Wireless Mesh Audio System

Stephan Stankovic (u0872863@utah.edu)
Steen Sia (u1031024@utah.edu)
Jeremy Wu (u0895969@utah.edu)
Jonathan Pilling (u6005524@utah.edu)
https://theboyzzzz.github.io

***Abstract*—In today's world, many households have an abundance of wireless or Bluetooth speakers laying around. However, due to software and hardware limitations, it is difficult to pair multiple speakers together. We aim to solve this issue by designing a wireless mesh audio system that allows users to pair any number of speakers together. This mesh audio system is comprised of a single transmitter and multiple receivers: the transmitter takes an audio input from the users device and sends the signal to each of the receivers simultaneously. Since wireless networks do not have the bandwidth limitations of Bluetooth, our system will allow users to pair as many speakers as they wish.**

***Index Terms*—Audio, auxiliary, bandwidth, Bluetooth, receivers, speakers, transmitter, wireless mesh audio system**

I. Introduction

A. Background

Speakers are becoming increasingly prevalent into our society. They are present in phones, laptops, cars, and most households will have at least a few speakers laying around. However, as prevalent as speakers have become, support for pairing multiple speakers together (especially of different brands) is surprisingly limited.

One popular solution for wireless speaker pairing is Bluetooth. Bluetooth is a low latency wireless communication protocol used for exchanging data over short distance radio waves [1]. While it is convenient and readily available in any smart phone or laptop, it does have bandwidth limitations. The low bandwidth means that only two Bluetooth speakers can be paired concurrently to one device. While this is enough for many situations, larger rooms often require more than two speakers to fill the room with sound.

In order to overcome the limitations of Bluetooth, our system uses a wireless mesh network. Wireless networks are designed to replace wired Ethernet connections and can handle much greater throughput than Bluetooth. By using a wireless network instead of Bluetooth, our audio system will not run into bandwidth limitations when pairing multiple speakers together. Our mesh audio system will be comprised of two main parts: a transmitter and multiple receivers. The transmitter will take an audio signal through an auxiliary input and send it to each of the receivers simultaneously.

There are some existing products that implement a wireless multi-room audio system similar to ours such as Sonos and Google Cast [2], [3]. However, the proprietary software/hardware used by these manufacturers make it hard or impossible to pair different brands together. For example, Sonos users can only stream audio to their speakers by using the Sonos app, while an Apple Homepod requires an Apple device to stream any kind of content. Another problem for many of these existing wireless audio solutions is that they are prohibitively costly. The Sonos connect, which acts as a receiver to allow users to stream audio to their own speakers, costs $349 for a single unit. The Google Chromecast audio is a much cheaper solution at $29, but streaming to the Chromecast audio is not supported by all devices/applications.

Our goal is to bridge the gaps in the wireless speaker market by creating a mesh audio system that does not require any proprietary software/hardware. We will keep our system simple and inexpensive. Each receiver will also have a simple auxiliary output, meaning that users will not be restricted to any existing ecosystem and they can connect to any brand of speakers.

B. Motivation

*1) **Wireless Mesh Networks**:* Bluetooth is currently the most popular solution for wireless speakers due to it is ease of use and availability; however, as explained previously, Bluetooth suffers from bandwidth limitations. Thus, it is not a viable solution for our project.

A traditional WiFi network is comprised of a single root node known as an Access Point (AP) and leaf

This work was supported by the University of Utah, submitted on December 13, 2019. Stephan Stankovic, Steen Sia, Jonathan Pilling, and Jeremy Wu are all with the University of Utah, Salt Lake City, UT 84112 USA (u0872863@utah.edu); (u1031024@utah.edu); (u0895969@utah.edu); (u6005524@utah.edu)

nodes that connect to the root. The root is responsible for arbitrating and forwarding all data between the leaf nodes. Since the AP must be directly connected to each of the leaf nodes, traditional WiFi networks suffer from limited coverage because every node must be in range to connect to the root node. Furthermore, another issue is that the network is susceptible to overloading as the maximum bandwidth is determined by the AP.

To overcome the range and bandwidth limitations of a traditional wireless network and Bluetooth, our system will use a wireless mesh network. A mesh network differs from a traditional WiFi network in that all nodes communicate directly with neighboring nodes instead of a central AP. In essence, this means that every node in the network that has children becomes an AP, and children with no other connections are known as leaves. Thus, a mesh network is easily scalable since it is not bandwidth limited by a single AP and it also leads to much greater coverage since each new node that is added extends the range of the network [4]. Currently ESP offers their own ESP-Mesh protocol, which we have successfully setup and transferred text, but we may use the AP+Station mode to create our own mesh protocol.

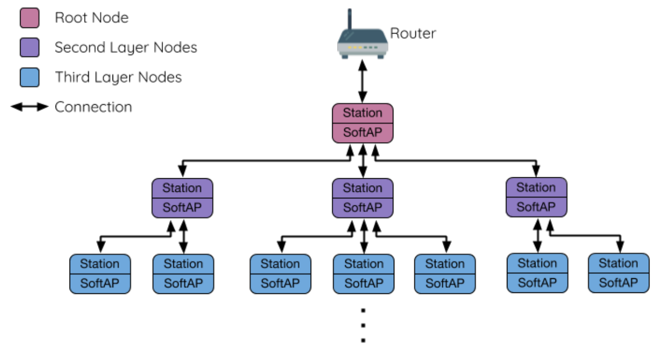


Fig. 1. Sample WiFi Mesh Network [5]

*2) **Huzzah ESP32 Board**:* The core component of our mesh audio system is the ESP32 board [6]. ESP32 boards have multiples features that are perfect for our project application. The existing features are as follows:

- **802.11b/g/n HT40 WiFi transceiver:** Built-in WiFi allows our boards to form a mesh network and communicate with one another.
- **Integrated Bluetooth:** Bluetooth will allow the root node to be paired to a smartphone for wireless control over the mesh network.
- **Low noise analog amplifier:** A low noise analog amplifier ensures clean audio transmission to each speaker.
- **2 x DAC:** Digital to Analog converters allow us to convert the digital audio signal to an analog output for our speakers.
- **PWM/timer input/output available on every GPIO pin:** Timers on every pin will allow us to keep our mesh audio system synchronized.

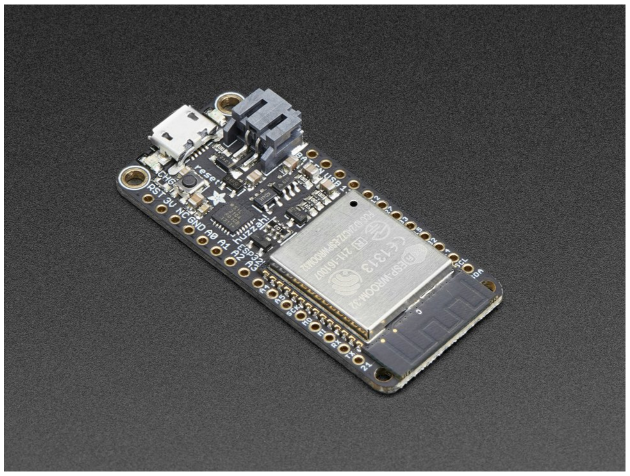


Fig. 2. Huzzah ESP32 Board [6]

C. Hardware

Hardware for this senior project will be our ESP32 boards connected to PCBs which will contain an audio jack and any additional components we deem are needed for the project. We are planning on putting these in 3D printed cases combined with batteries for portability. Batteries will be optional to allow users to take these wherever they go, otherwise users will be able to power the system with a micro usb. The optional battery will be any Lithium Ion Polymer battery. Other hardware we will be utilizing for our project will be speakers to showcase transmitted; synced audio; mobile phone to run and showcase our application; and a transmitter board.

II. Technical Details

A. iPhone Application

To control the mesh audio network, an iPhone application was developed using Swift. The application communicates with the root node in the network through BLE (Bluetooth Low Energy). The main features of the app are as follows:

- Connect to any BLE peripherals
- Add/remove devices from the network
- Rename devices in the network
- Mute devices
- Change volume

1) Swift Core Bluetooth Framework: The Core Bluetooth Framework provides classes that allow iOS devices to discover, explore, and interact with low energy peripherals. The two main components of Bluetooth communication are known as the central and the peripheral. In the case of the wireless mesh audio network, the phone application is the central, and the Esp32 devices are the peripherals. Peripherals constantly advertise the data that they have to any centrals in close proximity. When a central discovers a peripheral, it is able to act as a manager by connecting/disconnecting from the peripheral and reading/interacting with the data that is being advertised. An example of communication between peripherals and centrals is shown below, where the peripheral is a heart rate monitor, and the central is a laptop, phone, or tablet.

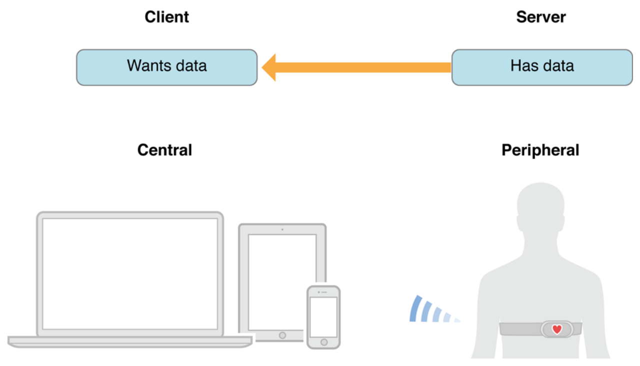


Fig. 3. Central and Peripheral Communication [7]

Once the central is connected to a peripheral, it may then request data through the peripherals services. A service is a collection of data which helps to implement a feature or functionality of the device. For example, in the mesh audio network, the root peripheral has the following services:

- Device Service
- Volume Service

Each service is also a collection of characteristics, which provide more specific data and functionality related to the service. The structure of the peripherals in the mesh audio network are detailed below:

- Device Service
  - Device Names
  - Connected Devices
- Volume Service
  - Mute
  - Current Volume
- Music Service
  - Play/Pause
  - Forward
  - Backwards

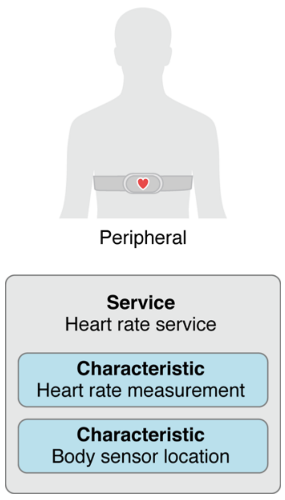


Fig. 4. Peripheral Data Structure [7]

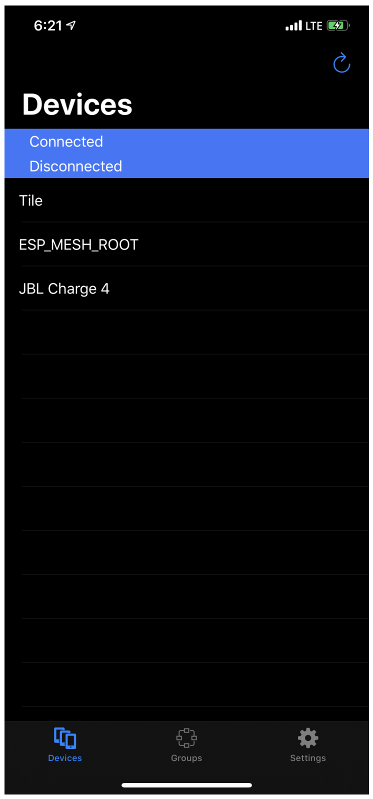
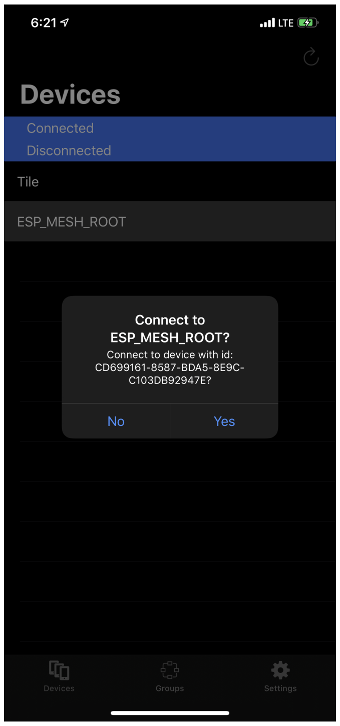


Fig. 5. Devices View

2) Application Details: Upon launching the application, users will be greeted with a Devices screen. The application will also automatically scan for any nearby Bluetooth Low Energy peripherals. If any are found, they will be displayed in the disconnected devices section. At this point, there are two options available:

- Connect to the device: Selecting a device will bring up a prompt which asks if the user would like to connect to the peripheral.

- Scan for new devices: By either clicking on the refresh button in the top right corner or pulling down, users may scan for new devices. This does not affect any currently connected devices.


6:21

Devices

Connected

Disconnected

Tile

ESP_MESH_ROOT

Connect to ESP_MESH_ROOT?

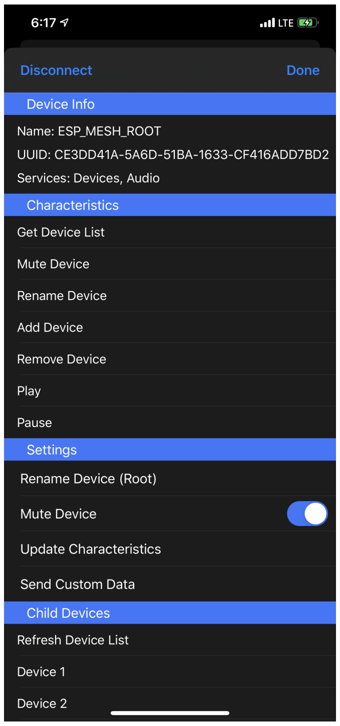
Connect to device with id: CD699161-8587-BDA5-8E9C-C103DB92947E?

No Yes

Fig. 6. Connecting to Devices

Once a user has connected to a device, it will move from the disconnected section into the connected section. At this point, it is now possible to select the connected device, which will bring up the device detail view. The device detail view contains the bulk of the application's functionality, and has the following sections:

- Device Info: Shows the name, universally unique identifier (UUID), and services of the connected peripheral.
- Characteristics: Shows all of the characteristics of the peripheral. Selecting a row in this section will open a text dialog box, allowing users to send custom data to the characteristic.
- Settings: Contains several settings for the mesh audio network:
  - Rename Device: Renames the root node
  - Mute Device: Mutes all speakers in the network
  - Update Characteristics: Refreshes the characteristic list and updates the values of each characteristic if they have changed.
  - Send Custom Data: Allows users to send custom data to a characteristic of their choice.
- Child Devices: Shows the child devices connected to the root node. Selecting a device presents a text box which allows users to rename the device.


6:17

Disconnect Done

Device Info

Name: ESP_MESH_ROOT

UUID: CE3DD41A-5A6D-51BA-1633-CF416ADD7BD2

Services: Devices, Audio

Characteristics

Get Device List

Mute Device

Rename Device

Add Device

Remove Device

Play

Pause

Settings

Rename Device (Root)

Mute Device

Update Characteristics

Send Custom Data

Child Devices

Refresh Device List

Device 1

Device 2

Fig. 7. Device Detail View

B. ESP32

1) WiFi Mesh: The WiFi mesh network is implemented using the ESP32 Mesh Development Framework (MDF). This framework allows a mesh network to be setup with no router. One of the ESP32s is set to be the root node (the device that manages the other devices in the network). This root node then connects to leaf or child nodes, which act as both as an access point and a station. Since all nodes act as an access point, each node extends the range of the network by allowing nodes that cannot reach the root node to connect to an intermediary node.

2) Transmission of Data: Data is transmitted through the mesh network with the use of the wireless broadcast function in the MDF. The broadcast function allows nodes to send data to a device with a different MAC address in the network. Since the root node maintains a table of the MAC addresses of the child devices, data can be transmitted to every device in the network by broadcasting to each MAC address in the table.

The original intention of this project was to stream music from an auxiliary input into the root node and broadcast it to the entire network. The analog to digital converter (ADC) on the board did not function properly due to hardware limitations. Since an ADC is required to convert analog input, the only alternative solution with the remaining hardware was to store music files in the root board's memory. Transmitting the music file across the mesh network involves reading 1024 byte blocks one at a time, then broadcasting the block to the rest of the network.

3) Bluetooth: Bluetooth is a wireless protocol that allows devices to exchange data with one another over short distances [8]. Bluetooth is typically used to transmit small packets of data between mobile devices since it has much lower bandwidth than traditional wireless networks [8]. Although the low bandwidth is not ideal for the wireless audio network, it's more than enough for the data transmitted between the root node and the iOS application.

The ESP32 is a flexible micro-controller which can behave either as a server or a client. Since our project revolves around a main transmitter, it will act as the server. The mobile application will act as the client that connects to the ESP32's server in order to access its services.

Bluetooth on the ESP32 uses a Generic Attribute Table to store the characteristics and services that will be advertised. It is crucial to have generic attribute table for BLE devices to send and receive standard messages. It is a contract that these devices abide to in order to trade information.

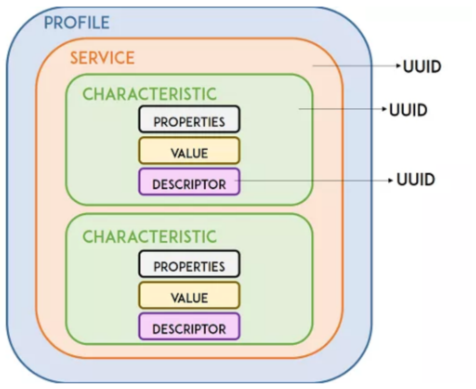


Fig. 8. Generic Attributes Data Structure [9]

Once a profile is connected to one client, it cannot be accessed by other clients simultaneously. Multiple services can be instantiated and can provide utilities with the use of one or more characteristics. Each characteristic has a declaration and value. They respectively describe the metadata contained in the characteristic and contain the raw data to be transmitted.

Characteristics can be advertised with a variety of permissions such as read, write or notify. Each characteristic also has a UUID which allows it to be easily identified. Our service contained the following characteristics:

- Get Device List
- Mute Device
- Rename Device
- Play
- Pause

4) I2S Communication Protocol: The way audio data was transmitted was serially using the I2S protocol. This protocol is specifically made for audio applications and allows for a low jitter connection. There is only one master and one slave to which data is being transferred from and to. I2S uses 3 different signals: word select, serial clock, and serial data. All of these clock signals were generated on the ESP32 boards. This made it fairly simple to send data since we would simply configure the number of bits, which was 16, and the sampling frequency, which we selected to be 11025 HZ. The WAV files were all sampled at that rate and therefore we had very poor audio quality. Data sent over the network is stored into memory. I2S then retrieves it, sends it to a direct memory access buffer, and then finally outputted to the on-board DAC.

An I2S configuration was also set up for root node which contained a built in ADC. However, the ADC was so inaccurate that it was unusable to decode audio. The difference in this configuration versus the one found above was that now the ESP32 was receiving data from the on-board ADC, placing it into a direct memory access buffer, where then I2S would transfer into memory which the WiFi could access.

5) Digital to Analog Converter (DAC): Digital to analog conversions were handled by the ESP32s built in DAC. There are two DACs on the ESP32, one was used for left channel audio data while the other handled right channel audio data. The DAC received data directly from the I2S protocol discussed above. Whenever the I2S data was available, it could be read from two GPIO pins on the board. The DACs were only able to achieve 8 bit resolution for the digital data that was provided which was unfortunately very low quality.

6) Analog to Digital Converter (ADC): Analog to digital conversions were supposed to take place where we stream audio from a auxiliary capable device, but unfortunately we ran into a hiccup with the PCB that we created, and the onboard ADCs are notoriously bad. After much research, we learned that others couldn't successfully get it to sample at the advertised 192KHz potential. Many users couldn't even get it to sample using the preferred method at all, and the ones who did just did analog reads from other GPIO pins. We didn't

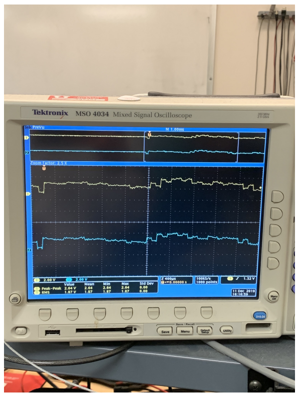


Fig. 9. Left Right Synchronization of Audio

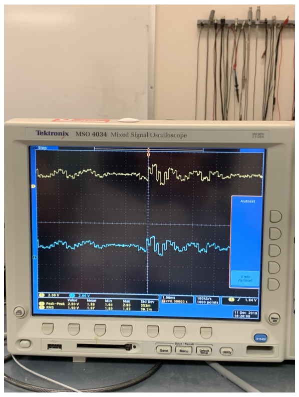


Fig. 10. Audio sync between 2 boards

use that method simply due to sampling being too slow in that case.

C. PCB

The PCB was designed in the first month of our project. We picked the TLV320AIC3204 audio codec from Texas Instruments as our main integrated circuit (IC) on the PCB. This would have allowed us to do analog to digital conversions as well as digital to analog. Therefore, we only needed to design and test one PCB for both the root node and the non-root nodes. The datasheet provided from Texas Instruments was used to select capacitor and resistor values and to wire up the board. Unfortunately, the codec was only sold in a Quad Flat No-Lead (QFN) form, which we found to be very hard to solder onto the board. The PCB wasn't used simply due to us getting the board back from soldering the week of demo day. We didn't have enough time to configure the board and verify that everything worked. Our original proposal didn't contain an audio codec, but due to the poor performance of the ADC and DAC found on the ESP32 chip, we later added one. Our original PCB was supposed to only contain the 3.5mm headphone jack ports and the RC circuit to reduce noise.

Similar to our original proposal, for demo day 3.5mm headphone jack breakout circuits were placed onto breadboards. We didn't have enough time to create a custom PCB for this purpose.

III. What we Learned

- Teamwork and Communication
- C
- WiFi and Mesh protocols
- Configuring Peripherals
- Simple Audio Signal Processing

A. Challenges

The team experienced many challenges throughout the course of the project. Many risk mitigation plans had to be put into effect. A lot was learned from circumventing these roadblocks.

1) Writing Micro-controller Code: One of our first challenges was beginning to write the code for our micro-controller. When researching we saw that the ESP32s had an Arduino IDE and a default Espressif Development Framework. We decided to go with the default framework such that we could have more access to the ESP32 versus, having to figure out how to rewrite everything in Arduino. We had a TA in Embedded Systems who helped us make this decision. Tom Becnel informed us how they wrote code for ESP32s out in the field; and he mentioned ”We normally use their example files and modify them for our use”. Taking his advice we began to look into the examples and learned that there was a Mesh Development Framework (MDF) that was created for this specific use. We found an example of a no router mesh system (ad-hoc) and decided to not reinvent the wheel. We heavily modified the original code by deleting and stitching in other examples as we saw fit.

2) PCB: From the beginning when designing the PCB we had challenges. Embedded Systems hadn't really taught us how to work with a variety of components and how to read their datasheets. When designing the PCB we had to really bring in knowledge from 2280 and online resources to decide what kind of capacitors and other various components we needed to add. Creating the schematic wasn't horrible since we could use online libraries which had the components we wanted to use. The difficult part was creating the board file with correct wiring since there were so many components.

A mistake we made was doing all surface mount soldering components besides the pin headers. Soldering the PCB was a extremely difficult due to the size of components, the placement, and the packaging of the audio codec. This QFN package was small with 0.5 mm pitch between pins. This proved difficult to work with, even with a stencil. The IC needed to be pushed down onto the PCB to prevent misalignment; however, when the IC was pushed down, the solder paste was spread between the pads causing shorts. This was hard to correct without the right equipment.

Lastly, it was difficult working with the audio codec since it is a fairly new chip offering from Texas Instruments. It sounded amazing on paper and had all the functionality we wanted, but when it came to powering and configuring it, it proved difficult. We found the register mapping, but didn't really understand the process in which to program it. Some documentation stated to provide a master clock signal first, some places said to power the board first then provide a clock signal. It was confusing and difficult.

3) ADC: Working with the on-board ADC was a challenging experience. After some research we learned that the ADC was inaccurate and required a special type of calibration to get it working at all. This was required to get the sampling rate up to the 44100 Hz which we wanted, but unfortunately we weren't able to achieve. We had another option of sampling from a GPIO pin, but that was extremely slow and didn't allow us nearly enough samples to get any sort of recognizable audio. The team learned a lot about how the analog voltages are converted and represented in binary.

B. Testing and Risk Mitigation

Testing each project task was just as important as developing the individual components. One thing that was imperative to progress was to always make sure that testing happened concurrently alongside implementation. This ensured that new functions and changes to the code base were working according to specification. Testing new modules would first test the core feature of the added system. After making sure the new implementation worked for normal use, edge case testing began. After edge case testing the module was deemed complete. As new system pieces were integrated, regression testing was done on existing technologies to ensure there were no new issues. Many issues were encountered throughout the semester. Risk mitigation plans were in place to resolve issues beyond the teams control and have a successful demonstration.

1) Hardware Testing: One of the core custom hardware pieces was the PCB with audio codec. One of the simple tests performed was doing an ERC check in Eagle. This was a simple check to ensure no basic errors were made with circuit design. Once the circuit was fabricated and the components were soldered, visual inspections on all soldered components was performed. During visual inspection the PCB was examined thoroughly. The visual inspection passed if no solder chips, tombs, or other abnormalities were found. Many boards failed this check under the microscope. One PCB passed this inspection near the end of the semester. This PCB was then hooked up to power, and voltages were traced with a multi meter. The first multi meter test was to check that the audio codec was receiving power and that the voltage reading was accurate. After confirming that the codec was powered, a phone was connected to the auxiliary input on the PCB. Following auxiliary traces with the multi meter confirmed that voltage readings were being delivered to the audio codec input pins. After these tests, serial data was attempted to be printed out on the console. This test was never passed successfully. An I2S/I2C timing issue seemed to be the probable cause of the serial data issue. To circumvent this issue, the built in ADC on ESP32 was used. More issues ensued during testing, analog voltage readings were sporadic and unpredictable. Another risk mitigation plan was used. This plan used a WAV file and got rid of the ADC conversion required. The final hardware testing was driving speakers with the ESP32's DAC pins. Being able to hear audio from the speakers passed this final test.

2) Mesh Testing: The first software module that was completed was the mesh network. The mesh network was primarily tested by sending test packets over the mesh network. These test packets contained integer information. Testing was first performed with one root and one leaf in the mesh. On successful, accurate data transfer, this test was passed. Testing the mesh's broadcast capabilities involved one root and multiple leaf nodes. This test was passed when integer data was being transferred simultaneously throughout the mesh. Functional testing was performed with a maximum of five leaf nodes. Communication continuously worked as expected.

After root and leaf testing the mesh network was tested with intermediate nodes. These tests also helped validate the self-healing properties of the mesh. The intermediate nodes provided an extra hop in the mesh network so that a leaf did not always communicate directly with the root. To perform this testing, the boards used were placed approx. 25m apart. If a leaf was too far from the root, it would connect to the mesh through the intermediate node instead. With the change in network configuration, data was still being transferred successfully. No data loss was noticed, but minor latency was noticed in this network configuration.

Transmission of audio data was broken into pieces. After the final risk mitigation plan was in place, a leaf board was used to verify that a speaker could be driven by a WAV file being played through the ESP's DAC. After this functionality was working, the WAV file was moved to the root node. This WAV file data was then packed and broadcast over the mesh network. Various I2S and WIFI buffers had to be adjusted for this to work. Once audio was successfully playing over the mesh, different sample rates, I2S buffer sizes, and delays were manipulated during testing.

Ensuring that audio was synced was one of the final tests for the mesh network. This was done by listening to the track while it was played through different speakers. Some minor latency issues seemed to happen at times, but most of the time these issues were not audible.

3) Mobile App Testing: Testing of the mobile application was done mostly by trial and error. Testing user interfaces with unit tests or conventional testing methods is challenging due to the fact that most of the mobile application relies upon graphical design. User interface design is also largely dependent on preferences that can vary from user to user.

4) Bluetooth Testing: Testing the Bluetooth integration was done through two methods: prototype application and mobile application. The prototype application utilized is the **nRF Connect for Mobile** [9]. An app offered in Android and iOS, which allowed us to use our phones to test our implemented ESP32 BLE server. In action, our BLE server would initiate advertising mode and allow the app to scan for devices. Upon scanning, we were able to connect to "ESP-Mesh" (BLE server) and see listed services and their respective characteristics. The first test simulation was to be able to send bytes of data to the client. The Bluetooth implementation consisted of a mock characteristic that had a value array of bytes containing "hello" represented in hexadecimal. Once read from the characteristic, "hello" displays without any issues. This marked the start of our read characteristics that were previously mentioned in the Bluetooth section.

The next test consisted of the app transmitting information to the BLE server. The app would be able to write any byte value and our BLE server should print out that same value. Fortunately, esp-idf had utilities to make data transfer easy [8] which allowed us to log the data sent from the client when a writing event occurs.

Finally once these characteristics became functional, we were able to test them using the actual mobile application. The phone application easily identified the BLE server under "ESP-MESH", connected to it, and accessed the UUIDs for each characteristic. We added one more characteristic which allowed read, write, and notify permissions. The phone app was able to enable/disable notification which respectively acts as a way to receive/block server information. This was a good function to implement in order to perform less tasks or essentially put the server and client interaction on stand by. The test from before remains, we made sure that our mobile app was able to read all characteristics simultaneously and parse characters and integers. Lastly, we sent integers from the mobile app to the BLE server and the server was able to print the same data. The transmission of data (reading and writing) performed seamlessly and had no delay.

C. 3D Printed Case

The purpose of the 3D printed case is to provide an enclosure for both the ESP32 and the custom PCB. The bottom case is designed to hold the ESP32 with pegs. It also features a hole for the mini-USB port so that it can go inside the case and attach to the micro controller.

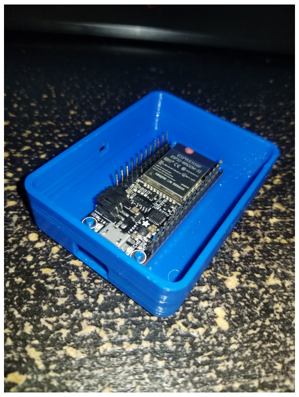


Fig. 11. Bottom Case

The top case is responsible for holding the custom PCB upside down. It has snap fit sides guaranteed to keep the case in tact. The purpose of this is so that we can have wires directly connecting from the PCB to the ESP32 and make it compact. There is also a hole for the top case to give enough space for the audio jack pins to connect inside the case.

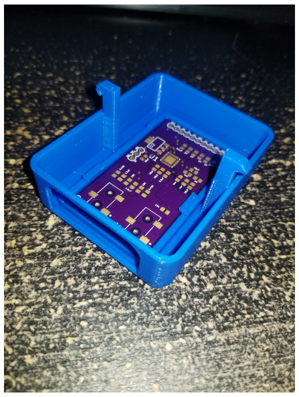


Fig. 12. Top Case

Lastly, the full case with the ESP32 and PCB together is represented below. But due to technical difficulties with the PCB, we did not opt to use the 3D printed design.

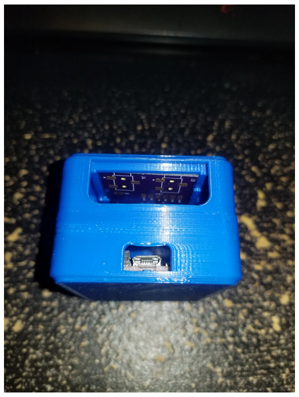


Fig. 13. Full Case

V. Conclusion

We have learned a lot about how to work as a group and present ideas to each other. Communication is vital in any group setting and we have learned how important this is in an engineering setting. We have also learned about the technologies we worked with throughout the semester. Moreover, we have learned about wireless mesh networks and how to implement them. Lastly, we have learned about using frameworks with the ESP-MDF and IDF. These were core frameworks used in our project. To summarize, the intent of the Audio Mesh project was to propose a new audio solutions system that pairs with multiple speakers of different brands. This project used a Wi-Fi mesh system to combat Bluetooth discrepancies. Each designated speaker, connected to their own PCBs, received audio data from the root board. The hope for this project was to provide a quality of life experience for consumers that desire flexibility in sound systems.

References

[1] S. Pandi, S. Wunderlich, and F. H. P. Fitzek, "Reliable low latency wireless mesh networks — from myth to reality," in *2018 15th IEEE Annual Consumer Communications Networking Conference (CCNC)*, Jan 2018, pp. 1–2.

[2] "Wireless speakers and home sound systems." [Online]. Available: https://www.sonos.com/en-us/home

[3] "Chromecast built-in." [Online]. Available: https://www.google.com/intl/en_us/chromecast/built-in/

[4] R. Bruno, M. Conti, and E. Gregori, "Mesh networks: commodity multihop ad hoc networks," *IEEE communications magazine*, vol. 43, no. 3, pp. 123–131, 2005.

[5] Espressif, "Esp-mesh concepts." [Online]. Available: https://docs.espressif.com/projects/esp-idf/en/latest/api-guides/mesh.html#mesh-concepts

[6] A. Industries, "Adafruit huzzah32 – esp32 feather board." [Online]. Available: https://www.adafruit.com/product/3405

[7] Apple, "Apple developer documentation." [Online]. Available: https://developer.apple.com/library/archive/documentation/

[8] Espressif, "espressif/esp-idf." [Online]. Available: https://github.com/espressif/esp-idf/blob/master/examples/bluetooth/bluedroid/ble/gatt_server_service_table/tut

[9] M. Wisintainer, Sheldon, Germán, R. Santos, Giovanni, Y. Tawil, S. Bello, S. Santos, Michele, Paul, and et al., "Esp32 bluetooth low energy (ble) on arduino ide," Jun 2019. [Online]. Available: https://randomnerdtutorials.com/esp32-bluetooth-low-energy-ble-arduino-ide/

Stephan Stankovic Stephan Stankovic was born in Salt Lake City, Utah. His background comes from the Balkan country of Serbia, which peaked his interest in Nikola Tesla. Began to pursue a bachelor degree in Computer Engineering at the University of Utah in August 2013. Expected graduation date, Fall 2019.

Steen Sia Steen Sia was born in Dumaguete City, Philippines in 1997. He moved to the United States in 2009 and is pursuing a bachelor's degree in Computer Engineering at the University of Utah. Currently, he is working as a Software QA intern at SelectHealth in Murray, Utah.

Jonathan Pilling Jonathan Wyatt Pilling was born in Salt Lake City, Utah in 1995. He's currently working on his B.S. in Computer Engineering from the University of Utah located in Salt Lake City. From 2016 to 2019 he worked at Huntsman Cancer Institute at the Information Desk. Since the summer of 2019 he's been working as a Software QA Intern for ChartLogic in Sandy, Utah. Mr. Pilling's awards include Hack the U 2016 - 1st Place (Hack to Save Homeless Pets) and multiple Dean's List awards.

IV. Bill of Materials

TABLE I
BoM

Item	Unit Cost	Quantity	Part #	Manufacturer	Description	Datasheet/Documentation
HUZZAH32	$19.95	15	3405	Adafruit	WiFi modules	https://www.espressif.com
ESP-IDF	$0.00	1	N/A	Espressif	Programming Guide	https://docs.espressif.com
Visual Studio Code	$0.00	1	N/A	Microsoft	IDE to code	https://code.visualstudio.com/
XB31 EXTRA BASS	$99.99	1	N/A	Sony	Bluetooth Speakers	https://www.sony.com/
Auxiliary Cable	$5.49	15	N/A	Syncfruit	3.5mm Premium Aux Cable	https://www.amazon.com/
PCB Creation/Labor	$3	15	N/A	Eagle	Fabrication of PCBs	https://www.autodesk.com/
Headphone jack	$0.95	15	N/A	Adafruit	3.5mm Stereo Headphone Jack	https://www.adafruit.com/

Jeremy Wu Jeremy Wu was born in Salt Lake City, Utah. He's currently a senior in the Computer Engineering program at the University of Utah. He currently works as a junior software developer at IDFL Laboratory and Institute.