

Software Controlled Physical Sound Sources

ECE 445 Design Document

Won Woo Lyu - Wlyu2

Micki Rentauskas - Mar3

Ben Sisserman - Bens3

Group 26

TA: AJ Schroeder

9/29/20

Table of Contents

1 Introduction	2-3
1.1 Objective	2
1.2 Background	2
1.3 High-Level Requirements	3
2 Design	4 - 23
2.1 Block Diagram	5 - 8
2.2 Physical Overview	9 - 11
2.3 Requirements & Verification	11 - 16
2.4 Software Design Plot	17
2.5 Circuit Schematic	18 - 19
2.6 Tolerance Analysis	20 - 21
2.7 COVID-19 Contingency Plan	22
3 Costs	23 - 24
4 Schedule	25
5 Ethics & Safety	26
6 Citations	27

1 Introduction *(Written by Won and Ben)*

1.1 Objective

We will build three unique Software Controlled Physical Sound Source (SCPSS) devices for our client, postdoctoral student Ryan Corey, who conducts research on hearing aids at the *Illinois Augmented Listening Laboratory* (IALL). 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 smaller 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.

Currently, Ryan uses speakers placed throughout the lab to demo prototypes. The speakers use pre-recorded sounds that must be originally recorded in anechoic chambers, which can be expensive. Speakers also direct their audio output, unlike their physical counterparts. Our physical sound sources will create a sound wave that would go in all directions and reflect off of the walls of the room creating a new sound that a speaker would not be able to mimic since the sound wave of the speaker will be directly away from the speaker along its axis [3]. Our product solves these problems by allowing various physical and electronic devices to be used and controlled by our own Python Module, which can be easily utilized by the IALL.

1.2 Background

Ryan Corey, a postdoctoral student at University of Illinois at Urbana-Champaign, does research on audio-processing for noise cancelling hearing aids using speakers placed throughout a room to emulate various sound sources for testing. He has requested that we create software controlled real sound sources for his lab. In his lab, most of the sounds are outputted from a series of speakers, and sounds are produced by using specialized software. The below picture is from the article [4] of his team working on Cooperative Listening Devices, and they used 12 speakers to test Cooperative Listening Devices. These speakers can be replaced by our project, which will produce various real sounds while not using a speaker.



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.

2 Design *(Written by Micki and Ben)*

Our design will trigger the sound devices from commands sent via a Python module that will be running on the host PC. Our Python module will initialize all SCPSS on the network by listening for broadcasts from the devices to the local router. The devices will broadcast their device type (relay, servo, or ringer) and ID (MAC address) when turned on, and will continue broadcasting until they are powered off or initialized by the control unit. The devices will receive and decode commands sent from a router communicating with the PC using an ESP32 microcontroller.

There are three different board types, each will be powered by a 6V power source, and consists of different mechanisms for sound generation with electronic components on a printed circuit board (PCB). The relay mechanism contains relays which will toggle wall power to various “switch type” electronic devices like a vacuum or blender. The servo mechanism will use a PWM (pulse-width modulation) from the microcontroller to control a servo motor to strike an object with a mallet. Finally, the ringing mechanism utilizes an H-bridge to amplify PWMs to create an LC resonant circuit to control a vintage telephone ringer.

2.1 Block Diagram

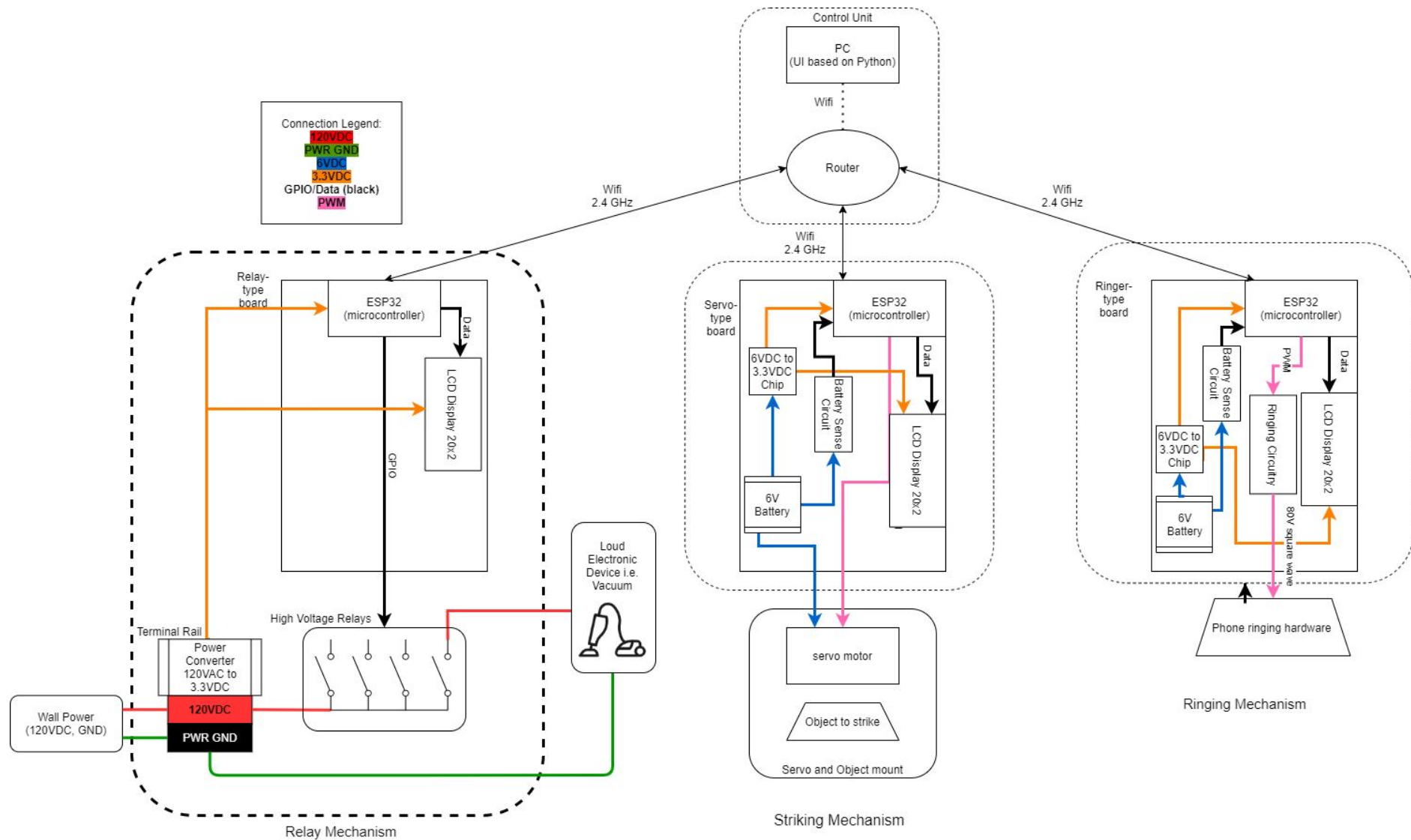


Figure 2. Block Diagram

2.1.1 Relay Board Block Diagram Enlarged

The Relay Board will not be powered via a 6V battery like the other two boards, because it already has power available from the power outlet which we can step-down to 3.3V for our microcontroller and LCD display. As you can see from the block diagram below, the ESP32 will manage the activation and deactivation of the relays using the microcontroller's GPIO pins. The 120VAC, PWR GND, and 120VAC to 3.3VDC converter will be placed on a terminal rail in order to safely manage the high voltage from the wall. The high voltage relays will be located on a separate, off-the-shelf board.

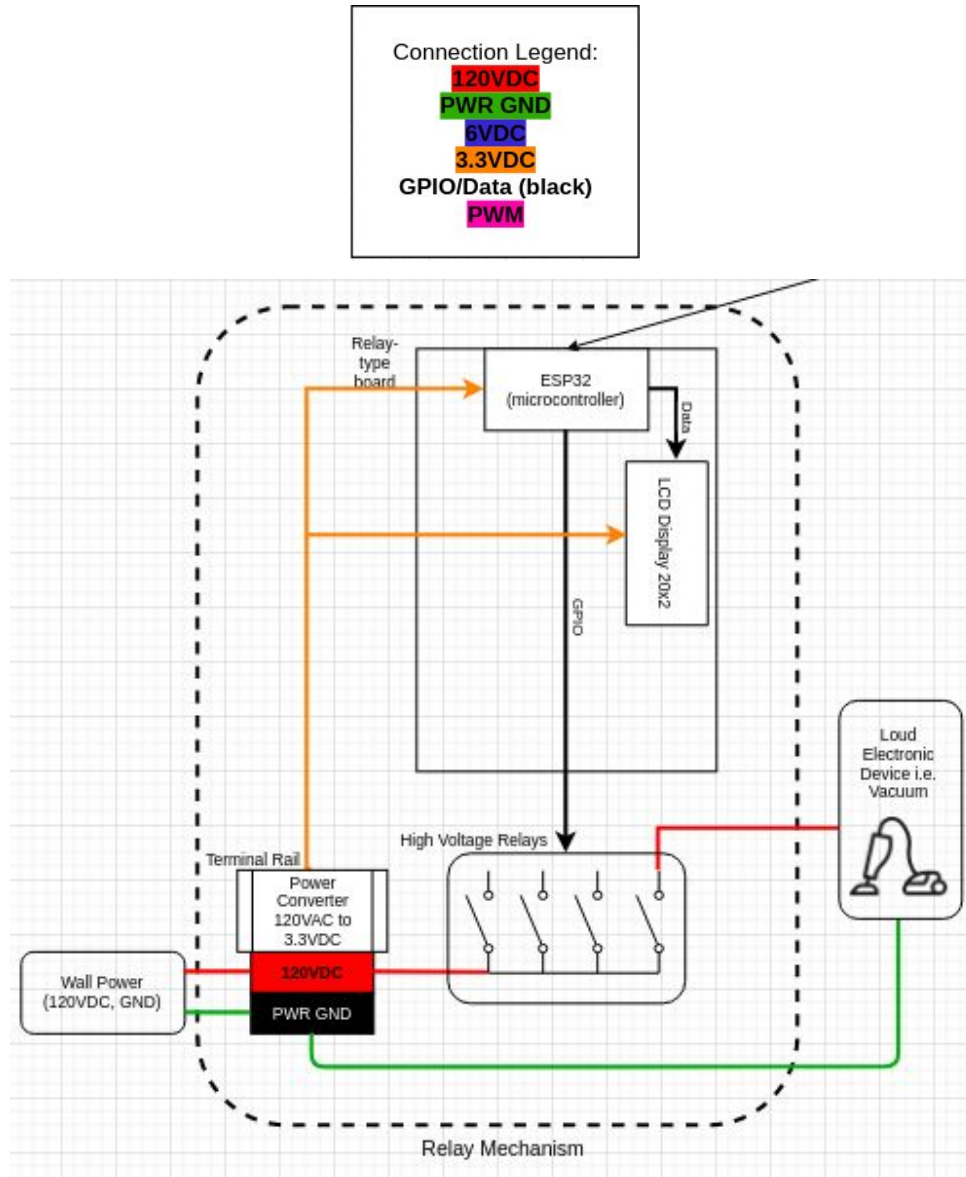


Figure 3. Enlarged Relay Board Block Diagram

2.1.2 Ringer Board Block Diagram Enlarged

The Ringer Board will be powered by a 6VDC battery and will two 180 degree out of phase square waves from the ESP32 and a 3.3VDC to 35VDC boost converter to effectively create a 60V PWM used in a resonant LC circuit to create a magnetic field from a very large inductor to oscillate the ringing hardware we extracted from an old phone. This type of sound was a specific request from our sponsor, and required reverse engineering an antique phone ringer.

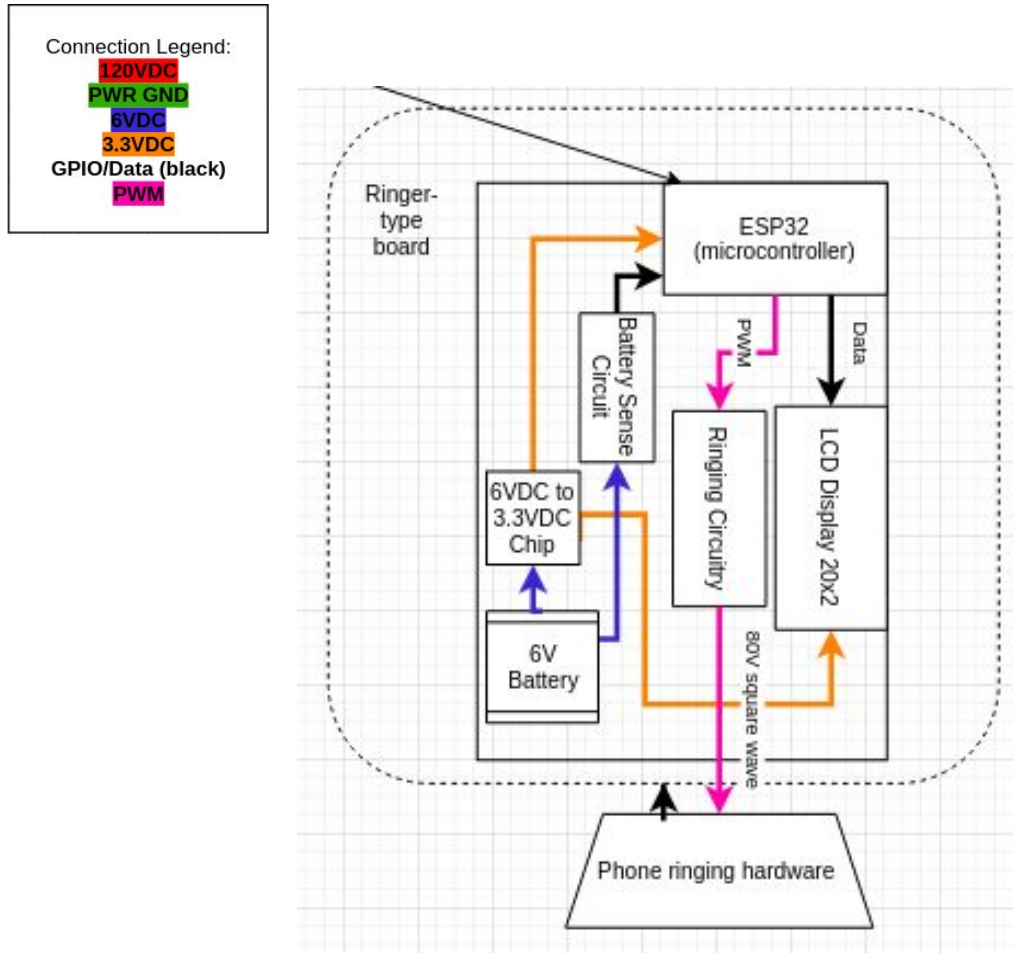


Figure 4. Enlarged Ring Board Block Diagram

2.1.3 Servo Board Block Diagram Enlarged

The Servo Board will include a mount to house the servo motor steadily as it strikes objects. The servo mount will be held down either by added weights or velcro and be attached to a mallet of some type to strike an object such as a drum or a bell. The microcontroller on this board will generate a pre-programmed PWM sequence to move the servo motor from a rest position to the position of the object to strike. This board will also be powered by a 6VDC battery.

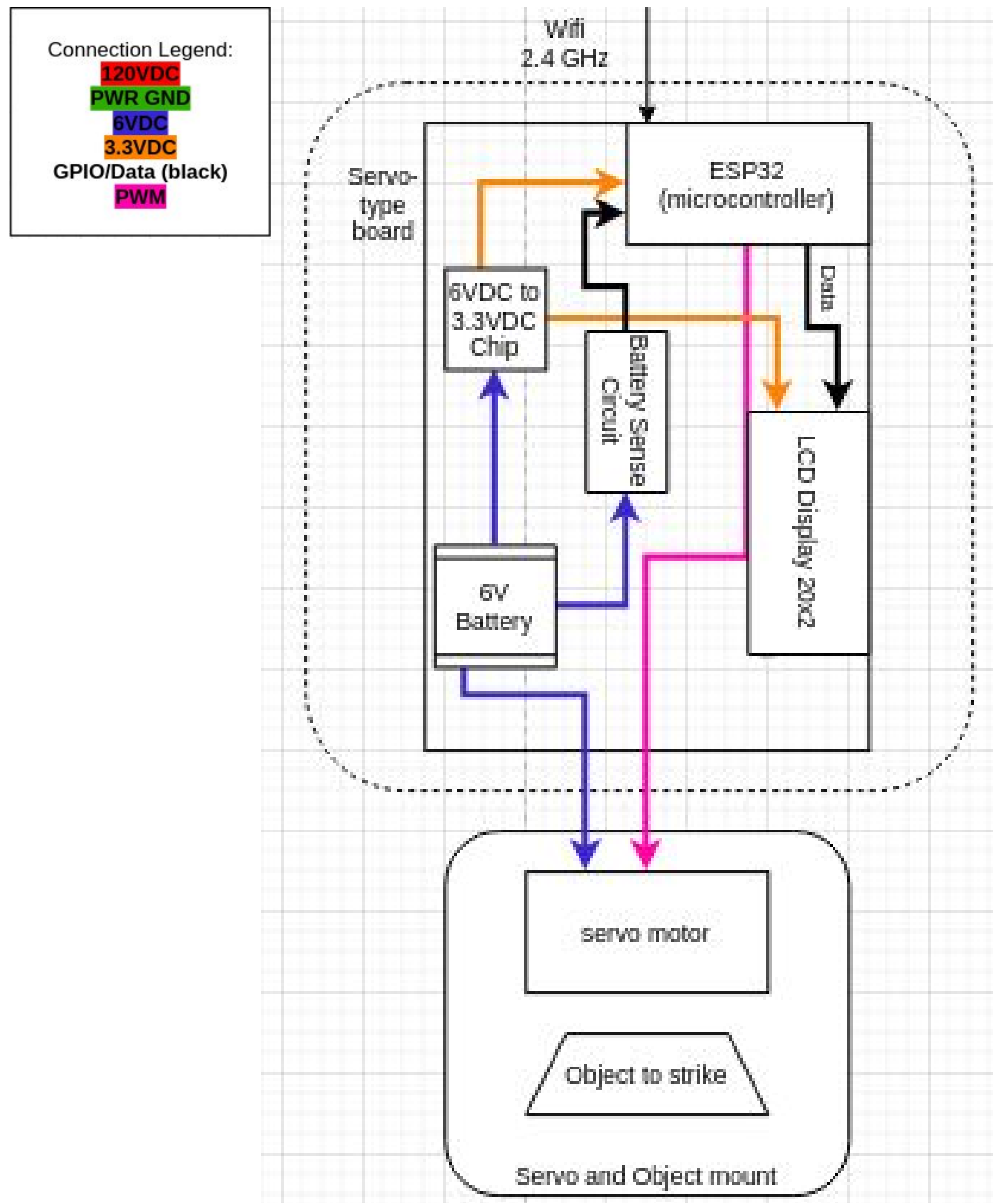


Figure 5. Enlarged Servo Board Block Diagram

2.2 Physical Design *(Written by Won, Ben and Micki)*

2.2.1 System Overview

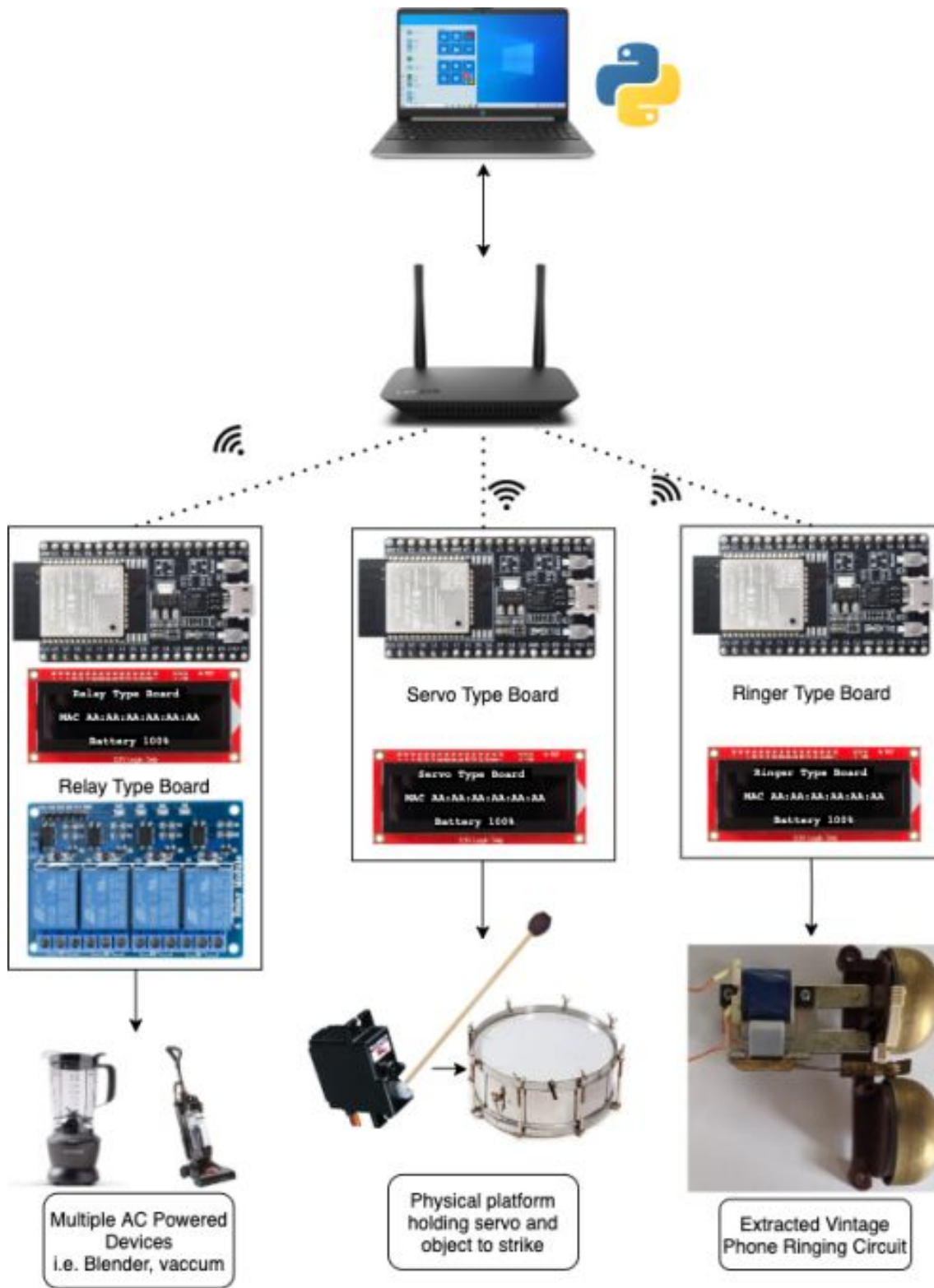


Figure 6. System Overview Diagram

2.2.2 Device Overview

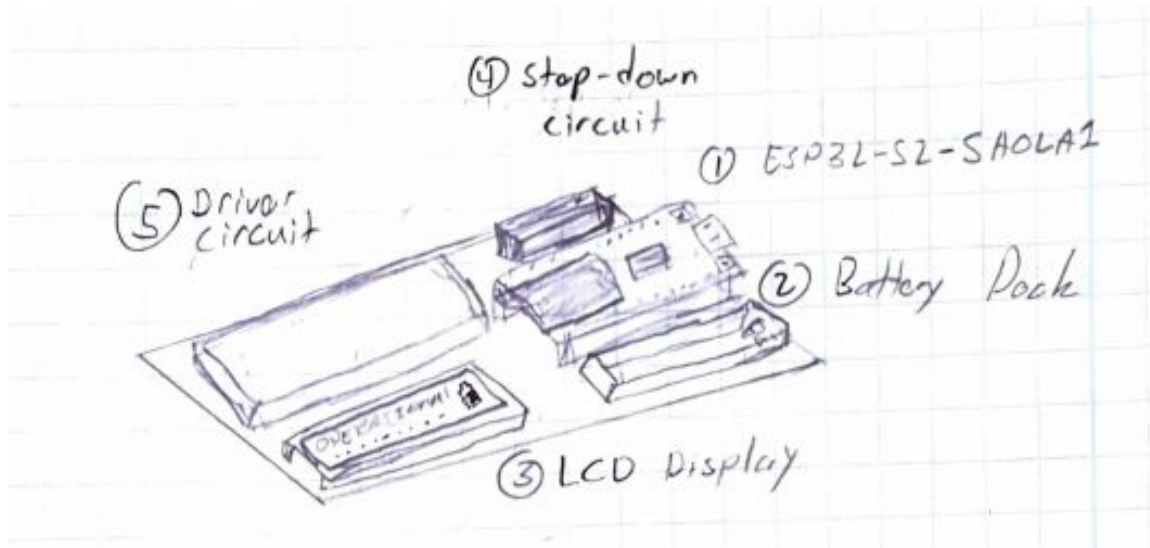


Figure 7. PCB Sketch

2.2.3 External Components

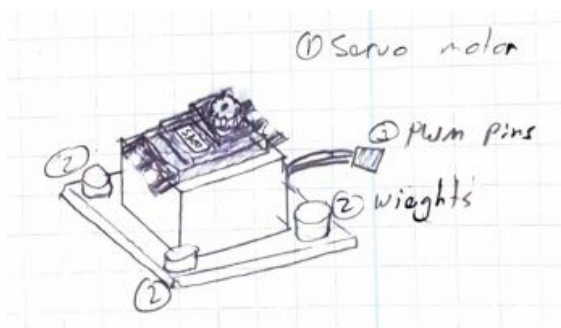


Figure 8. Servo motor mount

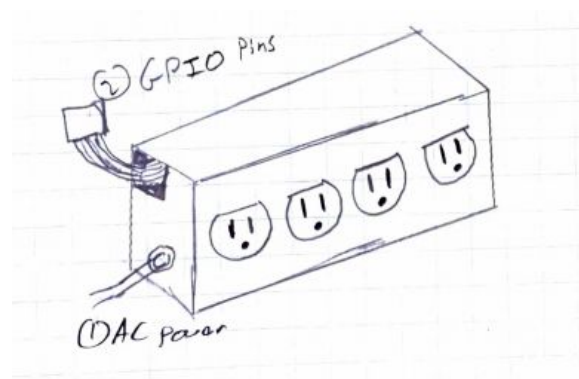


Figure 9. Relay enclosure

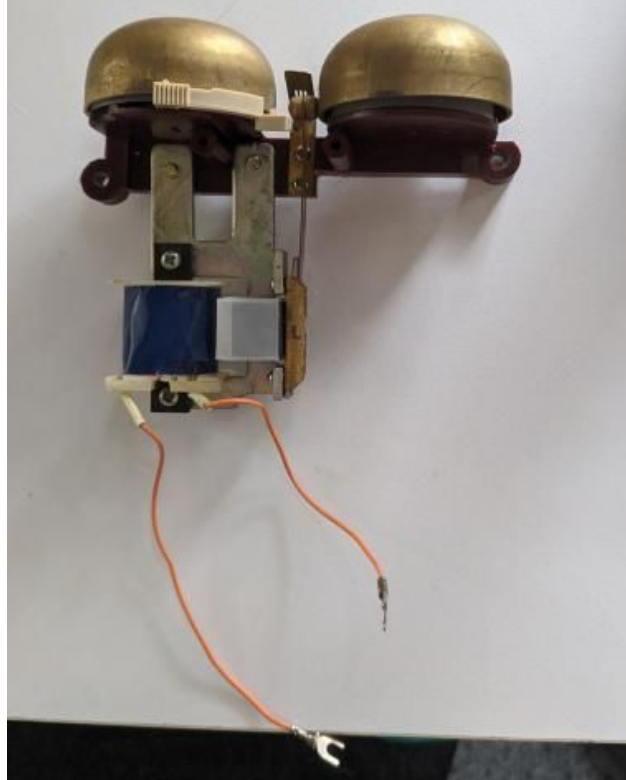


Figure 10. Physical Ringing Hardware

2.3 Requirements and Verification Tables *(Written by Won, Ben, and Micki)*

2.3.1 Control Unit

The control unit will consist of the user's PC and local router. The PC will communicate commands using a Python API that the user can integrate into their current Python scripts. The API should be limited to 1 Python module for simplicity of integration. The local router will send commands to the boards via Wifi using TCP communication, which will be set up separately for each board, due to its superior reliability.

Requirements	Verification
1. The Python module must be able to identify and establish TCP communication with all the boards.	1. Use the Python module on the host PC to initialize the boards. Use the API to print out the number of boards and their type. All available SCPSS devices should be listed.
2. Another requirement for this system is to ensure in software that all messages are consistent in their timing. We do not care about latency, but about latency consistency. Latency should not vary by more than 30 ms.	2. Measure time at right before issuing command in Python script and record timing of sound generation using a sPython audio listening library and record timestamp of sound. Repeat five times and check that

3. The Control Unit must be OS agnostic. The user PC should not be limited by the system	<p>results are consistent to within +/- 15 ms (30 ms).</p> <p>3. Repeat verifications 1 and 2 on Linux, Windows, and Mac.</p>
--	---

2.3.2 120VAC to 3.3VDC Power Converter

Terminal rail mounted power converter. Since the relay board uses wall power, power converter is essential to change power from wall power of 120VAC to 3.3VDC, which is appropriate power for the microcontroller and LCD display.

Requirements	Verification
1. Capable of supplying at least 1A for powering ESP32 and LCD display	<p>A. Connect to 120VAC supply, and check output pins for 3.3VDC +/- 0.5VDC with a multimeter.</p> <p>B. Load test with $R = \frac{V}{I} = \frac{3.3}{1} = 3.3\Omega$ resistor rated at minimum power</p> <p>$P = IV = (1)(3.3) = 3.3W$</p>

2.3.3 ESP32-S2-SAOLA1

Microcontroller must be able to receive signals over Wifi and decode commands. It should also be able to receive commands via a wired USB connection. Decoded signals control the relays and ringer boards via GPIO for the decoded duration, and the servo board takes PWM. Each board will have its own microcontroller.

Requirements	Verification
1. Receive and decode commands over 2.4GHz Wifi	1. Test receiving and sending a string over UDP and TCP.
2. Receive and decode commands via USB	2. Connect ESP32 to Host via USB and test sending and receiving strings.
3. Output 3.3VDC +/- .5VDC GPIO for given duration	3. Measure output of GPIO using multimeter.
4. Convert input from sense circuitry and translate into battery status	4. Display battery status on LCD monitor and measure with multimeter to verify result.
5. Output data to LCD display	5. Test outputting a few strings to the LCD display.

2.3.4 LCD Display

LCD display will report the status of the device to the user Indicating when it is broadcasting its identification on the network, successfully initialization of Control Unit, and when battery is low. Would display which relay is on and for how long. Size of display is 20x2, and each board will have its own LCD display. Difference between display on relay board with others' displays is that it will not display any information about low battery since relay board will be connected to wall power, which means that power is full as long as it is connected.

Requirements	Verification
1. Must be able to receive 3.3V signals from GPIO 2. Must be large enough to display strings of at least 20 characters	1. Send data to the display using the ESP32 microcontroller, and check that sent data is displayed. 2. Send data like in (1), using a string with 20 characters and checking they are all shown.

2.3.5 Off-the-Shelf Quad Relay Board

An off the shelf commercial relay board will be used to ensure that high-voltage of the outlet does not contact our PCB and damage/burn components. This board will have four relays which can toggle four different sound-producing objects.

Requirements	Verification
1. Toggle positive line voltage for four separate devices 2. Tolerate up to 12A for each relay at 120VAC 3. <10ms latency between received GPIO and switch action	1. Apply high signal to relay coil for each relay, and check for short across NO and COMMON 2. Use Electronic Load to test each relay separately. 3. A. Apply a square wave with 50ms period and 50% duty ratio B. Put oscilloscope probes on square wave and across NO and COMMON C. Set scope trigger at 0V so it catches when the relay closes D. Use horizontal measuring tool on scope to check delay between the PWM going high and the relay shorting

2.3.6 Terminal Rail

A terminal rail will be used in order to safely organize and connect positive and negative wall voltages as well as a 120VAC to 3.3VDC converter. It will provide a safe route for a circuit to terminate and reduce the risk of short circuit.

Requirements	Verification
1. Hold terminal blocks at 120VAC and PWR GND and provide isolation from each other 2. Support up to 15A of current.	1. Apply 120VAC across terminal nodes using a power supply and use an electronic load to apply 48A current.

2.3.7 Battery Sense Circuit

This block, which will be replicated for both the Servo Board and the Ring Board will monitor the voltage of our onboard battery and convert that to an analog signal between 0 - 3.3V so it can be read by the ESP32-S2 microcontroller.

Requirements	Verification
1. Convert voltage of the battery to a level that the GPIO of the ESP32 can handle and translate (3.3V) 2. Draw negligible current, at most 5 microAmps	1. A. Measure voltage of the analog output when the battery is charged, verify that it is above 3V but below 3.3V. B. After part A is verified, display analog value on LCD display, and compare to voltage measured on multimeter. Repeat this for the battery at high, medium, and low capacity. 2. Use a multimeter to measure current through the circuit and check that it is within bounds.

2.3.8 6V Battery

The battery must be able to supply 6VDC to the servo motor and 3.3VDC +/- 0.3VDC to the microcontroller. The battery must be able to keep supplying the power to the microcontroller, LCD Display and servo motor up to 3 hours with at least 1.5A.

Requirements	Verification
1. Must provide at least 1.5A for three hours of frequent use of both the microcontroller and the servo motor (once or twice a minute)	1. Run a Python script to turn on and off one of the SCPSS devices with a delay of 10 seconds between commands, and measure time until battery dies or low battery is displayed on LCD.

2.3.9 Step-Down Chip

Off the shelf chip which converts a 6VDC battery to 3.3VDC for powering the ESP32-S2 and the LCD display. This chip would not be necessary if we did not use a servo motor, microcontroller and display only need 3.3 V.. However, since the servo motor needs 6VDC, we decided to use a 6V battery and step down it for the other two.

Requirements	Verification
1. Must be able to step down 6VDC input to 3.3V +/- 0.3VDC output	1. Verify inputs and outputs using multimeters.

2.3.10 Servo motor

Servo motor will be equipped with a drumstick or plastic arm to hit objects. Powered by a 6V battery on the board and controlled by a 3.3V PWM signal from the ESP32. Servo motor will produce an approximate torque of 1.3kg/cm +/- 0.2 kg/cm, powered from a 6V battery. This amount of torque will be enough to equip with a drumstick and hit since the average weight of one drumstick is 40 g ~ 70 g.

Requirements	Verification
1. Must be able to take as input 3.3VDC PWM from ESP32 to control the motor. 2. Must provide enough torque to make the strike audible for at least 10 ft.	1. Use a function generator to create a 3.3V PWM with various PWMs and check that the motor steps with the PWM as expected 2. Attach a pen or pencil to the arm of the servo, and place next to a small piece of metal or a bell. Have the servo repeatedly strike the object and check that it is audible at above 10 ft distance.

2.3.11 Servo Mount

Servo must be securely attached to a mount that can hold the servo in place while it strikes a nearby object. This mount can be held down by weights or by velcro attached to the bottom.

Requirements	Verification
1. Able to hold servo near object to strike for at least 12 strikes before manual realignment is required.	1. Trigger motor to hit 12 consecutive times and listen for any noticeable changes in the sounds, while visually confirming that servo is in place.

2.3.12 Ringing circuit driver

We have acquired a ringing hardware from a vintage phone and reverse engineered it to see precisely what kind of control circuit is required. This circuit will take two 180 degree out-of-phase 3.3V PWM signals and use an H-bridge with a 35VDC boost converter to generate a 70VAC square wave to drive the ringing hardware.

Requirements	Verification
1. Take two 3.3V PWM signals from ESP32 and use it to toggle 60VAC across the H-bridge 2. Should be able to ring for at least 30 seconds continuously.	1. Use a function generator to output two 3.3V square waves to the circuit driver for the mechanism to activate. Listen and verify that the ring is consistent for at least 30 seconds.

2.4 Software Design Plot For Device Algorithm *(Made by Ben)*

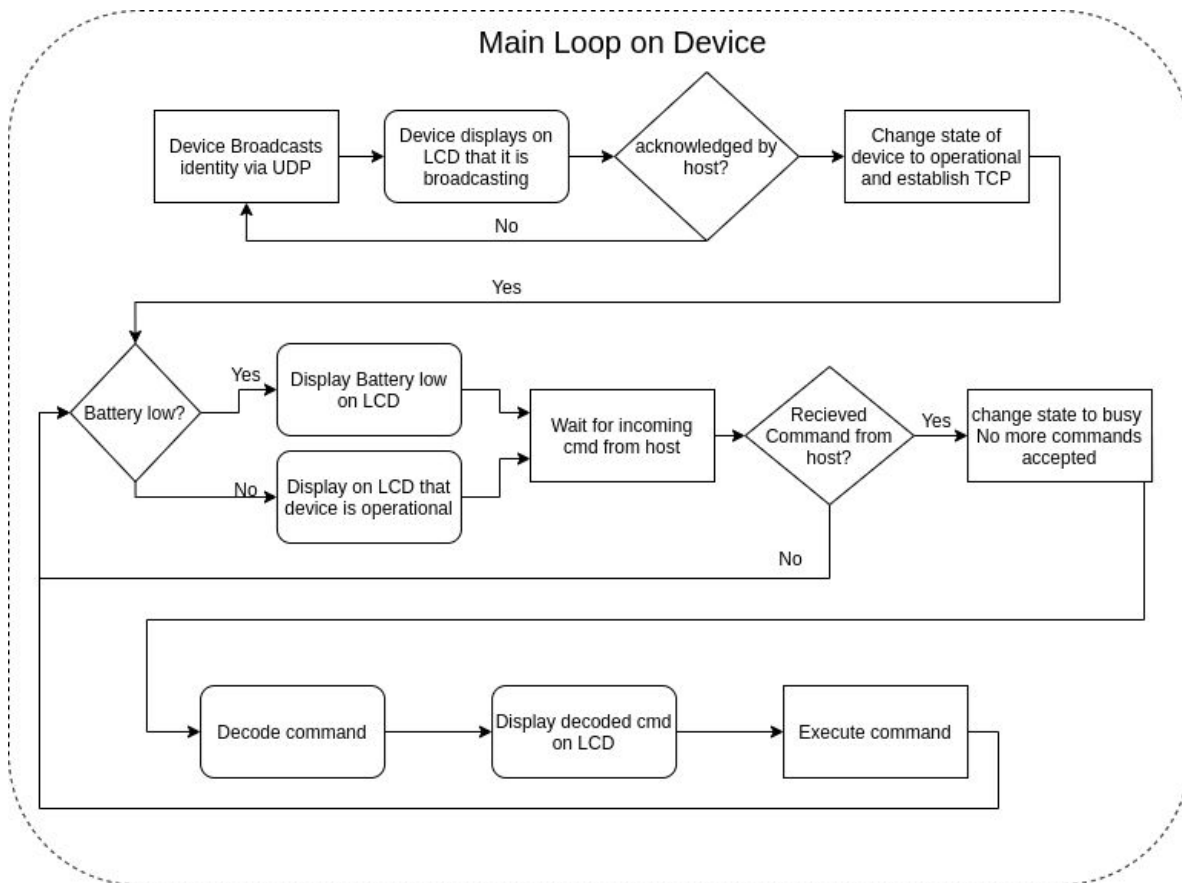


Figure 11. Device Main Loop Algorithm

2.5 Circuit Schematics *(Made by Micki)*

2.5.1 Relay Type Board Schematic

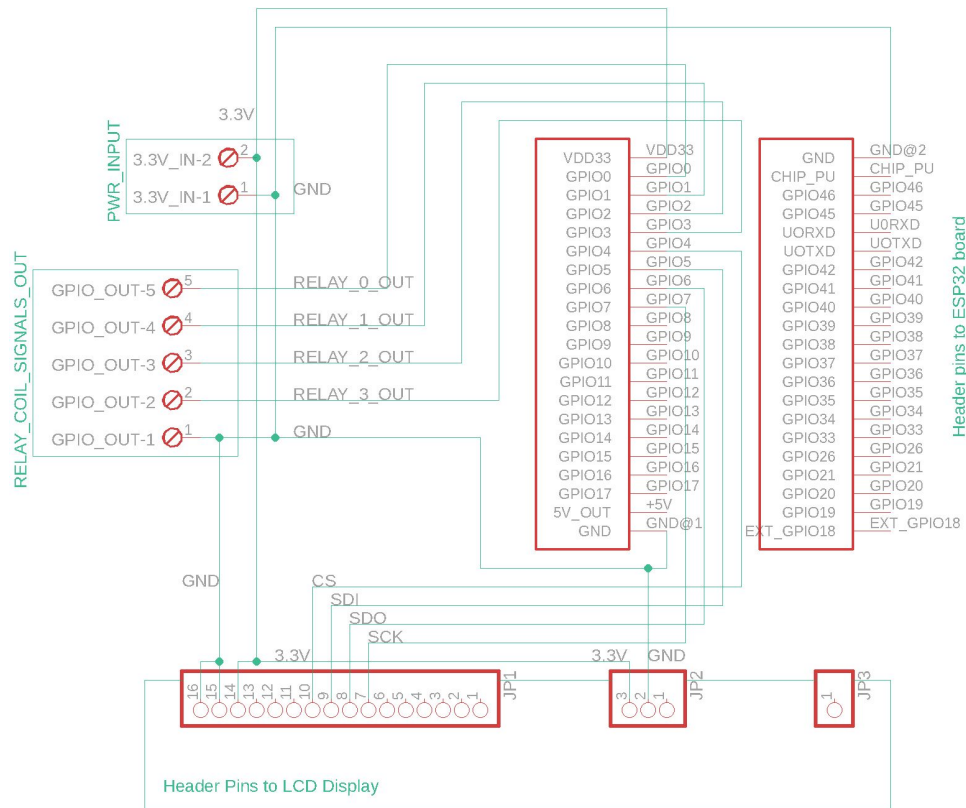


Figure 12. Circuit Schematic of the Relay Board

2.5.2 Relay Type Board Layout

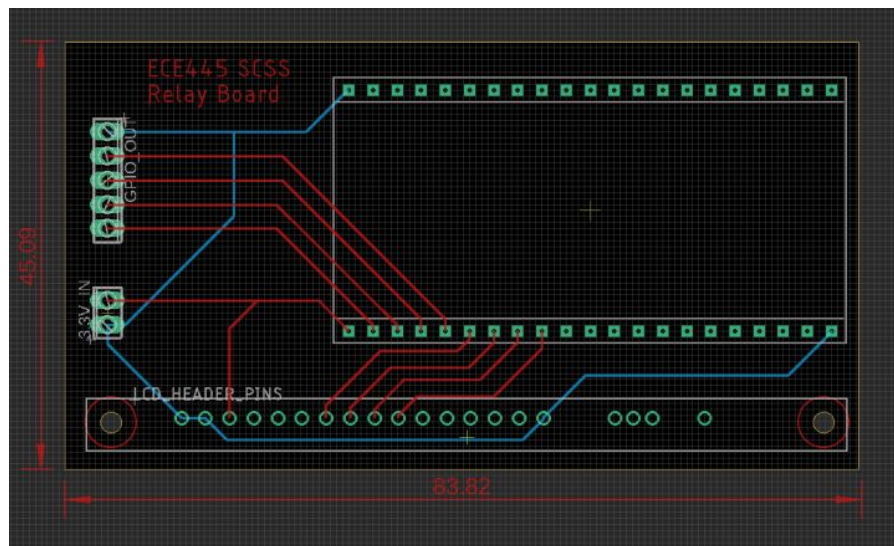


Figure 13. Relay Board Layout

2.5.3 Ringer Type Board Schematic

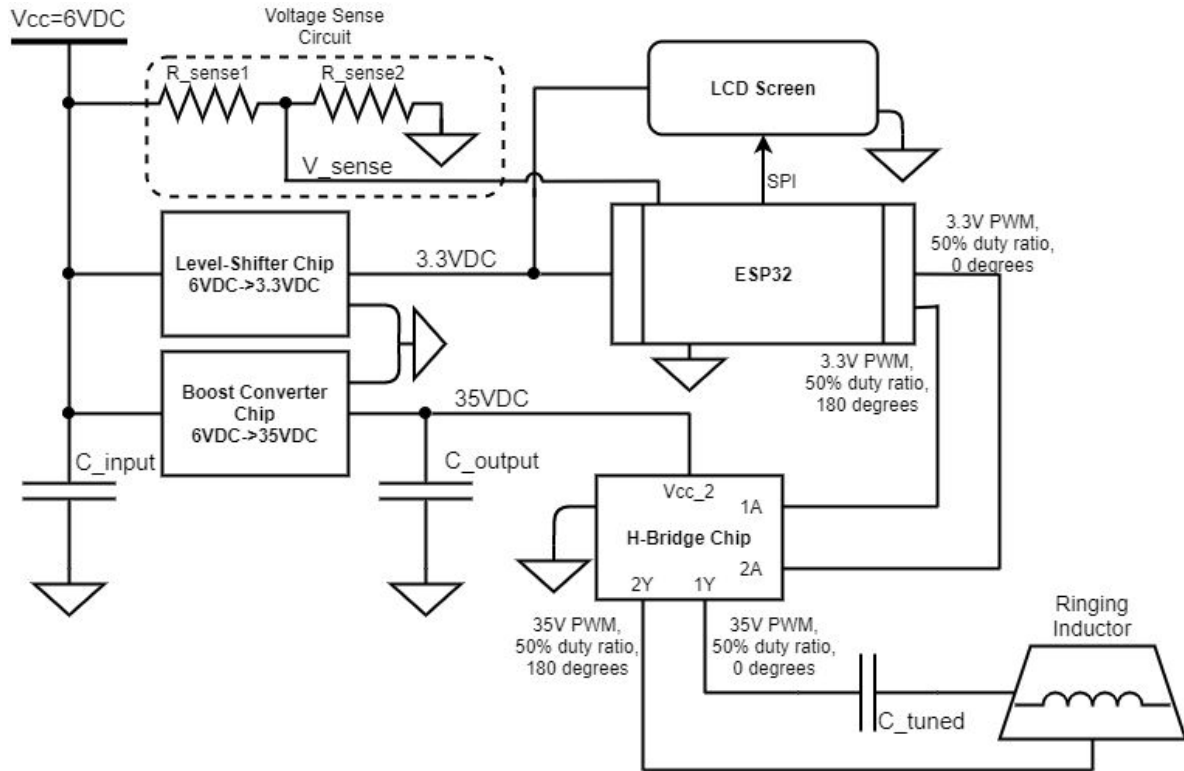


Figure 14. Ringer Type Board Schematic

2.5.4 Servo Type Board Schematic

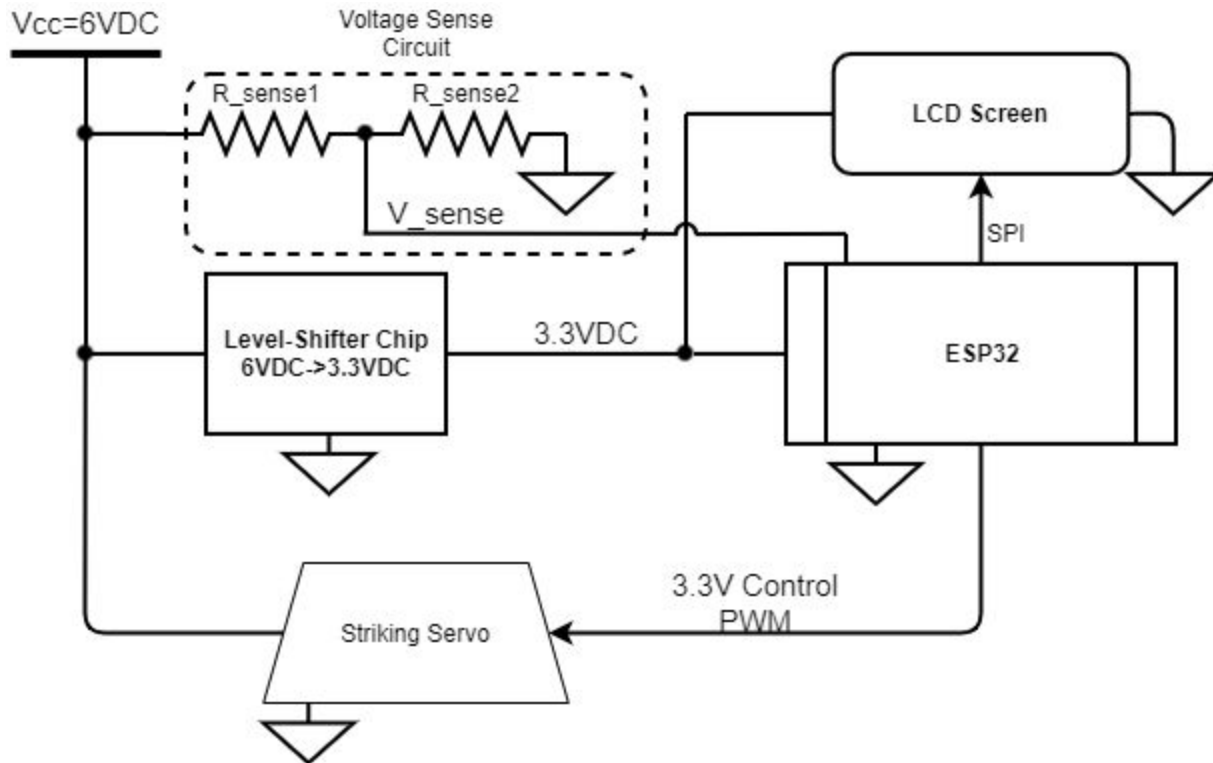
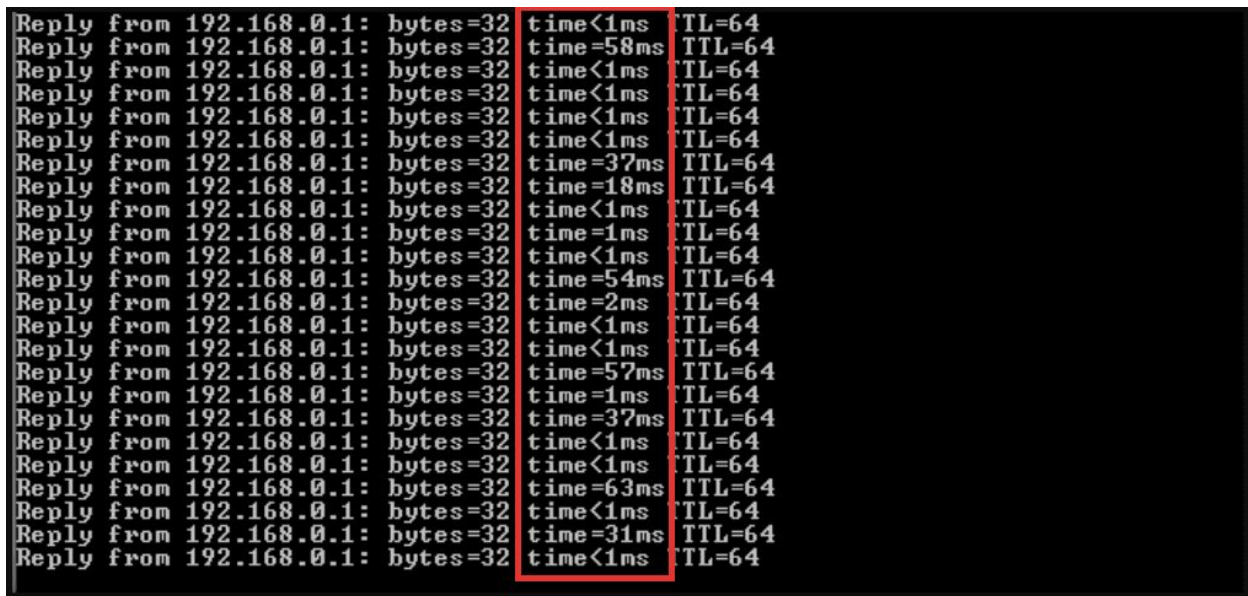


Figure 15. Servo Type Board Schematic

2.6 Tolerance Analysis *(Written by Won and Ben)*

The main risk for this project is the need for latency consistency. As mentioned in prior sections, the latency itself is not a risk factor. However, our sponsor has requested that the devices provide a latency consistency up to less than 30 ms of variation for repeatable testing, which means that we are limited by our network jitter. Network jitter is the inconsistency in latency of packets being delivered through the network [5]. Network jitter occurs during network congestion, interference, route changes, etc. Given that the IALL uses the IllinoisNet network, which is open to all university students, we can expect a decent amount of network jitter. There is a high chance that some packets could arrive with a 20 ms delay while others can arrive with a 60 ms or 10 ms delay. To calculate the network jitter, first we have to throw a PING to a certain destination such as the PC IP address in the terminal. This will lead the terminal to list PING brackets with inconsistent delay times. To find the average value of jitter, we have to average the time difference between each packet sequence in the list. An example of a PING list can be seen in Figure X below.



```
Reply from 192.168.0.1: bytes=32 time<1ms TTL=64
Reply from 192.168.0.1: bytes=32 time=58ms TTL=64
Reply from 192.168.0.1: bytes=32 time<1ms TTL=64
Reply from 192.168.0.1: bytes=32 time<1ms TTL=64
Reply from 192.168.0.1: bytes=32 time<1ms TTL=64
Reply from 192.168.0.1: bytes=32 time<1ms TTL=64
Reply from 192.168.0.1: bytes=32 time=37ms TTL=64
Reply from 192.168.0.1: bytes=32 time=18ms TTL=64
Reply from 192.168.0.1: bytes=32 time<1ms TTL=64
Reply from 192.168.0.1: bytes=32 time=1ms TTL=64
Reply from 192.168.0.1: bytes=32 time<1ms TTL=64
Reply from 192.168.0.1: bytes=32 time=54ms TTL=64
Reply from 192.168.0.1: bytes=32 time=2ms TTL=64
Reply from 192.168.0.1: bytes=32 time<1ms TTL=64
Reply from 192.168.0.1: bytes=32 time<1ms TTL=64
Reply from 192.168.0.1: bytes=32 time=57ms TTL=64
Reply from 192.168.0.1: bytes=32 time=1ms TTL=64
Reply from 192.168.0.1: bytes=32 time=37ms TTL=64
Reply from 192.168.0.1: bytes=32 time<1ms TTL=64
Reply from 192.168.0.1: bytes=32 time<1ms TTL=64
Reply from 192.168.0.1: bytes=32 time=63ms TTL=64
Reply from 192.168.0.1: bytes=32 time<1ms TTL=64
Reply from 192.168.0.1: bytes=32 time=31ms TTL=64
Reply from 192.168.0.1: bytes=32 time<1ms TTL=64
```

Figure 16. PING List Example

Credit to <https://www.pcwdld.com/network-jitter> for example

To find the average of the network jitter, we have to find out the total difference of time and divide it with the number of values that had variation above 1ms.

Difference 1 = 1 ms and 58 ms: 57ms

Difference 2 = 58ms and 1ms: 57 ms

Difference 3 = 1 ms and 1 ms: 0 ms

Difference 4 = 1 ms and 1 ms: 0 ms

.....

Difference 22 = 1ms and 31 ms: 30 ms

Difference 23 = 31 ms and 1ms: 30 ms

There are 23 differences from the beginning to end and 16 differences that have greater than 0 with total difference equals to 660. Finally, we can calculate the network jitter by dividing this total number of 660 by 16, which leads us to have 41 ms of network jitter. We formalized this process using the following pseudocode:

```
Let X be the list of times for ping responses.  
Let N be the total number of pings  
Let C be the number of jitter occurrences, initially set to 0  
Let S be the sum of jitters, initially set to 0  
For each  $x_i$  in X with  $i$  ranging from 0 to N:  
    If  $x_i \neq x_{i+1}$  :  
        Increment C  
        Set  $S = S + |x_i - x_{i+1}|$   
Jitter = S/C
```

In order to provide the 30 ms maximal jitter requested by our sponsor, we must do extensive measurement and testing on the network jitter of the IllinoisNet network, preferably from the IALL's access point, and use this measurement to create a jitter buffer in software. A jitter buffer on our system would use the timestamp of the command generated on the host PC, and wait until a set amount of time is passed before executing the command on the ESP32-s2 microcontroller. As we can see, the latency of these devices is not a bug, but a feature that must be set in software in response to the network jitter to ensure latency consistency.

2.7 COVID-19 Contingency Plan *(Written by Micki and Ben)*

In the case that we are moved to a completely online curriculum and we lose access to the senior design lab, we intend to shift focus from the relay-based device to the servo and ring based devices. This is because we would lack the equipment to safely work on and verify the functionality of high-voltage components, thus we can shift our focus to expanding upon the devices that are safer to test and measure in our own homes. This would mean that we would have to get more creative about our servo-based device, since it is the only one left that can output a variety of sounds. One option would be to expand the functionality of the servo to strike two objects, one to the left and one to the right of the servo, rather than simply striking one object. This would require an overhaul of our API and some of the code on the ESP32, but requires no change to the hardware/electrical design of the device.

3 Costs *(Written by Micki)*

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).

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

Part/Part #	Quantity	Manufacturer	Cost
Terminal Rail Power Converter (120VAC - 3.3VDC) PSK-10W-3-DIN	1	CUI Inc.	\$21.60
ESP32-S2-SAOLA1	3	Espressif Systems	Each: \$8 Total: \$24
LCD Display	3	Sparkfun	Each: \$7.49 Total: \$22.47
Terminal Rail 77-2293-ND	1	Phoenix Contact	\$5.43
Terminal Rail Partitions 277-2040-ND	10	Altech Corp.	Each: \$0.80 Total: \$8.00
Terminal blocks, various colors 277-3243-ND	9	Phoenix Contact	Each: \$1.34 Total: \$12.06
Ground terminal block 277-17411-ND	1	Phoenix Contact	\$4.51
Terminal block end plate 277-2038-ND	2	Phoenix Contact	Each: \$0.64 Total: \$1.28
6V Battery AmazonBasics AA High-Capacity Ni-MH Rechargeable Batteries	1 (6V) = 8 (AA)	Amazon	Each: 2.14 Total: 17.11
4 AA Battery Holder	2	LampVPath	Each: 2.99 Total: 5.98
Servo Motor FutabaS3003	1	Futaba	(already have)
6VDC to 3.3VDC level shifter chip AP1509-33SG-13	3	Semtech Corp.	Each: \$1.60 Total: \$4.80

H-Bridge	1	n/a	(already have)
22K Ω Resistor	2	n/a	(already have)
44K Ω Resistor	2	n/a	(already have)
High Voltage Quad Relay Board	1	n/a	(already have)
Various capacitors	n/a	n/a	(already have)
Vintage phone hardware	n/a	n/a	(already have)
Total			\$127.24

Our project is meant to be modular to accommodate future additions of sound devices and changing in and out devices, but for the scope of this class we are designing a system that can have four wall-powered sound sources, one servo “striking” type sound source, and one ringer type sounds source. We went over budget here but our sponsor is willing to cover additional costs. Finally, the actual sound sources aren’t included in the costs at the moment as we plan to use our own devices and when we ultimately give the system over to our sponsor, they can source the sound devices they want for their tests.

4 Schedule *(Written by Won, Ben and Micki)*

Week	Micki	Ben	Won Woo
10/5	Work on PCB for 3 boards, order minimal parts	Set up Github	Set up Github
10/12	Finish PCB for all three boards, plan basic tests for boards	Create protocol for message structure	Review UDP/TCP communication in Python
10/19	Breadboard with components then test on PCBs	Create and test Network Jitter buffer	Begin writing Python Module
10/26	Rework PCB if there is any problem	Begin embedded software for ESP32	Finish Python module on Host
11/2	Finalize hardware, build enclosures	Finish embedded software on ESP32	Finish embedded software on ESP32
11/9	Prepare Mock Demo	Prepare Mock Demo	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 Ethics and Safety *(Written by Won)*

Since the relay-type board will be 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 are planning to use a terminal block (aka terminal rail) to avoid any kinds of poor connected wires. Using it will lead to a convenient and safer way to distribute power from a single input source of the wall power to multiple outputs. We are responsible for our design and safety, and this safety concern is 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.

One other concern for the relay-type board is the use of high-power consumption electronics. These electronics are more likely to be used since they produce a lot of noise, for example vacuums, blenders, or hair-dryers are all high power consumption electronics that are very loud. However, a standard wall outlet may not be able to provide current for four of these devices at the same time. For this reason, the user must be aware of the amount of power their electronics require, and whether they need a high-power outlet rather than the standard.

Since we will lead the project based on using the school's wifi during the demo, we are responsible for our design that will prevent any circumstances that violate provisions of University policy over using the school's WiFi [7].

6 Citations

[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] Housing, U., n.d. *Connection Speeds*. [online] Illinois University Housing. Available at: <<https://housing.illinois.edu/Resources/Technology/Help/connection-speeds>> [Accessed 1 October 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.pcworld.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].