# Bluetooth Speaker

Lehi Alcantara & Clayton Ramstedt

# Overview

- Introduction/Motivation
- Demo
- Related Work
- Methodology & Schematic
- Hardware Implementation
- Software Implementation
- Protocol Implementation
- Results
- Conclusion

# Introduction/Motivation

- Why a Bluetooth speaker?
  - Because listening to music over copper wires is lame!
- This is a solved problem, but we were curious to know more about how Bluetooth works.
- Turns out the project was more complicated than we anticipated, and we ended up implementing our own custom protocol.

# Demo

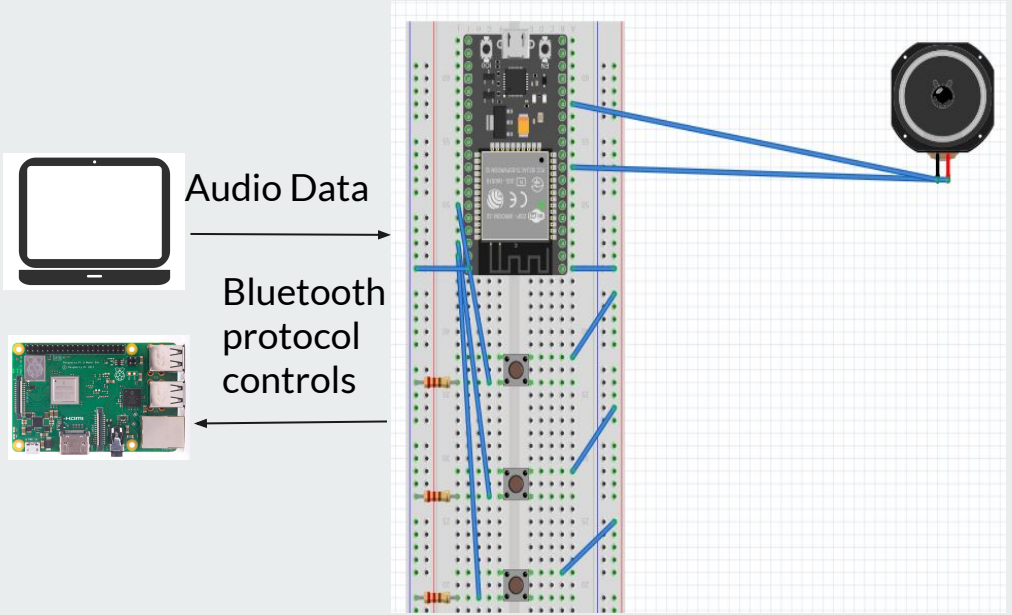
# Related Work

- Many tutorials online covering bits and pieces on how to work with bluetooth on the ESP32 (simple to hard)

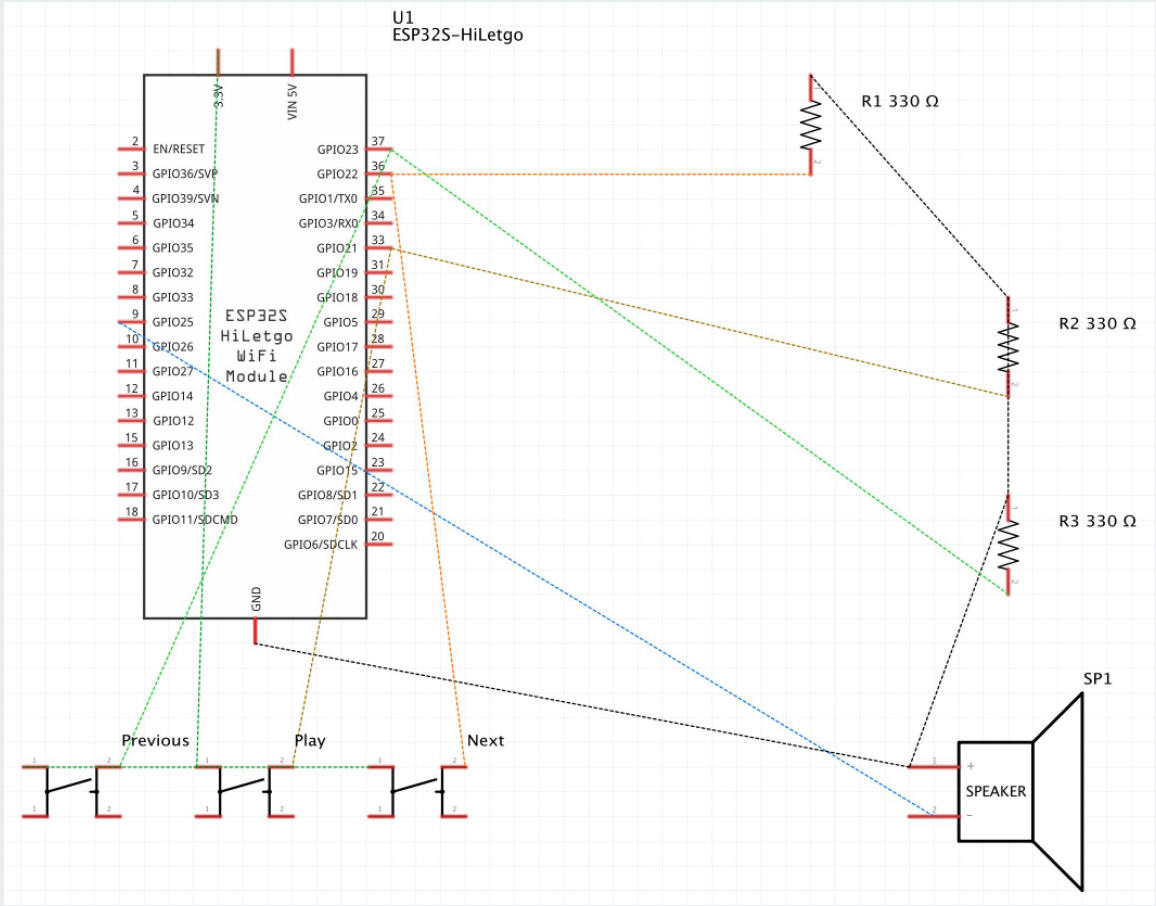
- Sophisticated Github project using ESP32 implementing A2DP and AVRCP profiles that are part of the bluetooth specification (It makes use of a real-time operating system and requires deep knowledge of bluetooth stack)
- Our intent was to build a bluetooth speaker and implement our own custom bluetooth protocol

# Methodology

- Implemented USB serial speaker communication first then bluetooth
- Send 512 bytes of Audio data via python using bluetooth socket (512 was the ESP32 buffer size)
- On Arduino side listen on bluetooth serial for data
- Play sound data until 256 bytes then request for more data to fill-in the buffer
- Listen for buttons interaction and act accordingly on the python script side



# Schematic


U1
ESP32S-HiLetgo
3.3V
VIN 5V
GND
EN/RESET
GPIO36/SVP
GPIO39/SVN
GPIO34
GPIO35
GPIO32
GPIO33
GPIO25
GPIO26
GPIO27
GPIO14
GPIO12
GPIO13
GPIO9/SD2
GPIO10/SD3
GPIO11/SDCMD
GPIO23
GPIO22
GPIO1/TX0
GPIO3/RX0
GPIO21
GPIO19
GPIO18
GPIO5
GPIO17
GPIO16
GPIO4
GPIO0
GPIO2
GPIO15
GPIO8/SD1
GPIO7/SD0
GPIO6/SDCLK
ESP32S
HiLetgo
WiFi
Module
R1 330 Ω
R2 330 Ω
R3 330 Ω
Previous
Play
Next
SP1
SPEAKER

# Hardware Implementation

- ESP32 microcontroller
- Speaker
- Resistors 330 ohms
- Buttons
- Jump wires

# Software Implementation

- Arduino IDE to flash the code to ESP32
- Pybluez Python library
  - Only worked well in Unix based systems.

# Protocol Implementation

- **Play**
  - If the ESP32 is going from a state of having no data in its buffer to having data in its buffer, it is assumed that the buffer now has 512 bytes in it.
  - The ESP32 keeps count of how many bytes it processes and after processing 256 bytes, it will send the *request more audio data* message to the host.
  - The host upon starting to play a new audio file will assume that the buffer of the ESP32 is empty and send 512 bytes of data initially and 256 bytes every time it receives a *request more audio data* message.

- **Pause** - On this state we stop the audio file transmission data to the ESP32

- **Next/Previous Song** - On the host, there was a queue of audio files that could be listened to. Normally the host will just send the sound data of each song sequentially to the ESP32, however when one of the change song buttons were pressed, the ESP32 sends a command character to the host that changes which audio file is being sent to the ESP32.

# Results

- Simple and functional bluetooth working with our custom protocol
- ESP32 is able to implement delays giving it a theoretical ability to play audio data back at a rate in the +100 KHz range

- Test it with 512 bytes, process and send message back:
  - Processing time per 512 bytes: **0.017 seconds**
  - Approximate time per byte: **34 us**
  - Max hz: **29418**

# Conclusion & Future Work

- Successfully built our bluetooth speaker with our custom protocol and were able to learn a lot about bluetooth protocol
- Future implementations includes:
  - Add Volume control
  - Send Audio data from smartphone to ESP32 speaker by using already existing audio profile
  - Adapting the protocol to b e able to send both sound data and commands to the ESP32