# Automated Speakers to Enhance Plant Growth by Playing Classical Music

## I. INTRODUCTION

Studies have shown that exposing plants to classical music at sunrise can encourage plant growth[1][2]. Despite the compelling evidence that supports this research, our team found little to no solutions on the market aimed towards either personal gardens or greenhouses. In this project, we aim to address this gap by creating a prototype widget that will automatically play classical music at sunrise and can be integrated into an existing garden as effortlessly as possible.

## II. CONCEPTUAL DESIGN PROCESS

We framed our opportunity as designing a system that plays music to the plant at dawn, as this is proven to facilitate the growth of the plant. Specifically, sunrise will be detected by the photoresistor, and the music will be played by the buzzer. If the music is accidentally played, it could be turned off by the button manually. This button acts only as a failsafe, and we approached our design with the goal of reducing the likelihood of music being accidentally played.

Our system consists of a buzzer, button, and photoresistor. To achieve our goal of easy assembling, we placed the buzzer and the photoresistor on different sides of the Pico so the wires would be easily installed. For the cover of the plate, as we valued usability and the long life of the device, we designed a chimney for the photoresistor to protect it from breaking and left a large enough opening for the button that fitted human fingers. The schematic of the circuitry is shown in figure 1.

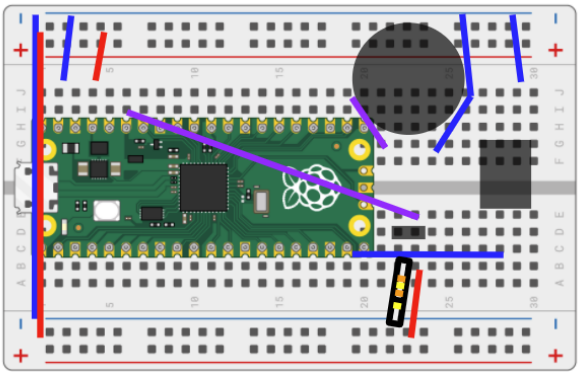

**FIG. 1:** The circuit diagram we modelled our circuit off of. The blue wires show connections to ground, red show connections to power, and purple show connections between individual components. The three gray translucent components are (from top to bottom): buzzer, button, photoresistor. A $1K$ resistor was used.

### A. Control System

To prevent false positives and false negatives, which will in turn reduce the need for human intervention, we needed a method to ignore momentary periods of darkness and daylight, and not register those as an additional night-day cycle.

We used two methods to accomplish this. First, to prevent rapid fluctuations near the transition from nighttime and daytime, we implemented a high and a low threshold. The light level is set to high if it goes above a certain threshold and is set to low if it goes below another, different threshold. If the light level is in between, it will take on the value of what it was previously set to.

Secondly, it will only register a change in daytime and nighttime if 30 minutes have passed continuously at the new light level. By doing so, briefly covering the sensor with a hand and uncovering it (such as by accident), will

not make the system believe there is another sunrise. This is illustrated below, where each "tick" represents 15 minutes.

## III. IMPLEMENTATION

### A. CAD Model

The structural component was adapted from our existing work with greenhouse widgets, specifically a multimodal lighting system. Instead of having two holes in the case for LEDs, we had a large hole for the piezo buzzer, and a chimney-like structure for the tall photoresistor. As we value efficiency, we used the straightforward method of using screws to fasten the lid to the main body as shown in the CAD figure below. One qualitative observation from previous work was that the hole for the button may be a bit too small and not be a comfortable fit for all fingers. To demonstrate greater empathy towards the stakeholders, we have tried our best to make the spacing around the button as big as possible to reduce awkwardness.

### B. Asynchronous Functions

The program was based on the UML diagram shown in the diagram below. Two sensors: a button and a photoresistor are constantly reading data (in the form of the light level and if the button is pressed or not), and constantly communicating this information to a Controller class, which adjusts global variables.

There is another function that reads these global variables and plays music depending on the values of these variables and how these variables change. All three tasks need to be performed concurrently, so we needed to incorporate asynchronous programming. Two common ways of handling asynchrony are via threading or interrupts, but neither is supported by *CircuitPython*[3]. Fortunately, *CircuitPython* recently introduced a package called *asyncio* that can perform the same tasks as other methods, which is what we used at the end[4]. The pseudocode is shown below, which describes how the button and photoresistor values change the relevant global variables, and how the music player uses these global variables. Note that while the button acts as a power off switch, what it does in practice is change the volume to zero. Once the next song plays (i.e., the next day), the volume is set to its default value.

### C. Playing Music

Another major feature of our system is a simple and straightforward implementation of encoding music and outputting it through the piezo buzzer. Using open-source code written by Charles Grassin and Stuart Memo, we were able to convert musical notes written in scientific pitch notation (where C4 represents middle C) to numerical frequencies[5]. We then modified the tone function from the *CircuitPython* package *simpleio*, to play a specified note for a specified duration at a specified loudness[6].

## IV. EVALUATION

### A. Verification

Due to the limited time span to work on this project, it is impractical to test this system with real inputs, since sunrise only happens once a day. Instead, we simulated daytime and nighttime by placing our hands over the photoresistor and reducing the time delay from 30 minutes to 5 seconds. We then calibrated the low and the high threshold from the photoresistor value when it is covered and uncovered.

By nature, it is difficult to test a combination of asynchronous functions. This is because not only does the order of the inputs

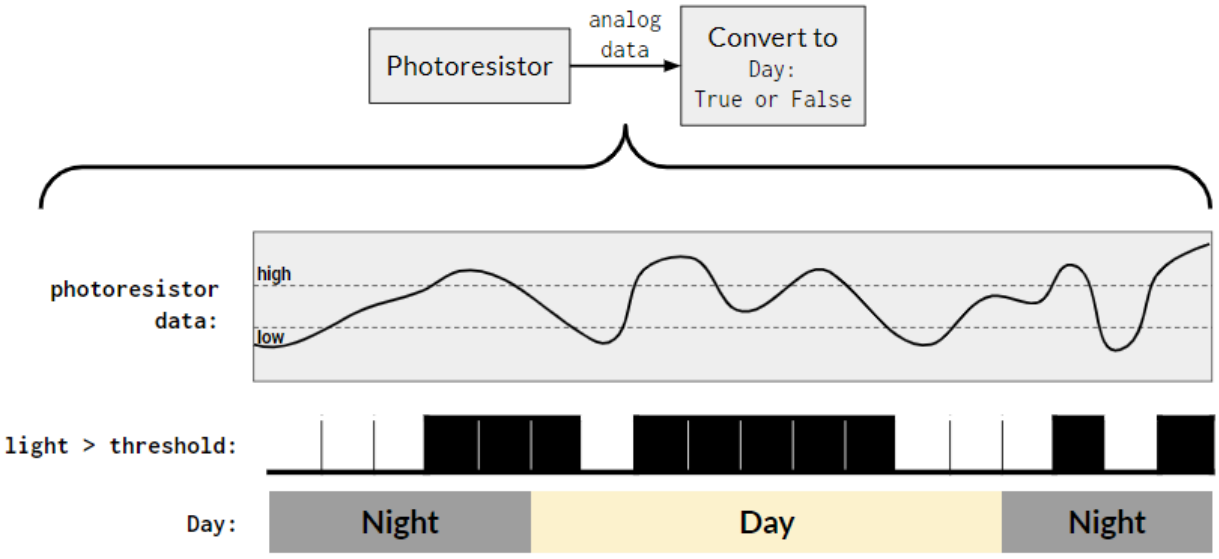


**FIG. 2:** Diagram showing how the day is distinguished from night from the photoresistor data. First, the light level is either set to high or low depending on two thresholds, and after two ticks (30 minutes) on high light, the time is set to day. After two ticks on low light, the time is set to night. If there are fluctuations in between, the system ignores it.

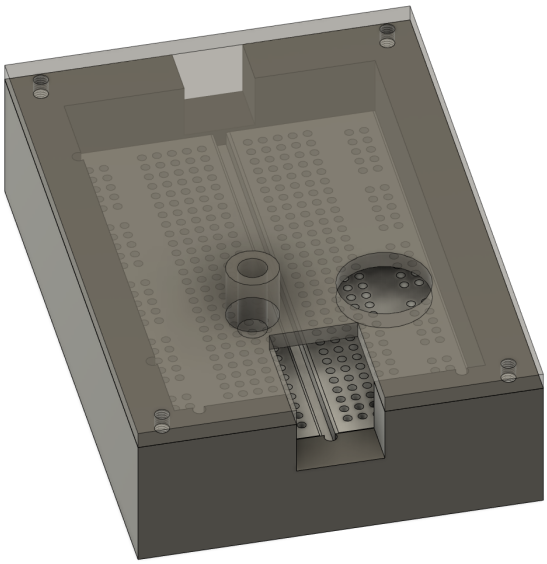

**FIG. 3:** CAD model of the widget case, where the breadboard is visible and the circuit components are not visible. The lid is transparent for visibility. The failsafe button goes in the center of the rectangular opening near the bottom.

matter, but the specific timings also matter. Therefore, we cannot test every single possibility unlike our previous projects. Instead, we tested the most valuable features that we expect to see:

- Under ideal situations, the system works as expected and the song only plays during sunrise.
- Whenever the buzzer plays a noise, pressing the button will always suppress it.

We also tried to test for edge cases, such as trying to time the button press and the music playing at the same time, and more. Fortunately, in all the edge cases we tried, the system works exactly as behaved.

The case also meets the technical requirements set for this project. Namely, placing it in different orientations and shaking it around does not affect the circuitry. We carried the system inside our bag to and from university for several days, and never had an

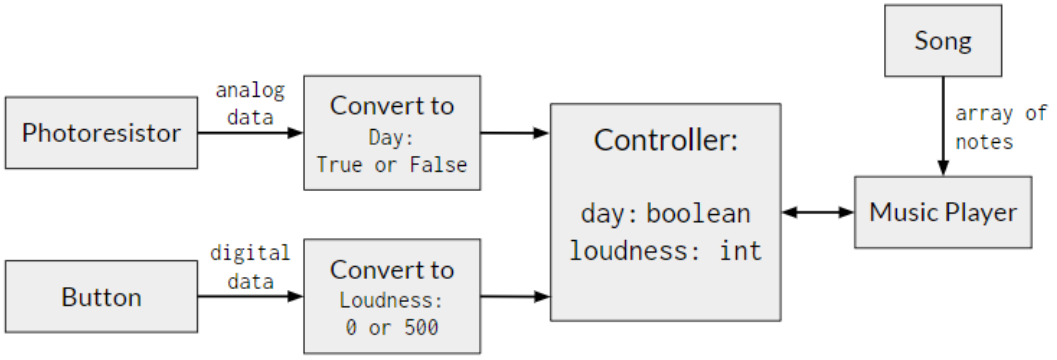


**FIG. 4:** A UML diagram of how various components interact with each other. Both the photoresistor and button is constantly reading data and updating the Controller class when necessary. These tasks still occur even when the music is being played.

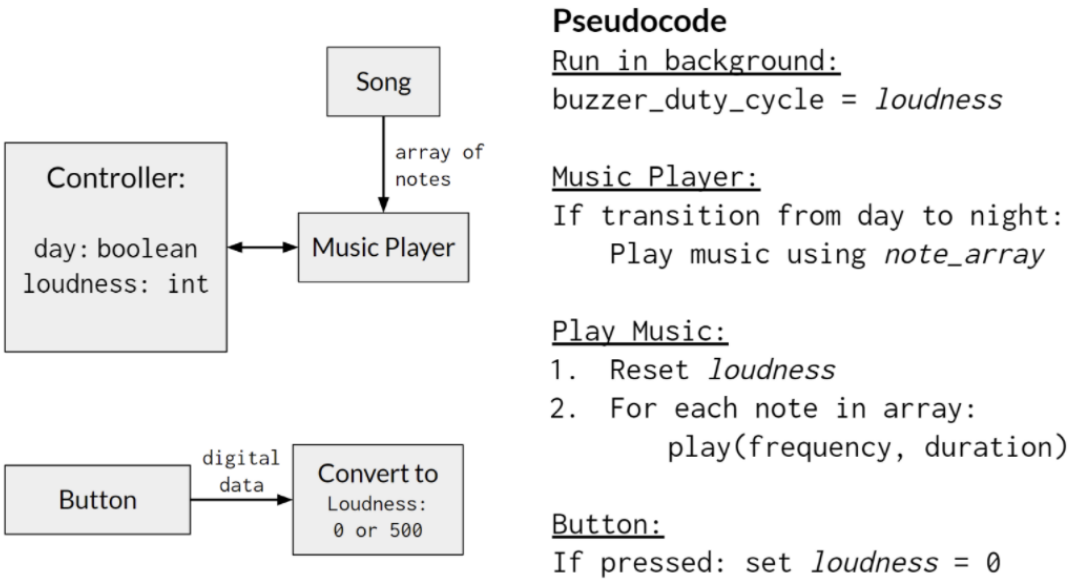


**FIG. 5:** Pseudocode that shows how data is being processed in the button and the conditions in which the music should be played. Refer to figure 2 for how the photoresistor processes data.

issue with parts falling out.

### B. Technical Issues

There were several technical issues that we encountered through the project, though we were able to address all of them. One issue we came across was that a naïve function that plays music by systematically changing the buzzer frequency and waiting a set amount of time cannot be interrupted by other functions. It will need to finish the song before any other part of the code can run. While other microcontroller compilers such as *Micropython* have built-in interrupt handlers, the *CircuitPython* compiler, which we are required to use, does not[7].

To resolve our problem, we consulted with

other people in the Adafruit CircuitPython discord, where we were given advice of how to proceed with the functionalities of *CircuitPython*. This was an extremely challenging part of the project, as none of our team members had worked with asynchronous programming before, so it was a new experience for all of us.

Another issue we encountered was that the specific buzzer was only able to play monophonic tones. That is, only a single frequency can be played at the same time and therefore limiting us to simple songs, such as the famous melody from Beethoven's *Fur Elise*. While this problem can be solved by introducing other buzzers, there are certain tricks one can use to play polyphonic tones on a single buzzer by carefully adjusting the duty cycle. However, the degree of control these techniques need is not available for us. If this project is continued, we would like to invest in a more powerful buzzer, or perhaps even a speaker.

## V. INDIVIDUAL CONTRIBUTIONS

Everyone had an equal contribution in the conceptual design phase. Chris, Julia, and I worked on the CAD model together while Hamshi and Annoah worked on planning out and building the circuit. We also worked on the initial programming part together during studio and worked out and implemented the logic shown in figure 2. Julia and a few others attempted to work on the interrupt capability of the button, but found that several packages were incompatible with *CircuitPython*. I picking up from where Julia ended by asking for advice through the Adafruit discord and was able to finish the programming part. I also made all the diagrams shown in this document.

## VI. REFLECTION

The biggest thing that I learned and was new to me was asynchronous programming. This was always an area that fascinated me, and is closely related to my first exposure to programming via *Scratch* in grade 6. I was able to have several while loops running concurrently and communicate between them by sending "messages," allowing me to make many games. When I eventually moved to more mature languages such as Javascript, Java, and Python, I was very surprised that I was no longer able to run several pieces of code at the same time and communicate between them. Therefore, learning asynchronous programming was always a goal of mine, and this widget lab finally gave me an excuse to learn it. While it sounds very dull, it was truly an extremely exciting experience for me!

Something that really helped my learning was talking to other people and asking people for their opinions and advice. This was true for learning asynchronous programming, which I have talked about several times in this paper already, but also for other parts. I originally thought that implementing a system that wouldn't be sensitive to brief fluctuations of light levels would be extremely difficult, but after talking it over with my group, we found another way to look at the problem that made it almost obvious! Similarly, I consulted my peers outside the group for thoughts on the audio capabilities of the buzzer. Both peers I consulted had experience working with audio but they disagreed about the capabilities of the buzzer. One thought that it was impossible to play polyphonic tones while the other thought that with some clever tricks, it was possible to play polyphonic tones on anything. Through these discussions, I learned a lot about audio, electronics, and the fundamental physics of how speakers worked!

The audio part of this project was related to

a past project I did last semester, where my team and I tackled the problem of polyphonic piano transcription (audio to MIDI file) via machine learning, specifically via the use of Transformers. One of the major parts of that project was working with different representations of music, and that experience helped a lot when I was making an intuitive interface for processing music. In the future, I would like to make the feature even more intuitive by taking in a MIDI file as an input. The main reason I didn't do that was because MIDI files are often polyphonic and in order to generate monophonic sound, I would need to pick out the most important notes. There is no simple algorithm to do so, which is why I didn't attempt it this time.

In ECE253, we also learned about interrupts. Initially I was very excited to apply my past knowledge about interrupts to this project. Unfortunately, *CircuitPython* does not support interrupts, and I was forced to find another way. Regardless, I found it very interesting how there can be several solutions to the same problems and how different developers can all think that their approach is the more superior.

---


[1] V. Chivukula and S. Ramaswamy, "Effect of different types of music on rosa chinensis plants," *International Journal of Environmental Science and Development*, vol. 5, pp. 431–434, Oct. 2014.

[2] K. Creath and G. E. Schwartz, "Measuring effects of music, noise, and healing energy using a seed germination bioassay," *The Journal of Alternative and Complementary Medicine*, vol. 10, pp. 113–122, Feb. 2004.

[3] K. Rembor, "Circuitpython: Frequently asked questions." AdaFruit, Mar 2022.

[4] D. Halbert, "Cooperative multitasking in circuitpython with asyncio." AdaFruit, 2022.

[5] S. Memo and C. Grassin, "Note to frequency." GitHub, 2020.

[6] S. Shawcroft, "Simpleio - simpler, beginner friendly io." Read the Docs, 2017.

[7] P. Sokolovsky, "Micropython: Writing interrupt handlers." Read the Docs, Mar 2022.