Georgia Institute of Technology

Design Document

Outta Time

Team A5: Jonathan Huang, Selene Jordan, Lindsey Lubin, Bo Han Zhu

Date: October 14th, 2022

Table of Contents

Date	Author	Comments
9/24/2022	Team	Document Created
10/12/2022	Jon	Edited Power Subsystem
10/13/2022	Bo Han	Added microcontroller selection to Trade-Offs
10/13/2022	Bo Han	Began and completed PCB Section (Main, UI, Power Boards)
10/13/2022	Bo Han	Added design and simulation details for Audio, UI, LED subsystems
10/13/2022	Lindsey	Added Software design and simulation details
10/13/2022	Selene	Began the Mechanical Section
10/14/2022	Team	Finalizing Sections
11/11/2022	Team	Finish up Document for Final Submission

Project Description

Outta Time is a robot thespian inspired by the white rabbit from *Alice in Wonderland*. It will recite lines, play audio, tilt the head, move an arm, and a path will light up when instructed by the director. It will operate from a standard 120V wall outlet and will be controlled by an ESP32 microcontroller.

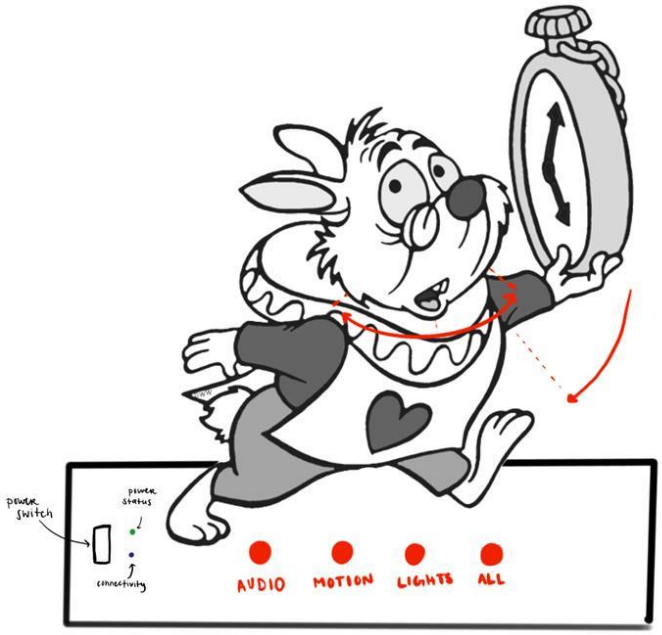

Figure 1. Preliminary graphic sketch of the robot thespian.

Hierarchical Design

From a top-level perspective the entire project encompasses 3 systems and 5 subsystems, all with individual components. The interactions between the subsystems and systems can be all summed up between Figure 2 and Table one below.

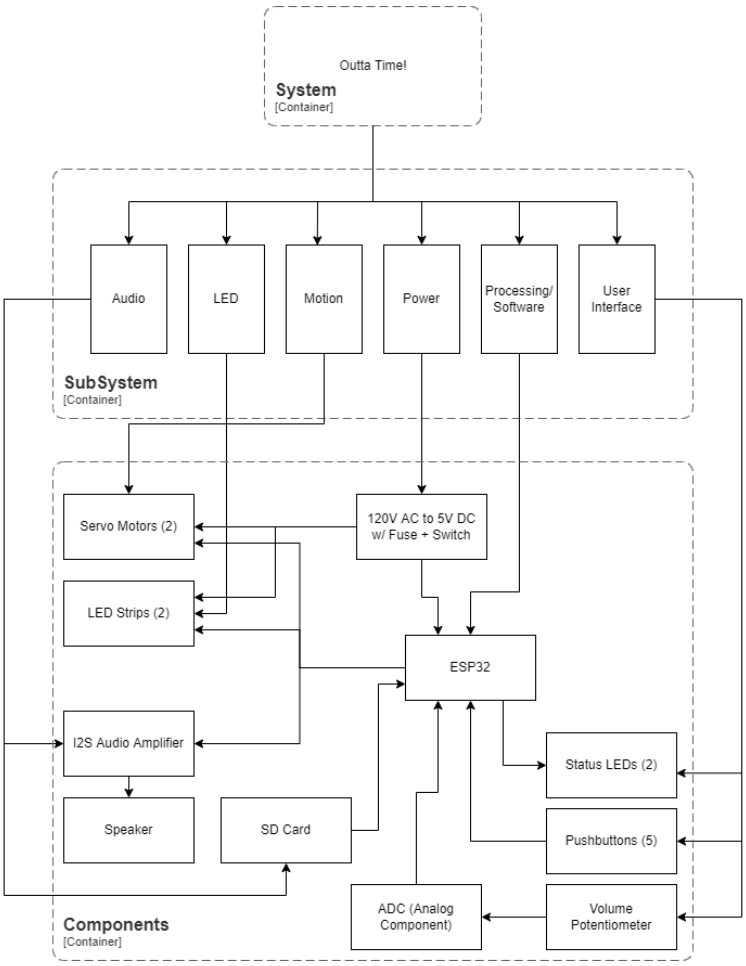

Figure 2. Project hierarchical diagram of systems, subsystems, and components

Table 1: Subsystem interface table

Interface	Source	Destination	Description
120V AC	System Input	Power	120V AC Input
WiFi	System Input	Processing	WiFi commands from Director to system
5V DC	Power	Processing	Converted 5V DC Power
5V DC x2	Power	Motion	Power to two Servo motors
5V DC x2	Power	LED	Power to two LED strips
5V DC	Power	Audio	Power to the audio amplifier
3.3V DC x7	Processing	User Interface	Logic high for five pushbuttons and one potentiometer; power to the ADC
PWM x2	Processing	Motion	PWM signals to control two servos
I2S (3 lines)	Processing	Audio	I2S communication to the audio amplifier
SPI (4 lines)	Processing	User Interface	SPI communication to the ADC
SPI (4 lines)	Processing	Audio	SPI communication to the SD card reader
GPIO x5	User Interface	Processing	Digital reading of the five pushbuttons
GPIO x2	Processing	LED	Digital communication to two LED strips
GPIO x2	Processing	User Interface	Power to toggle two status LEDs

Subsystem Designs

Table 2: Task Leads for each Subsystem

Team Lead	Jonathan Huang
Audio Subsystem Lead	Bo Han Zhu
LED Subsystem Lead	Bo Han Zhu
User Interface Subsystem Lead	Bo Han Zhu
Motion Subsystem Lead	Selene Jordan
Power Subsystem Lead	Jonathan Huang
Software	Lindsey Lubin

Audio Subsystem

Diagram

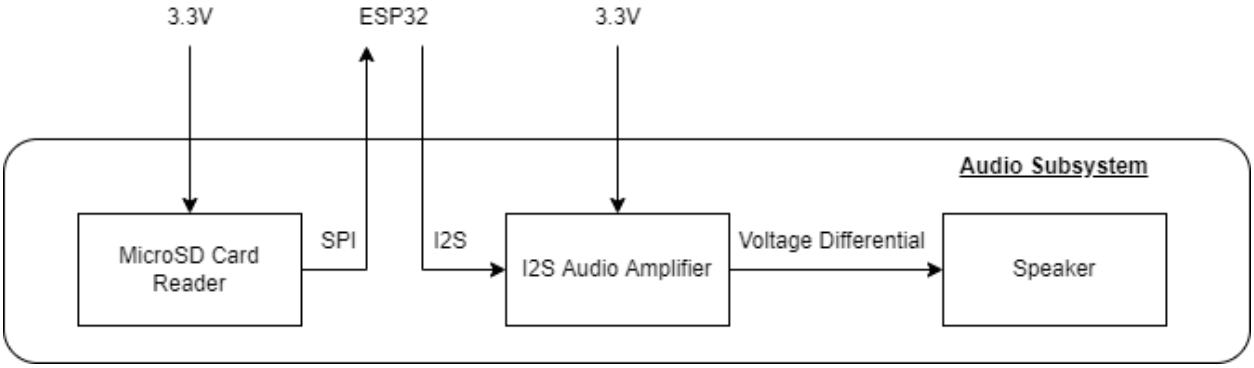

Figure 3. Audio Subsystem

Description

The audio subsystem plays the "I'm Late!" song from Alice in Wonderland. The song is stored in a MicroSD card, which is read by a breakout board before being transmitted to the ESP32. The ESP32 propagates this via I2S to the audio amplifier, which controls a speaker by outputting a voltage differential. The entire system is powered using 3.3V from the ESP32.

Design Process

The first step for audio was to determine what we wanted to play, as this would dictate the complexity of hardware and software needed. We chose to play a song in MP3 format from Alice in Wonderland that was about 10 seconds long. An external memory would be needed to hold such a large file. Second, to meet the 40-70 decibel range we selected, we found out that a simple speaker was not enough, and therefore needed to bring in an audio amplifier and a larger speaker.

Simulation

Two breadboard simulations were done for the Audio subsystem. The first simply verified that the ESP32 and I2S audio amplifier could drive a simple speaker. The audio file for this simulation was not stored in memory – it was pulled from an online radio station via the ESP32.

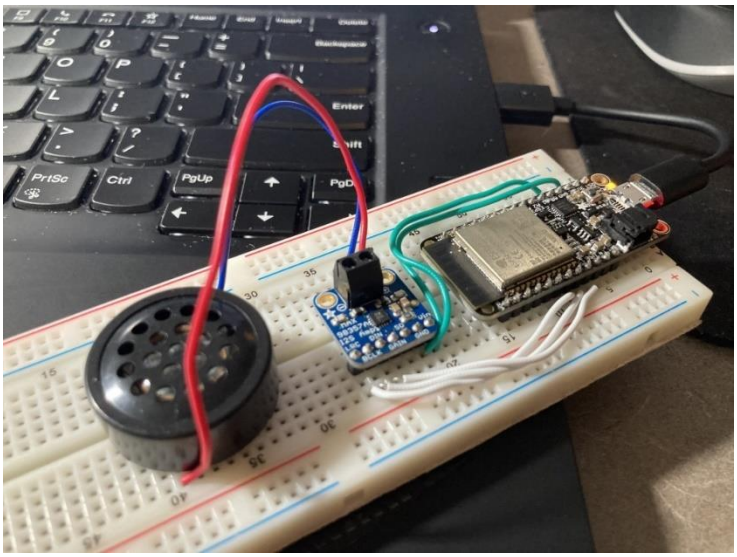

Figure 4. The ESP32 and I2S audio amplifier drive a simple speaker.

The second simulation involved the entire block diagram and the song we chose. A MicroSD card reader containing the MP3 song was connected via SPI to the ESP32. It was during this simulation that we realized that the simple speaker from the first simulation was too soft and could not meet the 40-70 dB requirement. Upon switching to a larger speaker, about 60 dB was consistently measured about a foot away.

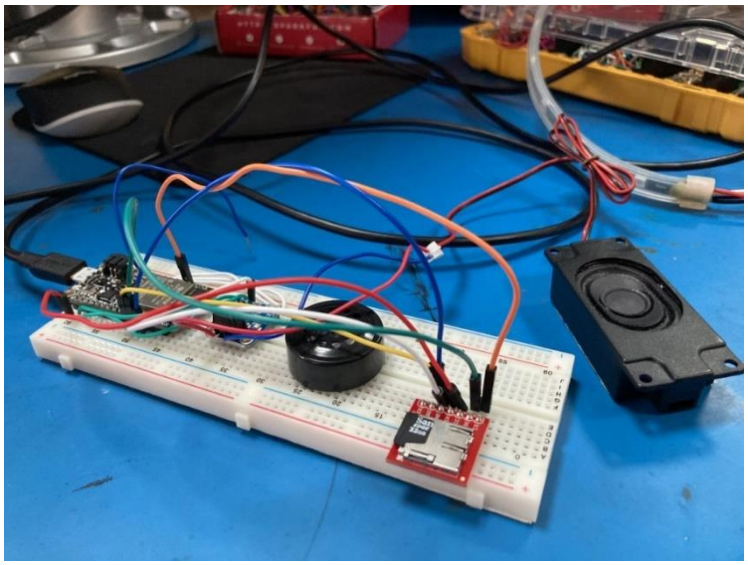

Figure 5. A larger speaker and MicroSD card reader are added to the first audio simulation. The simple speaker in the middle of the breadboard is disconnected and unused.

Motion Subsystem

Diagram

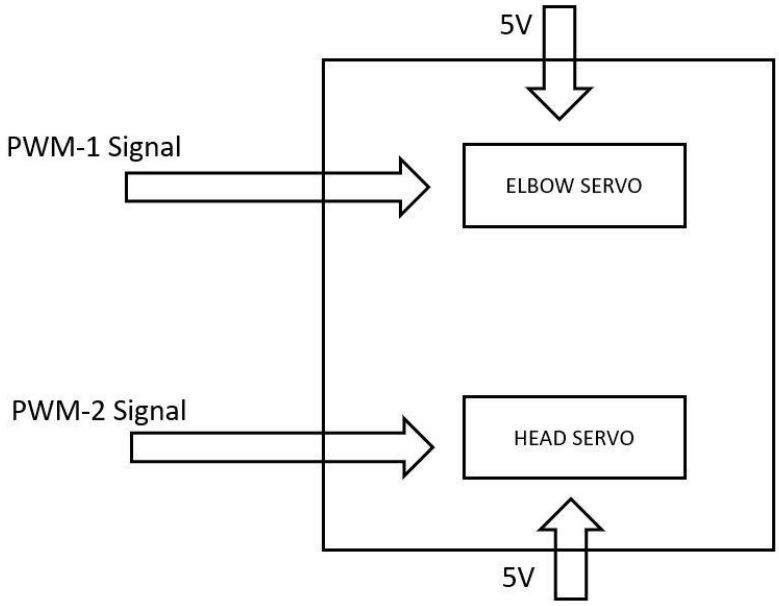

Figure 6. Motion Subsystem

Description

The motion subsystem will take 5V DC and receive 2 PWM signals that control the motor drivers. The motor drivers will then interpret the signals and power the motors appropriately to produce the desired motion for the rabbit's elbow and head.

Design Process

Our first motion design included the rabbit's arm and two legs moving in order to mimic a running motion. After discussion, we decided that this would be too complicated and costly for our budget. Instead, we decided that we would limit ourselves to two motions; the rabbit's head and arm. We also chose to simulate these motions with a standard servo motor capable of holding the necessary weight of the rabbit. The rabbit's head will move with a range of 60 degrees and the rabbit's arm will move 45 degrees.

Simulation

The motion simulation utilized a standard servo motor and the ESP32. For this simulation we connected the servo motor to the ESP32. The power on the servo motor is connected to an external power supply of 5V. The ESP32 and motor are also connected to common ground. Once I secured these connections, I then uploaded a standard sweeping code onto the ESP32. This simulation confirmed the servo motor was compatible with the ESP32 and could perform the range of motion we need for our rabbit.

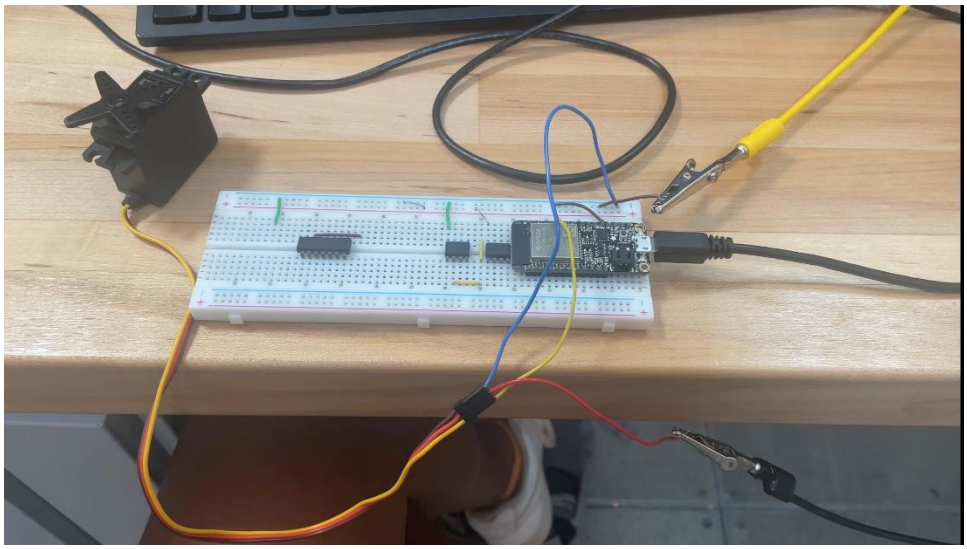

Figure 7. The motion simulation of a standard servo motor and ESP32 connected to perform a sweeping motion.

User Interface Subsystem

Diagram

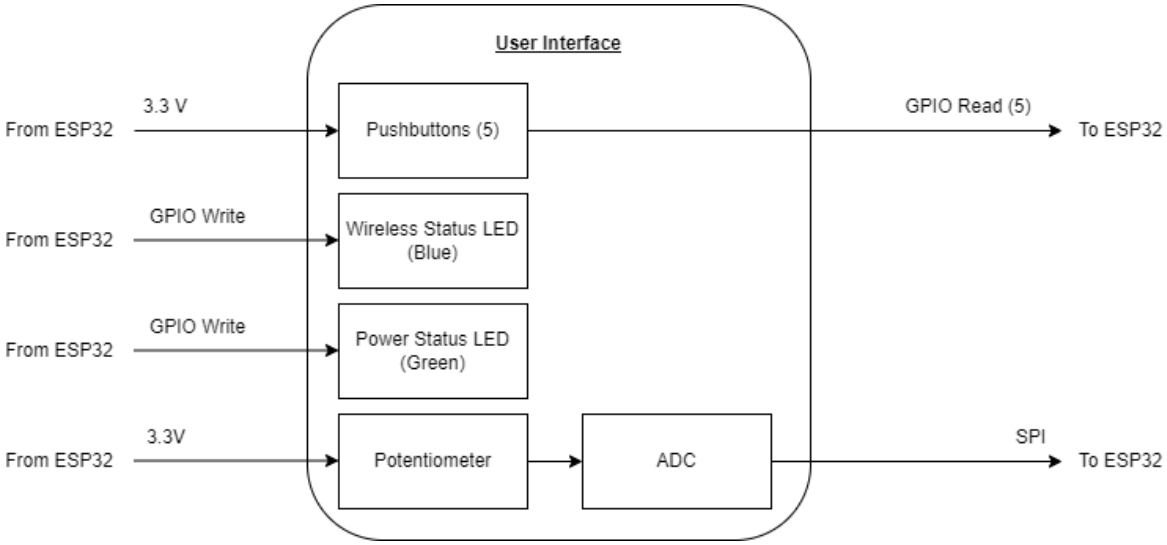

Figure 8. User Interface Subsystem

Description

The user interface subsystem serves as the bridge between system diagnostics/control and the user. There are five pushbuttons: one is to test all subsystems, three are to test a specific subsystem, and one is to initiate wireless connection with the Director. There is a potentiometer that serves as volume control for the speaker and two LEDs that display the status of wireless connectivity and power on/off. The status LEDs draw power from the GPIO pins of the ESP32 (3.3V) whereas the pushbuttons and potentiometer are supplied by the ESP32 3.3V output. Pushbutton feedback is communicated to the

ESP32 via GPIO while the potentiometer's analog reading is converted to digital by an ADC and then communicated to the ESP32 via SPI.

Design Process

We began the design with just three pushbuttons to test the Audio subsystem, Motion subsystem, and LED subsystem individually. However, as we had more software/electrical integration discussions, we began to realize that testing the system without wireless connectivity and toggling wireless connection were important functions, and so two more pushbuttons were added.

Initially, there were no status LEDs in our design. But because whether having power and wireless connection was such a difference-maker to how the system operated, we realized that displaying this information to the user was crucial. Thus, we added status LEDs for both.

When we designed based on using the Raspberry Pi Zero W, which had no on-board ADCs, we picked the MCP3008 to serve as an external ADC to handle the potentiometer output. However, after we switched to the ESP32, which had multiple on-board ADCs, this circuitry was no longer needed. Yet, we kept it in our design to fulfil the "one analog circuit" requirement.

Simulation

The pushbuttons, LEDs, and potentiometer control were first simulated in TinkerCad. An Arduino Uno digitally outputted signals to the LEDs. An optimal LED resistor value was chosen based on the Digikey LED Series Resistor Calculator (https://www.digikey.com/en/resources/conversion-calculators/conversion-calculator-led-series-resistor).

The Arduino Uno received pushbutton feedback as digital inputs and the potentiometer input as an analog input. TinkerCad only allowed us to use Arduino microcontrollers, but because the ESP32 development environment is based on the Arduino IDE and compiler (with an additional board package), we were confident that the results would translate over.

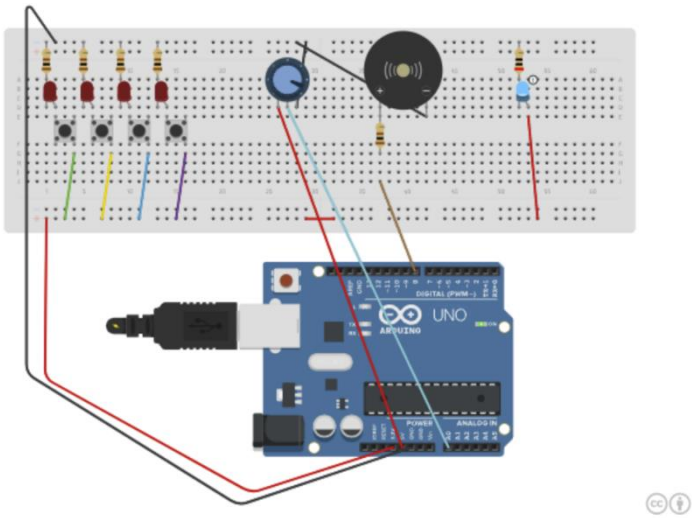

Figure 9. TinkerCad simulation with various user interface components. The potentiometer value is used to determine the frequency of a simple speaker, but it is also printed to the Arduino serial console.

Because we had a fair amount of experience hooking resistors in series with LEDs and adding pull-ups to pushbuttons, we did not breadboard them. However, we did breadboard the ESP32, ADC, and potentiometer to ensure that the MCP3008 ADC could transmit 10-bit digital values via SPI.

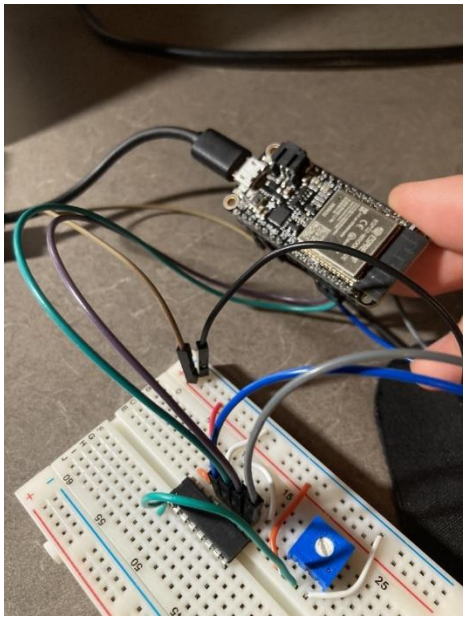

Figure 10. Variable Resistor serving as a potentiometer connected with the MCP3008 ADC and ESP32.

Power Subsystem

Diagram

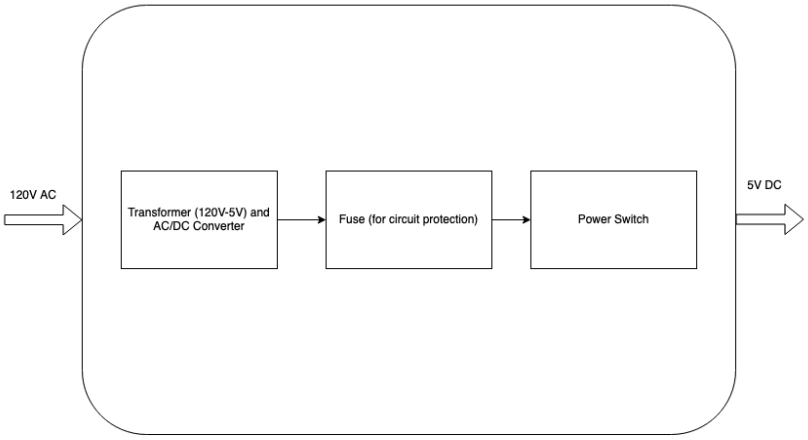

Figure 11. Power Subsystem

Description

The power subsystem will take in the required 120V AC at 15A-20A from a standard US wall outlet and output 5V DC. It will first go through a 15W (5V DC, 3A) transformer, then a 3A fuse (for circuit protection), then finally a power switch before distributing power to the other subsystems.

Design Process

We first began starting with a 5V, 2A power supply as we understood initially that all the parts needed 5V and that 10W should be able to supply all the power necessary for all components to function. In addition, we understood that it would be important to be able to protect our components from voltage and current surges, which is why we included a 2A fast-blow fuse as well. We also included a switch so that we could control when electricity would flow through our thespian.

However, after working on the power budget, we soon realized that all the components take up around 10W. In addition, we did not scale this value by 1.5x to ensure for all worse case scenarios, making the power being drawn about 15W. We wanted to keep the 5V, so we decided to change the current to 3A so that the power generated could satisfy the power drawn

Simulation

For our simulation of the power subsystem, we made a power budget to see how much power we needed to be drawn and how much power could be supplied. We found the values for each of the components using their datasheets online; we then summed all the power values up and multiplied them by 1.5 to get a worst-case power drawn and updated our power supplied to match that (5V, 3A).

Voltage Drawn

Component	Subsystem	Amount	Max Voltage	Max Current	Worst Case Power
MCU	Processing	1	5V	250 mA	1.25 W
Servo Motors	Motion	2	5V	800mA	4 W
LED Strips	LED	2	5V	60 mA each	1.2 W
Pushbuttons	User Interface	5	3.3V	3.3 mA each	0.0545 W
Status LEDs	User Interface	2	3.3V	60 mA each	0.396 W
ADC	User Interface	1	3.3V	425 uA	0.0014 W
SD Card Reader	Audio	1	3.3V	150 mA	0.495 W
I2S Audio Amplifier	Audio	1	3.3V	800mA	2.64 W
					= 10W * 1.5 ~ 15W

Voltage Given

Component	Subsystem	Amount	Voltage	Current	Power Supplied
MCU (3.3V Output)	Processing	1	3.3V	250mA	0.825W
Power Supply	Power	1	5V	3A	15W

According to my calculations, 15W should be enough for this 5V, 3A

Figure 12. Power Subsystem

LED Subsystem

Diagram

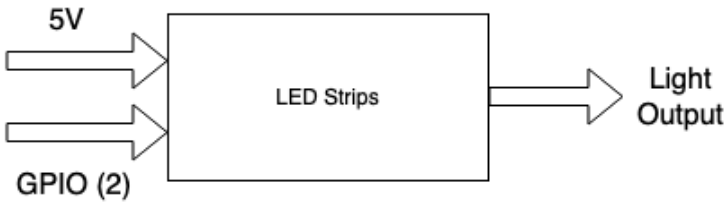

Figure 13. LED Subsystem

Description

The LED subsystem consists of 2 LED strips of 23 RGB LEDs each that will illuminate the Rabbit's "path". The LED strips are powered by the external 5V power supply; they cannot draw from the MCU due to a high-power draw. The Arduino Neopixel library is used to control the strips via one GPIO pin per strip.

Design Process

We decided to use LED strips for lighting from the beginning – discrete LEDs would take up too many GPIO pins and timing each one would be very difficult. The only question was which communication protocol should we use for control. After considering both PWM and SPI options, we decided to go with the WS2812, which could be driven by a single GPIO. SPI would require more connections, and at that time, we still used the Raspberry Pi Zero W, which had a finite number of PWM pins. The strip itself had hardware to propagate the one digital signal down the strip and a readily available Arduino library for us to use.

Simulation

We began by using TinkerCad to ensure that the Arduino Neopixel library could indeed handle a WS2812 LED strip with a single GPIO. Again, TinkerCad only allowed us to use Arduino microcontrollers, but because the ESP32 development environment is based on the Arduino IDE and compiler (with an additional board package), we were fairly confident that the results would translate over.

Figure 14. TinkerCad simulation of an Arduino Uno driving two WS2812 LED strips with no additional hardware.

Indeed, our second simulation was to hook up the ESP32 with the WS2812 strip in a bench testing environment with external 5V. We were able to successfully light up the LED path using the same library.

Figure 15. Bench testing of the WS2812 LED strip with the ESP32.

Trade-Offs and Alternatives

During design, we made two critical component decisions, which were the types of microcontroller and servo motors we used.

We originally began design with the suggested Raspberry Pi Zero W. Code development was to be done in micropython in a built-in OS environment, and the Pi Zero provided the required I/O interfaces as well as a built-in WiFi module and API for communicating with the Director. It could even drive speaker output directly via Micro-USB. However, its cost, $60, would have caused us to exceed our $160 budget by roughly $10.

Therefore, we pivoted to another microcontroller with built-in WiFi capabilities – the Adafruit ESP32 Feather. This board was much cheaper at $20 and provided enough I/O interfaces to cover each subsystem. The drawback was that we could not drive the speaker directly and needed to edit the audio subsystem to include an additional I2S amplifier. Another side benefit to using the ESP32 was that development could be done in bare-metal or RTOS C-code, which was a programming language and development environment our team was more acquainted with.

We originally planned on using a Tower Pro Micro Servo motor, however, after some testing we realized that the motor would not be able to hold the necessary weight. Instead, we decided to use the Tower Pro Standard Servo motor. This motor was significantly larger and able to hold the required weight, however, it was double the price of the mini servo motor at $12.

Software Design

The software state diagram pictured in figure 14 includes 4 different states: wait (setup), test, decode, and perform. This software state flow diagram illustrates how our thespian will communicate with the director and translate the instructions into actions. The basis for these 4 states is to ensure a working demo as well as to ensure that we can debug and test while offline.

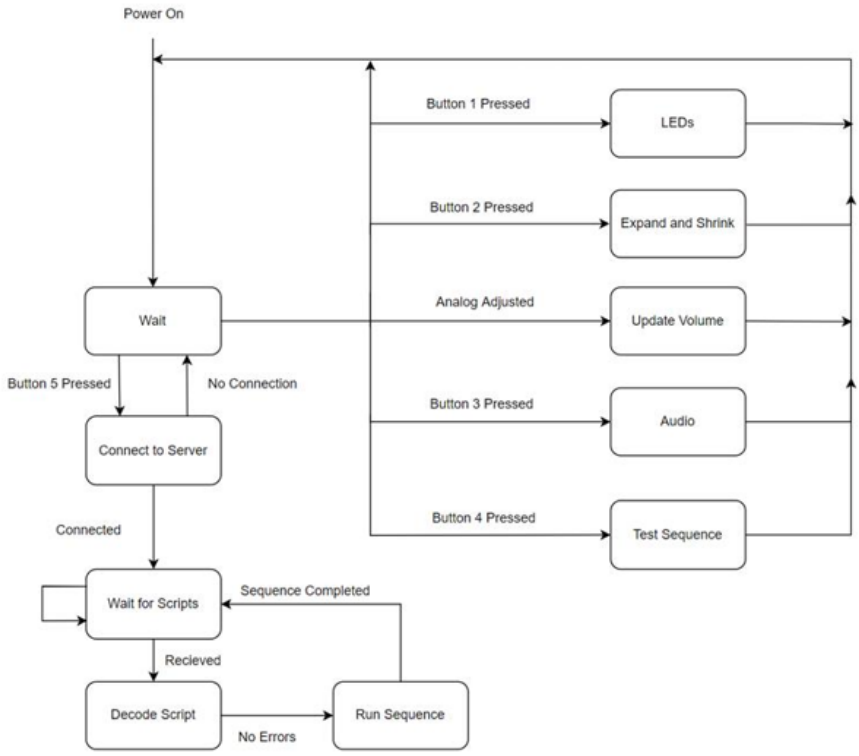

Figure 16. Software State Flow Diagram that includes the setup, test, decode, and perform.

In addition, we have included the software architecture as shown in figure 15. Each class is designed for easy integration and easy testing. This means, that each of the classes can exist independently for easy testing, however, if necessary, when integrated, the classes can be run together easily. The core of the software architecture is the Main class, which setups and runs test sequences for each subsystem based on push button input. The connectToServer class oversees connecting to the director; there are Motion, Audio, and LED classes that are meant to perform the tasks of their respective subsystems.

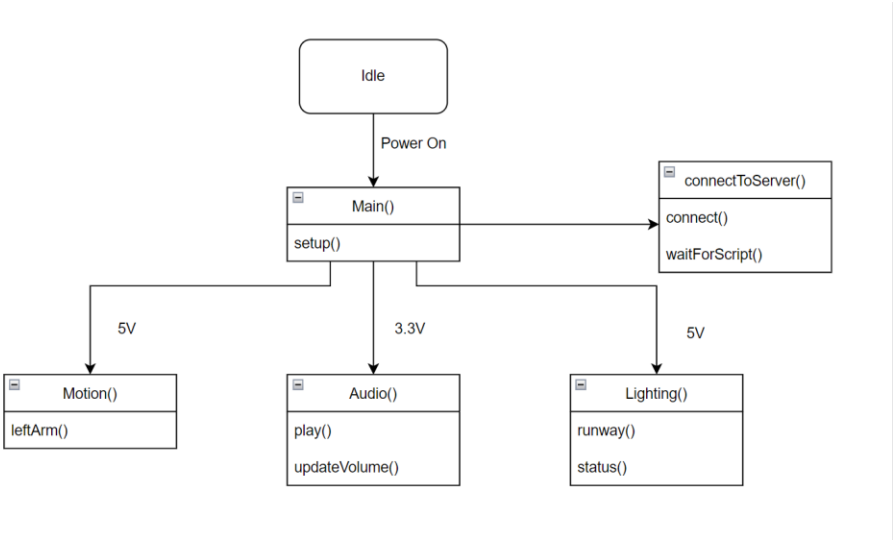

Figure 17. Software Architecture detailing all functions for a class

Software Framework

setup(): initialize global/local variables. Define inputs and outputs

connect(): attempt to connect to the director.

waitForScript(): if connection successful wait for script until timeout (10s). Flash light if nothing is recieved in time.

leftArm(): Rotate arm motor by 90 degrees (total of 2 sec)

play(): Play "Im Late" Music

updateVolume(): Read analogIn from potentiometer and update speaker volume accordingly

runway(): light up leds sequentially

status(): Sets status of blue LED. Input of 1 turns the LED on

Figure 18. Software Framework outlining the functions shown in the software architecture

Software Simulations

For simulation purposes, we developed a program to run the above architecture. The main purpose was to implement several try blocks to help the user see where an error occurs if the thespian were to fail. We aimed to be all encompassing in the event of connection failure. Since the software directly the user interface (status buttons, testing sequences, etc.) a form of debugging was needed in the initial stages to ensure everything was working as it should. Figure 19 shows the results of the program if it was run successfully.

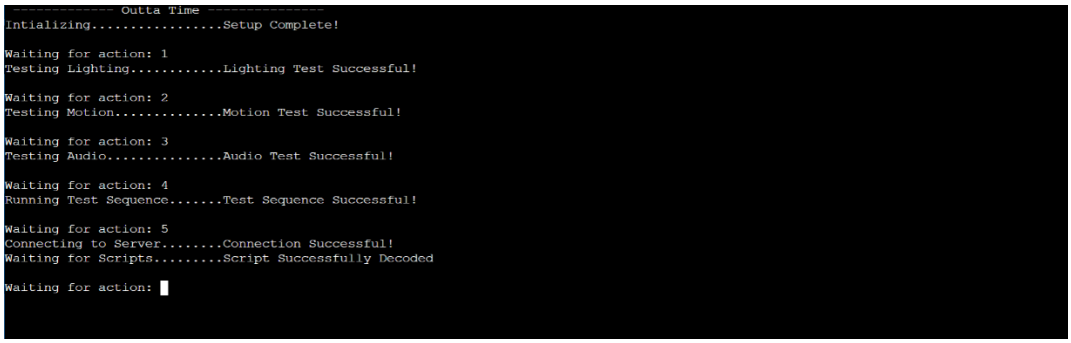

Figure 19. Successful run of program demonstrating button interaction

We also had a slight setback as we switched from the Raspberry Pi 0 W to the ESP32, however, it was easier for us to code on the ESP32 as it will be in C/C++. After the base program was run, we simulated connecting to the director via ESP32. Figure 19 shows the results of that simulation. We ran into issues using Ubuntu to run the programs as it lacked the updated libraries needed to do so. Running the director script in anaconda solved the issue.

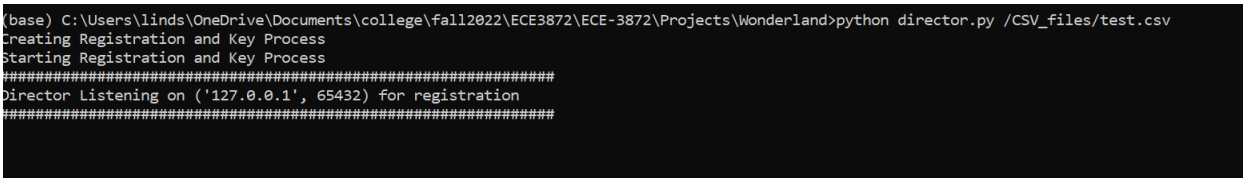

Figure 20. Successful connection to Director

Mechanical Design

Our theme is Alice in Wonderland and the character on top of the box is The White Rabbit. Our mechanical design utilizes two servo motors to rotate the rabbit's head and arm. The rabbit's holding a clock will be revealed from behind his body with a CCW motion. The rabbit's head will then spin back and forth in a CW and then CCW motion to show he is worried about being late.

To achieve these motions, the rabbit was designed to have three parts: the head, the body, and the watch. These three parts would be laser cut and then engraved to get the details of the rabbit. The head is in front of the body, the arm with the watch is behind the body, and the body is attached to the top of the box. Our two servo motors will be attached to the body at one end and then attached at their other end to the head.

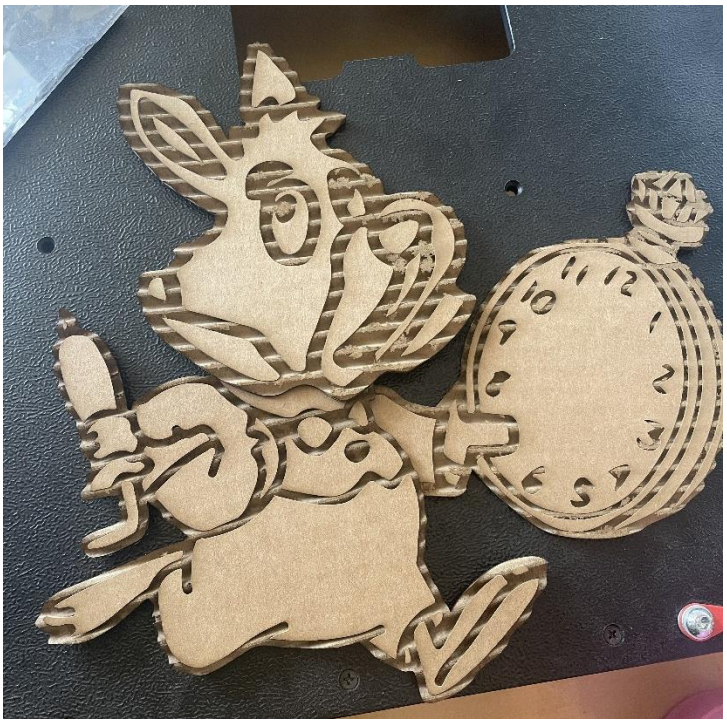

Figure 21. Cardboard laser cut test of rabbit design.

We went with a simple box design. On one side of the box there will be 4 test buttons and an LED to indicate if the system is on. On the front of the box, we will have an engraving with the GT logo and our team's name. The back of the box will be transparent by utilizing acrylic material. This will allow for our electrical connections to be visible. On the top of the box, we will place the rabbit as well as laser cut holes to fit LED strips. These LED strips will represent a path for the bunny to go down.

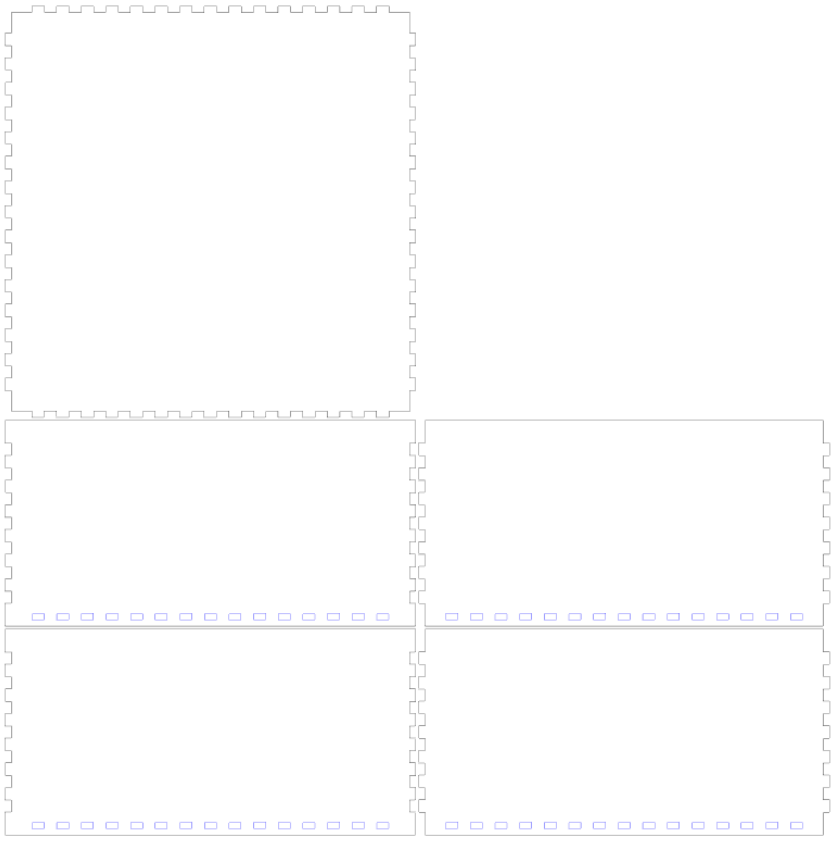

Figure 22. Illustrator .svg file for the base design, 20 cm x 20 cm x 10 cm

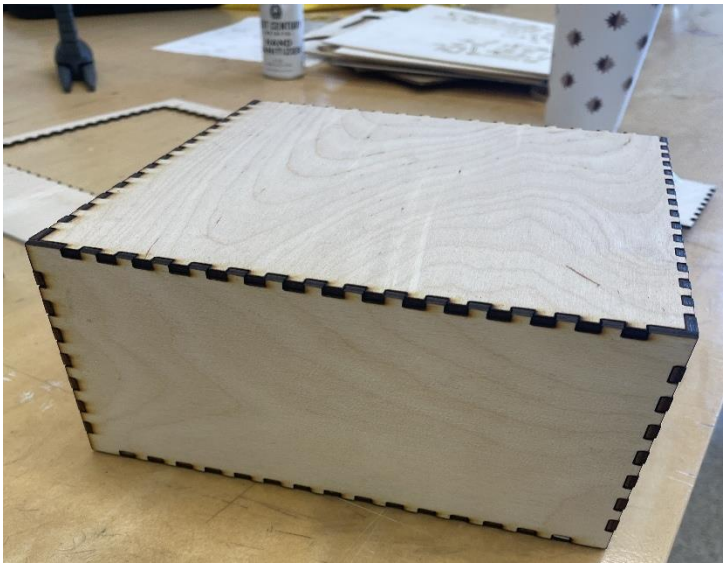

Figure 23. Laser cut of box design without engravings

PCB

Three PCBs created using EAGLE host the various subsystems:

1. Main Board: Processor, Audio, Motion, LED
2. User Interface Board: User Interface
3. Power Board: Power

Main Board

The ESP32 microcontroller is located on the Main Board. Through terminal blocks, the board receives 5V from the Power Board and provides 5V and control signals to the two servos (Motion subsystem) and two LED strips (LED subsystem). A 9-pin terminal block sends 3.3V from the microcontroller's voltage regulator to the User Interface Board with various analog/digital inputs and outputs. On the Main Board are the Audio subsystem's SD card reader and I2S amplifier and an ADC to convert the analog volume signal from the User Interface Board.

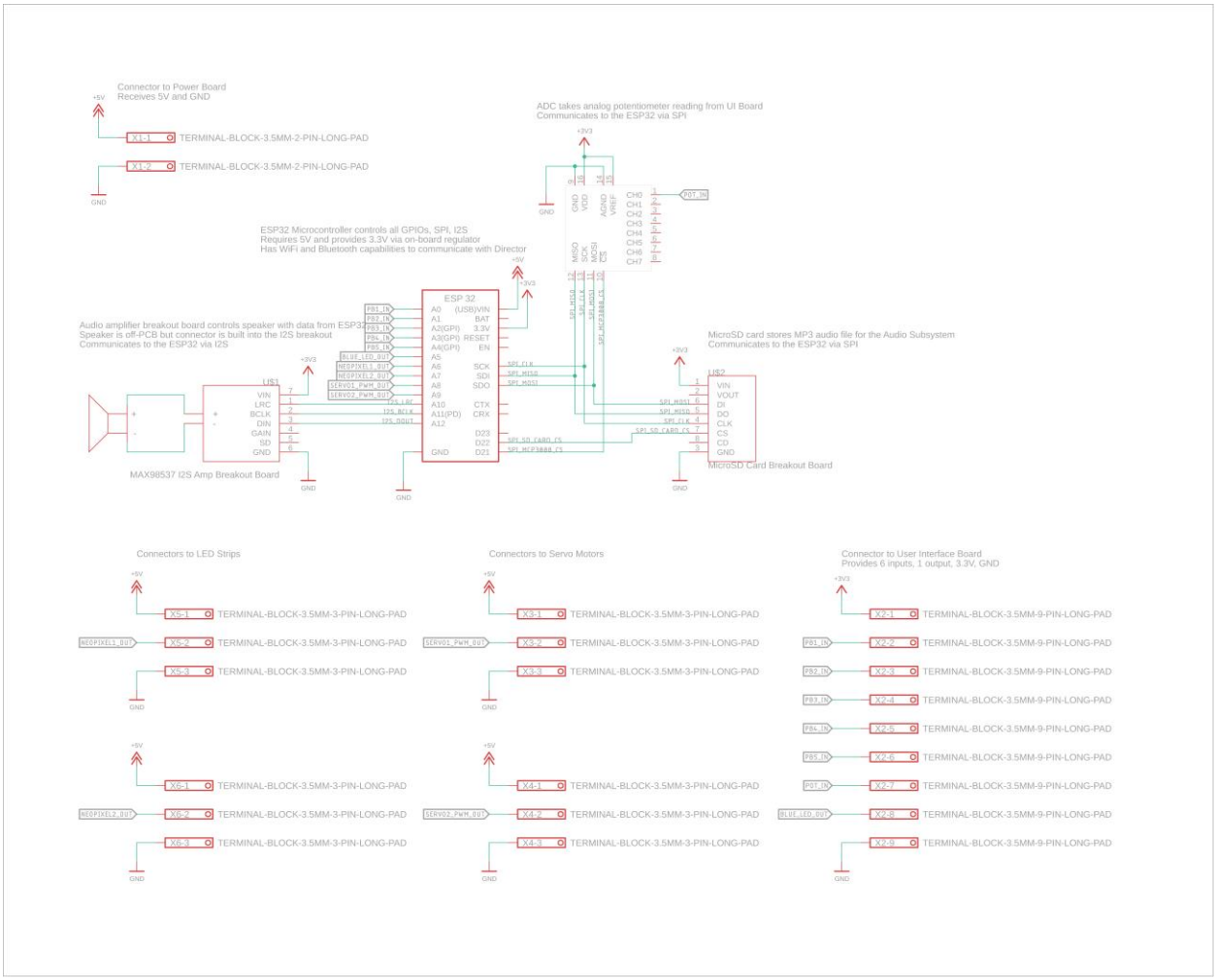

Figure 24. EAGLE schematic of the Main Board. The ESP32 in the middle directly connects to the I2S amplifier, SD card reader, and ADC. Terminal blocks on the top and bottom connect to other PCBs (User Interface Board, Power Board) or external components (servos, LED strips).

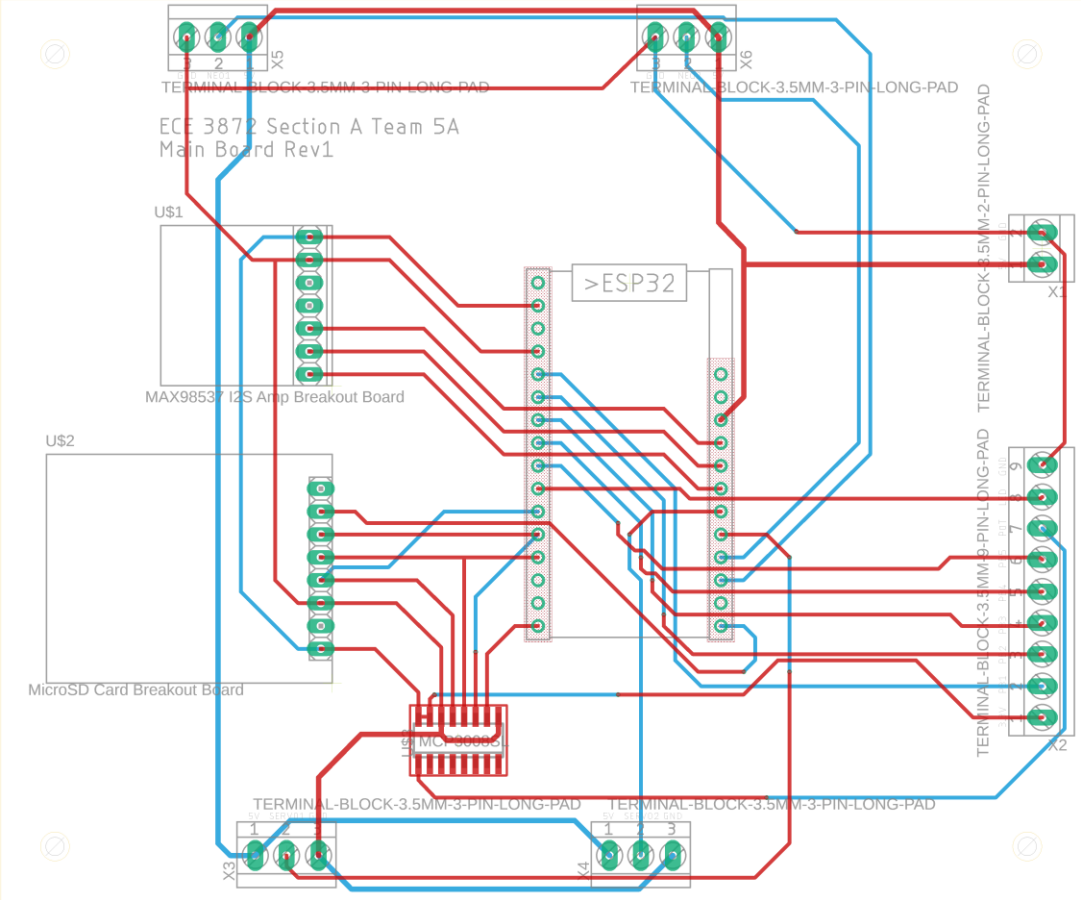

Figure 25. EAGLE board for the Main Board. Notice the terminal connectors being on the edge of the board, with silkscreen text for each pin's function.

User Interface Board

The User Interface Board hosts components for user inputs or outputs. A green and a blue status LED, respectively, show the power status and wireless connection status of the system. Four tactile pushbuttons allow the user to test various subsystems individually and another enables/disables wireless connection. A potentiometer provides analog input to the system to control the volume. This board receives 3.3V from and communicates using digital and analog signals with the Main Board.

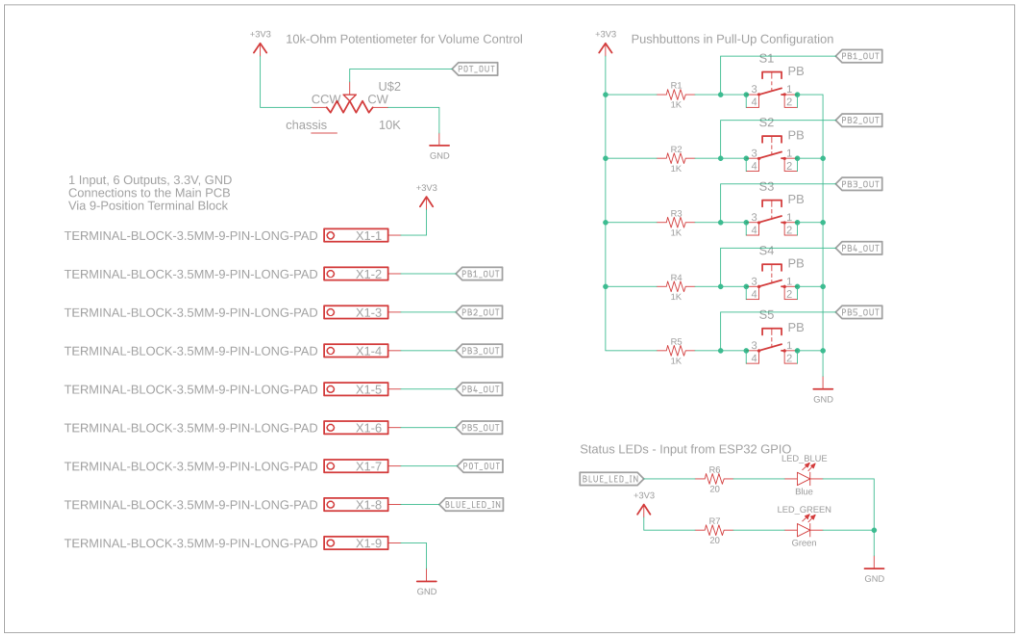

Figure 26. EAGLE schematic of the User Interface Board. The terminal block on the bottom-left provides all connections. Pushbuttons have hardware pull-ups and are debounced in software.

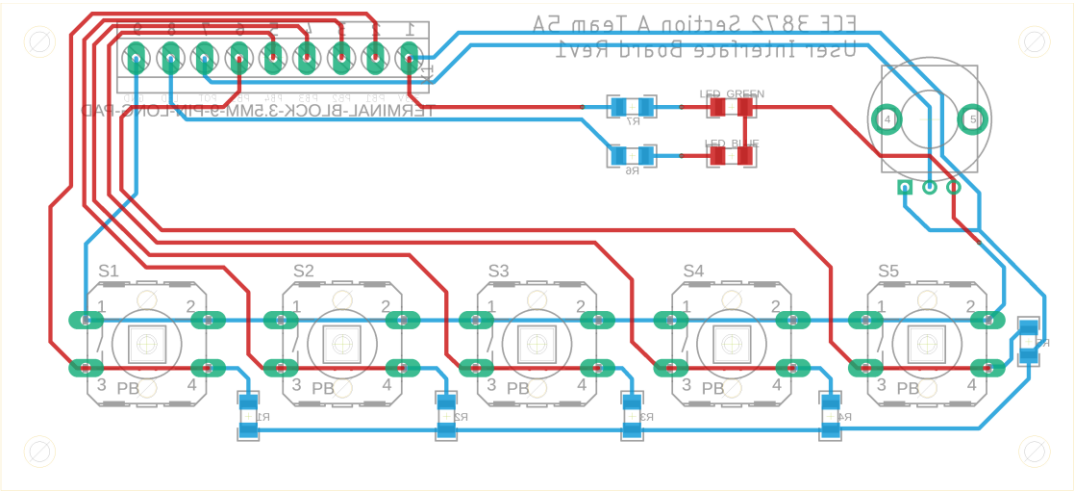

Figure 27. EAGLE board for the User Interface Board. Notice that all user facing components (pushbuttons, LEDs, potentiometer) are on the top layer and the other components (terminal block, resistors) are on the bottom layer – cutouts from the side panel are needed only for user facing components.

Power Board

The input to the Power Board is 5V 3A from a wall transformer. A 3A fuse on the power rail ensures that the system does not exceed that limit in case the wall transformer has a power surge. A switch toggle allows the user to turn off power. Outputs are connected to the Main Board, which further distributes 5V throughout the system.

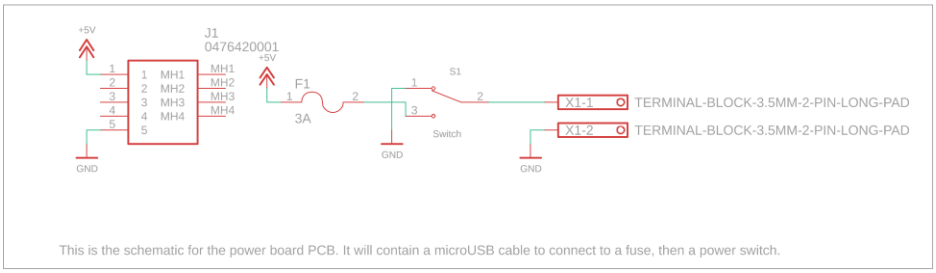

Figure 28. EAGLE schematic of the Power Board. A Micro-USB female connector receives 5V from the wall transformer and a terminal block sends the fused 5V to the Main Board.

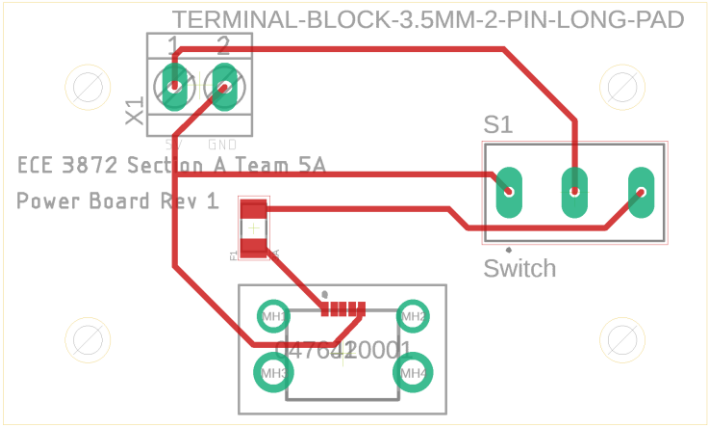

Figure 29. EAGLE board for the Power Board. Unused data pins on the Micro-USB connector are purposely left floating.

For the Power Board, we ended up needing to cut the trace connecting the GND line with the switch. This is because when the microcontroller had a cable plugged in (i.e. for firmware uploading or serial debugging), switching the Power Board "off" would short the 5V from the USB cable and GND. Hence, we left the "off" state as a floating value instead of GND.

Schedule

Figure 22. describes what our schedule looks like. It is less granular than expected because our group decided to house all concrete action items within the team meeting notes (Figure 23.). Each concrete action item is related to the subsystem(s) that a particular team member is working on for the week along with a concrete due date that is influenced by our schedule. Every meeting, each team member will give status updates, and if a task is finished, we will go back to when the task was originally assigned, and strike through the task to indicate that it has been completed.

ECE 3872 Group A05 Schedule

	Week Number												
Task	1	2	3	4	5	6	7	8	9	10	11	12	13
Brainstorm (Overall Project Goals)	Jon	Jon											
Design Top Level (Electrical) Block Diagram	Bo Han	Bo Han											
Align Mechanical System Requirements	Selene	Selene											
Design Software Block Diagram	Lindsey	Lindsey											
Proposal													
Electrical Design			Bo Han	Bo Han									
Mechanical Design (Includes Box Design)			Selene	Selene									
Software Design			Lindsey	Lindsey									
Audio Subsystem Design				Jon	Jon								
Motion Subsystem Design				Selene	Selene								
User Interface Subsystem Design				Bo Han	Bo Han								
LED Subsystem Design				Bo Han	Bo Han								
User Interface Subsystem Component Selection				Bo Han									
LED Subsystem Component Selection				Bo Han									
Audio Subsystem Component Selection					Bo Han								
Motion Subsystem Component Selection				Selene	Selene								
Power Subsystem Design				Jon	Jon								
Plan Software Simulation				Lindsey	Lindsey								
Draft Software Archictecture Diagram				Lindsey	Lindsey								
Draft Software Framework				Lindsey	Lindsey								
PDR Document				Jon	Jon								
Build Preliminary Test Plan					Jon								
PDR													
Audio Subsystem Simulation/Refinement						Bo Han							
LED Subsystem Simulation/Refinement						Bo Han							
User Interface Simulation/Refinement						Bo Han							
PCB Grouping/Connectors Brainstorm						Bo Han							
Power Subsystem Simulation/Refinement						Jon	Jon	Jon					
Powerboard PCB							Jon	Jon					
Motion Simulation/Refinement						Selene	Selene	Selene					
Software Simulation/Refinement						Lindsey	Lindsey	Lindsey					
Work on Software Framework						Lindsey	Lindsey	Lindsey					
Simulate Software for Lighting Subsystem							Lindsey	Lindsey					
Simulate Software for Audio Subsystem							Lindsey	Lindsey					
Simulate Software for Motion Subsystem							Lindsey	Lindsey					
Attempt To Connect to Director							Lindsey	Lindsey					
Create Electrical Simulation Video							Bo Han						
User Interface Board PCB Design							Bo Han						
Main Board PCB Design							Bo Han						
Complete Electrical Portions of CDR Document								Bo Han					
Complete Parts Order Form								Bo Han					
Complete Design Docuent							Jon	Jon					
Complete BOM							Jon	Jon					
CDR													
Build Audio Subsystem									Bo Han				
Build LED Subsystem									Bo Han				
Build User Interface Subsystem									Bo Han				
Build Power Subsystem									Jon	Jon			
Build Motion Subsystem									Selene	Selene			
Connect microControlller to Director									Lindsey				
Write software descriptions for CDR									Lindsey				
Upload all simulation videos/documentation										Lindsey			
Code Software Components										Lindsey			
Test Audio Subsystem									Bo Han				
Test LED Subsystem									Bo Han				
Test User Interface Subsystem									Bo Han				
Test Power Subsystem										Jon	Jon		
Test Motion Subsystem										Selene	Selene		
Electrical/Software Integration										Bo Han			
Electrical/Mechanical Integration										Bo Han			
Buffer Time to fix Electrical issues											Bo Han		
Test Software										Lindsey	Lindsey		
Final Integration											Jon	Jon	
Finish Deliverables											Jon	Jon	
Final Inspection and Demonstration													

Milestones	Bo Han
Tasks	Selene
	Lindsey
	Jon

Figure 30. Team Schedule.

- Action Items
  - Everyone
    - CDR is Due: Sunday!
  - Lindsey
    - Simplify wording in test document: Friday
    - Upload software simulations (screenshots of simulations): Friday
    - Record and voiceover video simulations and add to simulations folder: Friday
    - ~~Connect to ESP32 with director: Friday~~
    - Software Design Section of the CDR: Friday
  - Bo Han
    - ~~Complete parts order: Thursday~~
    - ~~Finalize BOM: Thursday~~
    - ~~Update electrical subsystems in the CDR with design/simulation process: Thursday~~
    - ~~Electrical Design in Design Document: PCBs etc: Thursday~~
    - ~~Trade off in design document: Thursday~~
  - Selene
    - ~~Finish simulation of motors (get video and voiceover): Friday~~
    - ~~Get a final picture of the rabbit for CDR: Friday~~
    - Do mechanical design section for CDR: Friday
    - Do motion subsystem for CDR: Friday
    - ~~Tradeoffs in CDR~~
    - Change all numbered requirements from Inspection: Friday
  - Jon
    - ~~Finalize schedule: Friday~~
    - Power subsystem design: Friday
    - Review everything when it's done: Saturday
    - ~~Review CDR for any more tasking: Friday~~

Figure 31. Sample Action Items List for 10/10/2022.

Integration

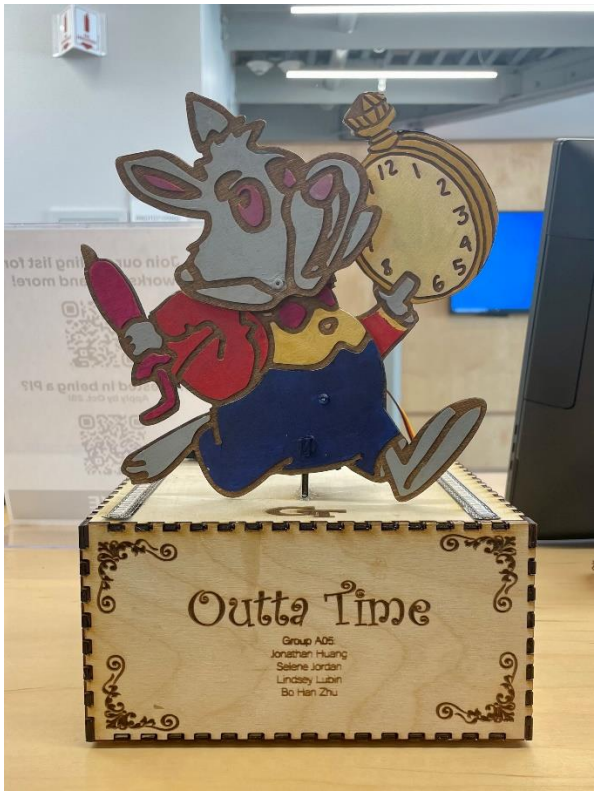

Figure 32. The front of the complete robot thespian.

Figure 32 shows the front of the complete robot. To achieve this final product, we had to go through mechanical-electrical integration and software-electrical integration. For mechanical-electrical integration, we were able to mount the PCBs by drilling 1/8-inch holes through the wood. To hold the PCB in place, we used nuts and screws, with an additional nut standing in the way between the PCB and the wood serving as a standoff to reduce shock to electrical components. For the user interface board, we tested many wood cutouts to ensure the pushbuttons, potentiometer, and status LEDs would stick through the wood and be accessible to the user.

One issue we ran into was during the mounting of the servos. The servos were held in place behind the thespian using wood. However, we attached everything before ensuring the servos could rotate. This was disastrous as the head servo did not have enough spacing – it could not move its load due to friction with the wood in-between. We tried to drill a larger hole through the wood in-between to minimize friction, but the drill got too far and permanently damaged the servo instead.

For software-electrical integration, we were able to copy over many of the firmware scripts we used during simulation of individual subsystems and modify the pin definitions according to the EAGLE schematic. We created a thread for each function in FreeRTOS. One issue we ran into was overloading one core of the ESP32 and so we moved the audio subsystem (the most CPU-intensive operation that also required uninterrupted execution) to its second core.

Software High-Risk Components:

- Multithreading using an RTOS library most of us never even heard of before (FreeRTOS)
- Although the ESP32 had built-in wireless connectivity, connecting to the Director was not ensured due to MCU-specific implementations. Using this microcontroller also left us “stranded” as other groups could help each other out on this obstacle using the Pi.

Hardware High-Risk Components:

- LED strip soldering and mounting. The LED strips did not have soldermask on its pins and therefore we needed to find a way to allow the solder to stick.

Mechanical High-Risk Components:

- Servo mounting. We could not really use screws as the servos did not have housing for them and our setup caused too much friction between components.

Repository Management:

- We have a GitHub repository that stores both our simulation scripts and final firmware:
  - https://github.gatech.edu/llubin3/ECE3872-A05
- We used Microsoft Teams/OneDrive for shared file management.

Configuration Controls/Diagnostics:

- Pushbuttons:
  - One pushbutton tests the overall system without wireless connection.
  - Three pushbuttons test the audio, LED, and motion subsystems respectively.
  - One pushbutton initiates wireless connection.
- Status LEDS:
  - A blue LED indicates wireless connection status.
  - A green LED indicates power status.
- Potentiometer:
  - Controls the volume output of the speaker.