Software Controlled Physical Sound Sources

By
Ben Sisserman - Bens3
Micki Rentauskas - Mar3
Won Woo Lyu - Wlyu2

Final Report for ECE 445, Senior Design, Fall 2020
TA: Anthony Schroeder

9 December 2020
Group No. 26

Abstract

This document outlines the design and fabrication process of our senior design project. We designed three separate devices, each capable of producing their own distinct sounds. These devices function identically in terms of software, but each controls a different mechanism for producing sound. The first board controls an array of relays to power electronics on and off. The second board controls a servo motor to strike objects repeatedly to generate noise. The third board uses a vintage phone's ringing hardware to generate a ringing noise on command. All three boards communicate over WiFi, and hence need a jitter buffer to limit latency variation to no more than 30 milliseconds for consistency during the testing of hearing aids. We successfully implemented the Relay board and the Servo board, and verified the performance of the jitter buffer on these devices. Unfortunately, we encountered issues controlling the ringing hardware for the Ring board.

Contents

1. Introduction

1.1 Problem

Testing hearing aids has always been a task to audiologists since soundfield speakers with the introduction of background noise through a soundfield speaker cannot replicate real-world experiences. Improvements of technologies have made a smallear leap between lab testing of hearing aids and actual real-world listening, but these systems are very complex and still not fully capture the real-world sounds [1]. They are complex, because replication of real-world environments requires a variety of different types of sounds from different distances and angles. The best way to produce a sound from a speaker that yields accurate real-world results is to put eight speakers around the person for every 45 degrees [2], which tells that it will be harder to produce accurate real-world sound if the lab environment gets complex.

1.2 Solution

Our project is an array of software-controlled physical sound sources (SCPSS) consisting of three different modules for driving different sound types. Our system is for the Illinois Augmented Listening Laboratory, and most specifically post-doctoral student Ryan Corey, who is doing research on noise-cancelling hearing aids. Currently, their test system for prototypes is an array of speakers which they play sounds on via wired MIDI cables [4]. The issues with this setup are that the sound samples must be recorded in anechoic chambers, which is an expensive process, as well as the fact that speakers are unidirectional, and can only produce sound along the axis it is facing [3]. Our system offers a solution to these problems by using physical sound sources such as blenders, struck objects, and ringing bells, while still capable of being automated by controlling via a Python script on the researchers' PC.

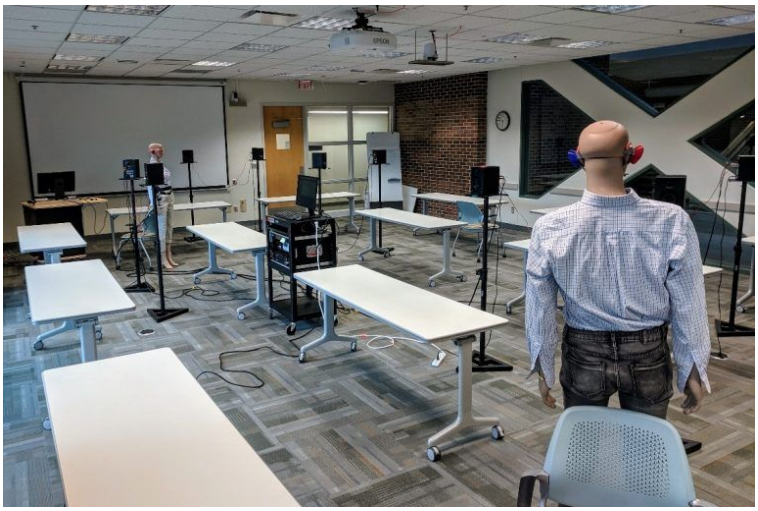

Figure 1. The conference room used for the massive distributed array dataset

1.3 High Level Requirements

- The devices must be able to control at least three different physical sound sources.
- Latency variation must be kept to a minimum. Latency itself is not a concern, but latency must not vary by more than 30 milliseconds.
- Devices must function with at least six feet of distance between devices for simulating real-world environments and circumstances.

1.4 Visual Aids

The visual aids on parts of the project and the overall physical diagram are shown in the following figures. First, we have the host, which is a researcher's PC and this will run the python module. Using the router that is connected to the host, this will facilitate communication between the host and devices over Wifi. The microcontroller that communicates with Wifi will control the devices, and devices are connected to the physical sound sources. The Relay board will make sounds by controlling electronics such as a blender or vacuum, the servo board will strike an object using the servo motor and a metal arm, and the ringing board will ring vintage phone ringing hardware.

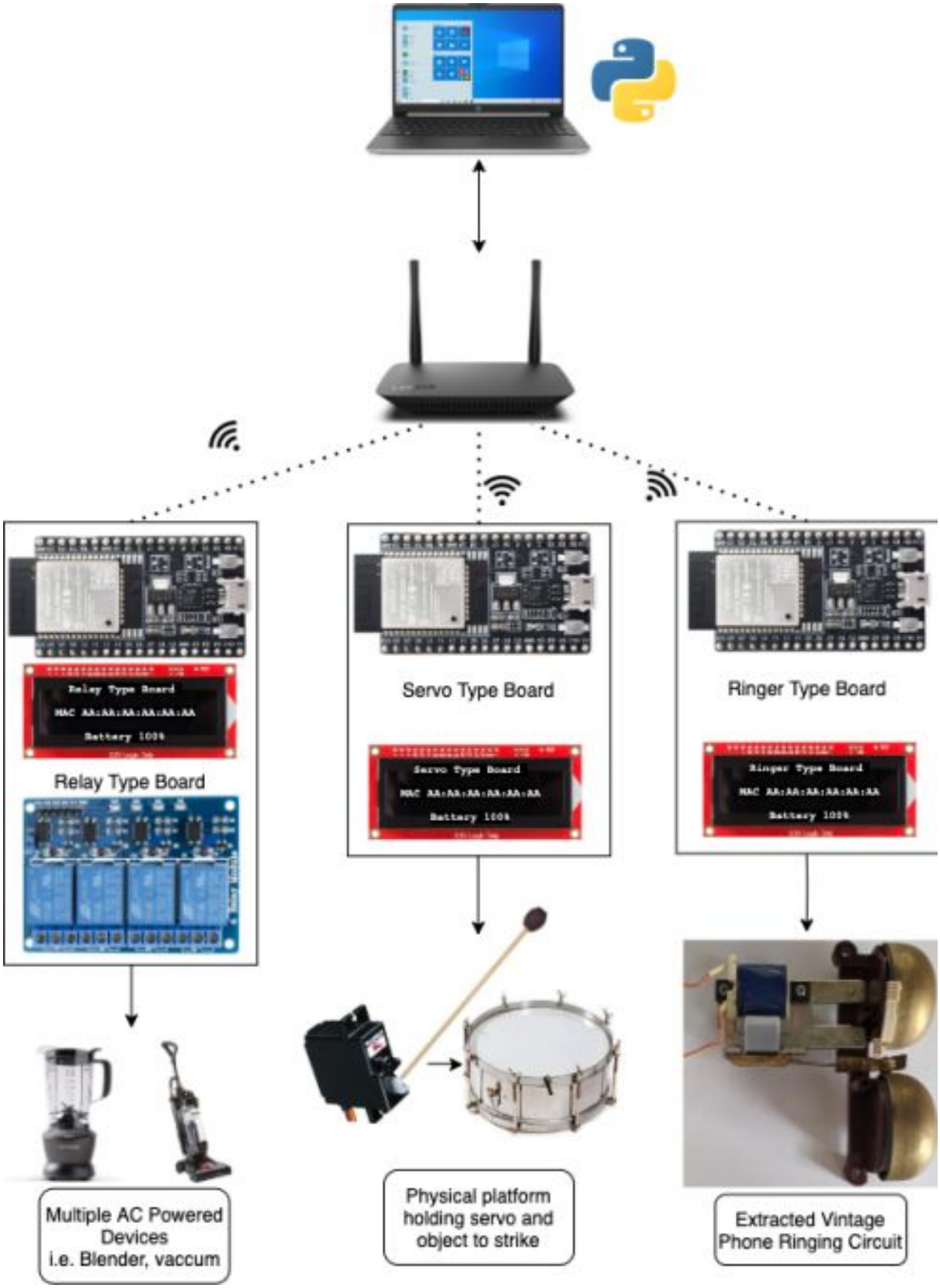

Figure 2. System Overview Diagram

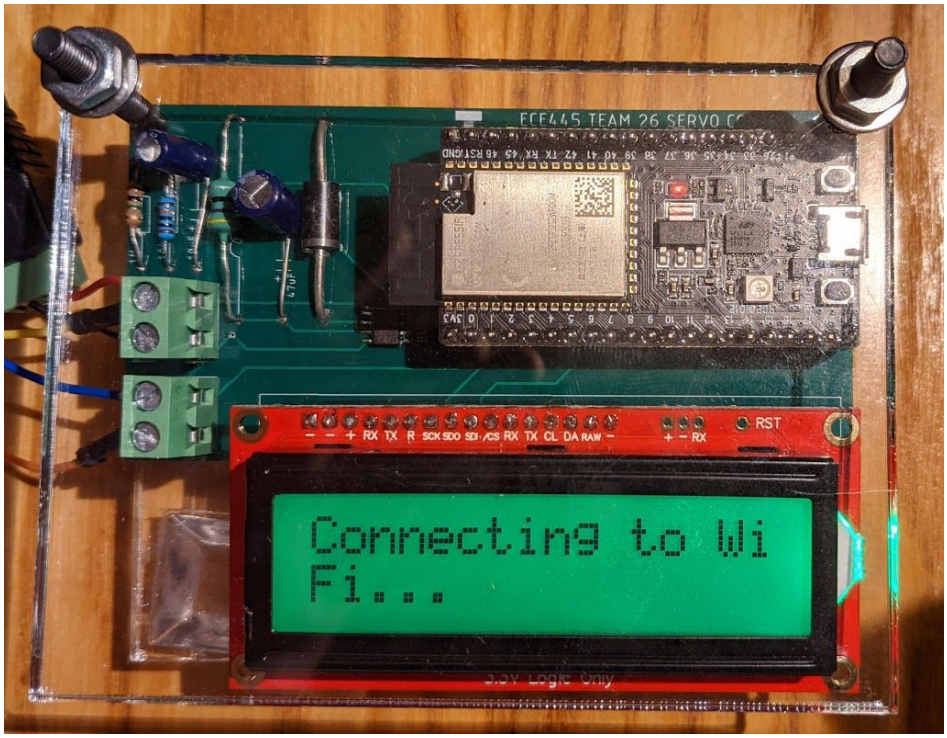

Figure 3. Servo PCB

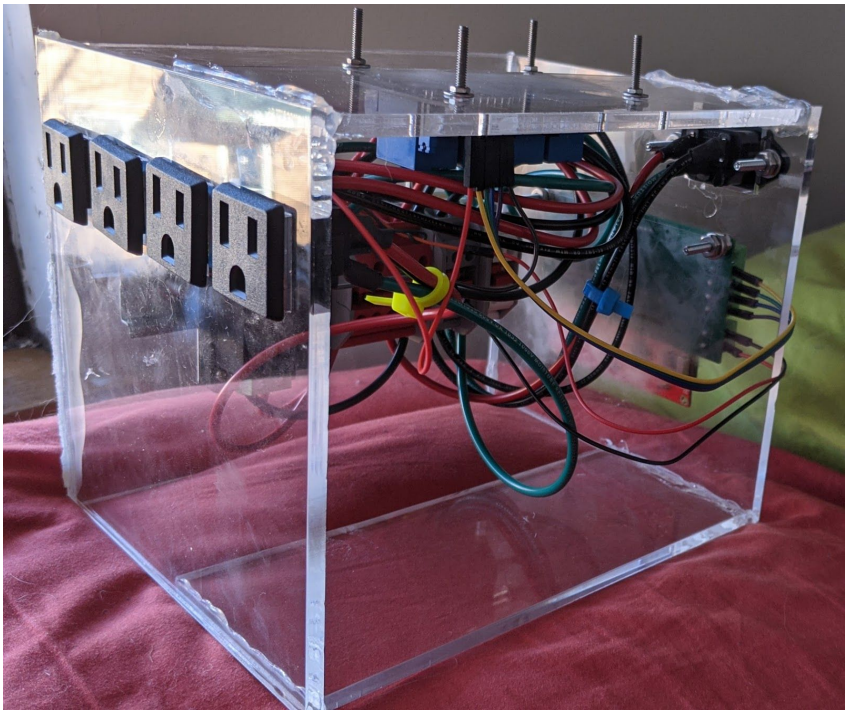

Figure 4. Relay enclosure

1.5 Block Diagram

The full block diagram for the SCPSS system can be found in Figure 6. Our project was divided into 2 main parts, hardware and software, with subcomponents for each. Hardware consisted of three different boards with similar communication systems, but different circuitry for driving the sound sources: relay-type, servo-type, and ringing-type. The software consisted of a python module for the Host to control the boards, implementing the jitter buffer on all three boards, programming the microcontrollers to communicate over TCP and their LCD displays.

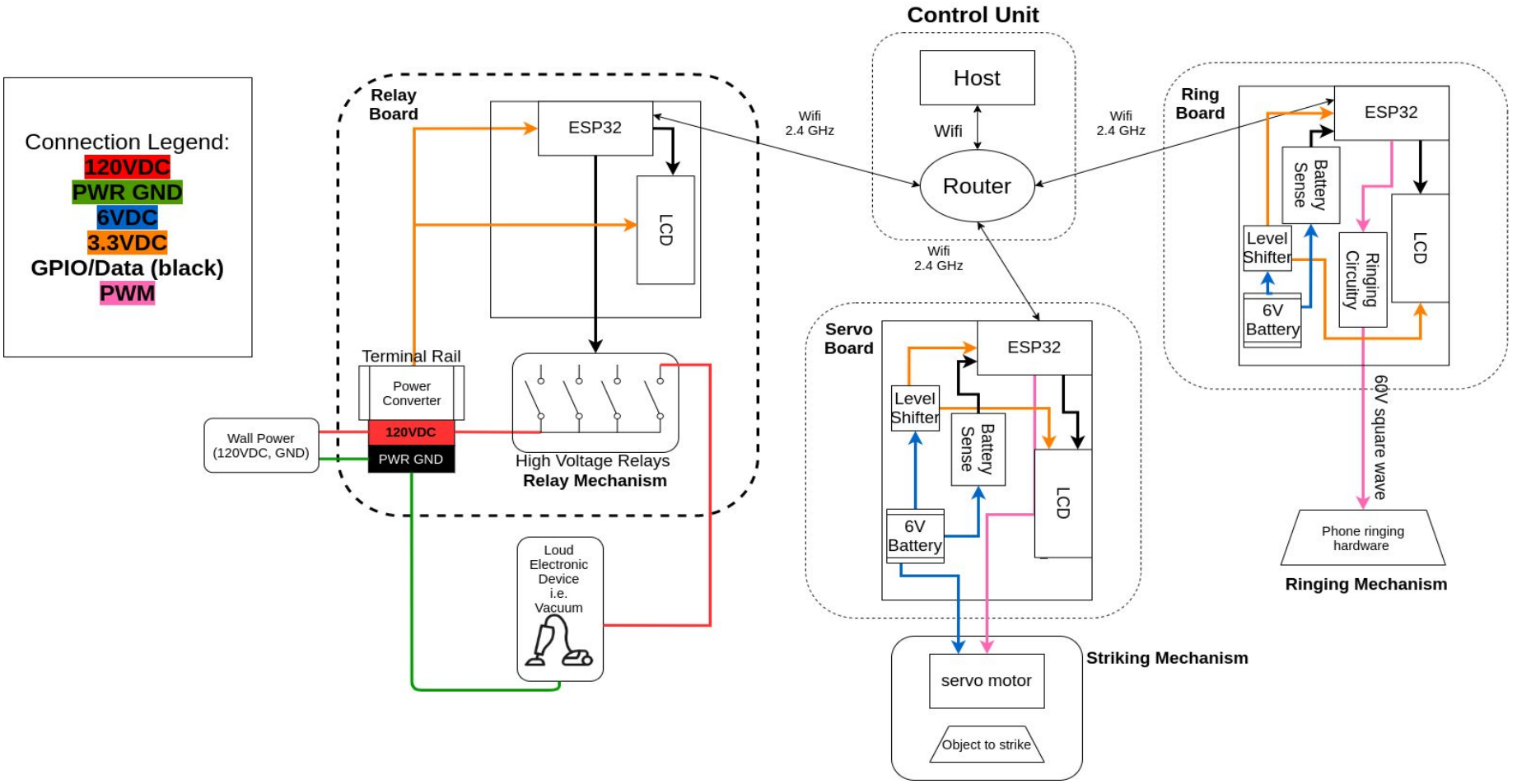

Figure 5. Overall System Block Diagram

2 Design

We opted for a system with three different variations of sound by considering the typical types of sound found in the average building or home. The Relay Board controls a variety of wall-powered devices such as a vacuum or blender, the Servo Board controls a servo used to strike objects and produce sound, and the Ring Board contains circuitry to drive an old-fashioned ringing phone. The last sound type was a specific ask by our sponsor, and upon researching how phone bells are driven (using ~+/-100V square waves), we decided that this warranted its own separate module.

2.1 Design Procedure

2.1.1 Relay Board Hardware

For the relay board, we opted to use a secondary off-the-shelf board with high-power relays to switch wall-powered sound-producing devices on and off. The only inputs needed from the communication module is a GPIO to toggle to the relays, which is supplied for a set duration by the ESP32. We opted for relays rather than MOSFETs because as mentioned in the high-level requirements, latency itself is not a concern, so we could use slower relays for operation, which are also cheaper and less likely to fail in the case of faults.

Due to the use of wall voltage in this module, for safety reasons we opted to use a terminal rail and terminal blocks to provide high isolation across nodes and simplify wiring substantially. For additional safety, the hardware was enclosed by a laser cut acrylic enclosure (shown in Figure 4) which still allowed access to the hardware in case modifications are needed, and the sound devices were plugged into mounted wall plugs to eliminate the need to modify any cords. The actual circuit schematic and board layout can be found in Appendix A Figure 13 and 14.

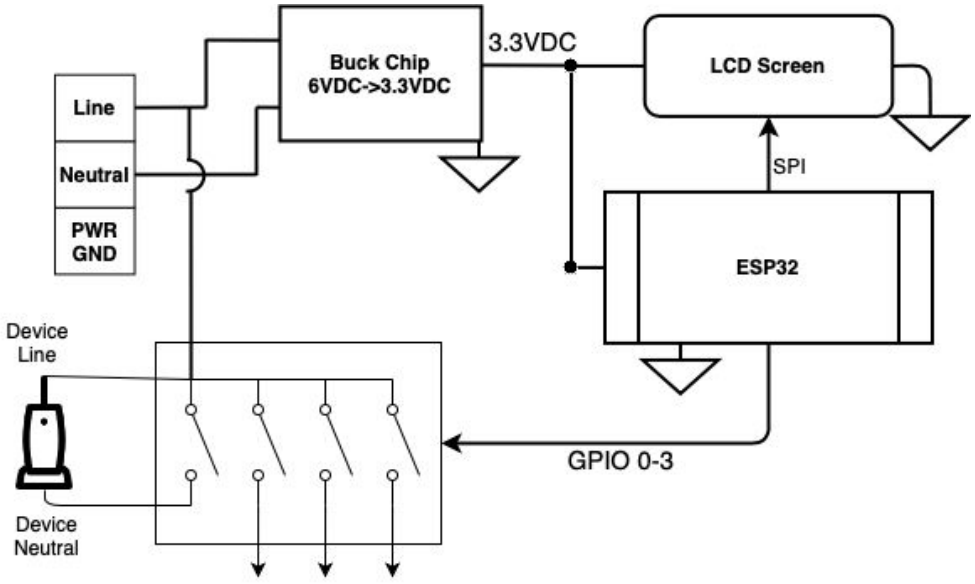

Figure 6. Relay Board Schematic

2.1.2 Servo Board Hardware

For the servo board, we had a simple goal of making sound by striking the object, and making the board to be free on placing. To achieve the first goal of striking, we are using the servo motor, which will be controlled by the GPIO of the microcontroller, ESP32, by sending 3.3VDC of PWM (pulse width modulation). For the second goal, this board has an independent power source of battery pack that outputs 6VDC, and this gives power to LCD screen, ESP32 and the servo motor. Since 3.3VDC is the power that is needed to turn on the LCD Screen and ESP32, the board uses a buck chip to lower the power source from 6VDC to 3.3VDC.

Since the board is relatively dependent on the power source of the battery pack to become independent from the position of placing, we needed a way to show to the user that the battery is low or not on the board. For this, the board uses a voltage sense circuit, which will detect the specific power that the power source outputs and changes the background color of the LCD screen as high for green and low for red. Figure 7 shows the flow of the servo board and helps to understand the above description easily. Additionally, to create a neat appearance and prevent short from surface contact, the entire board was enclosed in a laser cut acrylic enclosure as shown in Figure 3. The actual circuit schematic and board layout can be found in Appendix A Figure 15 and 16.

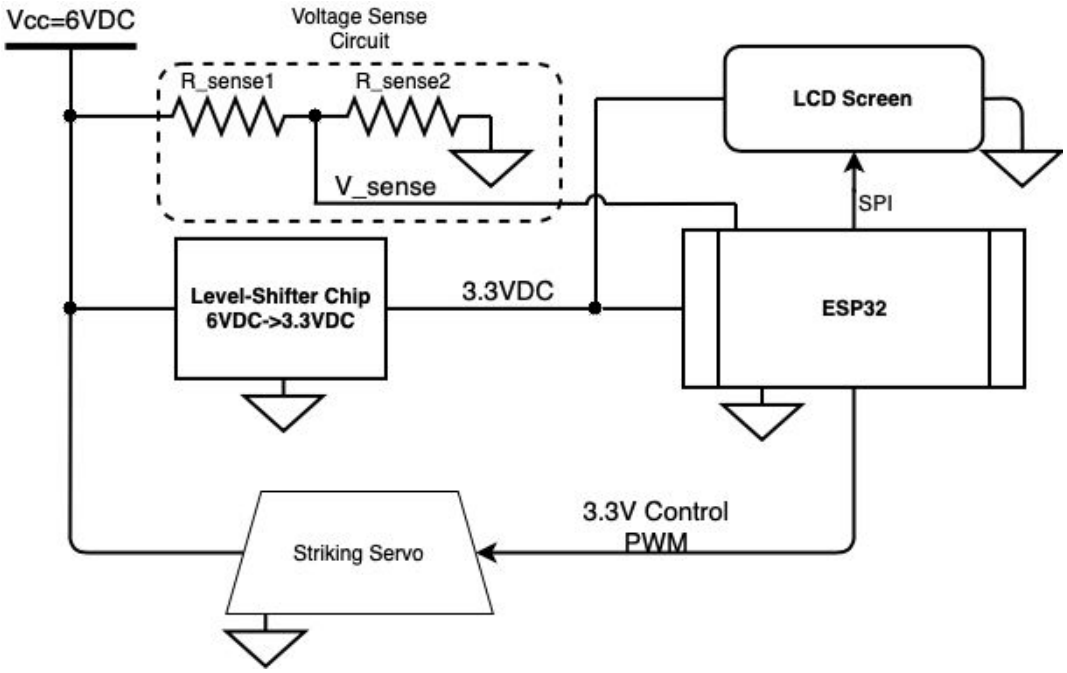

Figure 7. Servo Type Board Schematic

2.1.3 Ring Board Hardware

Our sponsor specifically requested a vintage phone ringing sound, and after researching how they worked and disassembling the phone I found at a thrift shop, we concluded that the hardware is driven by a ~100V, low frequency square wave [8]. This square wave causes an alternating current in a solenoid, which according to Ampere's law [9] creates an oscillating magnetic field and thus causes the metal hammer between the bells to oscillate and ultimately ring the pair of bells. To keep materials across

boards similar, we opted to use the same 6VDC battery pack as the servo board and step it up to around ~60VDC.

In order to alternate positive and negative 60V across the solenoid with MOSFETs, an H-bridge is needed since the required gate to source voltages for the FETs becomes fairly high if the source of the FET is at 60VDC. See Fig. 8 for the layout of the ringing hardware driving circuitry. Additionally, using an H-bridge allows for the prevention of shorting our 60VDC output from the boost chip by adding a several nanosecond delay between switching. Finally, there is also a capacitor in series with the ringing solenoid, tuned at resonance to create an LC resonant circuit. The equation for the resonant frequency is as follows:

$$\omega_0 = \frac{1}{\sqrt{LC}}$$

Equation 1: Resonant freq. for an LC circuit

Which can be rearranged to:

$$C_{tuned} = \frac{1}{\omega^2 L}$$

Equation 2: Tuned capacitor value for resonance

Creating a resonant LC circuit minimizes power losses and thus heat generation by maximizing parallel reactance and minimizing series reactance, and the power is able to oscillate between the inductor and capacitor efficiently [10]. The actual circuit schematic and board layout can be found in Appendix A Figure 17 and 18.

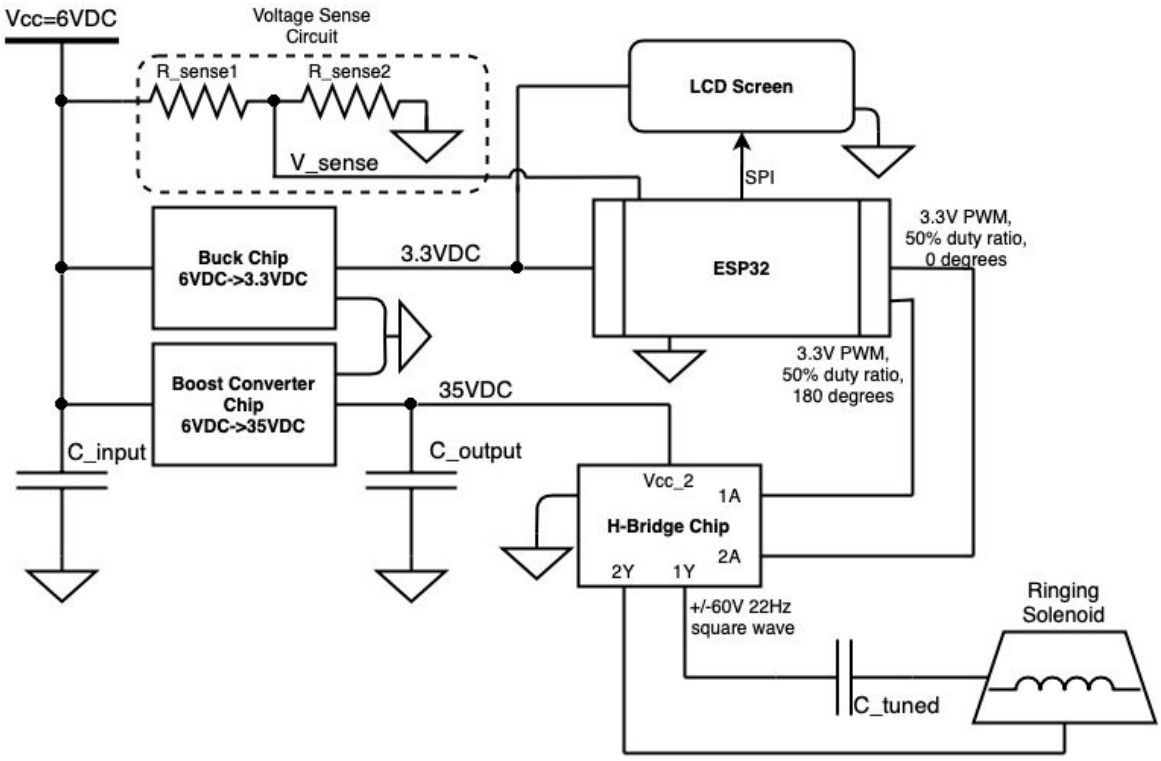

Figure 8. Ringer Type Board Schematic

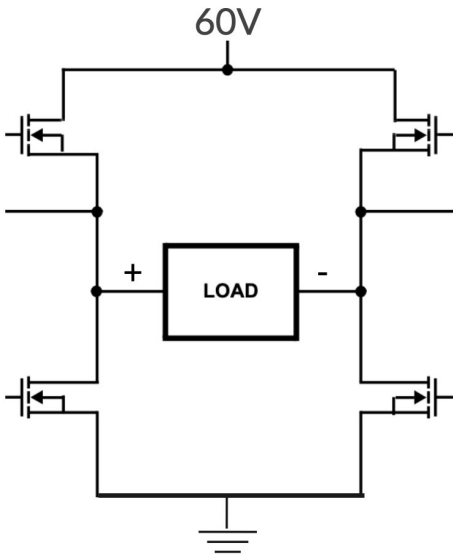

Figure 9. H-Bridge and MOSFET Circuit Diagram

2.1.4 Battery Sense Hardware

Since the ESP32 input pins have a max voltage rating of 3.3V, in order to sense the battery voltage to estimate the battery status, we had to use a voltage divider to scale it down. Also, using a voltage divider is advantageous because the sense circuitry will draw little current. The servo and ring boards, as seen in Figure 7 and Figure 8, both have the battery voltage divider, and the voltage at the ESP32 input can be found using the following equation:

$$V_{sense} = \frac{R_2}{R_1+R_2} V_{battery}$$

Equation 3: Voltage divider equation

Which can be reorganized for the battery voltage:

$$V_{battery} = \frac{R_1+R_2}{R_2} V_{sense}$$

Equation 4: Battery voltage from voltage divider

2.1.5 Jitter Buffer

For our implementation of the jitter buffer, we planned to calculate the latency of our incoming commands from the Host PC using a timestamp encoded into the command, and then delay the execution of the commands until by the difference between the latency and the size of the buffer. This method creates a floor value for the latency that messages must satisfy to be executed. By making this buffer large enough, messages are almost always buffered and hence execute with little variation in latency.

Originally, we intended to calculate the latency by deducing the time difference between the ESP32 microcontroller and the Host PC. Using the time difference between the two computers, the timestamp on the message, and the internal clock of the ESP32, we could calculate the latency of the message:

$$Latency_{actual} = Time_{esp} + Time\ Difference - Timestamp_{Host}$$

Equation 5: calculating latency

2.1.6 Communication Software

For communication we needed to use a protocol that would facilitate communication over WiFi between the Host and the boards. The two protocols we considered were UDP and TCP. We initially planned to use UDP since it was simpler to implement, had greater speed, and we had prior exposure with this protocol. However, as we continued to think about our design, we decided to use TCP because of its reliability. Reliability for us was a more important factor than speed, because if one of our devices does not receive a single command during a test by the IALL, that whole test needs to be repeated. Although TCP is slower than UDP, that would not hinder our project goals since we use our jitter buffer to slow down messages anyway.

2.1.6 LCD display

In order to communicate with the user directly on the state of the boards, we decided to use the SPI interface on the ESP32 to control an LCD display. We wanted to show on this board the IP and Port of the board for the Host to establish communication, the current state of the device and most recent commands for easier use and troubleshooting. We also wanted to display the status of the batteries to the user. After getting familiar with the Sparkfun SerLCD, we decided to change the backlight on the LCD display to indicate that the batteries need to be charged.

2.2 Design Details

2.2.1 Relay Board Hardware

The method for turning wall-powered devices on and off was designed in the same way wall switches work. The line input to the devices is held at 120VAC, while the relays toggle shorting and opening the connection from line to neutral to turn the device on or off, respectively. The ground off all of the devices were tied together to the power ground connected to the wall and also tied to the metal body of the terminal rail to ensure a safe path to ground for everything in the case of a fault.

2.2.2 Servo Board Hardware

To strike the object, the servo motor needs to move from 0° to 180° repeatedly while it is turned on. Since every motor has its own PWM to move its own position, we first had to find out how Futaba S3003 servo motor's PWM looks like. Below Figure X. shows how the output of PWM should look like from the GPIO of the ESP32 to control the motor as we desired. With 20 ms of the duty cycle, the high signal of 3.3VDC needs to repeatedly hit from 0 ms and around 3 ms, 1 ms for 0° and 2 ms for 180°.

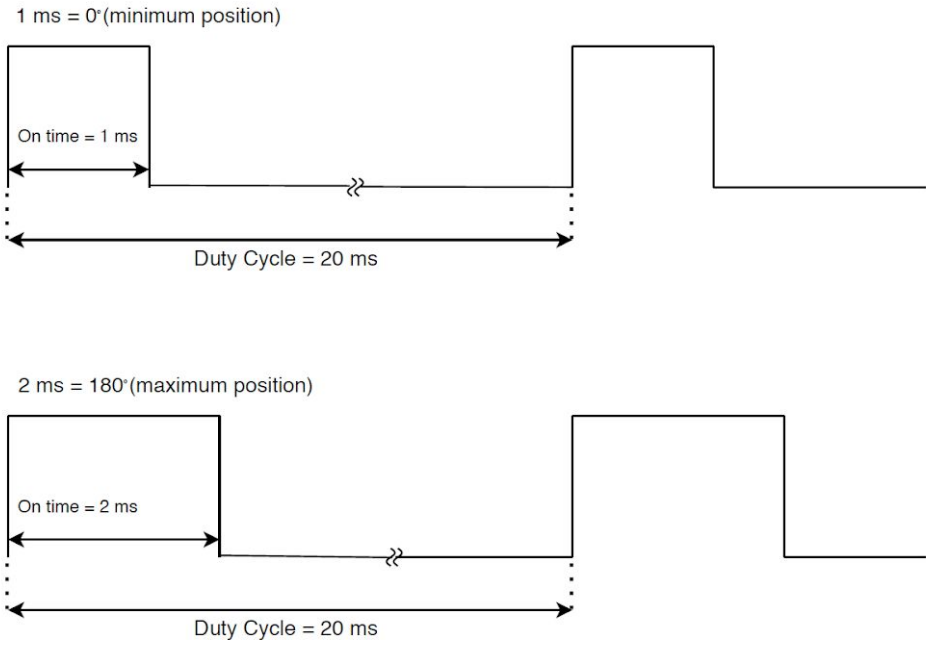

Figure 10. Pulse Width Modulation for Servo Motor

2.2.3 Ring Board Hardware

The solenoid's inductance was measured to be 41.6842 H, and the frequency we chose to operate at was 22Hz, so the value for the tuned capacitor can be plugged into Equation 2 as follows:

$$C_{tuned} = \frac{1}{(2\pi 22)^2 41.6842} = 1.2\mu F$$

2.2.4 Battery Sense Hardware

The ESP32 can map the sense voltage serial value to a 0V to 3.3V range in order to convert it to a meaningful value, and then use Equation 4 to estimate the battery voltage and therefore state of charge. Plugging in the resistor values (R1 as 1MΩ and R2as 100kΩ) we get:

$$V_{battery} = \frac{1M+100K}{100K} V_{sense} = 11 V_{sense}$$

2.2.5 Jitter Buffer

When implementing the jitter buffer, we encountered issues trying to calculate the time difference between the ESP32 and Host PC. After trying a few different approaches we found online with inconsistent results, we decided to be a little more creative about how we calculate the latency. The key of our approach is that we do not need the actual latency of the commands, only the variation of the latency, thus we can use an approximated latency by using a constant reference. When the ESP32 and the Host PC initialize their TCP connection, synchronize their clocks by creating timers on both the Host and the ESP32. These clocks are not entirely synchronized because of the initial latency of the first TCP message, but this initial latency is constant and thus can be used as a reference for approximating the latency. We have outlined the method below.

Initialize Timers on both Host and ESP32

$$Time_{esp} = Time_{Host} + Latency_0$$

When command received, estimate latency by assuming $Time_{esp} == Time_{Host}$

$$Latency_{est} = Time_{esp} - Timestamp_{pc}$$
$$= Latency_{actual} + Latency_0$$

Since $Latency_0$ is constant, we can use $Latency_{est}$ to measuring variation

Equation 2: calculating approximate latency

One additional aspect of the jitter buffer that should be noted is that network jitter can vary from one network to another. This means that a buffer that is large enough to work on my network may not be large enough to work on the IALL's network. For this reason, we decided to include in our python module a command for creating custom buffer sizes. This way, if the user is experiencing bad performance, they can increase the size of the buffer.

2.2.6 Communication Software

For implementing communication between the Host and the ESP32 using TCP, we created a finite state machine that would bring the board to an operational state only once the board had established communication with the Host. Using this FSM, the boards can always reinitialize communication if for any reason WiFi connectivity or the Host become unavailable without having to reboot the device.

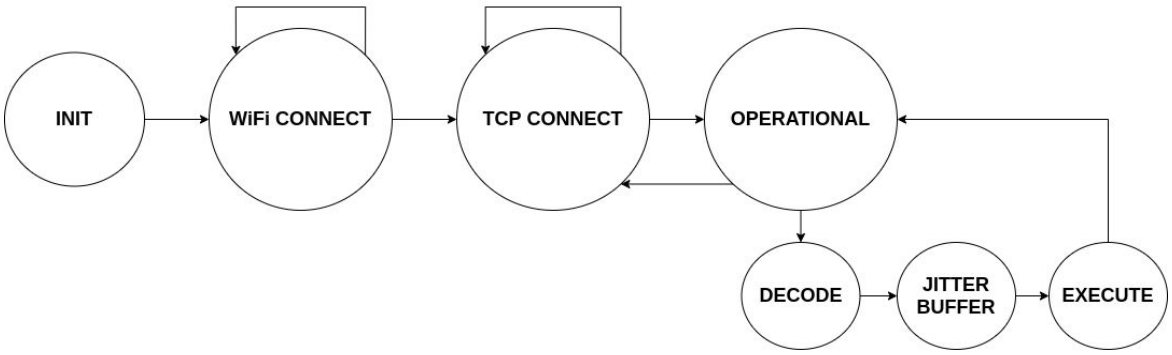

Figure 11. Finite state machine of ESP32 for establishing communication

The following is a detailed explanation of the states and their transitions:

1. **INIT -** Initialize data structures for WiFi, TCP, and SPI interfaces.
2. **WiFi CONNECT -** Repeatedly attempt to connect to WiFi using the current credentials. Display on LCD "Connecting to WiFi..."
3. **TCP CONNECT -** Wait for a TCP client to connect. Display the IP and Port on LCD for the user to initialize an SCPSS object in Host that establishes communication with the ESP32.
4. **OPERATIONAL -** In this state we are waiting for an incoming stream of bytes from the Host. If the TCP connection is lost, we move back to **TCP CONNECT**. Once a message arrives, the device moves to the decode state.

5. **DECODE -** The decode state uses our string encoding to get commands along with the time stamp.

   Encoding Scheme:

   Example: C10T1534E

   C - Indicates the command. The first character is taken as the on or off bit, the next character is an optional field that is used to indicate which of the four relays to activate on the Relay board.

   T - Indicates the timestamp in milliseconds on the Host when the message was sent. This numeric string is converted to an integer.

   E - Indicates the termination of the current command.

6. **JITTER BUFFER -** Delays execution of command if approximate latency is smaller than the buffer size, otherwise continues to **EXECUTE.**
7. **EXECUTE -** The current command is executed on the respective mechanism and the ESP32 returns to **OPERATIONAL** for the next command.

2.2.6 LCD display

In implementing the SPI interface we countered some confusion finding the pins on the ESP32 for SPI, and this confusion was exacerbated due to using the Arduino IDE for programming the ESP32, since the Arduino core for ESP32 has its own pin configuration. Fortunately, the ESP32 has the capability to use almost any pin for any of its compatible interfaces. We used the Arduino SPI library to set up a virtual SPI data structure that would let us use any pins for the SPI interface.

3 Verification

3.1 Relay Board

Following assembly and component level testing (see Appendix B, Table 3), as well as testing for any shorts in the circuit with a handheld multimeter, we plugged the system into wall power via power strip with an internal circuit breaker in case of any unintentional high currents. Next, we plugged a lamp into each of the four outlets to make testing quieter, and manually triggered the input pins with a voltage source and checked that the lamps turned on and off as expected. Once each relay was checked, we connected the ESP32 to header inputs on the board and connected to it on our PC. We then triggered relays once at a time and in combination in a basic test script and manually checked that the lamps were toggling as expected. Finally, to verify our current rating and high-level requirements, we plugged in a blender and a vacuum cleaner and ran our test scripts and checked for the response, and also moved our specified six foot distance away from the host PC. The relay board module overall was a success, and we were able to control two different sound devices, with the ability to expand to four devices, from our six foot distance requirement.

3.2 Servo Board

To accomplish level testing (see Appendix B, Table 4), we first had to check if the output of PWM from the ESP32 is the same as we expected like Figure 10. With using the oscilloscope in the lab, we correctly outputted the PWM with the duty cycle of 20 ms that repeatedly hit 0 ms and 3 ms from the specific GPIO of ESP32. This level test was checking mostly the software part of the servo board with simply connecting the motor to ESP32. Another software part in verification is being able to change the background color of LCD when battery is low. Since we had a battery pack that has a low battery, we were able to find out that checking the 6V battery pack's voltage whether it is lower than 5V was enough to detect that power source is low or not through using the multimeter. By implementing a checking output of battery less than 5V with ESP32, we were able to change the color of LCD background by green or red depending on the battery status.

Servo board also had various hardware subsystems (see Appendix B Table 2) such as buck converter, LCD display and battery sense circuit that were also used in ring board. For the buck converter, we were able to check the output of 3.3V from converting 6V of the battery pack using the multimeter. Also using the multimeter, we were able to verify that the battery sense circuit outputs 3.3V. Last part of the verification is the LCD Display, and the printing command on the LCD display is verified by checking the output on the display when the command was inputted from the host. Putting all these software and hardware subsystems together, we were able to build the servo board that turns on and off with various functions.

3.3 Ring Board

During verification, we ran into an issue with the output of the H-bridge on this board that caused the ringing hardware to fail. Instead of a neat +/- 60V square wave, there was high frequency and high amplitude ringing at the transitions of the waveform. This effectively created a much higher frequency signal that caused the direction of the magnetic field to change at the same frequency, around 100kHz, and the force on the striking mechanism was not in one direction long enough to cause it to move far enough to strike the bells. The desired square wave was at 22Hz, and applied a force on the hammer long

enough to strike one bell before the magnetic field changed directions and the hammer was forced in the other direction, striking the other bell, and so on, oscillating for the requested duration.

We believe there are two main reasons for this undesired wave form: PCB layout for the boost converter and the MOSFETs. In the PCB design the boost converter was spaced out more than necessary, and through hole components were chosen in many places in order to enable trying out components with different parameters, and this created undesirable inductances because of long traces and excess wire. From [11], we case see that in switching converters, when the unwanted inductances are charged in the first part of the switching cycle, when the switches alternate the inductances cause resonance with parasitic capacitances across the switches and thus create large voltage spikes which are then dampened. Also, since this ringing was so large and the boost converter and H-bridge were only rated for up to 100V, we went through all of our spares of these components while debugging

Given the accelerated nature of the course, we were not able to make an additional PCB order to try to mediate these issues. Given more time though, the first objective would be to make the traces for the boost converter as short as possible and choose as many surface mount parts as we can, and then add pads across MOSFETs for the addition of snubber circuits to suppress voltage spikes. Next, we would measure parasitic inductances and impedance and calculate values for the snubber circuitry. As before, we would also be sure to put the max rated capacitance at the input and output of the converter. All these efforts should eliminate the large ringing we observed and hopefully get the ringing board working.

3.4 Jitter Buffer

For verification of the Jitter buffer, instead of trying to record the activation of the sound, we made use of the serial communication monitor to report the approximate latency and the time passed from the timestamp to the execution of the command. To quantify the effectiveness of differently sized buffers, we performed 5 rounds of sending 200 commands for increasingly larger buffers and then took an average of their performance. As you can see below, for smaller buffers, some messages arrive with over 30ms variation from the buffer size, while when using larger buffers, like 200ms, the buffer covers all incoming messages and we have achieved our high-level requirement.

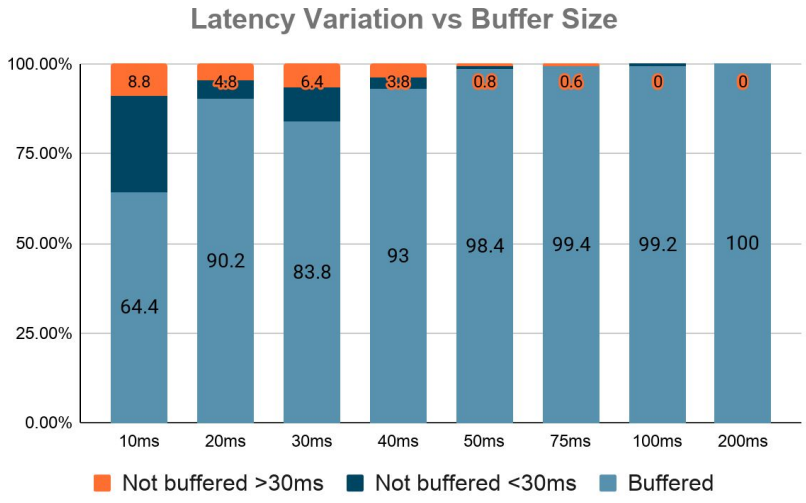

Figure 12. Performance of Jitter buffer for different buffer sizes.

4 Costs and Schedule

A summary of total hardware and labor costs follows.

4.1 Parts

In Table 1, you can find the cost breakdown for the project, the per-unit cost difference for bulk purchasing, and the total costs.

Table 1. Parts Costs

Part	Manufacturer	Retail Cost ($)	Bulk Purchase Cost ($)	Actual Cost ($)
Terminal Rail Power Converter (120VAC - 3.3VDC) (PSK-10W-3-DIN)	CUI Inc.	$21.60	$19.89	**$21.60**
ESP32-S2-SAOLA1	Espressif Systems	$8	$8	**$24**
LCD Display	Sparkfun	$19.95	$19.95	**$59.85**
Wall cord (EPS614-ND)	Inventus Power	$4.03	$2.34	**$4.03**
Wall input socket (Q335-ND)	Qualtek	$0.88	$0.49	**$0.88**
Power outlet (Q227-ND)	Qualtek	$0.99	$0.55	**$3.96**
Terminal Rail (277-2293-ND)	Phoenix Contact	$5.43	$4.06	**$5.43**
Terminal Rail Partitions (277-2040-ND)	Altech Corp.	$0.80	$0.55	**$8.00**
Terminal blocks, various colors (277-3243-ND)	Phoenix Contact	$1.34	$0.92	**$20.10**
Terminal block jumper pieces (3030161)	Phoenix Contact	$0.67	$0.46	**$6.70**
Ground terminal block (277-17411-ND)	Phoenix Contact	$4.51	$3.11	**$4.51**
Terminal block end plate (277-2038-ND)	Phoenix Contact	$0.64	$0.47	**$1.28**
6V AA Ni-MH Rechargeable Batteries	Amazon	$2.14	$2.14	**$17.11**

4 AA Battery Holder	LampVPath (Amazon)	$2.99	$2.99	**$5.98**
Servo Motor FutabaS3003	Futaba	$14.70	$10.03	**$14.70**
Boost converter (6VDC to 30VDC) (MC34063ADR2G)	ON Semiconductor	$0.60	$0.20	**$3.00**
6VDC to 3.3VDC buck chip (AP1509-33SG-13)	Semtech Corp.	$1.60	$0.64	**$9.60**
180uH inductor (B82144F2184J000)	TDK Electronics	$0.84	$0.36	**$3.36**
Diodes (MBRA140T3G)	ON Semiconductor	$0.41	$0.10	**$4.10**
H-Bridge (HIP4082IBZ)	Renessas	$4.40	$2.21	**$17.60**
5 terminal output (2368-25-E700-05-ND)	NTE Electronics, Inc	$1.27	$1.27	**$1.27**
2 terminal output (2368-25-E700-02-ND)	NTE Electronics, Inc	$0.51	$0.51	**$2.55**
Header pins set (S1011EC-40-ND)	Sullins Connector Solutions	$0.66	$0.34	**$1.98**
22KΩ Resistor	n/a	(already have)	n/a	**n/a**
44KΩ Resistor	n/a	(already have)	n/a	**n/a**
High Voltage Quad Relay Board	Amazon	$22.76	$22.76	**$22.76**
Various capacitors	n/a	(already have)	n/a	**n/a**
Vintage phone hardware	n/a	$3.00	n/a	**$3.00**
Plexiglass sheet	Home Depot	$21.99	n/a	**$43.98**
Total:				**$311.32**

4.2 Labor

Our fixed development costs are estimated to be $40/hour, 10 hours/week for three people. We consider approximately 60% of our final design in this semester (16 weeks).

$$3 * \frac{\$40}{hr} * \frac{10hr}{wk} * \frac{16\ wks}{0.6} * 2.5 = \$80,000$$

4.3 Schedule

Week	Micki	Ben	Won Woo
10/5	Finalize relay board PCB	Set up Github	Set up Github
10/12	Finalize servo and ring board PCBs and parts	Create encoding for message structure	Set up ESP32S2
10/19	Breadboard with components then test on PCBs	Firmware and Python code for TCP communication	Work on Servo Motor with ESp32S2
10/26	Reworked ring PCB	State machine for embedded software	Connect Motor with TCP communication
11/2	Finalize hardware, build enclosures	Jitter buffer development	Finalize Servo Board and Ring Board
11/9	Prepare Mock Demo	Jitter buffer testing	Prepare Mock Demo
11/16	Prepare Demonstration	Prepare Demonstration	Prepare Demonstration
11/23	Begin Final Report	Begin Final Report	Begin Final Report
11/30	Prepare Presentation	Prepare Presentation	Prepare Presentation
12/7	Finalize Final Report	Finalize Final Report	Finalize Final Report

5 Conclusion

5.1 Accomplishments

The relay board and servo board can be considered as the finished part in the project. We are certain that our sponsor will easily use our project as soon as he gets the boards and their documentation. Since there has been a full enclosure for both of relay and servo boards, we are certain that it will protect any users from the high voltage of electricity, and boards are already prepared to be used in any kind of demo or research for IALL.

5.2 Uncertainties

The only uncertain part is the project's ability of using the school Wifi. For the demo of this project, we simply used one hotspot from the phone; however, using school Wifi can be necessary for IALL to use the project for their research.

5.3 Ethical considerations

Since the relay-type board is connected to the wall power, we have to care about any components that are connected to this board since there is a possibility of getting an electric shock. In order to avoid this, we used a terminal block (aka terminal rail) to avoid any kinds of poor connected wires. Using itl led to a convenient and safer way to distribute power from a single input source of the wall power to multiple outputs. This ethical consideration was inspired by safety concerns from an implementation of the IEEE Code of Ethics Section I.1, "disclose promptly factors that might endanger the public or the environment" [6]. We are also considering ordering a commercial relay board to separate between sensitive components like the microcontroller and the AC power. Even though the project was built with this ethic consideration, the user still needs to be careful when using the project since there can be a potential danger while using it.

5.4 Future work

Revising the ringing module in order to get it functioning would be the first priority for future work, and would entail redesigning portions of the PCB and choosing several replacement components. Once this board is finalized, a new plexiglass enclosure will need to be cut to enclose it in a similar manner as the servo board.

On the software side, the Git repository containing the firmware and module for using the system need to have documentation finalized in order to ensure there is no confusion in using the system, and if issues arise, the lab can debug the code as needed.

Finally, we would like to expand the functionality of the jitter buffer by improving the method of clock synchronization between the microcontrollers and the Host PC, so that we can calculate the actual latency of the commands and provide consistency across instances of the devices. At the moment, we can only guarantee consistency within the same test on a single device.

References

[1] Miller, A., 2020. *Extending Hearing Aid Testing Beyond The Walls Of The Sound Booth | Phonak Audiology Blog - Phonak Pro - Life Is On*. [online] Phonak Audiology Blog - Phonak Pro - life is on. Available at: <https://audiologyblog.phonakpro.com/extending-hearing-aid-testing-beyond-the-walls-of-the-sound-booth/> [Accessed 28 September 2020].

[2] Staff, H., 2007. *Developing And Testing A Laboratory Sound System That Yields...*. [online] Hearing Review. Available at: <https://www.hearingreview.com/practice-building/practice-management/developing-and-testing-a-laboratory-sound-system-that-yields-accurate-real-world-results> [Accessed 28 September 2020].

[3] Physics Lecture Demonstration Facility. 2014. *How Does A Candle Flame Respond To A Sound Wave? - Question Of The Week 2014 Summer Girls Special Part 1*. [online] Available at: <https://lecdem.physics.umd.edu/question-of-the-week-archive/154-qotw-020-with-answer.html#:~:text=Sound%20propagates%20as%20a%20longitudinal,the%20speaker%20along%20its%20axis.> [Accessed 28 September 2020].

[4] Corey, R., 2019. Massive Distributed Microphone Array Dataset | Innovation In Augmented Listening Technology - University Of Illinois At Urbana-Champaign. [online] Innovation in Augmented Listening Technology. Available at: <https://publish.illinois.edu/augmentedlistening/massive-distributed-microphone-array-dataset/> [Accessed 2 November 2020].

[5] Wilson, M., 2019. *Network Jitter - What Is It And How To Monitor It With Software/Tools*. [online] PC & Network Downloads. Available at: <https://www.pcwdld.com/network-jitter> [Accessed 1 October 2020].

[6] Ieee.org. n.d. *IEEE Code Of Ethics*. [online] Available at: <https://www.ieee.org/about/corporate/governance/p7-8.html> [Accessed 28 September 2020].

[7] Campus Admin. Manual. 2001. *Appropriate Use Of Computers And Network Systems*. [online] Available at: <https://cam.illinois.edu/policies/fo-07/> [Accessed 29 September 2020].

[8] Telephone Ring Voltage Tech Bulletin. (2018, September 14). Retrieved December 08, 2020, from http://www.sandman.com/knowledgebase/ring-voltage-tech-bulletin

[9] N. N. Rao, Fundamentals of Electromagnetics for Electrical and Computer Engineering, Prentice-Hall, 2009.

[10] Kudeki & Munson, Analog Signals and Systems Prentice Hall, 2009.

[11] Ringing at switching nodes: Basic Knowledge. (n.d.). Retrieved December 09, 2020, from https://techweb.rohm.com/knowledge/dcdc/dcdc_pwm/dcdc_pwm03/3164

Appendix A Circuit Schematics and Board Layout

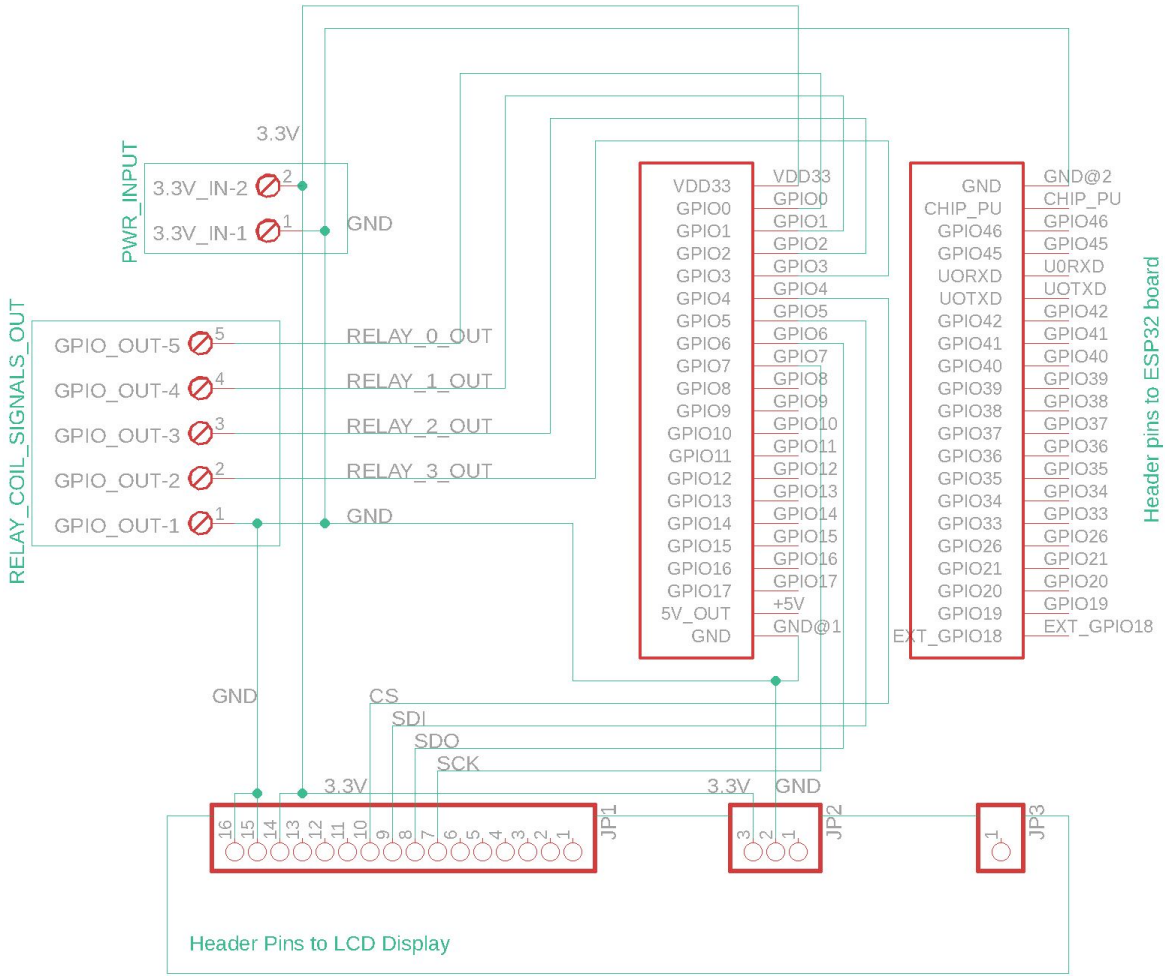

Figure 13. Circuit Schematic of the Relay Board

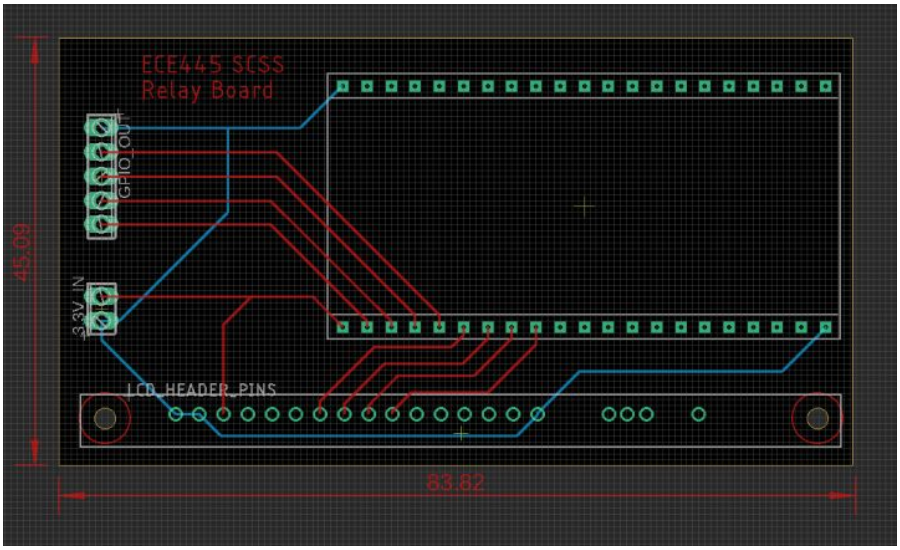

Figure 14. Relay Board Layout

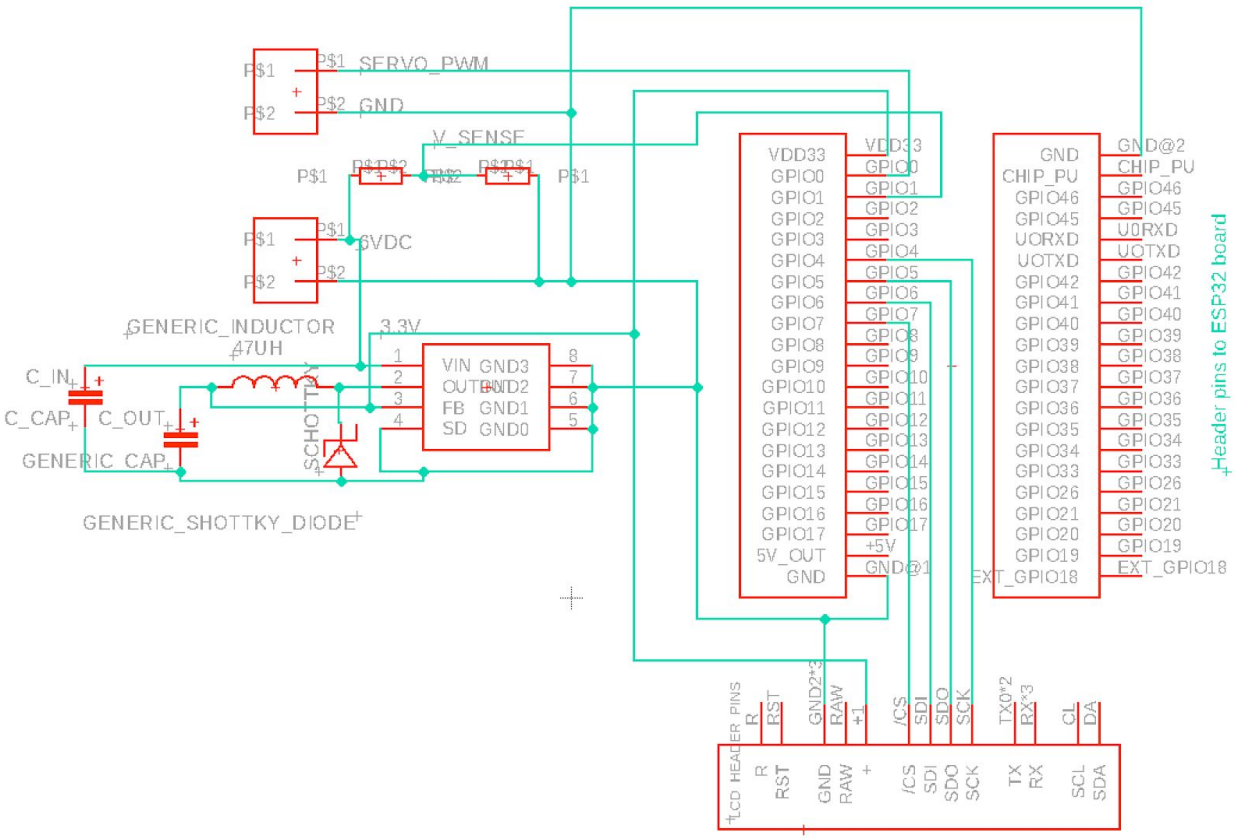

Figure 15. Circuit Schematic of the Servo Board

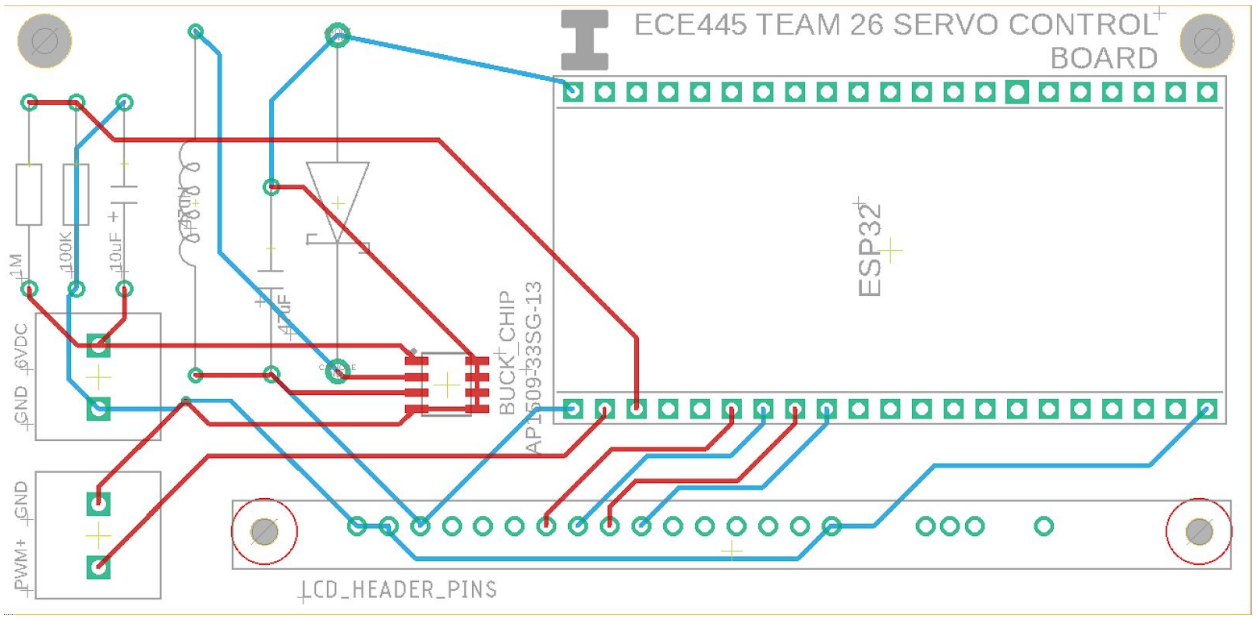

Figure 16. Servo Board Layout

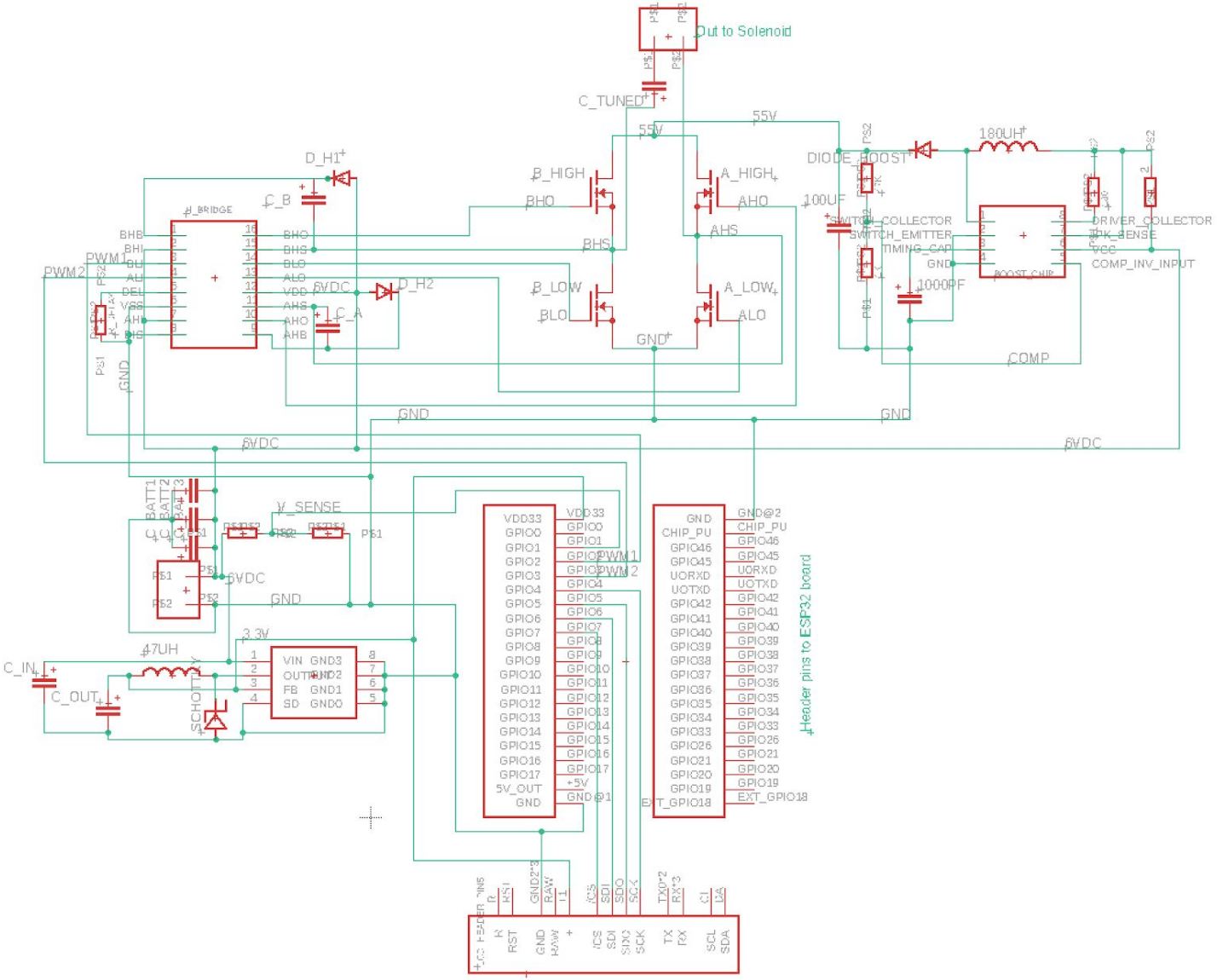

Figure 17. Circuit Schematic of Ring Board

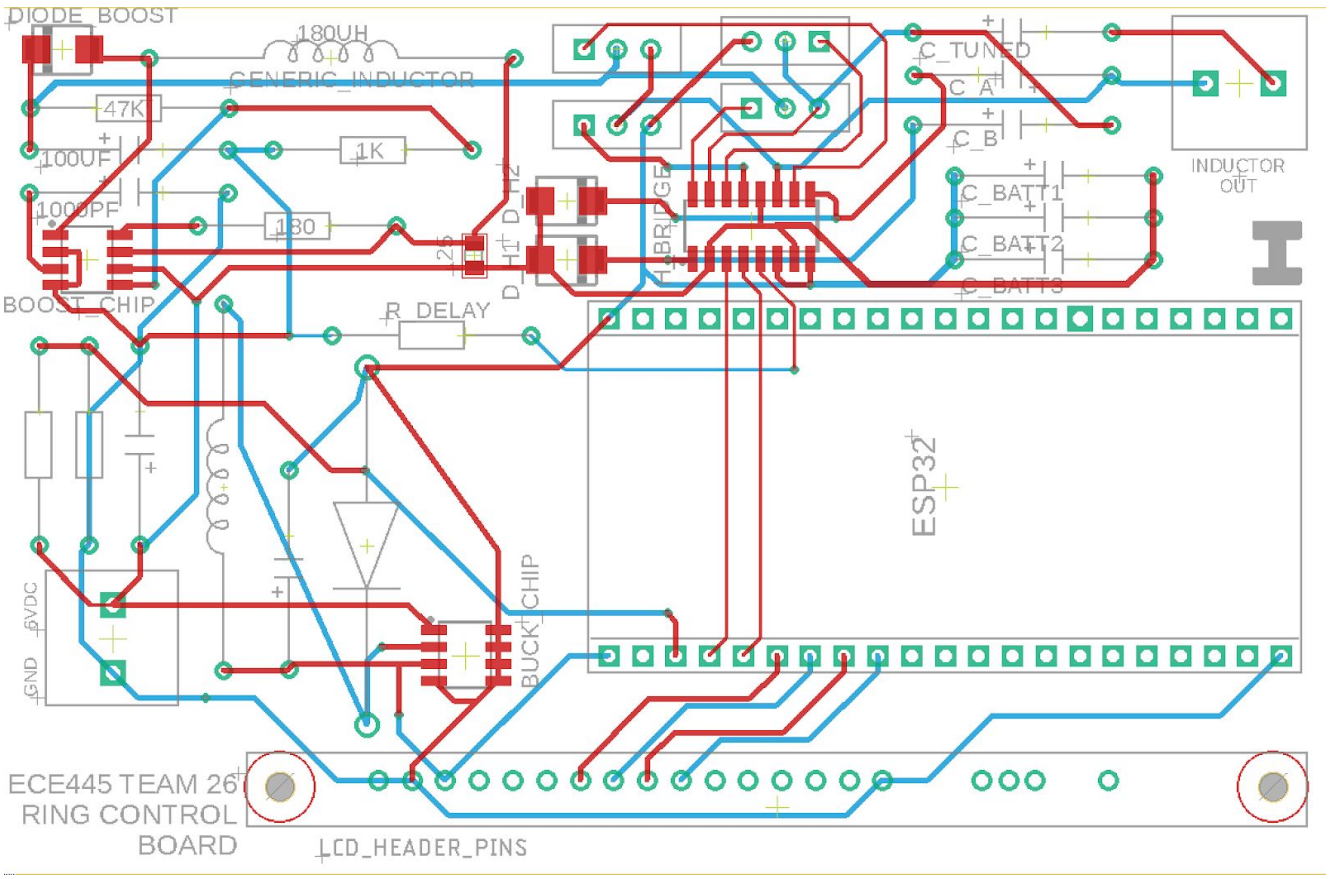

Figure 18. Ring Board Layout

Appendix B Requirement and Verification Tables

Table 2. System and Common Components Requirements and Verifications

Subsystem	Requirement	Verification	Verification status (Y or N)
ESP32	Jitter Buffer prevents network jitter greater than 30ms	Sent 200 messages over TCP with Buffer = 200ms	Y
	Output GPIO for relay board	Upload firmware setting GPIOs high and low and check with multimeter	Y
	Output PWMs for servo and ringing board	Upload firmware outputting PWMs with a set duty cycle and period and verify with oscilloscope	Y
LCD Display	Display Text and battery status over SPI	Displayed strings of various lengths and switched between charged and discharged batteries	Y
Buck Converter	Step down 6VDC input to 3.3V +/- 0.3VDC output	Attach to power supply and check with multimeter	Y
	Be able to supply at least 1.5 A	Check with multimeter while powering system	Y
Battery Sense Circuit	Scale battery voltage to a value the ESP32 can handle using a voltage divider	Convert serial input value on ESP32 and multiply by resistor divider equation and check that it's approximately equal to the measured battery voltage	Y
	Draws at most 5 uA	Checked with multimeter	Y

Table 3. Relay Board Requirements and Verifications

Subsystem	Requirement	Verification	Verification status (Y or N)
Relays	Open/close with GPIO command within 10ms	Checked on scope by applying square wave to the input pin and measure time difference from opening/closing and square wave rising/falling	Y

	Tolerate up to 12A for each relay at 120VAC	Tested with sound-producing devices drawing around 11 A	Y
Terminal Rail	Provide high isolation between circuit nodes	Check that multimeter shows overload or tens of megaohms of resistance between nodes	Y
Terminal Rail Mounted Power Converter	Step down 120VAC to 3.3VDC +/-5V	Plug into wall and check voltage output with multimeter	Y
	Be able to supply at least 1.5 A	Check with multimeter while powering system	Y

Table 4. Servo Board Requirements and Verifications

Subsystem	Requirement	Verification	Verification status (Y or N)
Servo Motor	Provides enough torque to make the strike audible for at least 10 ft	Tested with chime x10 in large room	Y

Table 5. Ring Board Requirements and Verifications

Subsystem	Requirement	Verification	Verification status (Y or N)
Boost Converter	Convert 6VDC to 60VDC +/-5V	Connect 6VDC power supply and probe output	Y
	Provide at least 1.5A current	Load test with electronic load in lab, setting current to 1.5A	Y
H Bridge	Output +/-60V square wave	Probe with oscilloscope	Y*
Vintage Phone Hardware	Ring for at least 30 seconds continuously	Send command from Python module for 30 seconds of sound and time ringing	N*

**See design section for analysis*