Electrical & Computer ENGINEERING
Carnegie Mellon

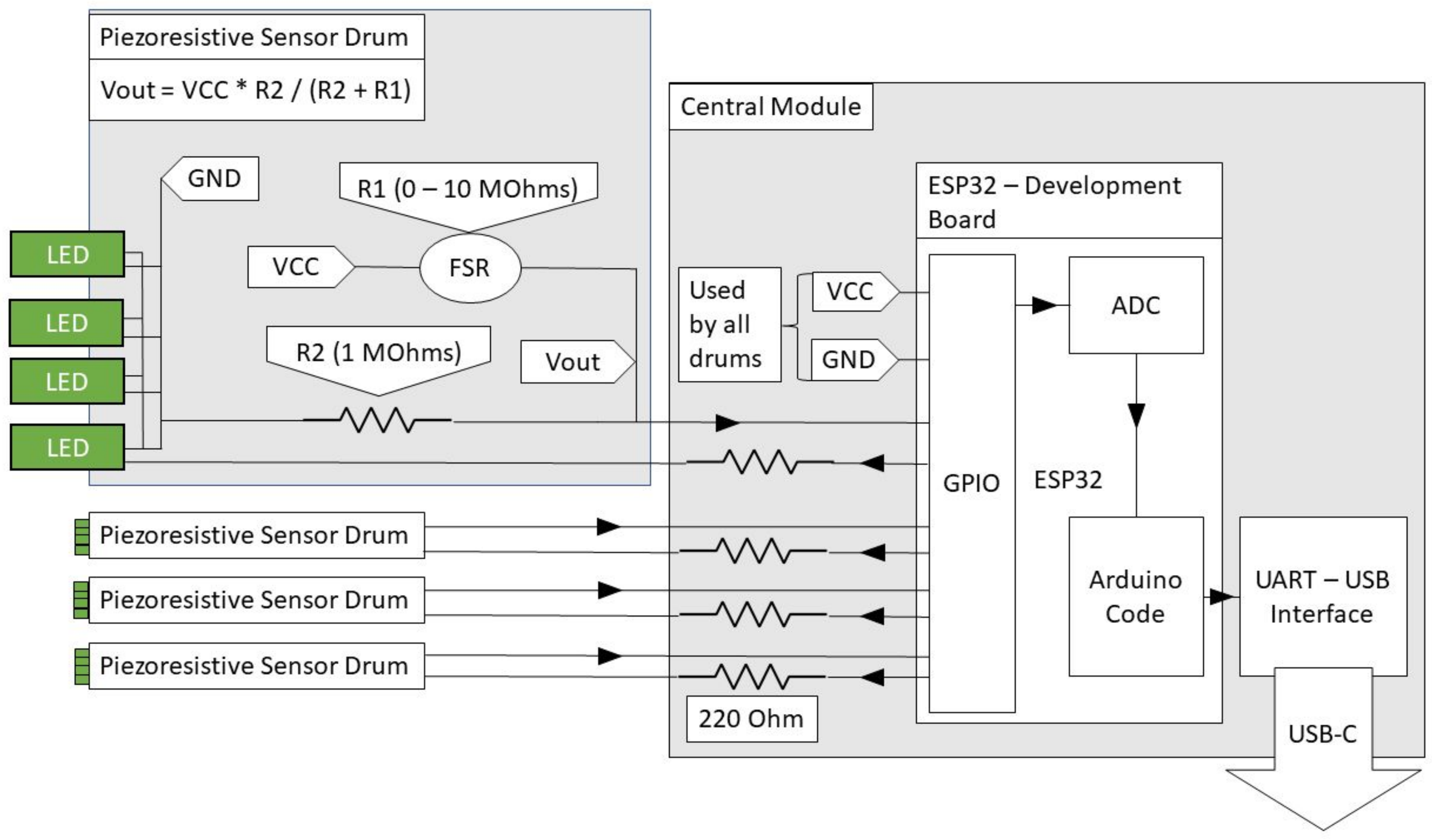
# Hit It!

**A5: Stephen Pupa, Shreya Ramesh, George Whitfield**
18-500 Capstone Design, Spring 2022
Electrical and Computer Engineering Department
Carnegie Mellon University

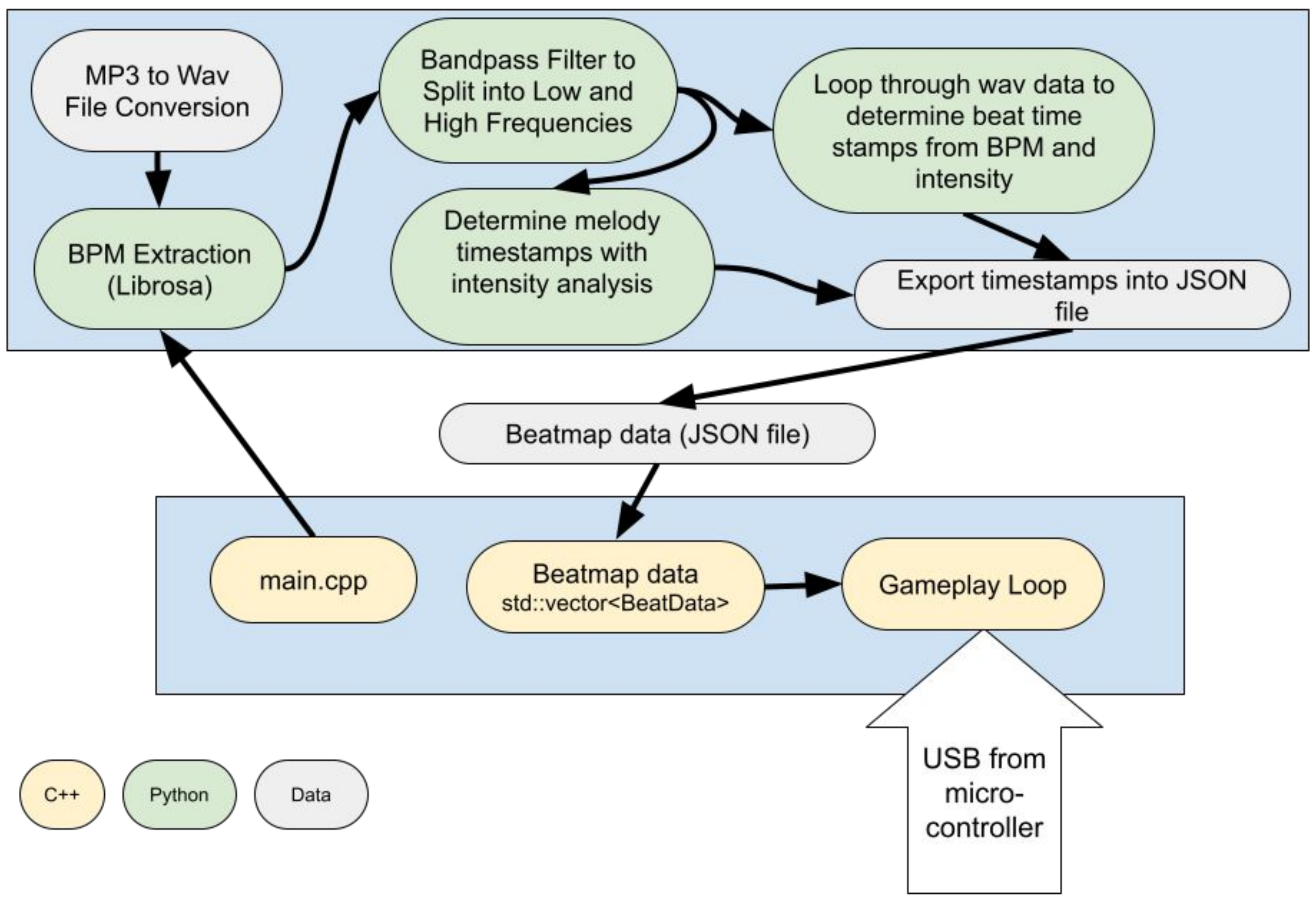
## Product Pitch

Hit It! is a rhythm game where the user hits drums to the beat of music. Hit It! has an **engaging interface**; users can interact with our colorful and custom designed interface by hitting four drum modules that correspond to buttons within the game. The drum modules were carefully designed to be **small and portable** so that the player can easily set up the game anywhere. In addition, the player can customize their experience by **importing their own music**. The game uses signal processing algorithms to automatically generate a sequence of notes called a “beatmap” that the player must hit to the beat of the music. The ECE areas used are hardware systems, software systems, and signal processing.

## System Architecture


Piezoresistive Sensor Drum
Vout = VCC * R2 / (R2 + R1)
GND
R1 (0 – 10 MOhms)
VCC
FSR
R2 (1 MOhms)
Vout
LED
LED
LED
LED
Piezoresistive Sensor Drum
Piezoresistive Sensor Drum
Piezoresistive Sensor Drum
Central Module
ESP32 – Development Board
Used by all drums
VCC
GND
ADC
GPIO
ESP32
Arduino Code
UART – USB Interface
220 Ohm
USB-C

This block diagram depicts the hardware implementation of our system, consisting of 4 drum modules and a central connecting/controlling module. Each drum contains an FSR-based voltage divider and feedback LEDs. The voltage divider output is read by the ESP32, which transmits that information to the PC and powers the LEDs.


MP3 to Wav File Conversion
BPM Extraction (Librosa)
Bandpass Filter to Split into Low and High Frequencies
Determine melody timestamps with intensity analysis
Loop through wav data to determine beat time stamps from BPM and intensity
Export timestamps into JSON file
Beatmap data (JSON file)
main.cpp
Beatmap data std::vector<BeatData>
Gameplay Loop
USB from micro-controller
C++
Python
Data

This above diagram shows the flow of execution of our software. Game play code begins when the user starts the game in main.cpp. This will then call the beat map generation Python code which filters the user-inputted audio file, runs an intensity analysis, and then outputs a JSON for the game play code to read. The beatmap is outputted onto the screen from our ‘draw’ function, which is called every time the game draws to the graphics window.

## Conclusions & Additional Information

QR code to visit our blog!

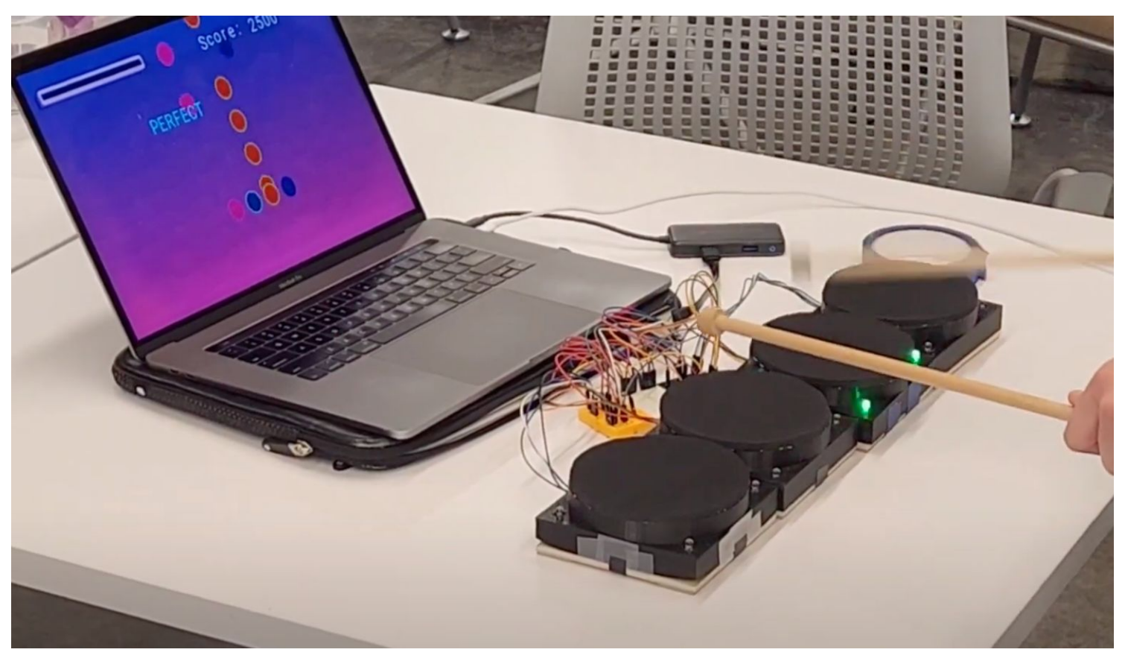
Overall, we are happy with the results from this project. However, due to various time constraints, we had to create a less involved game than we originally expected. We had to compromise on some of the aesthetics as well as the beat map cohesion with the song. If we had more time, we would consider expanding on the functionality of the game in regards to audio files inputted or even additional characteristics of the game.

## System Description

### Software

Modern video games are typically created with video game engines, which create layers of abstraction in the tech stack that make video game development easier for the programmer. To enhance the educational value of this project, our team chose to create Hit It! without a game engine, because we wanted to learn more about the technical details of game development such as graphics programming, audio signal processing, and cross platform compilation.

C++14 - Gameplay and graphics programming
- Compiled with CMake
- Open Graphics Library (OpenGL) used for graphics display
- PortAudio library used for realtime audio playback



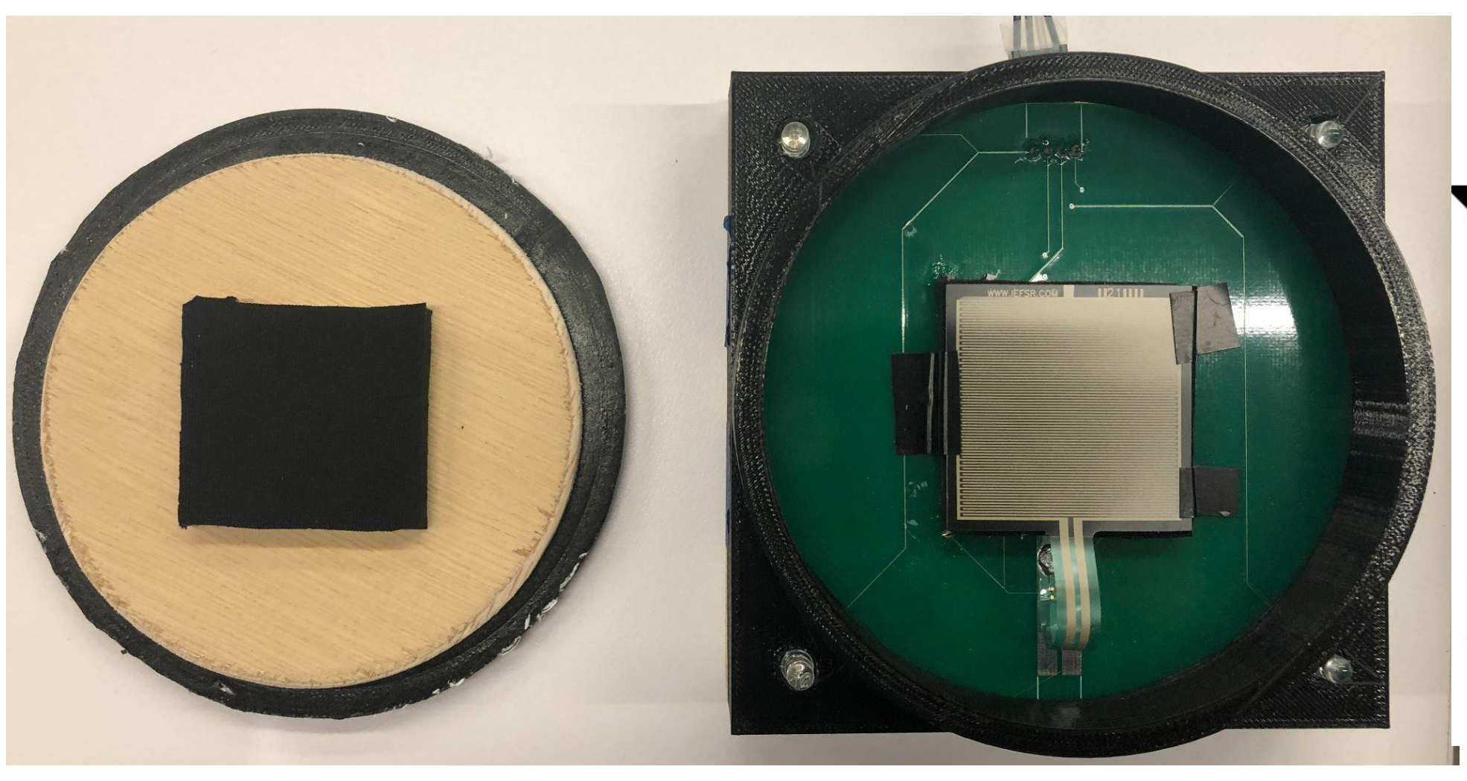
View of final system

Python 3 - Signal processing for beatmap generation
- Numpy and Scipy for the wav file manipulation
- Librosa to extract the BPM of the given audio file
- Scipy bandpass filters to differentiate between buttons

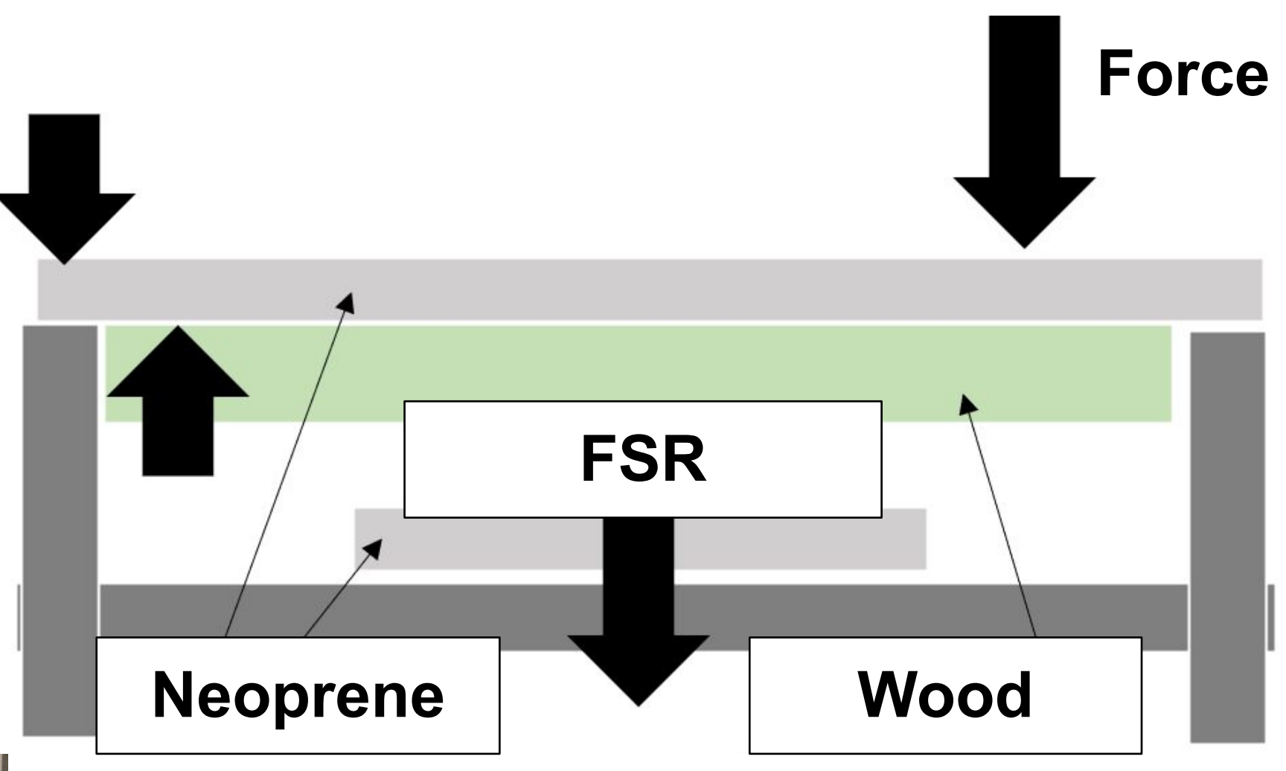
### Hardware

The drum sensors are created using Force Sensitive Resistors (FSRs), which change their resistance by several orders of magnitude when force is applied. The ESP32 microcontroller then reads this change through a voltage divider and respond accordingly by powering that drum’s LEDs and transmitting that information to the PC.

A custom drum housing was also designed in Fusion 360 and 3D printed to provide a compact and robust housing for the drum’s internal PCB. Forces applied to the drum are redirected into the FSR using sheets of Neoprene rubber and wood to form a central supporting column, which also provides the drum pad elasticity to achieve a satisfying bounce when struck.



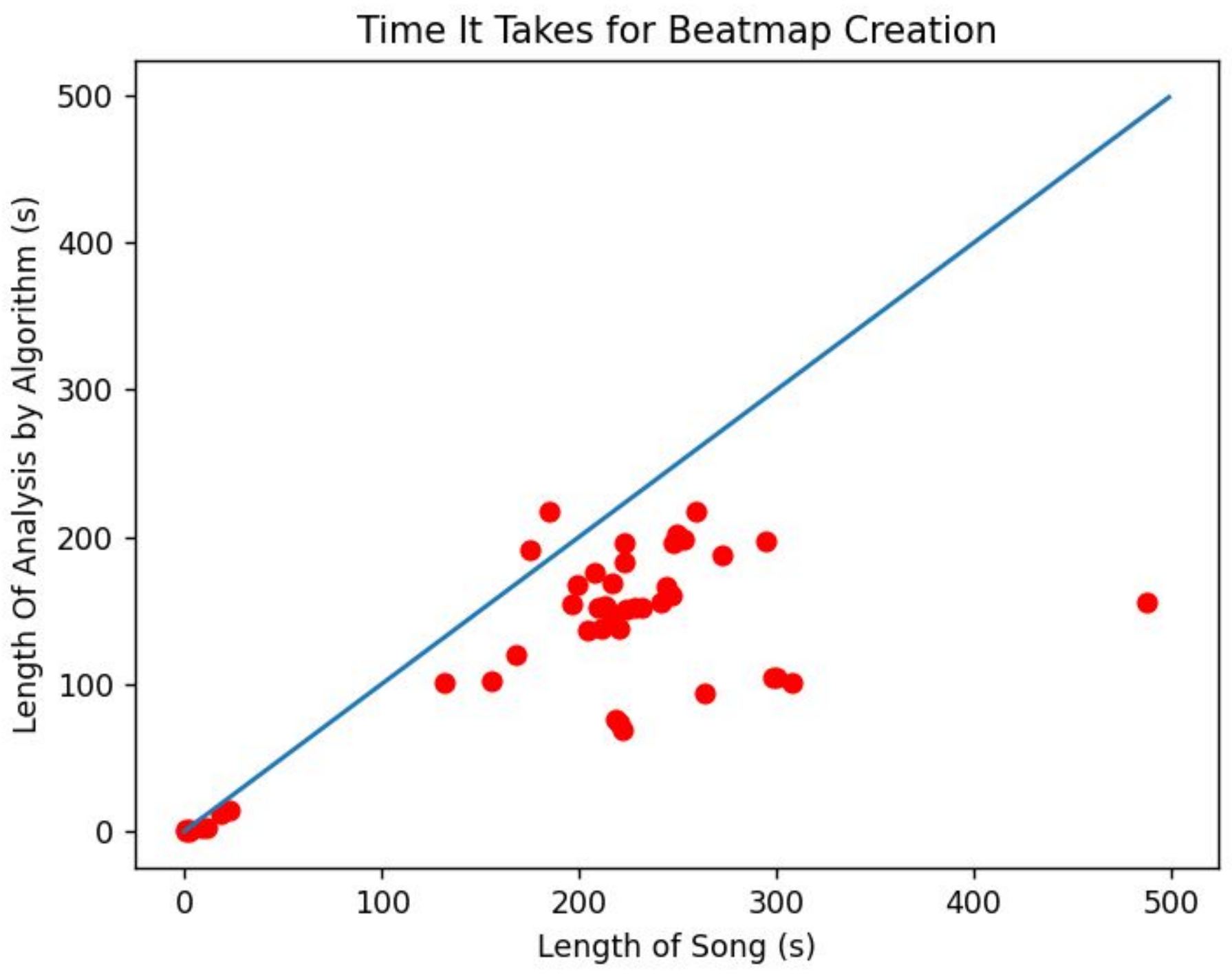
View of drum module internals. The FSR and LEDs are fixed to a PCB which routes relevant signals out through a latch connector.



Force Distribution inside module. The FSR is propped upon a central pillar so that most force is redirected through it.

## System Evaluation

| Area | Goal | Method of Testing | Results |
|---|---|---|---|
| Drum Recognition Accuracy | >99% Recognition Rate | Each drum was hit 250 times in the center, 250 times in the center using gravity, and 250 times on the edge | 100% when centered<br>96% on edge |
| Latency | < 37 ms | An IPhone’s slow-mo video feature was used to record the difference in frames between the drum hit and receipt of signal. | ~37.5 ms +- 5 ms |
| Compactness | < 15,000 cm^3 | Each module was measured at 720 cm^3, or 3600 cm^3 with 5 modules | 3,600 cm^3 or 3.6 Liters |
| Ease of Setup | < 1 minute Setup Time | 10 random participants were asked to setup the game given a brief explanation. | Average of 42.57 seconds |
| Length of Beat Map Generation | Length of Audio File | Use ‘time’ module in Python to measure execution time | 96% success rate |
| Accuracy of Beat Map Generation | 80% Accuracy | Compared the number of points created in an average beat map to the expected number of points from the Librosa Module | 67% Accuracy Rate |



50 beat maps were created and the times it took to create them were plotted. The blue line represents the user requirement, stating that we had a 96% success rate