
# Design Document

## *Bill the Chimney Sweep*

*Team #7: Luke Gillette, Apuroop Mutyala, Marko Stepniczka, Zekai Wang*

**Date: November 15th, 2022**

# Table of Contents



# Revision Record

| **Date** | **Author** | **Comments** |
|---|---|---|
| Sep 25, 2022 | Team | Document Created |
| Oct 13, 2022 | Marko | Software Section Updated |
| Oct 16, 2022 | Zekai | Mechanical Section Updated |
| Oct 16, 2022 | Luke | Added PCB schematic |
| Oct 18, 2022 | Luke | Added final PCB board and schematic |
| Nov 15, 2022 | Marko | Software and Electrical Sections Updated |
| Nov 15, 2022 | Zekai | Mechanical Sections Updated |

## Project Description

We will design a chimney box where Bill the Lizard will rise out from. Bill will be strapped onto a threaded rod and move vertically as the rod is propelled by motor and gear. Light will shine out of Bill's eyes as it speaks. The box underneath will look like a brick chimney with red LEDs emulating fire. The box will have a speaker, an on/off button and several mode buttons on one side, and all other parts will be hidden inside.

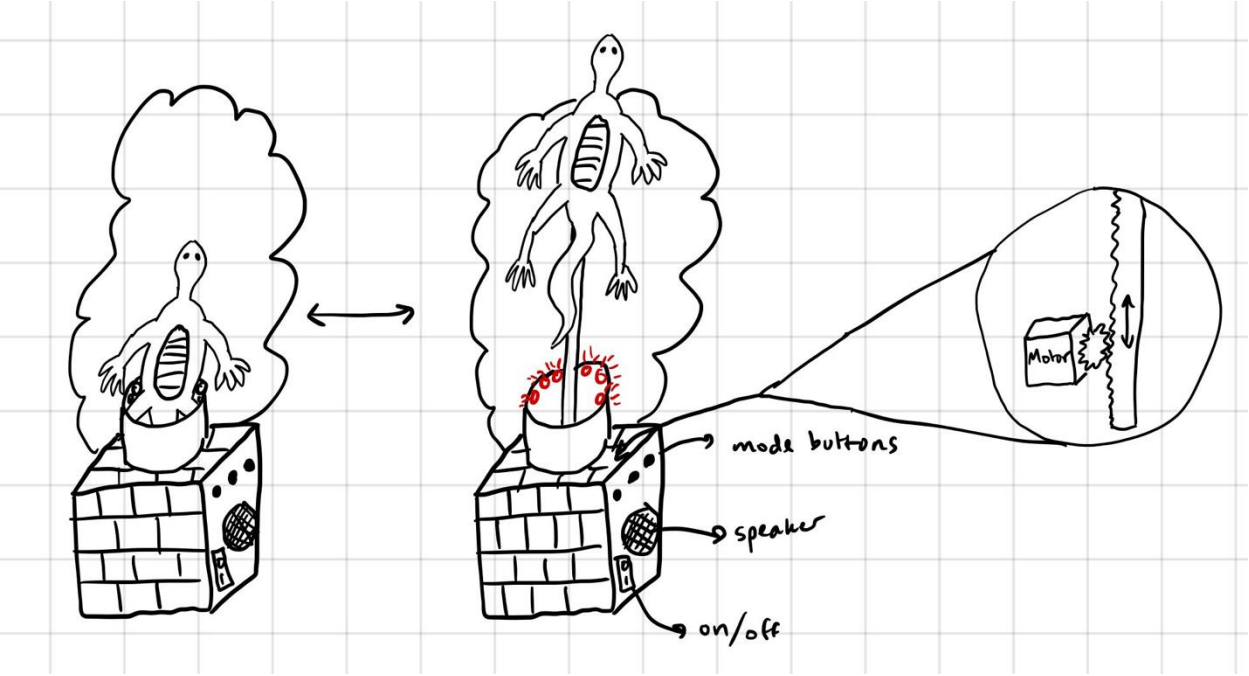


*Figure 1: Our initial illustration of the Chimney Box*

## System Design

The system will include inputs from the user to specify behavior in the form of a switch and multiple push buttons, as well as 120V AC power, and wireless transmission from the director

which will determine audio output. There will be five subsystems, including the processing, peripherals, power, audio, and user interface subsystems. The processing subsystem will be built off a Raspberry Pi Zero 2 W and will handle the main logic as well as wireless communications. The peripherals subsystem will include a 4.5V Stepper Motor, a dual H-Bridge Motor Driver, and an array of Red and Orange LEDs. The power subsystem will have a 120V AC to 5V DC transformer and a user input switch. The audio subsystem contains the PWM Audio Amplifier and a 0.1W speaker.

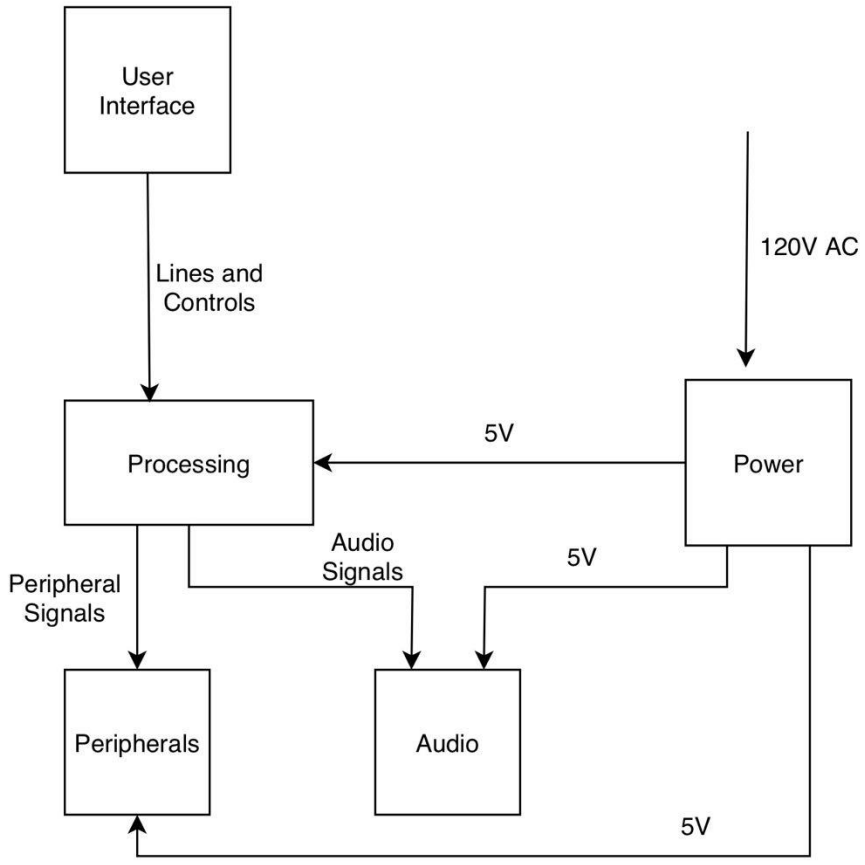


*Figure 2: System Diagram*

| Interface | Source | Destination | Description |
|---|---|---|---|
| Lines and Controls | User Interface | Processing | Lines of the Character |
| Peripheral Signals | Processing | Peripherals | PWM signals to control the motors |
| | | | DC output to control and power LEDS |
| Audio Signals | Processing | Audio | PWM signals to output audio |
| 120V AC | System Input | Power | 120V AC input power |
| 5V | Power | Processing | 5V DC power 10W |
| | | Peripherals | 5V DC power 5.4W |
| | | Audio | 5V DC power 0.1W |

*Table 1: Sub-system inputs and outputs*

## Subsystem Designs

Bill the Lizard will be based off five subsystems. These include the processing, peripherals, power, audio, and user interface subsystems. The functions of each of these subsystems are detailed below.

### User Interface Subsystem

The user interface subsystem will not need to be powered, as it will just provide pushbuttons for the user to interact with. It will provide 4 different pushbuttons to specify three different testing modes that the user can choose from, as well as a production mode which will incorporate all of the modes together. This will allow the user to test the audio, the movement, and the lighting of the system separately, as well as put it all together for the final product. The pushbuttons will produce digital signals, which will be output to the processing subsystem. It will also provide a switch which will be translated into a digital signal and output to the power subsystem to control the power to the rest of the subsystems.

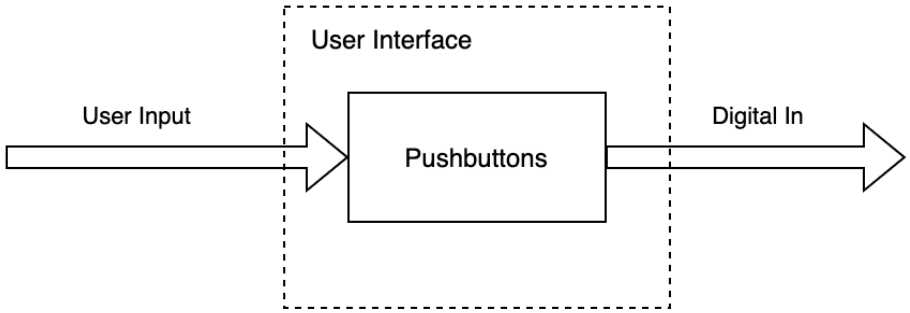


*Figure 3: User Interface Subsystem*

### Audio Subsystem

The audio subsystem will receive 5V DC from the power subsystem, along with a PWM input from the processing subsystem. It will send these inputs to the Class D amplifier, and the amplifier will then send an analog audio signal to the speaker, which will output sound.

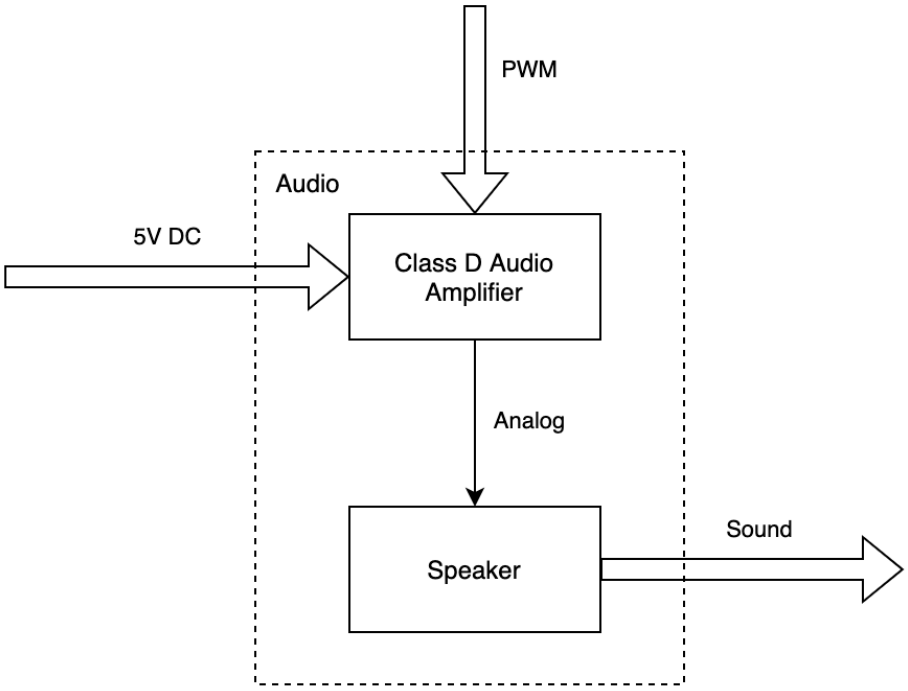


*Figure 4: Audio Subsystem*

## Peripherals Subsystem

The peripherals subsystem will receive 5V DC from the power subsystem, as well as a PWM signal and digital signals from the processing subsystem. It will send the power and PWM signals to the H-Bridge motor driver, which will control the 4.5V Stepper Motor through the 2 motor leads. It will also send digital signals to the LEDs, making them emit light which Bill is extended.

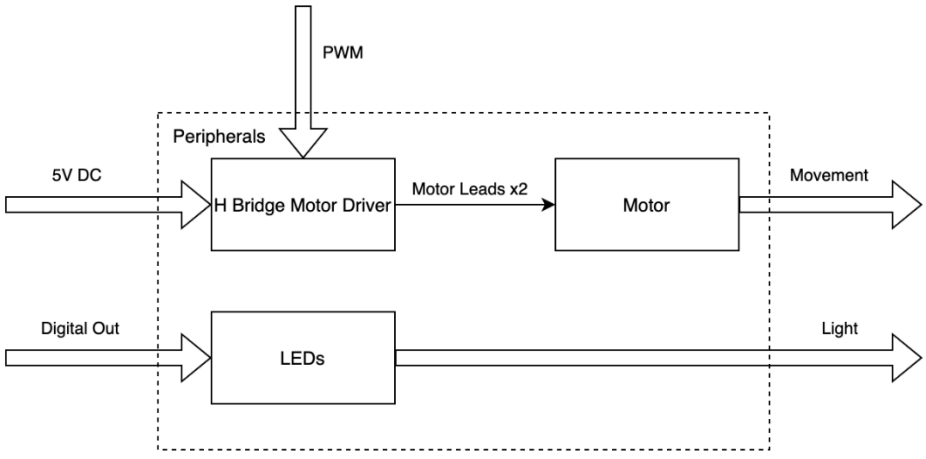


*Figure 5: Peripherals Subsystem*

## Power Subsystem

The power subsystem will take the 120V AC signal from the wall and input it to the 120V AC to 5V DC transformer. It will then send the 5V DC signal through the power switch which will be

opened/closed based on input from the user interface subsystem before outputting it through a fuse to the rest of the subsystems.

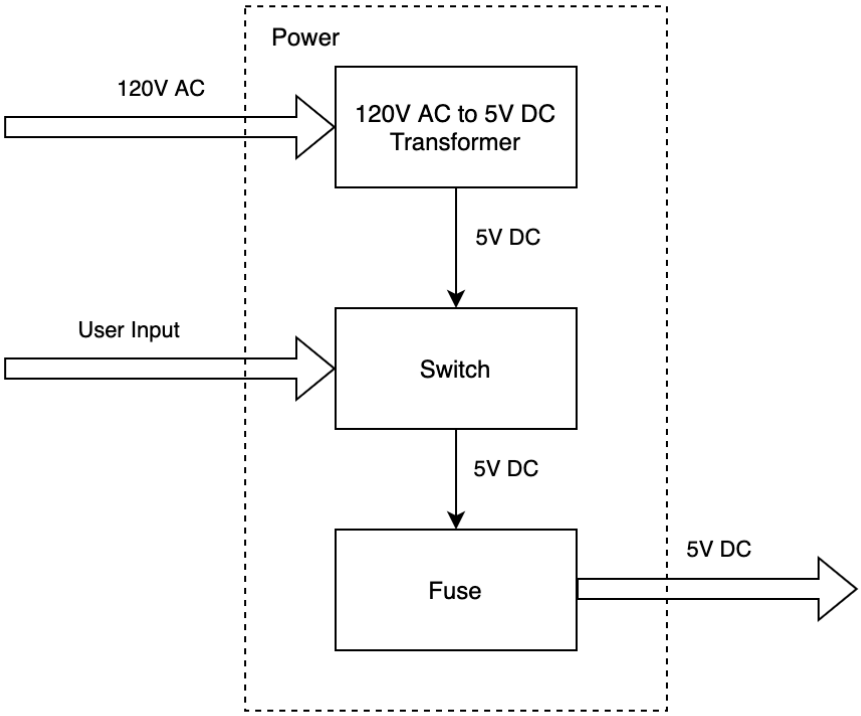


*Figure 6: Power Subsystem*

## Processing Subsystem

The processing subsystem will take 5V DC from the power subsystem, as well as the digital signals from the user interface subsystem and the wireless input from the director. It will use the 5V DC signal to power the Raspberry Pi, and the Raspberry Pi will use the digital signals to determine its state in the state machine. It will then output digital and PWM signals to the peripherals and audio subsystems to help them function correctly.

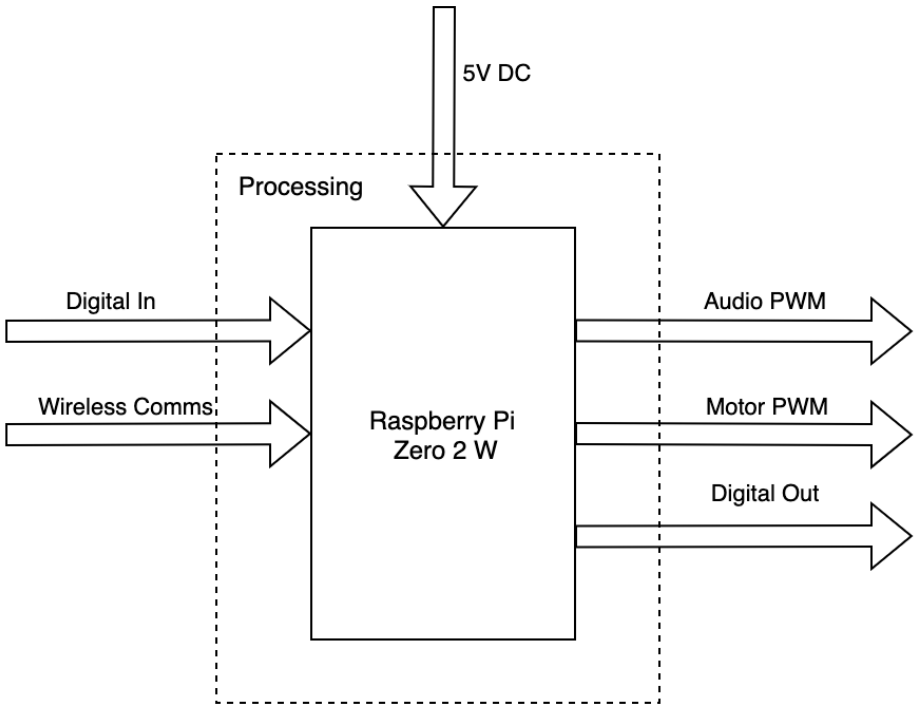


*Figure 7: Processing Subsystem*

## Constraints, Alternatives, and Trade-Offs

Direct linear movement is easy to achieve with parts such as solenoids, so at first that seemed like the obvious choice for raising and lowering Bill. However, it was found that due to the electromagnetic nature of solenoids, they are more geared towards producing rapid movements than slow and precise movements. So, alternatives were searched for, and a stepper motor was chosen due to its accuracy in rotations. While rotational movement won't directly raise and lower Bill with the desired effect, a system can be designed with a threaded or saw-toothed rod to be moved by a gear on the stepper motor, which will translate rotational movement to translational movement.

Choosing a microprocessor required looking at a couple of different options as well, namely choosing between Raspberry Pi and an Mbed Board. The tradeoffs were as follows:

1. Raspberry Pi
    a. More functionality
    b. Internet connectivity built in
    c. Full operating system onboard, so will be slower
    d. More complicated to work with
2. Mbed Board (LPC 1768)
    a. Less functionality
    b. No built-in internet connectivity
    c. Can have RTOS, e.g., can be faster
    d. Team has more experience so easier to work with

The Raspberry Pi was chosen mainly due to its included internet connectivity, because members of the team had worked with external Wi-Fi chips in the past, and they tend to be a hassle.

Sticking to the budget is also a constraint, as all parts chosen must come to cumulative price of less than $160 while also achieving the desired functions.

## Electronic Design

Our electrical design is based on a Raspberry Pi Zero 2 W, a custom PCB, and several breakout boards. The figure below shows how all the previously mentioned components in the sub-systems connect. The custom PCB has not been designed yet, but figures for that will be added when it has been.

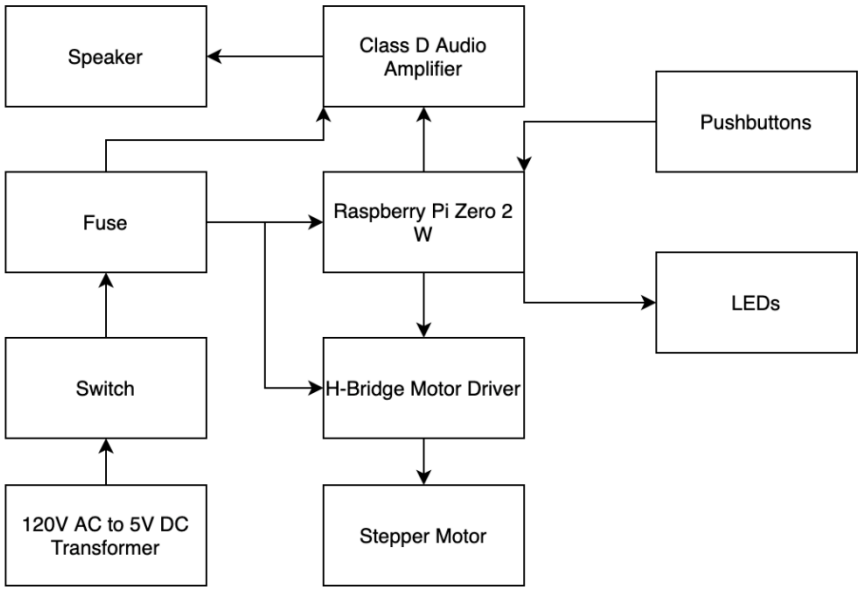


*Figure 8: Electronic component block diagram*

Figure 9 shows our analog circuit, which is a low-pass filter from our analog output pin on our Raspberry Pi to the A+ pin on our Audio Amplifier. This helps reduce noise by cutting out high frequency signals and static and lets us increase the volume of our speaker without much more noise.

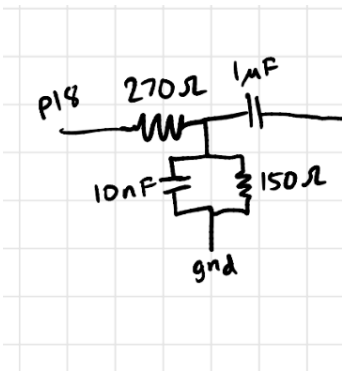


*Figure 9: Hand-drawn analog circuit*

Figure 10 shows our current PCB designs using Eagle software. We still have some questions about a couple of things, so we have not yet switched it over to the board design yet. This will be updated in the coming days. After testing on a bread board, we moved over all necessary components to the schematic in Eagle.

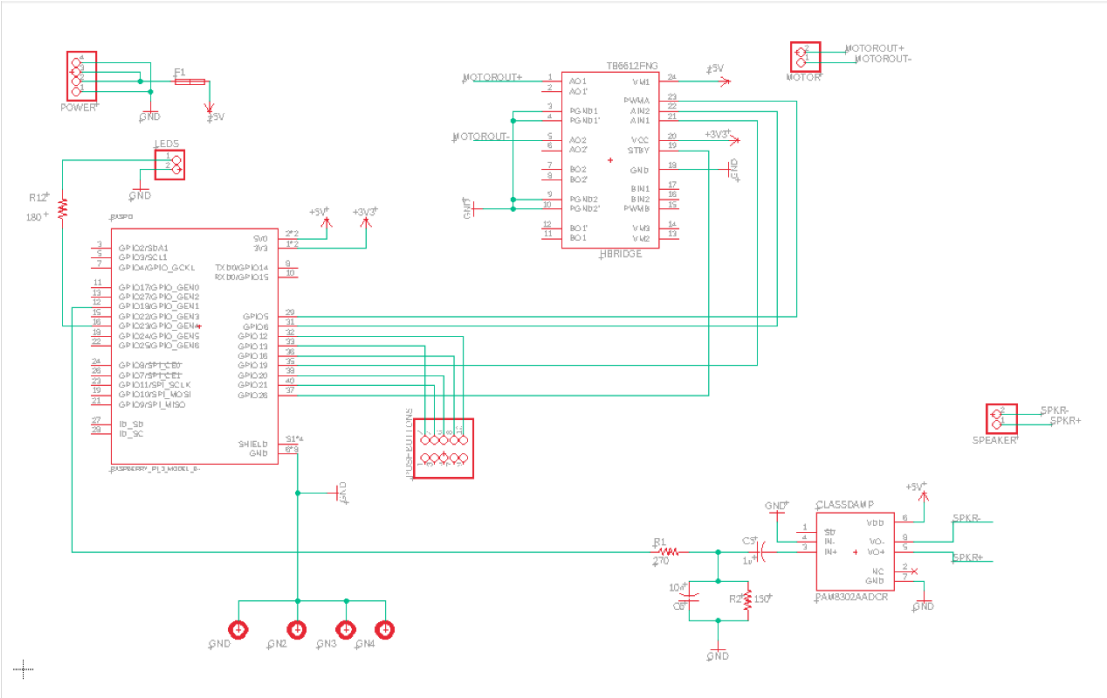


*Figure 10: Final schematic created in Eagle*

Figure 11 shows our final PCB design. Using our schematic, we then created the PCB design in Eagle. This is a two-layer design with our team's name, group number and section shown on the silk screen.

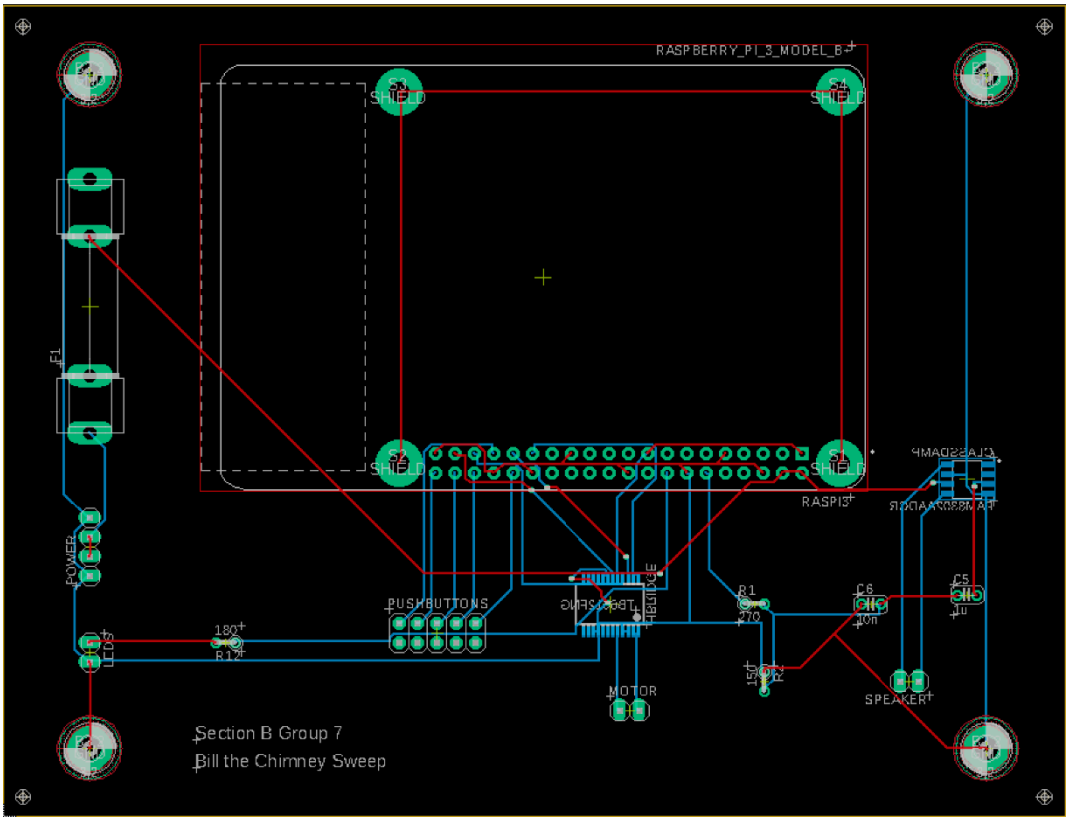


*Figure 11: Final PCB design created in Eagle*

## Mechanical Design

Our mechanical design features a threaded-rod and gear system to lift and lower a Lizard figurine. We decided the most convenient and efficient way was to 3-D print these parts. A stepper motor will be used to turn the gear and lift the rod, thus lifting Bill. We also decided to use a custom laser-cut design for our box.

The figure below details our initial vision for Bill the chimney sweep. During the first weeks of the semester, we chose this character and based our design off the character bill the lizard from 'Alice in Wonderland.'

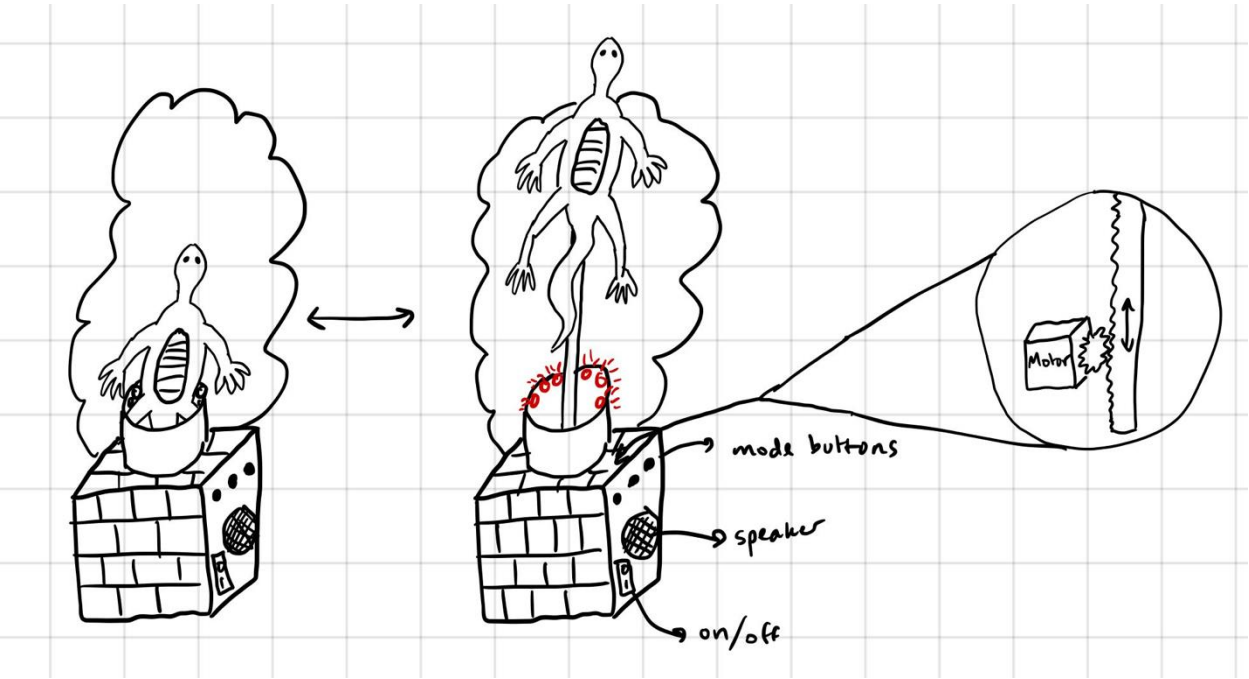


*Figure 12: Rough sketch of the overall design for Bill the Chimney Sweep*

The next figures show our hand drawn box panel designs. From these simple drawings, we will use Adobe Illustrator to create files to then laser cut our boxes. Our box design will be modeled like a chimney so the box will have a brick-like looking structure. These details were left out of the drawings to avoid clutter and highlight the main components.

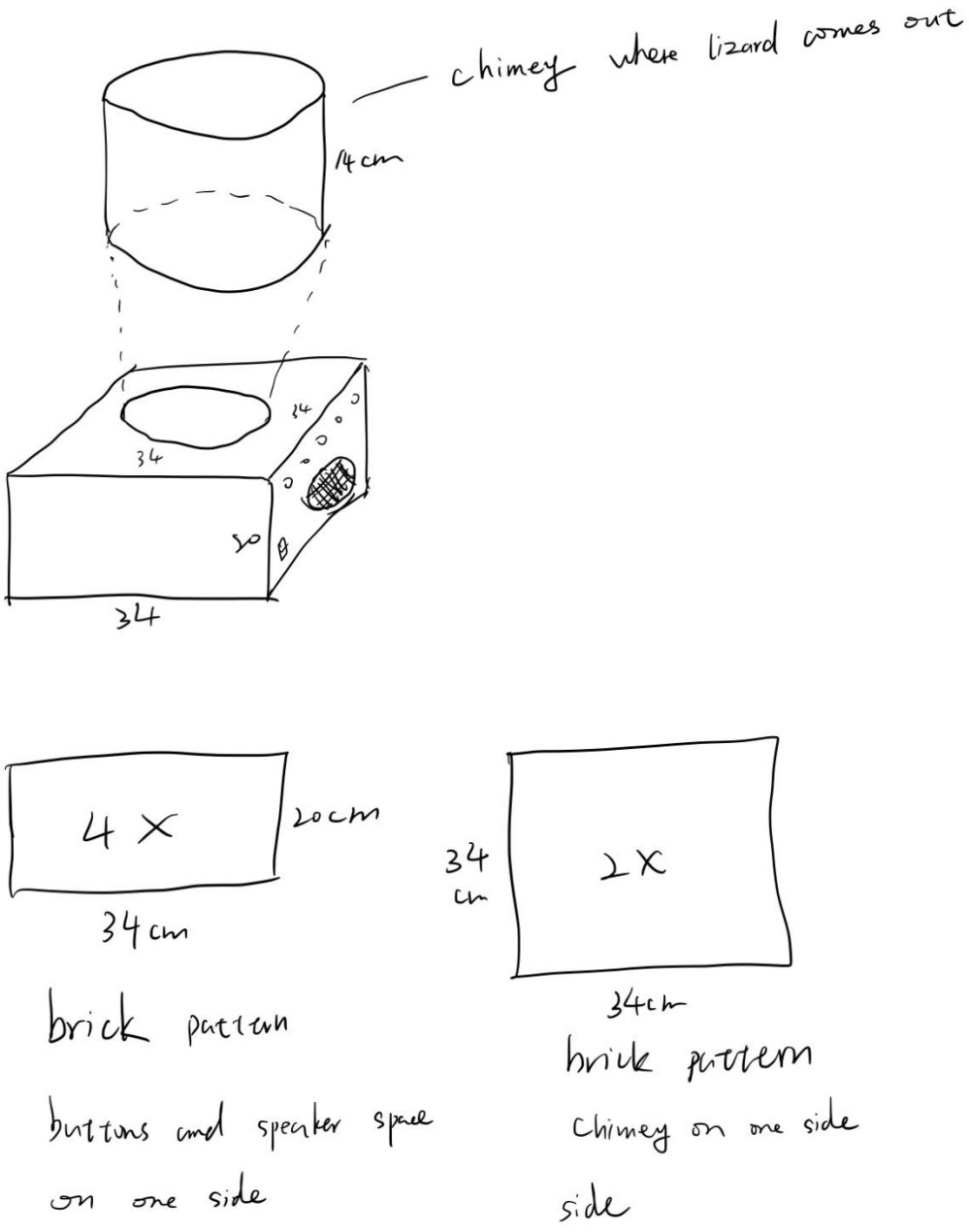


*Figure 13: Rough sketch of exterior design of the box*

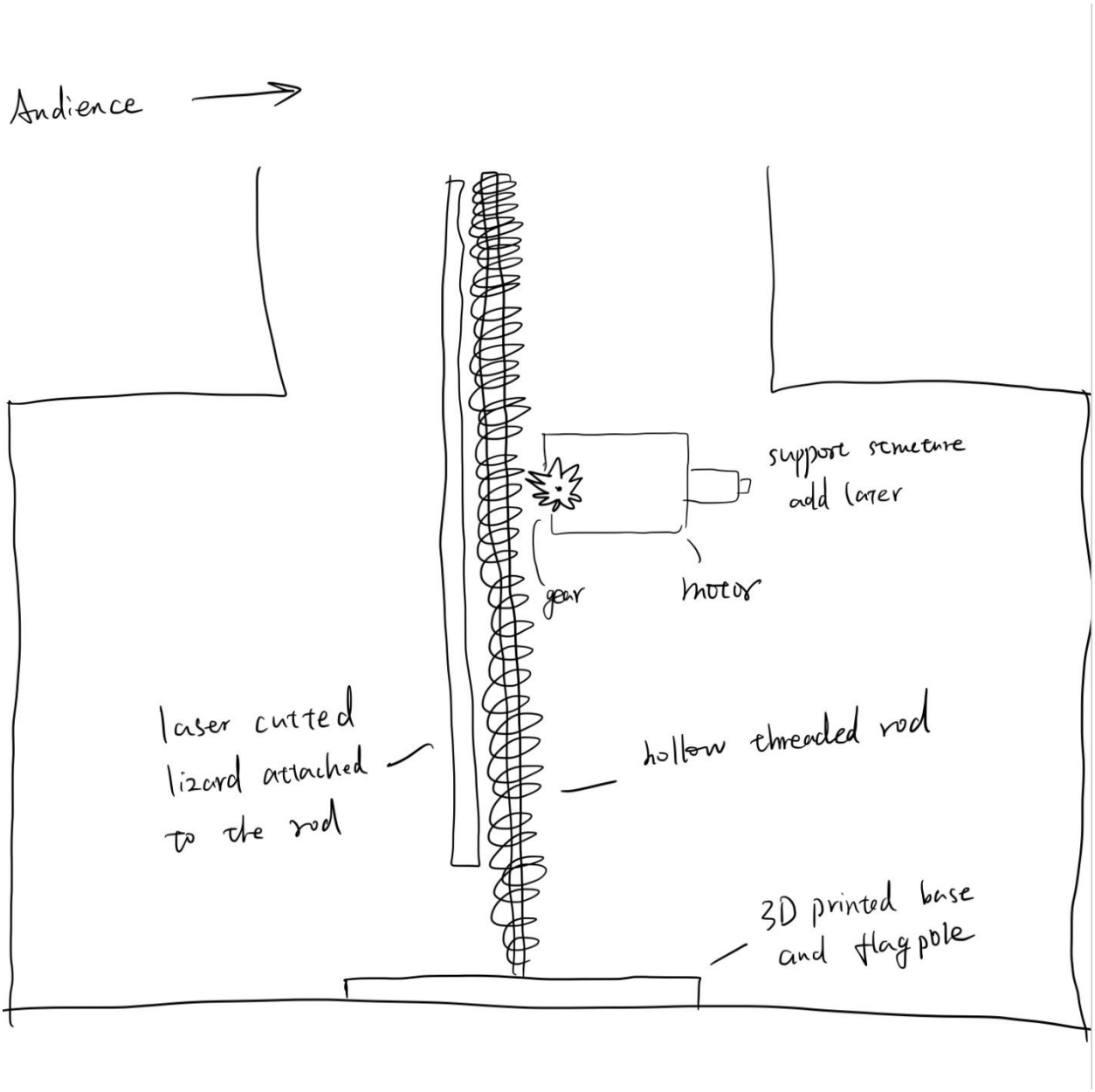


*Figure 14: Rough sketch of the interior design of the box that moves the lizard*

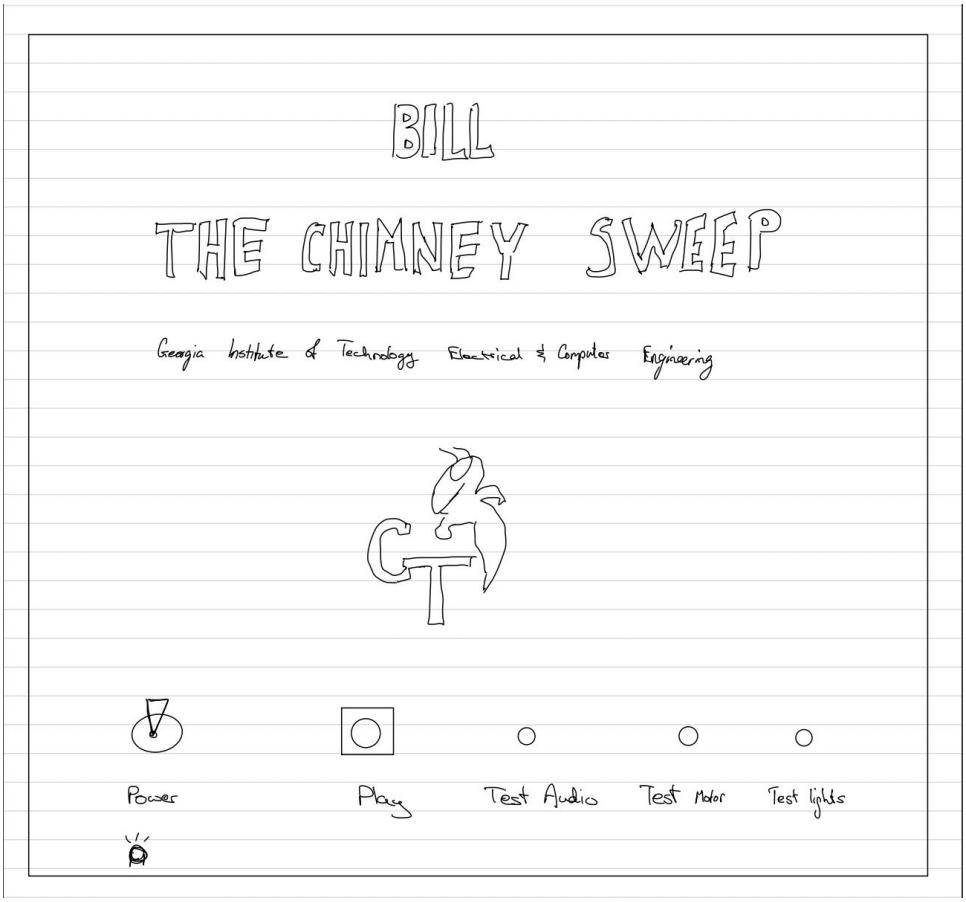


*Figure 15: Rough sketch of the front face of the box showing button layout*

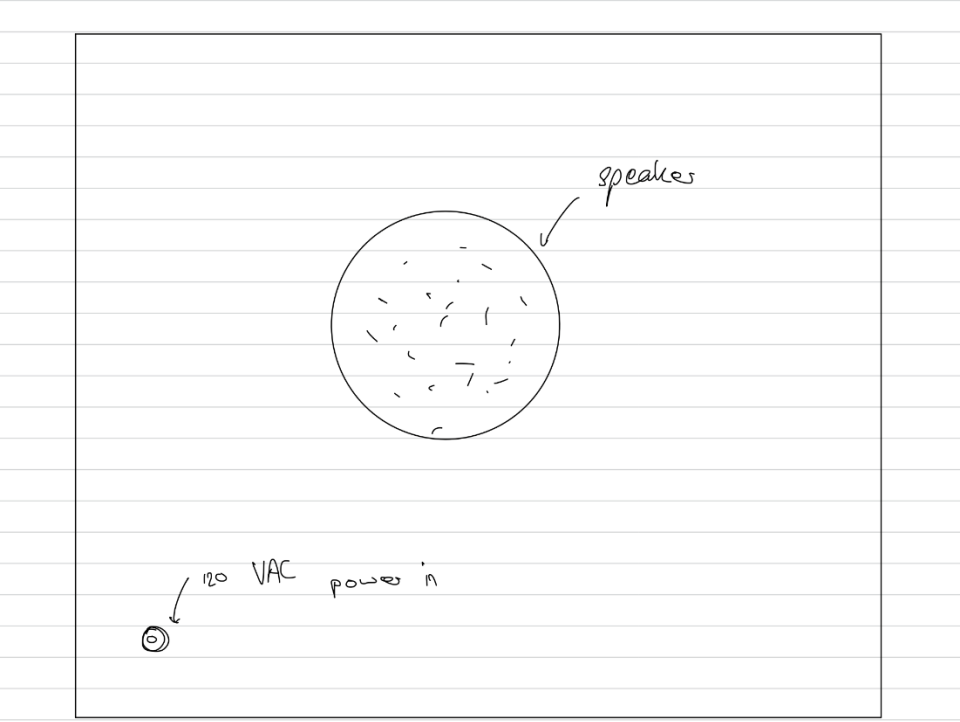


*Figure 16: Rough sketch of the rear face of the box, showing speaker, and input power layout*

One main feature of our project will be the rod which will lift and lower Bill out and into the box. In order to accomplish this, we plan to implement a rod and gear system. We will have a motor which will turn a gear and hence move the serrated rod up and down which Bill will be attached to. The cube-shaped box will be made of 6 sheets of laser cut wood and the chimney will be made of 4 sheets of laser-cut wood with laser-engraved brick pattern. The design will be finalized and be sent to production as soon as the electrical systems complete and we can arrange the space inside the box.

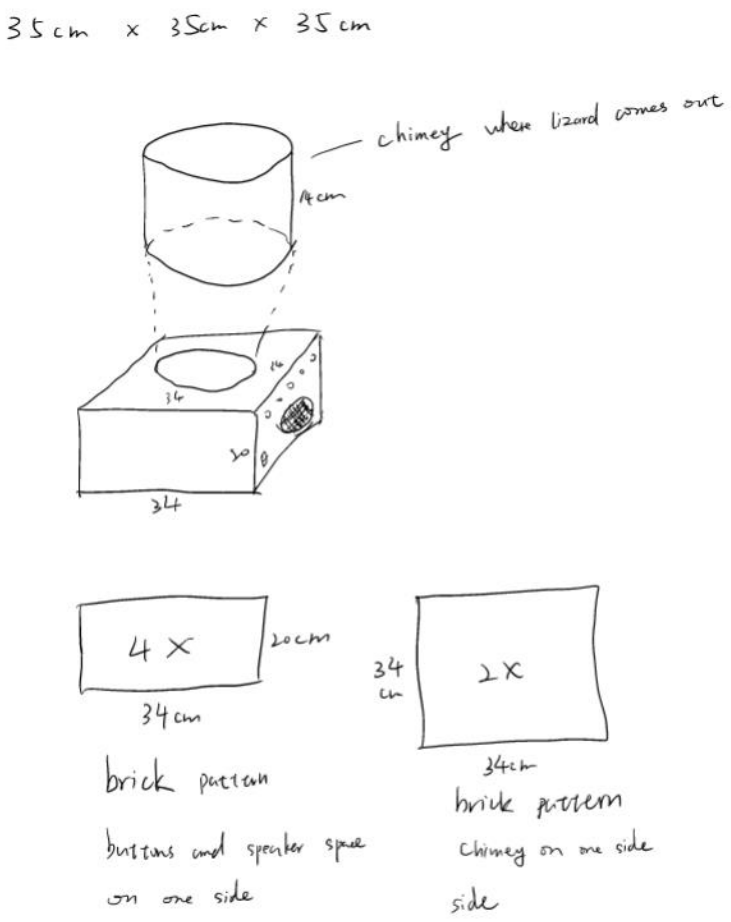


*Figure 1: Rough sketch of the exterior of the box in an orthogonal view and dimensions for laser cutting*

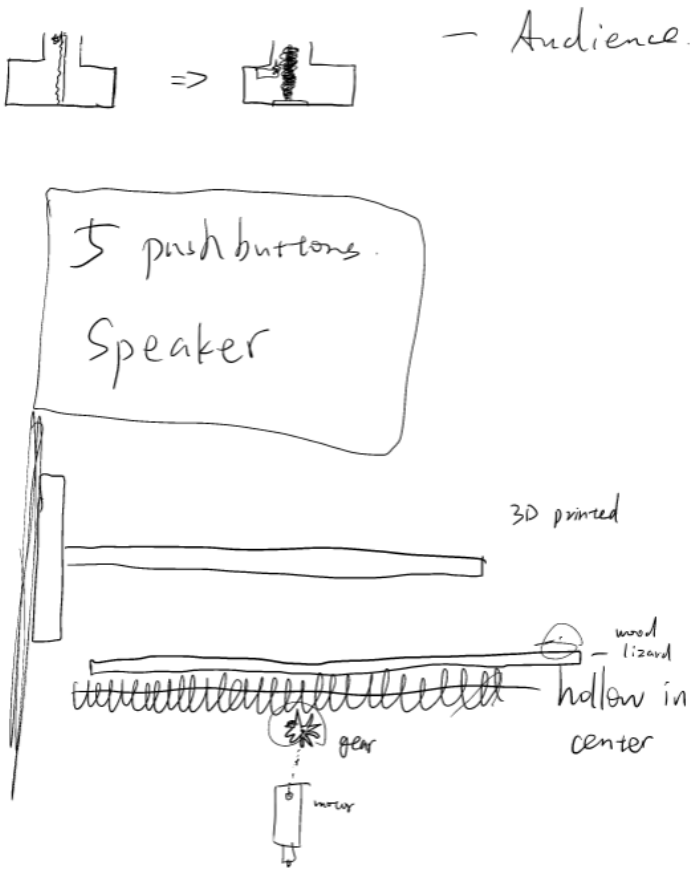


*Figure 18: Rough sketch of ideas to implement the interior mechanism for lifting Bill*

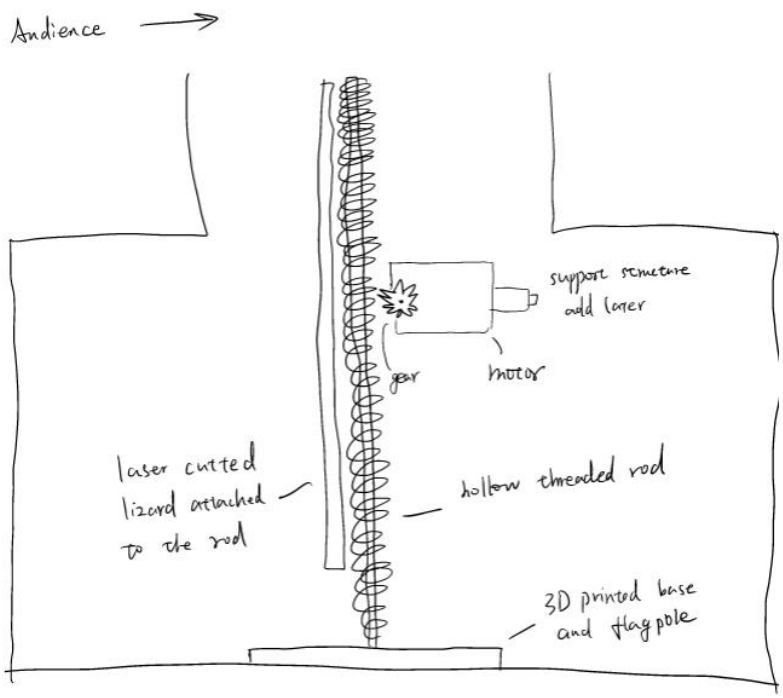


*Figure 19: Detailed sketch of interior moving structure*

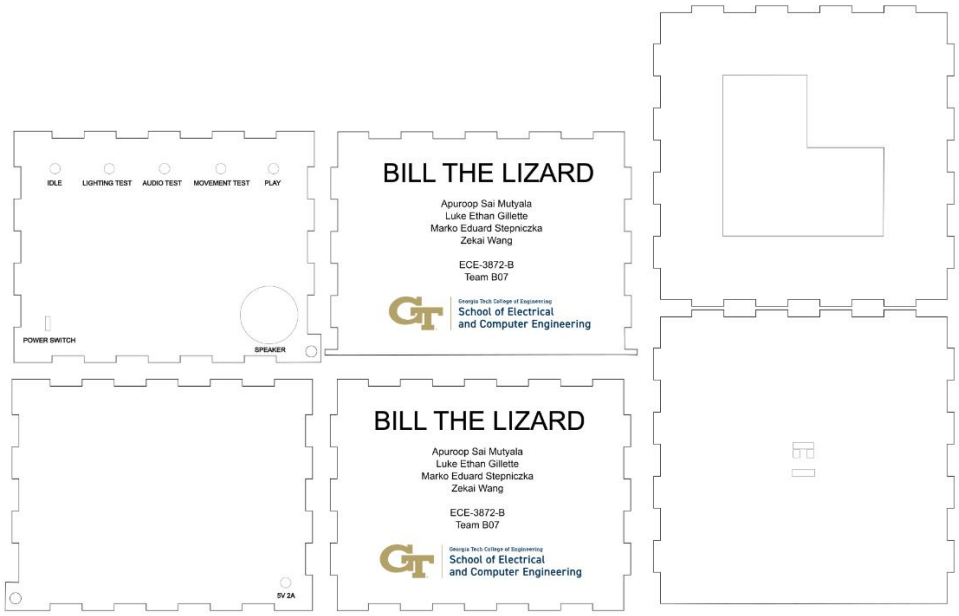


*Figure 20: Adobe Illustrator design for laser cut and engraved main portion of the box 25x25x19cm*

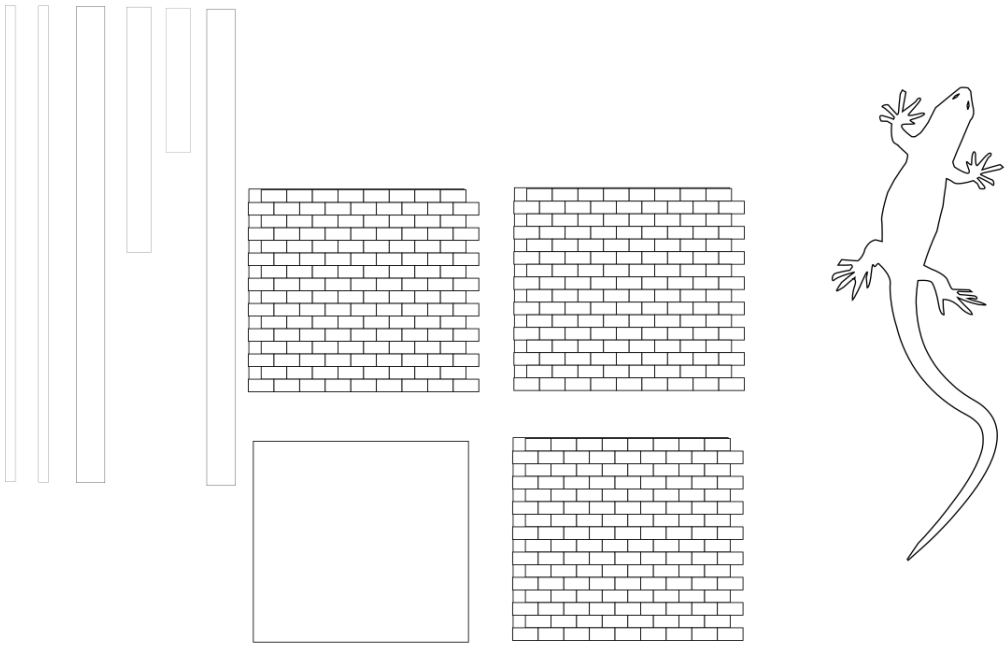

*Figure 21. Adobe Illustrator design for laser cut and engraved chimney portion of the box 15x15x14cm, internal support structure, and Bill the Lizard 12x33cm*

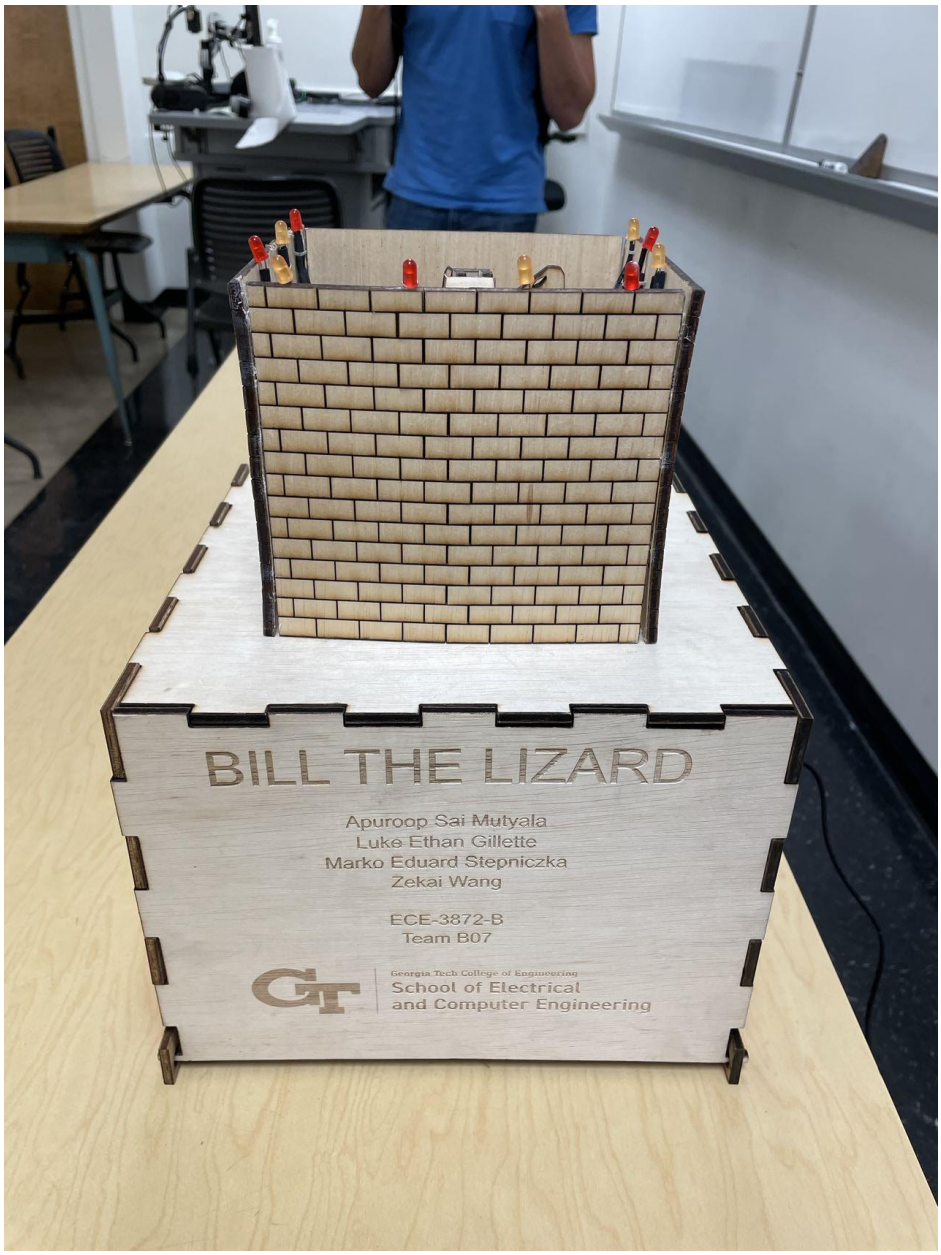


*Figure 22. Our assembled laser-cut box design*

One of the main features of the product is the vertical motion of Bill the Lizard. In order to accomplish this, we carefully designed a threaded rod and gear system to allow movement of laser-cut Bill the Lizard, we have also implemented a support system using laser-cut wood pieces around the moving rod to ensure the movement stay vertical. We have attached the motor to the lower size of the top of the main portion of the box. The Rod and gear were part of a complete motion set purchased from Amazon.

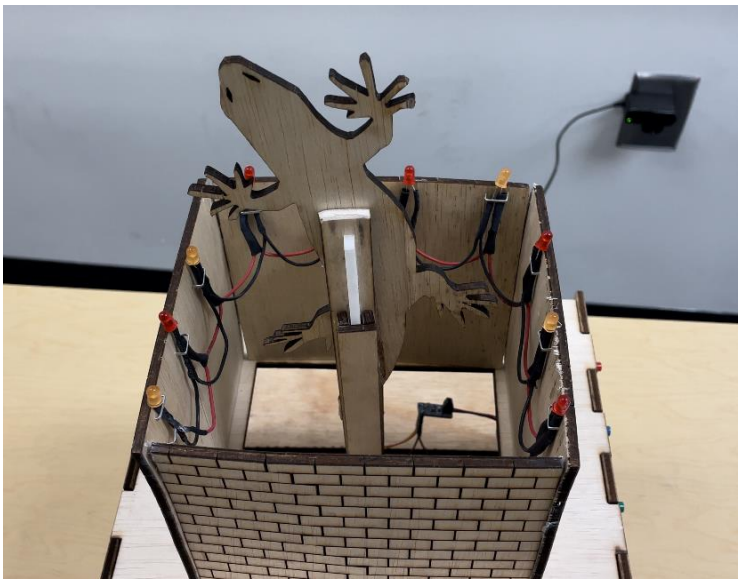

*Figure 23. Bill the Lizard is attached to the rod using superglue*

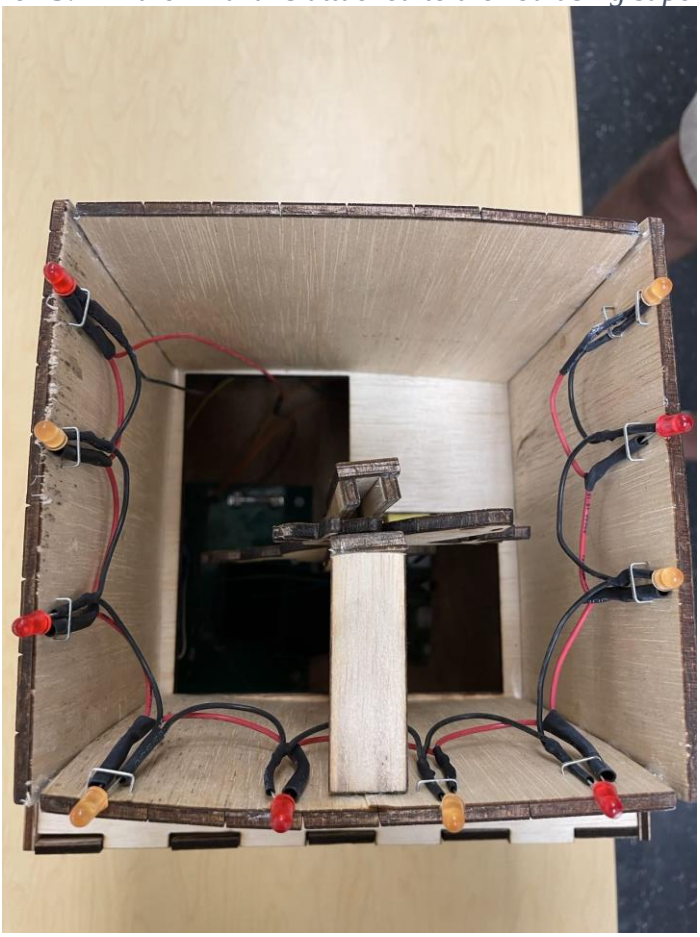

*Figure 24. Bird view of support structure that keeps motion vertical*

## Software Design

This section includes our software simulations and hierarchy.

Hierarchy:
Our software design includes 11 different states— idle, determine testing mode, repeatedly raise/lower Bill, loop default sound clip, blink on/off LEDS, testing end, connect to director, wait for line, decode file, play lines, mechanical movement, turn on LEDs, and production end, as shown in Figure 23. These states allow us to design software architecture to allow control of the Raspberry Pi to send signals to play sounds, read lines, turn LEDS on/off, and to raise and lower Bill.

The design features a host of different I/O objects from the gpiozero Raspberry Pi module to control and get input from our pushbuttons, motor, speaker, LEDs, and more. These I/O objects allow us to interact with our system and move it from state to state, and then see the desired outputs on our peripherals.

Our code is built from the following states:

- State 1 – Idle
- State 2 – Test Lighting
- State 3 – Test Audio
- State 4 – Test Movement
- State 5 – Production (Everything together)

Each of these states corresponds to different threads from the _thread Raspberry Pi module, with the idle state corresponding to the main thread, and the test states corresponding to their own new threads, and the production state corresponding to all the testing threads at once.

Our pushbuttons each are attached to their own interrupt routine, which lets them change the global state variable to transition between states on button presses. This allows us to externally control which state our system is in, and which behaviors it exhibits.

The hierarchy of our software design has now been described, including our state machine, multithreading, libraries, low-level interrupt routines, and expected behaviors.

Software Simulations:
We performed software simulations. Below is a description of our software simulations, where our software simulations are described showing that we did software simulations. The description of our software simulations is detailed in the next two paragraphs – it is not missing.

Here is the description of what we tested in our software simulations: we have performed various software simulations in order to ensure that our design functions as desired. First, we set up a state machine and a multi-threaded design in Python on our Raspberry Pi to handle all our software logic. These states allow us to progress through different behaviors as described in Figure 13. We tested moving through from state to state using pushbuttons, and each state would then decide which threads it needed to function correctly. In our simulations, the expected behavior was that we could click each pushbutton to move to the idle state, the motion testing

state, the lighting testing state, the sound testing state, or the play state which performs everything together. The outcome of these simulations was that we were able to verify that we could progress through our state machine and our threads worked together and separately.

Here are the results of our software simulations: the specifics of each result were also successful. For our motion testing state, not only did we see movement of the motor, but we were able to control it in both forward and reverse directions, which will let us raise and lower Bill after converting the rotational motion to linear motion with a gear and a threaded rod. For our lighting testing state, we were able to power all our LEDs with one control pin which decreases the need for extraneous wiring and is the behavior we want in our system. For the sound state, we were able to hear speech clearly from the audio clip we played, which means we should be able to hear the lines sent from the director clearly. The play state was able to run every functionality together at once.

The success of our simulations can be seen here.

Our connection to the director could not be established, which was the case for every other group as well and was out of our control.

Every part of our software design and simulations has now been detailed, including the hierarchy of our software, the goal of our simulations, and the results of our simulations.

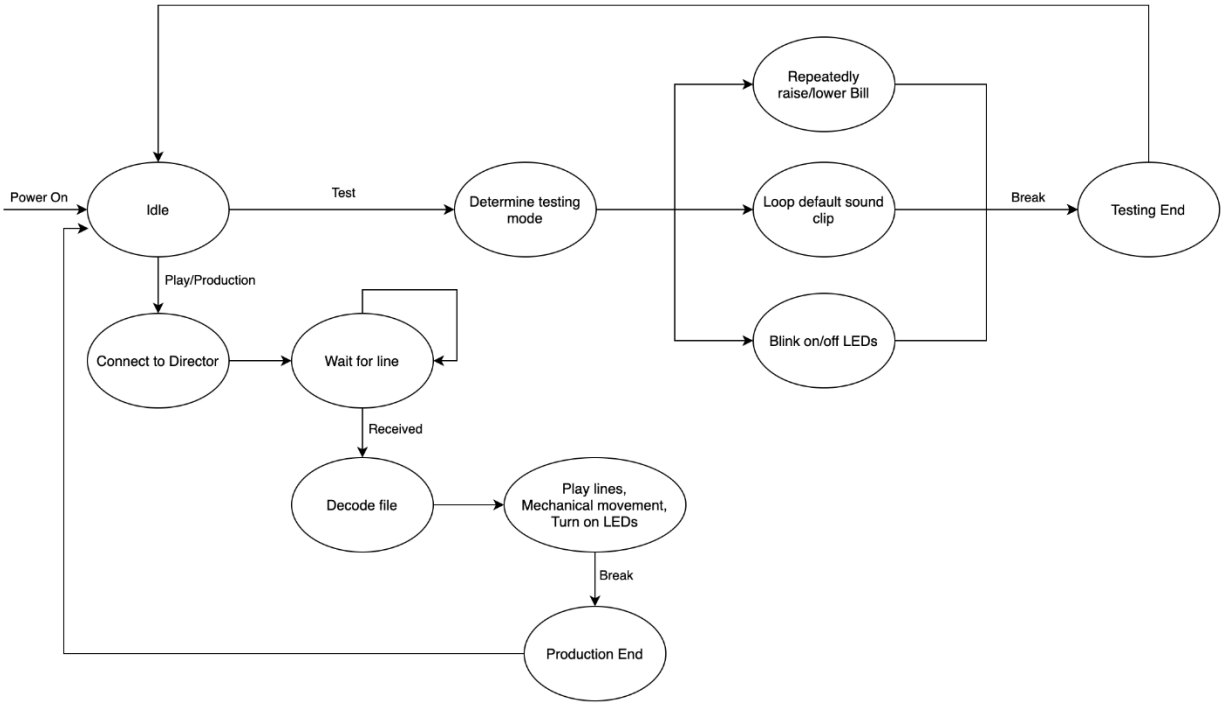


*Figure 25: Software state machine diagram with states: Idle, Determine testing mode, Repeatedly raise/lower Bill, Loop default sound clip, Blink on/off LEDS, Testing End, Connect to Director, Wait for line, Decode file, play lines, mechanical movement, turn on LEDs, and Production End.*

## Schedule

The schedule for completion of the project is shown in Table 2 where the task lead is indicated on each task.

| Name: Apuroop Mutyala, Zekai Wang, Luke Gillette, Marko Stepniczka | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Project: 3872 Project Schedule Fall 2022 | | | | | | | | | | | | |
| | Week Number | | | | | | | | | | | |
| **Task** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** |
| Brainstorm | Luke | | | | | | | | | | | |
| Design Top Level Block Diagram | Marko | | | | | | | | | | | |
| Design Software Block Diagram | Marko | | | | | | | | | | | |
| **Proposal** | | | | | | | | | | | | |
| **Software& Electrical** | | | | | | | | | | | | |
| ******Processing Subsystem****** | | | | | | | | | | | | |
| **Design** | | Luke | Zekai | Marko | Apuroop | Marko | Marko | | | | | |
| **Simulation** | | | | Marko | Apuroop | Marko | Marko | | | | | |
| **Build** | | | | | | | Marko | Marko | Marko | | | |
| **Test** | | | | | | | | Marko | Marko | Marko | | |
| ******Peripherals Subsystem****** | | | | | | | | | | | | |
| **Design** | | Apuroop | Luke | Zekai | Marko | Marko | Marko | | | | | |
| **Simulation** | | | | Zekai | Marko | Marko | Marko | | | | | |
| **Build** | | | | | | | Marko | Marko | Marko | | | |
| **Test** | | | | | | | | Marko | Marko | Marko | | |
| ******Sound Subsystem****** | | | | | | | | | | | | |
| **Design** | | Marko | Apuroop | Luke | Zekai | Marko | Apuroop | | | | | |
| **Simulation** | | | | Luke | Zekai | Marko | Apuroop | | | | | |
| **Build** | | | | | | | Marko | Apuroop | Luke | | | |
| **Test** | | | | | | | | Zekai | Marko | Apuroop | | |
| ******Power Subsystem****** | | | | | | | | | | | | |
| **Design** | | Zekai | Marko | Apuroop | Luke | Luke | Luke | | | | | |
| **Simulation** | | | | Apuroop | Luke | Luke | Luke | | | | | |
| **Build** | | | | | | | Luke | Luke | Luke | | | |
| **Test** | | | | | | | | Luke | Luke | Luke | | |
| ******User Interface Subsystem****** | | | | | | | | | | | | |
| **Design** | | Zekai | Marko | Apuroop | Luke | Apuroop | Apuroop | | | | | |
| **Simulation** | | | | Apuroop | Luke | Apuroop | Apuroop | | | | | |
| **Build** | | | | | | | Apuroop | Apuroop | Apuroop | | | |
| **Test** | | | | | | | | Apuroop | Apuroop | Apuroop | | |
| **Mechanical** | | | | | | | | | | | | |
| ******Motor and Rod****** | | | | | | | | | | | | |
| **Design** | | Luke | Marko | Luke | Marko | Luke | Marko | | | | | |
| **Simulation** | | | | Luke | Marko | Luke | Marko | | | | | |
| **Build** | | | | | | | Zekai | Zekai | Zekai | | | |
| **Test** | | | | | | | | Zekai | Zekai | Zekai | | |
| ******Enclosure****** | | | | | | | | | | | | |
| **Design** | | Apuroop | Zekai | Apuroop | Zekai | Zekai | Zekai | | | | | |
| **Simulation** | | | | Apuroop | Zekai | Zekai | Zekai | | | | | |
| **Build** | | | | | | | Zekai | Zekai | Zekai | | | |
| **Test** | | | | | | | | Zekai | Zekai | Zekai | | |
| **PDR** | | | | | | | | | | | | |
| **CDR** | | | | | | | | | | | | |
| **Processing Subsystem Integration** | | | | | | | | | Luke | Apuroop | | |
| **Peripherals Subsystem Integration** | | | | | | | | | Marko | Zekai | | |
| **Sound Subsystem Integration** | | | | | | | | | Luke | Zekai | | |
| **System Integration** | | | | | | | | | | | Apuroop | Marko |
| **System Test** | | | | | | | | | | | Luke | Zekai |
| **Final Inspection and Demonstration** | | | | | | | | | | | | |
| | | | | | | | | | | | | |
| **Milestones** | Luke | | | | | | | | | | | |
| Tasks | Apuroop | | | | | | | | | | | |
| | Marko | | | | | | | | | | | |
| | Zekai | | | | | | | | | | | |

*Table 2: Schedule to complete Bill the Chimney Sweep (August 24th- December 7th)*

## Integration

One of our requirements was that each subsystem will need to work independently and work together. In order to make sure they work together properly; we will implement a series of tests. Once we simulate the software first, we can then test each subsystem individually. Once each sub-system works individually, we will then integrate them. By taking this approach we can test smaller aspects of the design rather than trying to do it after integration, which will be a much bigger problem. Once the subsystems are verified, we will then add the mechanical aspects such as the LEDs and gear/motor.

Each system was tested separately at first, including the audio, movement, and lighting systems, and each subsystem was tested separately as well. Audio, movement, and lighting were tested separately using separate threads in our software design. After they all worked separately, we tested them all together by running each thread at the same time. Once they all worked together, our system's functionality was fully integrated.

## Conclusion

This semester, our team designed and built Bill the Chimney Sweep. The project was broken down into subsystem designs, and a schedule was made with subsystem leads for each week to make sure we were on track. This enabled us to meet our goals to complete the project.

We were able to stick to our schedule throughout our project, which enabled us to reach our end goal of a fully functional robot, subject to limitations by the director. The design process was followed by our members, and it helped us work together as a team and produce quality work. While we created a successful end-product, there are areas for improvement, such as getting the director to work, and adding more designs and color to our box for visual effects.

We learned how to build a project's hierarchy and test plan before building the actual product, which was valuable in its own right. That being said, we are proud of what we were able to produce as well.