB, which would allow us to improve audio quality and develop a better overall product. A byproduct of this would be that we would free up the extra GPIO pins we used for speaker output, which would allow us to improve our user interface on the speaker and add buttons for volume control and power. We would also be able to add LEDs that show Bluetooth connection and the volume level that the speaker is currently at.

5.3 Ethical Considerations

As we anticipated in earlier stages of the project, there are minimal ethical concerns and risks with our product that we developed. One potential concern is that the volume may be too loud after it is automatically adjusted. To combat this, we have made sure that the user has the highest priority for controlling volume on the speaker, and we made sure that our speaker does not allow the volume level to exceed 95%. Another concern that may be a little far fetched is that if the speaker isn’t in driving mode and the user is driving, they can be distracted from driving by trying to adjust the volume on the speaker, and potentially risking the safety of others with their distracted driving. However, this type of behavior from the user defeats one of the purposes of our product, which is to increase driver safety by making it easier to complete a task (changing the volume of the music in the car) that is traditionally considered to be a distraction. Thus, we feel that despite the potential risk that this creates, we are still abiding by the IEEE Code of Ethics [8], as our intent with the project is the responsible use of a product that we are creating to help the public mitigate a risk that exists even without our product.

References

[1] Espressif, “Wi-Fi + Bluetooth + Bluetooth LE MCU module”, ESP32-WROOM-32 datasheet, Aug. 2016 [Revised Feb. 2023].

[2] Robert Feranec. "How to Make Custom ESP32…" *YouTube*, Nov. 22, 2022 [Video file]. Available: https://www.youtube.com/watch?v=S_p0YV-JlfU. [Accessed: May 2, 2023].

[3] T-vK, “T-VK/ESP32-ble-keyboard: Bluetooth LE Keyboard Library for the ESP32 (Arduino IDE compatible),” *GitHub*. [Online]. Available: https://github.com/T-vK/ESP32-BLE-Keyboard. [Accessed: May 2, 2023].

[4] J. Pollicino, “Beats by Dre Pill Portable bluetooth speaker...,” *Engadget*, 16-Oct-2012. [Online]. Available: https://www.engadget.com/2012-10-16-beats-by-dre-pill-bluetooth-portable-speaker-ears-on.html#:~:text=An%20internal%20battery%20is%20said,via%20its%20Micro%2DUSB%20input. [Accessed: Feb. 2, 2023].

[5] J. Blom, “Bluetooth basics,” *SparkFun*. [Online]. Available: https://learn.sparkfun.com/tutorials/bluetooth-basics/common-versions. [Accessed: May 2, 2023].

[6] ArduinoGetStarted, “ezButton Library for Arduino,” *GitHub*. [Online]. Available: https://github.com/ArduinoGetStarted/button. [Accessed: May 2, 2023].

[7] Grainger Engineering Office of Marketing and Communications, “Salary Averages,” *Electrical & Computer Engineering at UIUC*. [Online]. Available: https://ece.illinois.edu/admissions/why-ece/salary-averages. [Accessed: May 2, 2023].

[8] IEEE, “IEEE code of Ethics,” *IEEE*. [Online]. Available: https://www.ieee.org/about/corporate/governance/p7-8.html. [Accessed: May 2, 2023].