WYDZIAŁ MATEMATYKI
i INFORMATYKI
Uniwersytet Łódzki

Bohdan Yeromenko

Student number: 384885

Embedded Audio Analysis for a Modernized Retro Toy

Bachelor's thesis in the field of Computer Science

Thesis prepared under the supervision of prof. dr hab. Alexander Shapoval, Department of Artificial Intelligence and Nonlinear Analysis

Łódź, 2025

Introduction

The project aims to contribute to the problem of existing interactive toys and audio producing devices, since they are often limited in their functionality, offering only pre-set patterns of behavior. This prevents them from realizing their full potential and creating a unique, bright and dynamic user experience.
The goal of this project is to modify the popular "Big Mouth Billy Bass" toy so that it can analyze an audio stream in real-time and synchronize its movements (body movement and mouth opening) with the played music from the Bluetooth-connected smartphone with a minimum possible delay.
This problem reflects a broader limitation of many consumer audio devices and interactive toys, which typically rely on static pre-programmed behaviors. In a world where personalization, interactivity, and real-time response are increasingly valued—particularly in smart devices and IoT systems—upgrading such toys represents a unique intersection of embedded systems, signal processing, and user experience design.

The original "Big Mouth Billy Bass" toy from the 2000s featured only two popular songs of the time, and its movements were synchronized exclusively to those tracks. This limited the user's ability to choose music and reduced the interactivity of the toy. Due to the fact that the toy was produced in large quantities and gained wide popularity, I believe that it is important that each user enjoys the functionality of this device with full use of it for their own needs and according to their own preferences, adding personalization to such toys rapidly expands the number of potential users, which in turn increases the number of sales and popularity of the device. Modern technological trends allow us to repeatedly increase the functionality of the toy, making it more attractive to users of all ages.
The improved toy is expected to connect to a smartphone via Bluetooth, functioning as a Bluetooth speaker. The key upgrade will be the device's ability to synchronize its movements with the beat of any music track in real time, thanks to a frequency analysis algorithm — a feature not available in the original version.

This approach is expected to increase interest in the device among users of various age groups and stimulate the creation of similar products, allowing users to choose sound according to their personal preferences.

The final product of the project will be a Bluetooth speaker in the form of an interactive toy, with the ability to synchronize movement to the rhythm of a music track.

This project holds significant potential for future enhancements. It is anticipated that control via Wi-Fi could be added, enabling integration with smart home systems and virtual assistants. Additionally, the original toy casing could eventually be replaced with customizable enclosures, made to order in any shape or size.

Compared to the original version and other attempts to modernize it, the upgraded toy of this project will adapt to any music track and connect to a smartphone or other devices via Bluetooth. Additionally, the device will be extremely easy to control — simply connect it, and the process will be fully automatic.

This new product will offer the ability to customize the user experience, significantly expanding the potential user base across different age groups.

One of the key features that distinguishes this project from the original toy and other modifications is the ability to implement real-time analysis of incoming audio frequencies thanks to the FFT algorithm. Moreover, to the wireless Bluetooth connection that will be implemented in the future, it is possible to add integration with Wi-Fi and other communication methods, which will significantly simplify control over the device.

This project not only gives a second life to a well-known old toy, but also shows that it's possible to make everyday devices more interactive by combining real-time sound analysis and simple mechanical parts. Even with limited hardware resources, it's possible to achieve smart behavior and dynamic reactions, which could be useful not just for entertainment but also for other types of embedded systems that rely on sound or user interaction.

Part 1 – Engineering

1.1 Analysis of system requirements

The device being created must have the following functionality: connect to other devices via Bluetooth or Wi-Fi, play an audio stream via the above-described means of communication, ensure the movement of the mouth, head and tail of the toy in time with the incoming audio stream.

The toy will have a limited range of movements due to the design features of the body, namely: opening and closing the mouth, moving the head by bending the neck to the left (towards the user) and back to the original position, moving the tail to the left (towards the user) and back to the original position.

The device is being developed as an interactive Bluetooth speaker, which means that all modern devices such as phones, tablets, laptops can be connected to it, and in the future, a connection via Wi-Fi may be implemented.

A device of this type should provide uninterrupted and fast audio playback and, most importantly, movement in time with the incoming audio stream. Due to the design and construction features: connection via Bluetooth, the presence of real-time audio stream analysis, as well as the presence of a transmission for electric motors and the electric motors themselves, the permissible response delay is from 100 to 500ms. One of the key aspects of such a project is minimizing delays to ensure uninterrupted audio playback and timely movement of the toy.

Initially, the device was conceived as autonomous, that is, one USB Type-C cable will be used to power the microcontroller with the ability to be powered from a Powerbank or adapter. Also, a battery case built into the original toy for 4 AA batteries will be used to power the electric motors, which will provide an acceptable level of autonomy and battery life.

To activate the device, the user will need to press the only button on the front panel and connect via Bluetooth. After that, if there is power, the device will behave like a typical Bluetooth speaker. It is also worth noting that the number and size of components differs from the original configuration of the original toy and requires the need to fit new components into the same basic toy case to ensure compactness.

1.2 Components selection

To implement the hardware of the device, there are following components needed:

- **ESP32 DevKit (WROOM-32, 30 pin)**
- **MAX98357A Amplifier (I2S)**.
- **YD 40-17 Speaker (3 W, 4 Ohm)**
- **DC motor 3V-6V type 270 (2 pcs)**
- **Drv8833 H-Bridge**
- **Adjustable power supply (3-12V, 3A)**
- **"Big Mouth Billy Bass" original toy**
- **Consumables (soldering iron, wires, etc.)**

Each of these components was selected based on specific technical requirements and tasks to ensure the necessary functionality of the device. The design elements and the reasons for their selection are described in detail below.

ESP32 DevKit (WROOM-32, 38 pin) microcontroller was chosen as an alternative to low-power Arduino and excessively powerful and expensive Raspberry pi, it combines the necessary power for this project, relative cheapness and the presence of a large number of libraries for this platform. It is used as a central microprocessor that combines the tasks of receiving a signal via Bluetooth and Wi-Fi, playing audio using an amplifier and speaker, as well as algorithmic analysis of the incoming audio stream in real time, which allows you to control the movements of the motors in time with the music. The main reason for abandoning the low-power Arduino in favor of the more expensive and powerful ESP 32 is the presence of the 520 KB of RAM and 240MHz CPU which gives us the opportunity to use more advanced algorithms for analyzing the audio stream, simultaneously controlling three DC motors. Such power of the microcontroller gives us the opportunity in the future to increase the functionality of the device by adding new functions such as Wi-Fi control and also adding a visual component in the form of a screen with an equalizer, without cutting its computing power in relation to the main task, namely, analyzing the audio stream in real time. Also, an important selection criterion is the presence of a built-in Bluetooth module and the presence of a large number of ready-made libraries and easy integration into common development environments. It is also worth noting that ESP32 has 38 pins and a variety of different interfaces (GPIO, I2S, SPI, I2C, UART, PWM, Bluetooth) for connecting peripheral devices.

MAX98357A amplifier is an excellent choice due to its compactness, ease of connection and compatibility with a chosen microcontroller. Unlike an analog connection, the I2S digital stream is much more resistant to interference from component wiring and power supplies. The ESP32 microcontroller is unable to directly output analog audio since it lacks a DAC, so it becomes clear that it is necessary to convert the digital signal to analog using the amplifier's built-in converter. The MAX98357A is a Class D audio amplifier, which means it has high power efficiency and low thermal output, which in total gives us an ideal candidate for a device that must operate for a long time without reboots and shutdowns.

YD 40-17 is speaker with a ferrite magnet, which, unlike a piezoelectric speaker, has a much better sound quality with similar dimensions, which is critical when installing such a speaker in the case of a small toy. When comparing the sound quality of a piezoelectric speaker and a speaker with a ferrite magnet, the superiority of the latter becomes obvious. It is also important to emphasize that a speaker with a ferric magnet provides much better reproduction of various frequencies compared to a piezoelectric speaker. Also, this speaker has a resistance of 4 Ohms, which makes it an ideal candidate for an amplifier MAX98357A, due to its ability to stabilize the output current for low-power speakers 3–8 Ohms.

DC motor 3V-6V type 270 are DC 3-6V motors which are already pre-installed in the original toy, which is suitable for my purposes in all respects: power supply from four 6V, 11,000 rpm and AA batteries powered, as well as the ability to increase the speed of rotation with increasing voltage.

H-Bridge drv8833 motor driver is a common dual-channel H-bridge motor driver, it was chosen because it allows to control two motors at the same time, and in either direction. This driver can provide up to 1.2A of DC current to each motor, and in some situations even more, which is an important criterion in toys of this kind. This driver also includes overheat protection in the form of a heatsink and the ability to control motors using PWM (pulse wide modulation), it is easy to adapt to ESP32 and is suitable for most motors used in toys.

Adjustable power supply is the best option since the ESP32 microcontroller requires 5V and has its own USB-C type connector which is using for power supply and firmware. This connector is the most optimal option in view of the fact that it is already implemented on the microcontroller chip and also it is worth to admit the large prevalence of power supplies and USB-C type cables. It is important that USB-C adapters have built-in protection against overheating, overvoltage, and short circuits, which makes the use of the toy safer.

PLP52 customizable network adapter - it is an external power source that has several operating modes of output voltage, while consuming 220 V from a standard socket. Was chosen due to the disadvantage of alkaline batteries, which are tend to heat up quickly and lose voltage under prolonged power loads such as the movement of electric motors. As a result of numerous tests, this power source was chosen as the most reliable, because the voltage from the wall sockets guarantees stable and safe operation of the device at 6V permanently.
It is worth noticed that the adapter does not directly participate in powering the microcontroller and sound reproduction modules. It serves only to power motor group.

“Big Mouth Billy Bass” original toy was chosen due to its wide popularity, and is also ideal for the implementation of such a project due to the pre-installed transmission for electric motors, the electric motors themselves, which provide movement of the mouth, neck and tail of the toy, significantly save time and resources when creating such a project. There is a presence of a case for batteries and a hole in the case for a speaker too. The original toy also has a pre-installed control board, which will later be removed and replaced with an ESP32. Also subject to replacement is a cheap and low-quality pre-installed speaker that does not meet the requirements of the project

Consumables which are used during the assembly process. Consists of typical materials for such work, namely: wires, connectors, solder, flux, electrical tape, heat shrink tubing, glue gun/double-sided tape, plastic ties, breadboards for testing in the early stages, as well as tools such as a soldering iron, pliers, screwdrivers.

1.3 Block diagram of the system

To better understand the internal structure of the device, there is a logical block diagram represented, it shows the connections between the device components:

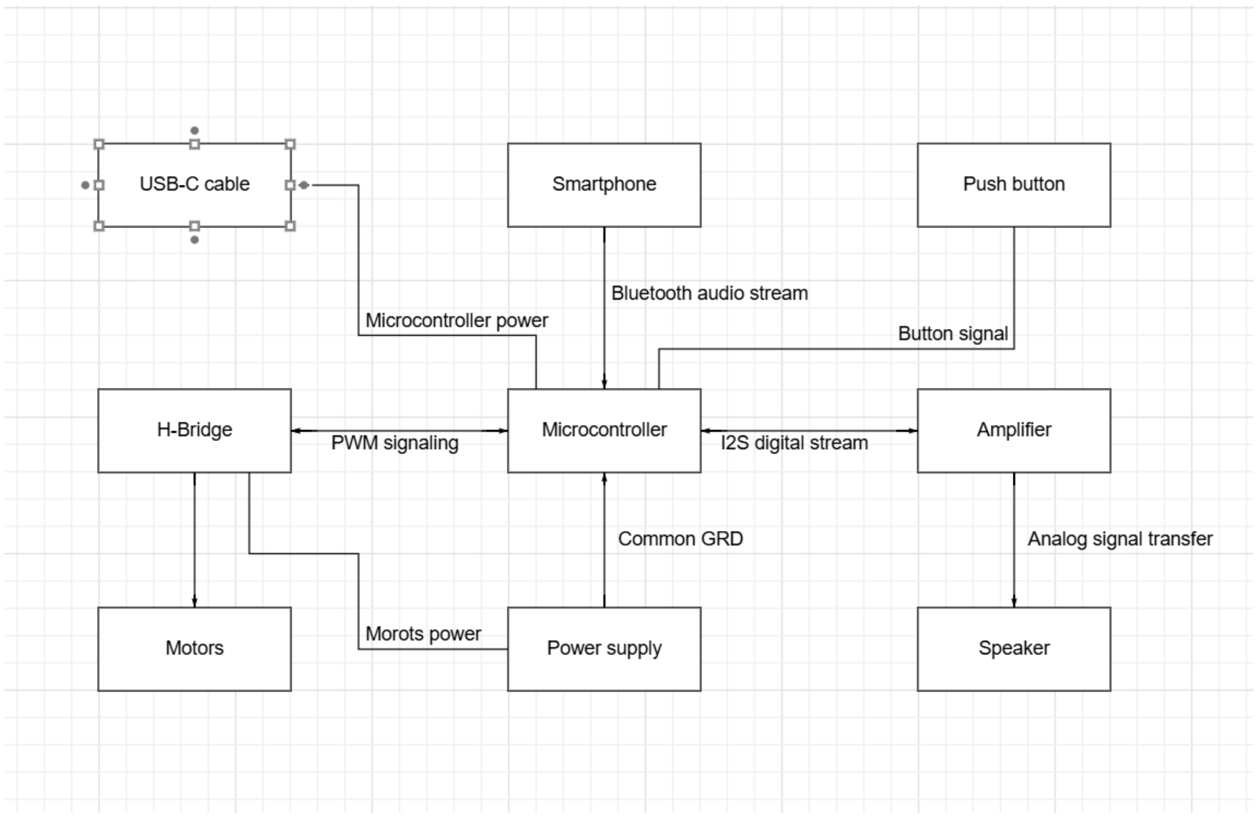


Figure 1 -Block diagram of the components

The central element of the system is the ESP32 microcontroller, which is responsible for receiving a signal from an external source (e.g. smartphone) via Bluetooth. The received signal is used to solve two key tasks of the device: playing audio through the device's speaker and control of the movement of the built-in DC motors in the real time. The sound is transmitted in a digital form via the I2S interface to the MAX98357A amplifier, which converts the incoming signal into the analog, so that it can later be played to the speaker. At the same time, the ESP32 analyzes the audio stream in real time using the Fast Fourier Transform (FFT) algorithm and, based on it, generates PWM (pulse-width-modulation) signals that are transmitted to the drv8833 motor driver, which in turn controls three DC motors, thereby ensuring the movement of the toy in time with the reproduced sound. The microcontroller is powered via the built-in USB-C input, while the motors are powered using 4 AA batteries. It is worth noting that the motor driver, microcontroller and the motor power supply have a common GND, which is necessary to keep the proper working of the electric motor control logic.

1.4 Electrical diagram of the system

To implement this project, an electrical circuit was developed that ensures the correct operation of all components of the device: microcontroller, amplifier, speaker, motor driver and motors. This section will present a detailed connection diagram of all components as well as a detailed description of their functions and contacts.

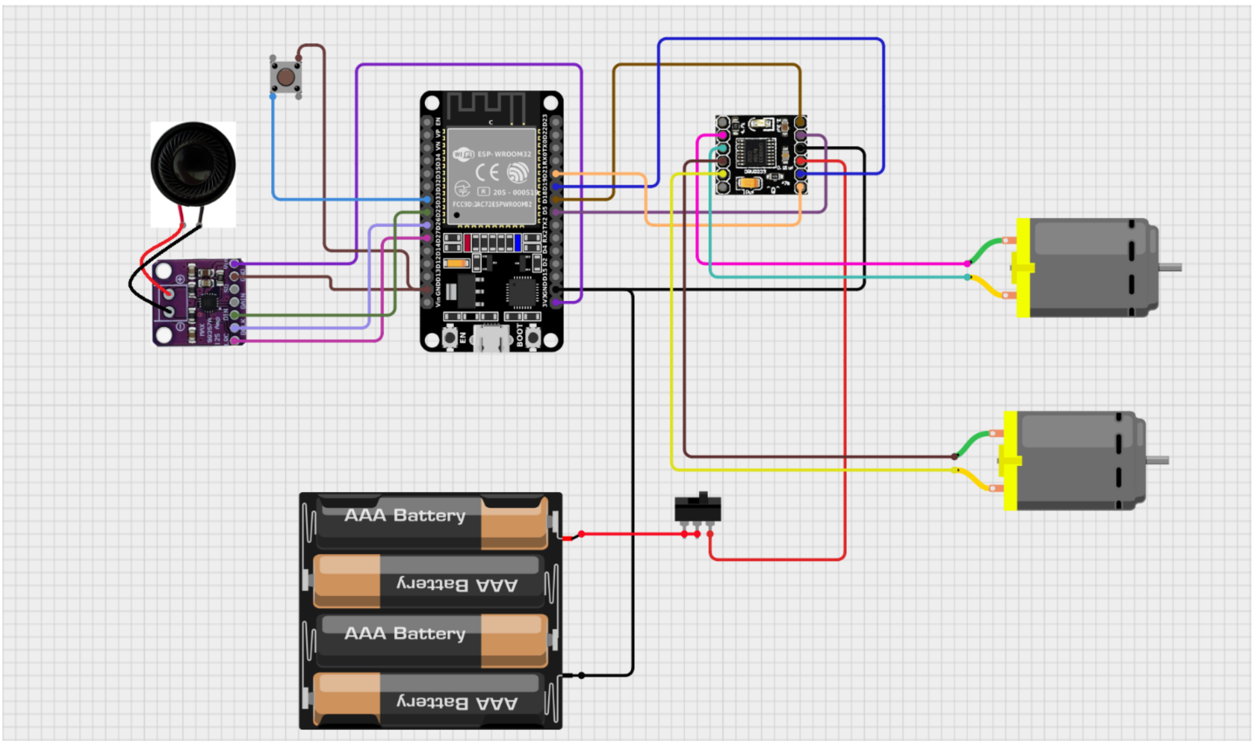


Figure 2 - electrical circuit diagram

1.5 Pinout

a) Connecting ESP32 to MAX98357A and to the speaker.

The I2S interface is used to transmit the audio signal from the ESP32 to the MAX98357A amplifier. It uses following pins to connect to ESP32:

I2S Data (DIN) -> D22: This pin is responsible for transmitting audio data from the microcontroller to the amplifier.

I2S LRC (WS) -> D25: This pin is used to synchronize the transmission of the left and right channels.

I2S Clock (BCLK) -> 26: This pin is used to transmit the clock signal.

MAX98357A(VCC) -> ESP32 (3.3V) Power pin

MAX98357A(GND) -> ESP32 (GND) Power pin

MAX98357A is connected to the YD 40-17 speaker via VCC and GND output of the amplifier and the following pins of speaker.

b) Connecting ESP32 to drv8833 and to motors.

To control the movement of the toy elements (mouth, tail, head) DC motors are used. Each motor is connected to one of the channels of the drv8833 driver:

OUT 1-2 -> Motor 1

OUT 3-4 -> Motor 2

The drv8833 is powered by an external 6V source (battery case), which is connected to the VCC pin and GND of the motor shield. The VCC pin of the drv8833 is connected to VCC of the ESP32 to keep the motor control logic works.

Pins IN1, IN2, IN3, IN4 are used to set the direction of the motors rotation and connected with ESP32 GPIO pins in the following way:
IN1 -> D19

IN2 -> D21

IN3 -> D5

IN4 -> D18

c) Power connection.

ESP32 microcontroller is powered by the USB-C cable and has a pre-installed connector for it. This provides a stable 5V power of the device.

Motors are powered by 6V external power module which is represented by PLP52 customizable network adapter due to the fact that the standard alkaline batteries cannot provide the required voltage due to the mechanical motor group system (gear transition and the rotating frequency slows motors, increasing the voltage in “slipping» mode).

It is important to notice that all components have the same GND to provide the correct work of all components and signals synchronization.

d) Buttons connection

There are a push button and the switched represented over the scheme.

Push button is connected to the D33 pin and the GND of ESP32 to generate a signal which starts the Bluetooth connection procedure.

The switcher is connected between the VCC power pin of the drv8833 motor shield and the battery case.

e) General comments

All connections between components (e.g. between ESP32 and MAX98357A or drv8833 and motors) are made using well-insulated wires and heat shrink tubes to prevent short circuits and provide the safe way of the device using.

It is important that the microcontroller, amplifier and motors operate with different power sources (ESP32 with 5V, and motors with 6V), but the common GND must be connected for the control signals to work correctly.

1.6 Assembly of the device

a) Preparing for assembly.

The assembly will be divided into two parts: the acoustic part will contain an amplifier and a speaker, and the mechanical part will contain DC motors and a motor driver. This approach will simplify installation in the device case, minimize the chance of errors, and simplify testing. Also, such necessary tools for assembly as a soldering iron, multimeter, screwdriver, pliers, heat shrink cables, electrical tape and glue will be prepared.

b) Removing the original components

The original production toy contains a pre-installed microcontroller board and a speaker; they are subject to removal and replacement due to non-compliance with the technical requirements of this project. It is important to carefully remove them without damaging other pre-installed mechanisms and devices such as electric motors and their transmission, as well as the device power button and battery case. If necessary, replacement of wires and soldering on contacts of pre-installed components will be performed to increase the overall reliability of the device.

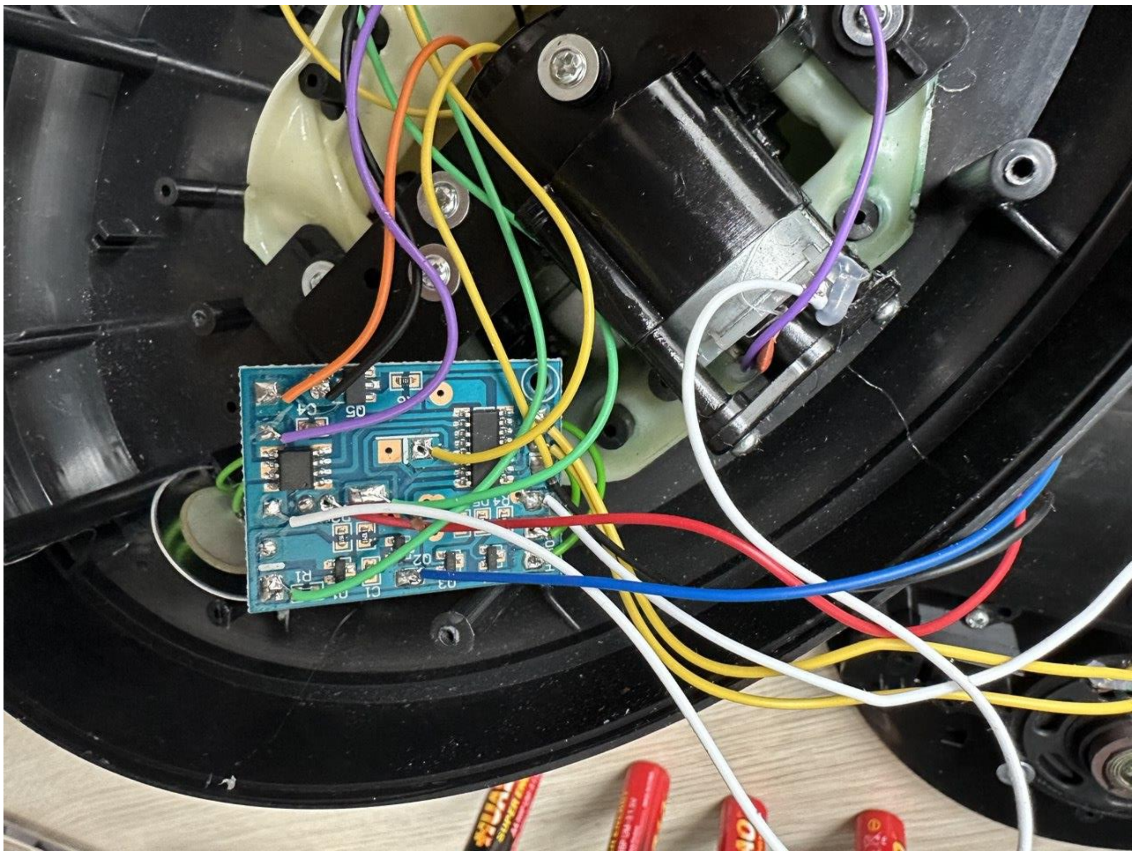


Figure 3 - Original components

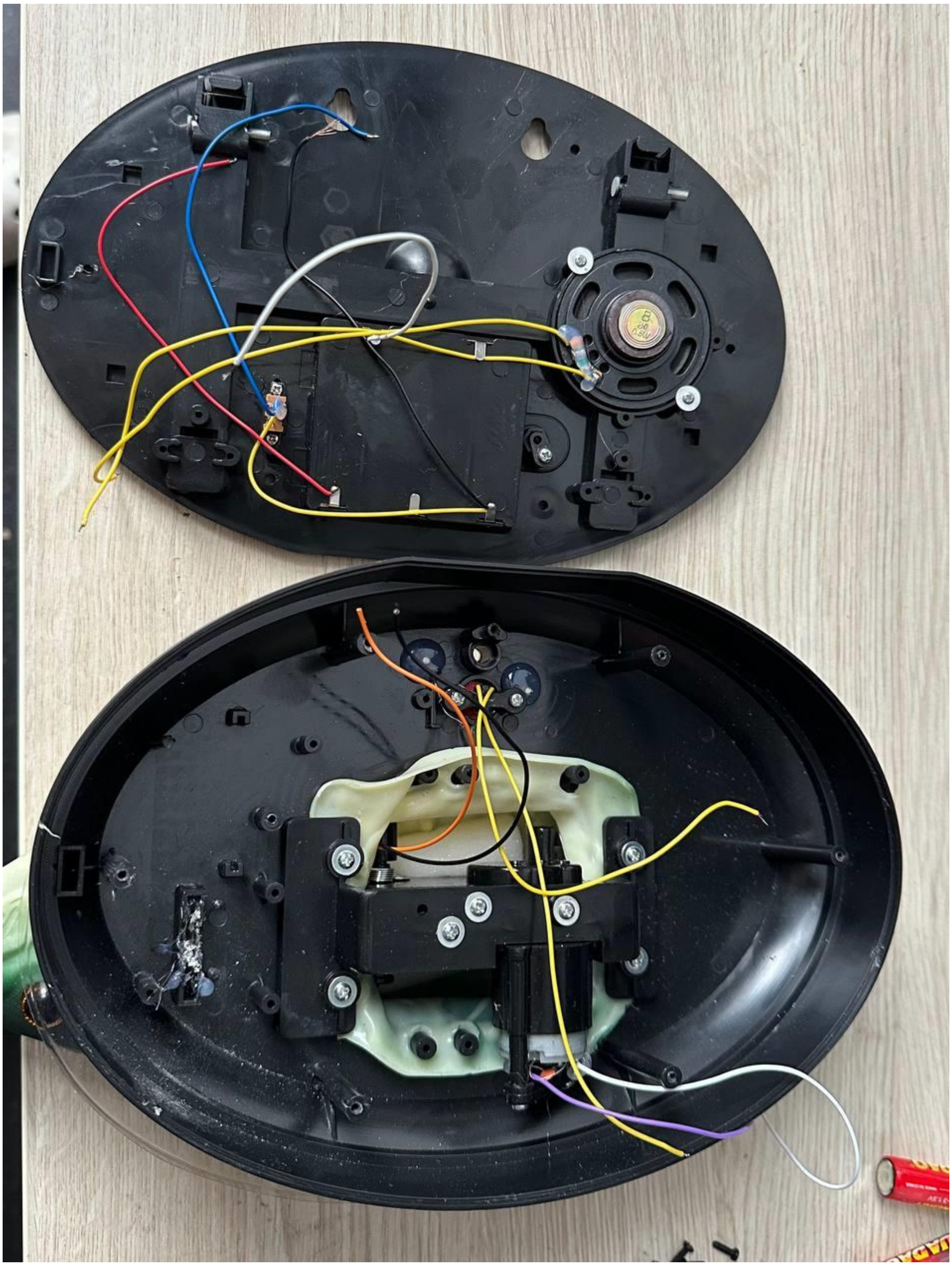


Figure 4 - Removing original components

c) Checking the assembly and functionality

To test the functionality of all modules, it is necessary to write a simple firmware sketch that will affect the functionality of all modules and include all possible options for their behavior. For this, it is reasonable to divide them into two groups: acoustics and movement. This will significantly simplify the testing process and minimize time costs and errors.

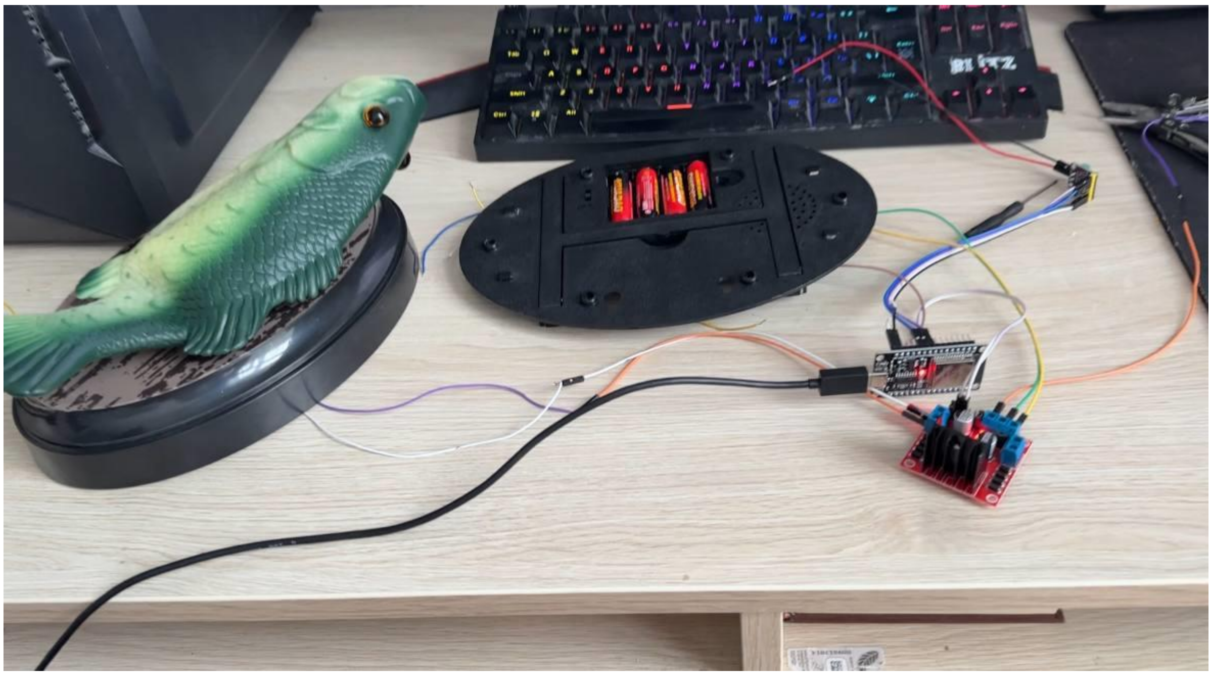


Figure 5 – Motor group pre-check

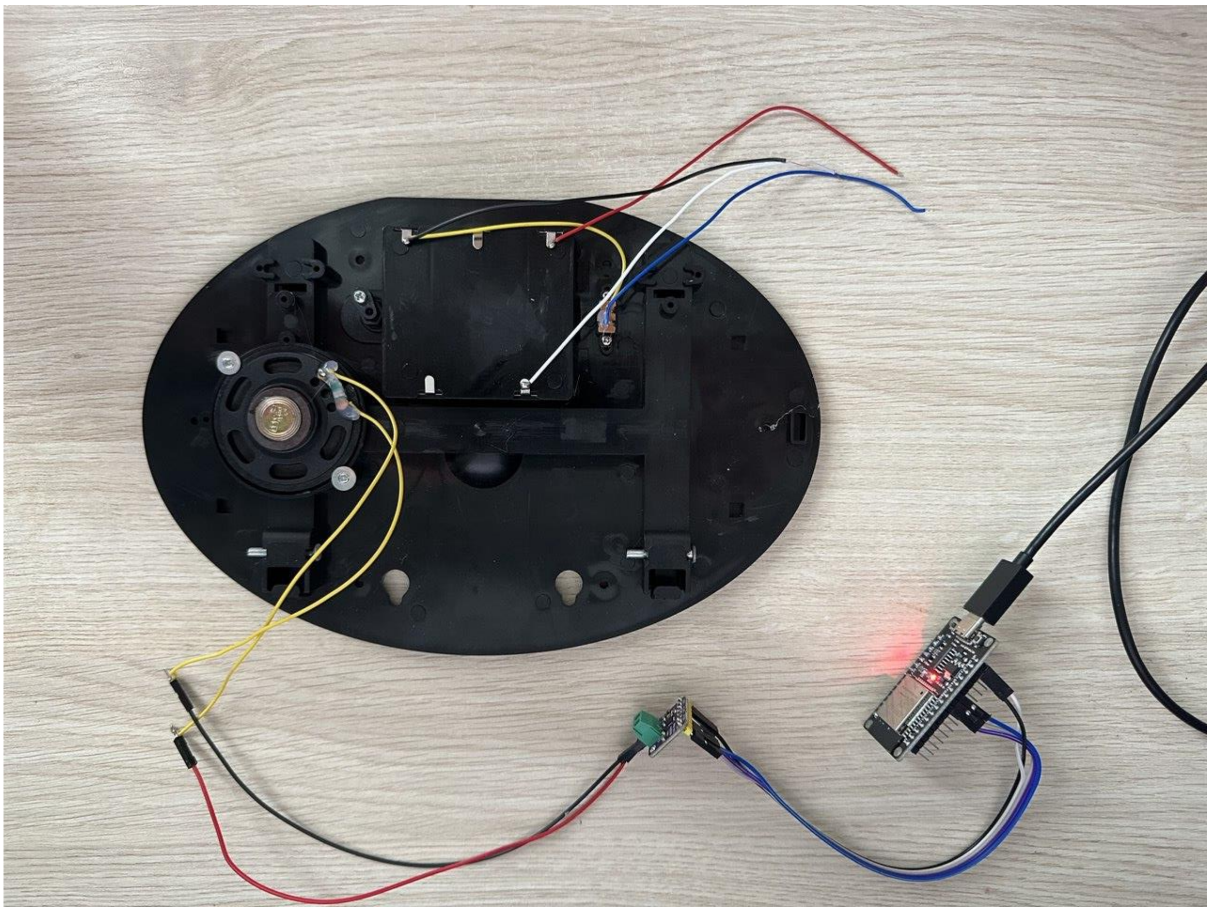


Figure 6 - Audio group pre-check

d) Placement of components inside the toy case

Since the space inside the original toy case is limited, careful planning must be done to place the new components inside the case so that the back cover can be closed without damaging the new components. Double-sided tape or mounting glue will be used to secure the new components inside the device body.

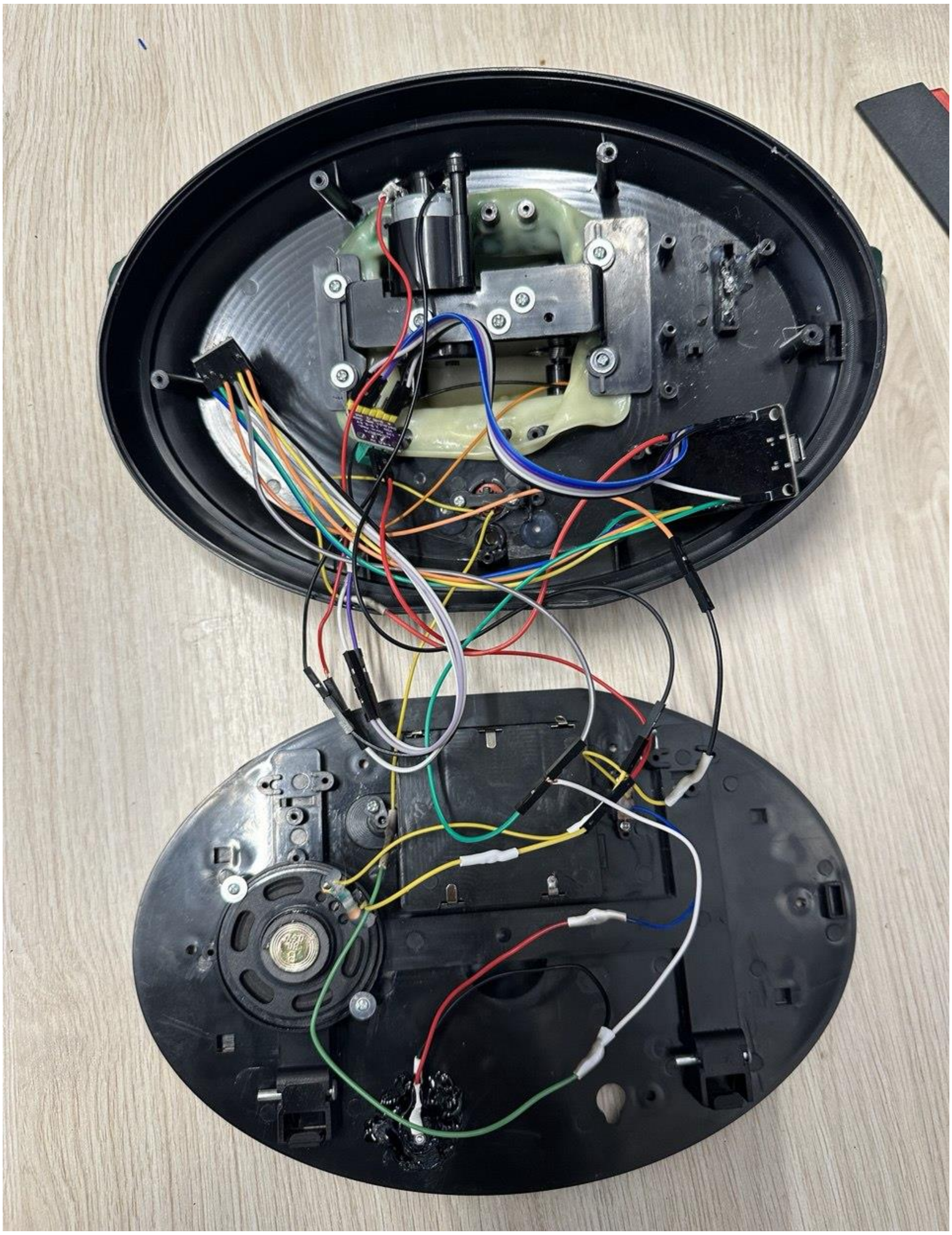


Figure 7 - Replaced components

e) Power supply connection

To ensure a pleasant appearance, it is necessary to minimize the number of wires that will stick out from the case of the toy. Thus, during the final assembly of the device, USB cable remained for power supply and final configuration of the microcontroller, as well as an external power source for the motors. It is worth noting that for the external power source, a hole was made in the case for the connector, and the connector itself was attached with liquid glue to the back wall of the device.

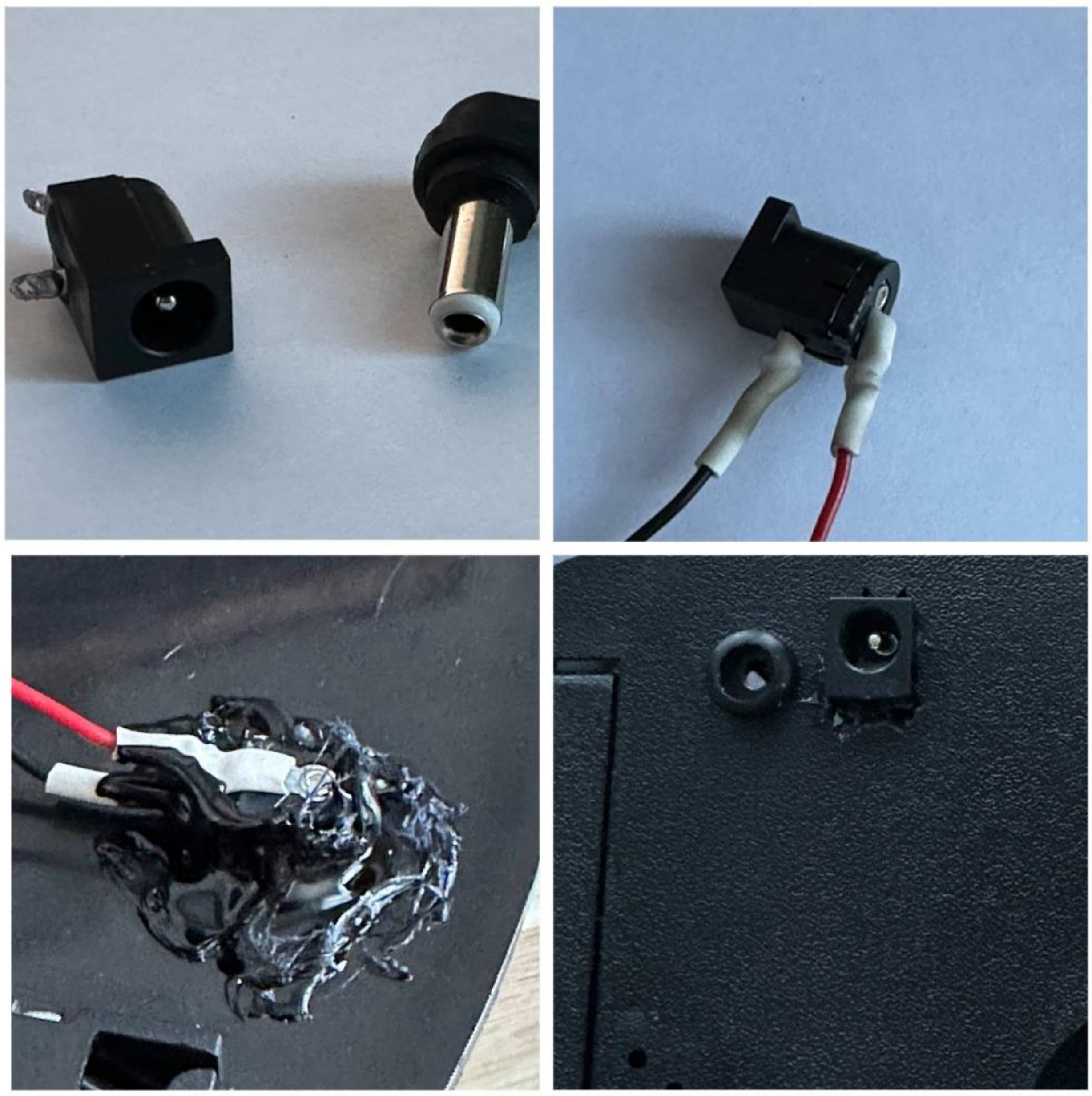


Figure 8 - Motor group power supply instalation

f) The case and aesthetics of the device

Due to the competent placement inside the case and soldering of connectors to the back cover, it was possible to achieve an acceptable appearance of the device, minimize the number of wires sticking out of the case, as well as ensure a reliable connection of all components and eliminate the possibility of a short circuit inside the case.

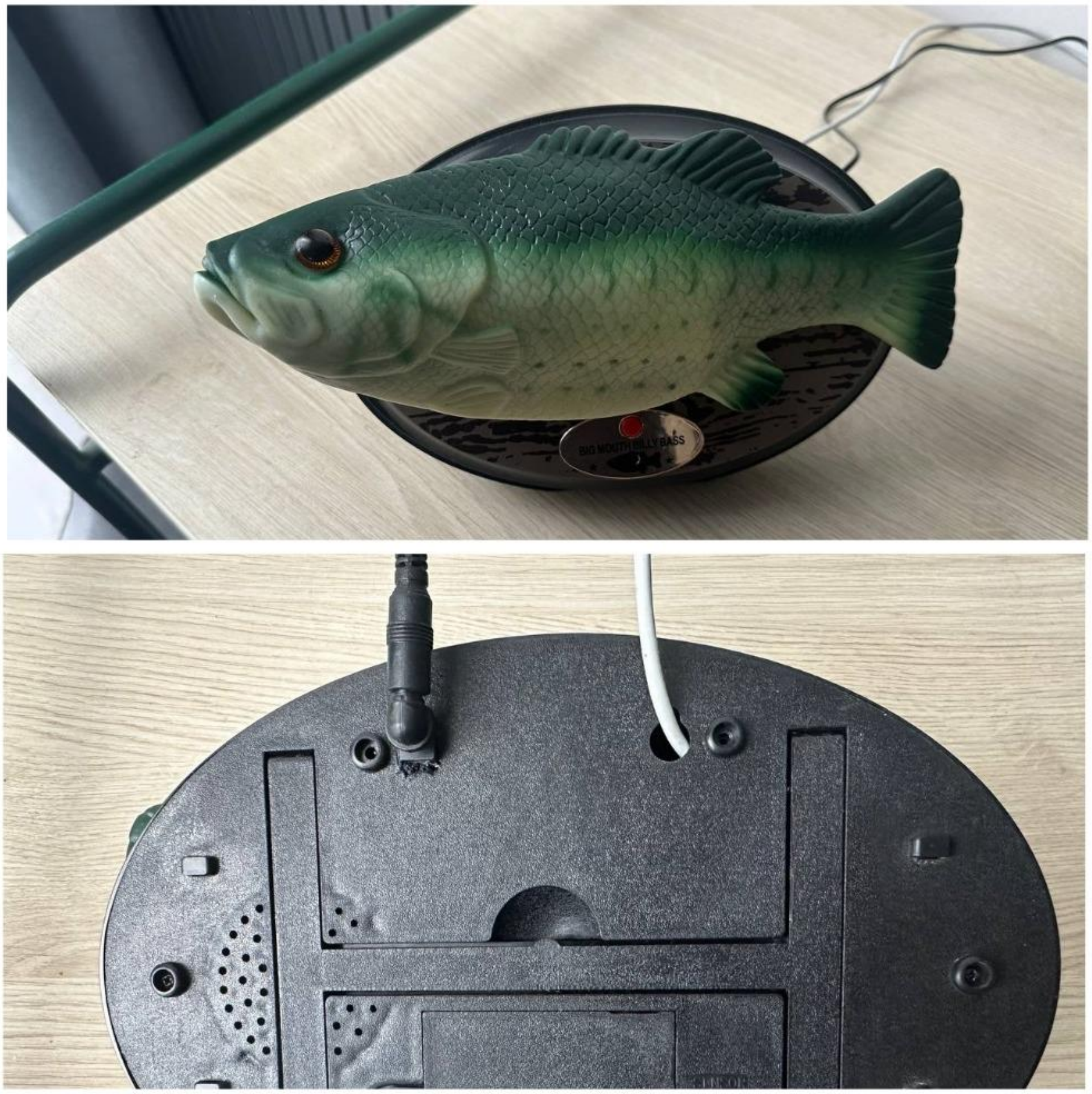


Figure 9 - Final view of the device

Part 2 – Programming

2.1 Software Architecture Overview

Due to the fact that the ESP 32 microcontroller was chosen for the implementation of this project, the development of software is influenced by some requirements such as: the use of the development environment for Arduino IDE microcontrollers, which was chosen due to its availability, ease of connection of the necessary libraries and user-friendly interface, as well as the use of C++ as the main programming language, since it is the standard for programming microcontrollers.

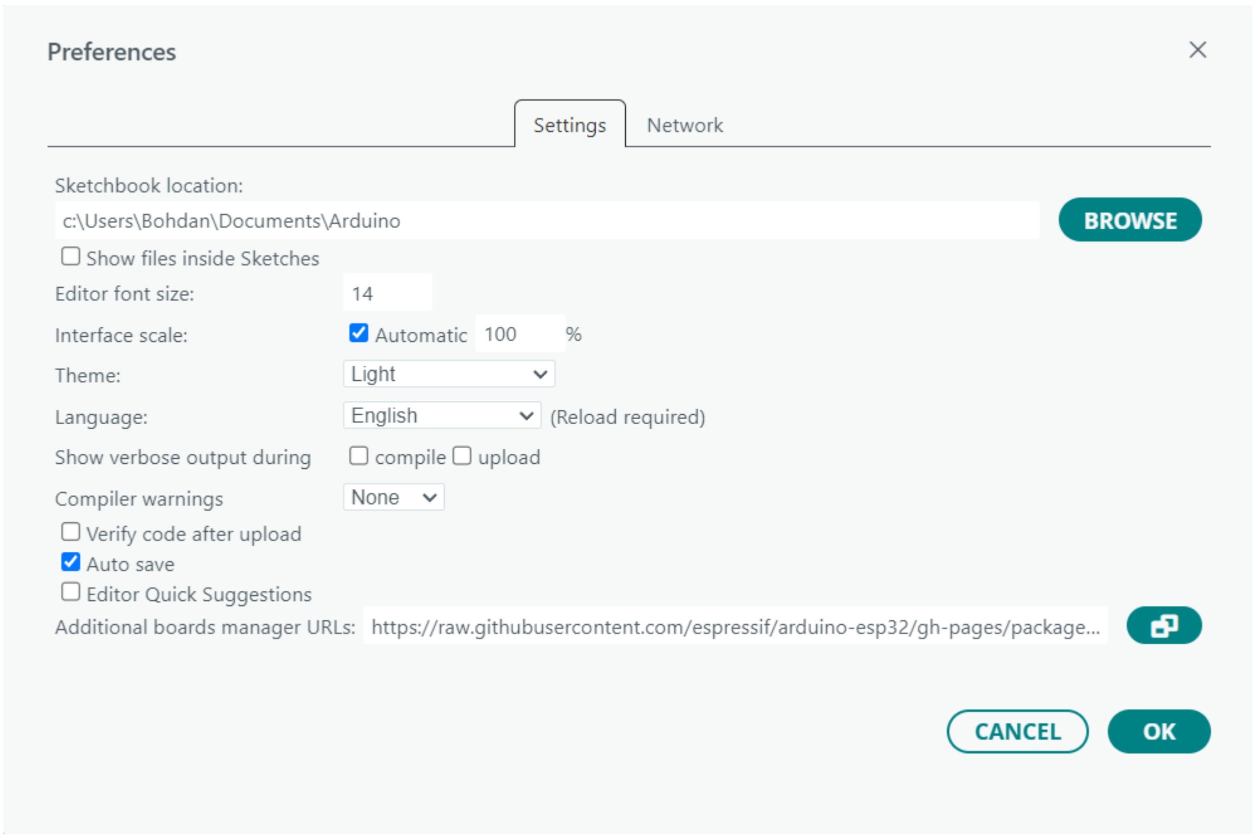
The general architecture of the software solution for this device consists of several interconnected modules: a Bluetooth module for receiving audio from an external device such as a smartphone, an I2S module for playing sound using an amplifier, an FFT module that serves to analyze audio frequencies and plays an important role in constructing motor control signals, and a motor control module that is responsible for the frequency and direction of movement of electric motors.

Among the main tasks that the project's program code must solve, the following can be highlighted: establishing a Bluetooth connection with another device to transmit an audio signal, playing an audio signal via the I2S interface, processing an audio signal using the FFT algorithm, constructing the logic of electric motor movement based on information obtained during audio analysis, and generating signals for electric motor movement.

Among the main technical difficulties in the software part, it is worth to highlight the complexity of setting the bass and vocal thresholds, as well as synchronization and ensuring fast action in real time or as close to it as possible.

2.2 Development Environment Setup

To start working with the microcontroller ESP32 in the Arduino IDE, there is a need to pre-configure the development environment, namely, add a URL for a specific microcontroller model in the board manager, and also find, select and install additional packages for working with this version of the microcontroller.


Preferences

Settings | Network

Sketchbook location: c:\Users\Bohdan\Documents\Arduino

☐ Show files inside Sketches

Editor font size: 14

Interface scale: ☑ Automatic 100 %

Theme: Light

Language: English (Reload required)

Show verbose output during ☐ compile ☐ upload

Compiler warnings None

☐ Verify code after upload

☑ Auto save

☐ Editor Quick Suggestions

Additional boards manager URLs: https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package...

Figure 10 - Arduino IDE board manager URL

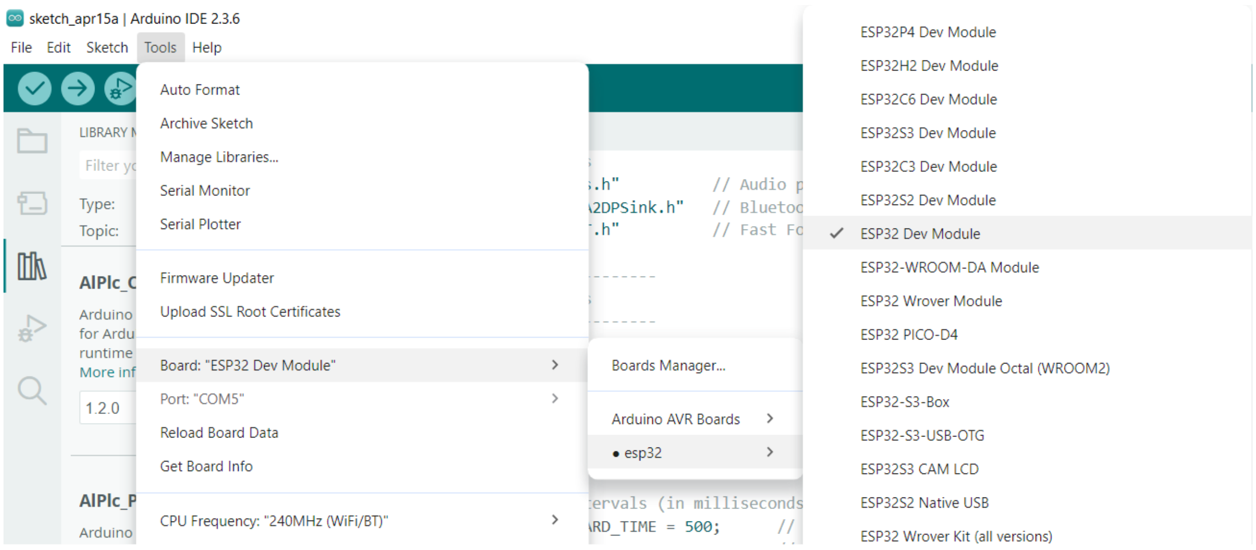
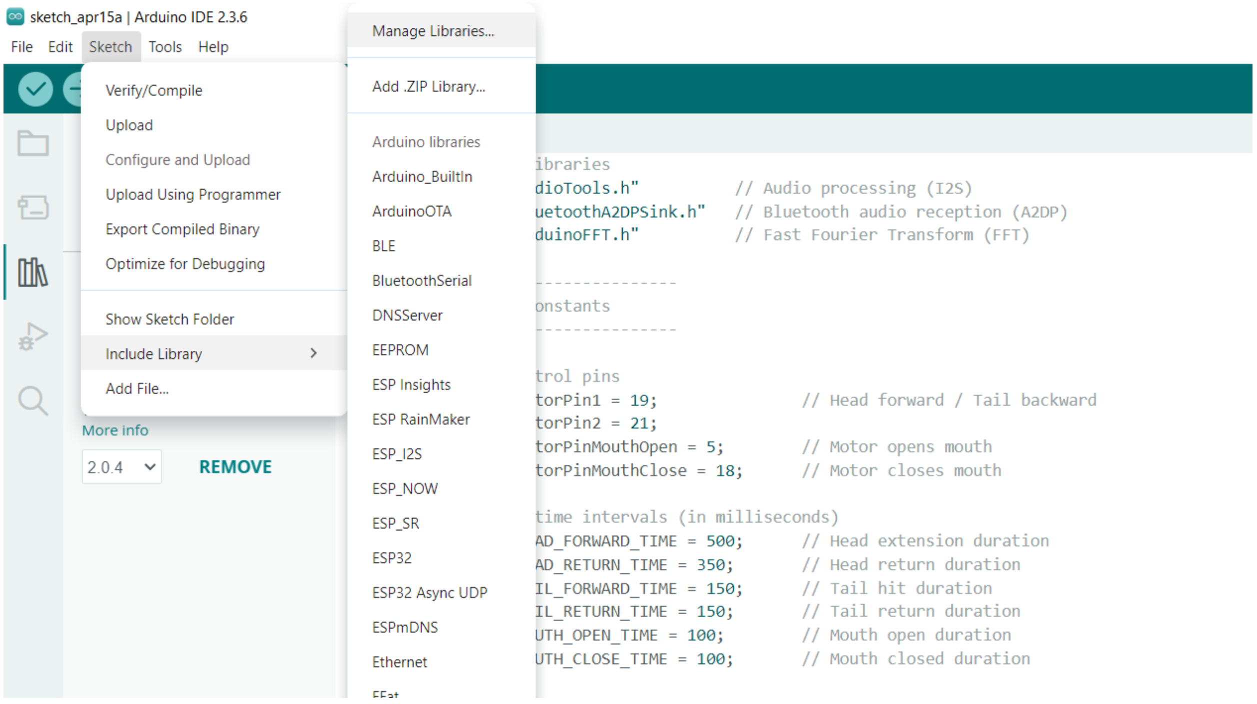


Figure 11 - Arduino IDE board selection

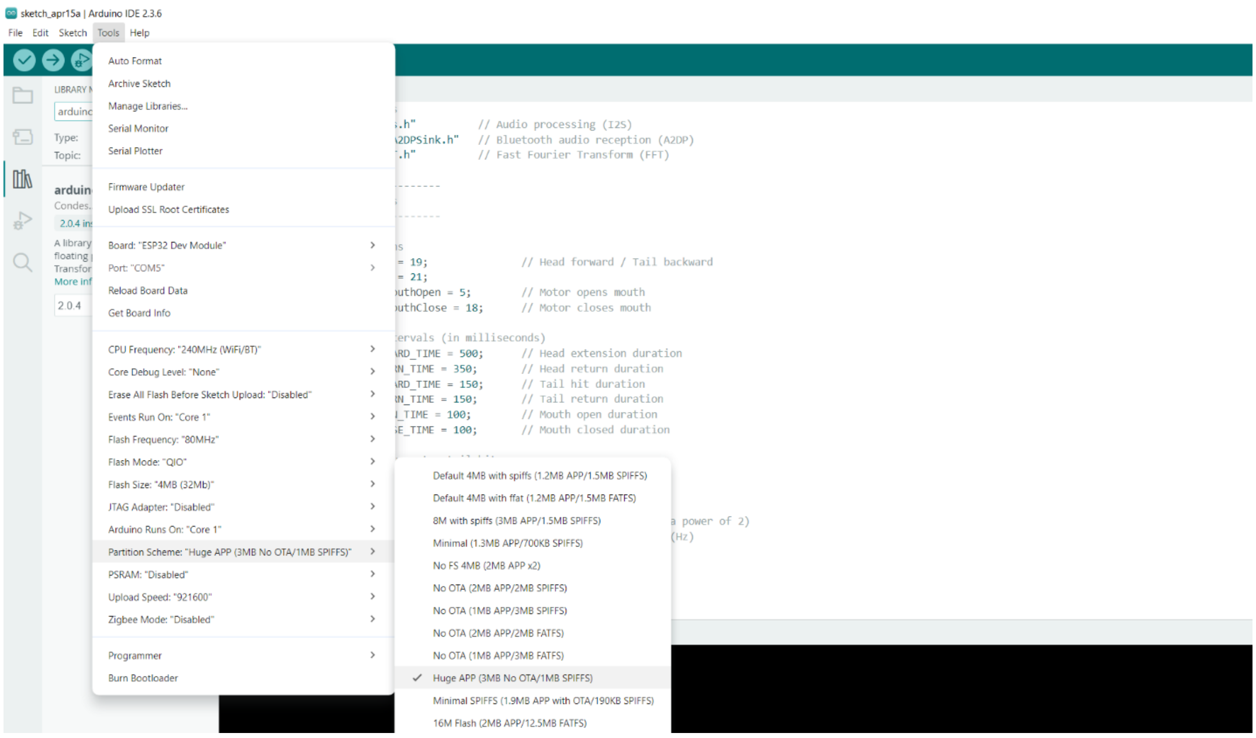
To ensure the correct operation of the program, it is necessary to install several libraries such as: AudioTools.h, BluetoothA2DPSink.h, and arduinoFFT.h. They can be installed through the library manager in the Arduino IDE or downloaded directly from the developer's page on GitHub and imported into the project manually using Add .ZIP Library option.



```
ibraries
dioTools.h"          // Audio processing (I2S)
uetoothA2DPSink.h"   // Bluetooth audio reception (A2DP)
duinoFFT.h"          // Fast Fourier Transform (FFT)

---------------
onstants
---------------

trol pins
torPin1 = 19;              // Head forward / Tail backward
torPin2 = 21;
torPinMouthOpen = 5;       // Motor opens mouth
torPinMouthClose = 18;     // Motor closes mouth

time intervals (in milliseconds)
AD_FORWARD_TIME = 500;     // Head extension duration
AD_RETURN_TIME = 350;      // Head return duration
IL_FORWARD_TIME = 150;     // Tail hit duration
IL_RETURN_TIME = 150;      // Tail return duration
UTH_OPEN_TIME = 100;       // Mouth open duration
UTH_CLOSE_TIME = 100;      // Mouth closed duration
```

Figure 12 - Arduino IDE library manager

It is worth noting that due to the fact that the BluetoothA2DPSink.h library cannot function correctly without AudioTools, which in turn is a common solution for a huge number of situations and, as a result, takes up a huge amount of space in the microcontroller's memory during compilation and loading. In this regard, we have expanded the partition scheme from the Default 4MB with spiffs (1.2MB APP/1,5MB SPIFFS) to Huge APP (3MB No OTA/1MB SPIFFS). This change allowed us to compile the program without exceeding the available memory limits of the microcontroller.



Board: "ESP32 Dev Module"
Port: "COM5"
CPU Frequency: "240MHz (WiFi/BT)"
Core Debug Level: "None"
Erase All Flash Before Sketch Upload: "Disabled"
Events Run On: "Core 1"
Flash Frequency: "80MHz"
Flash Mode: "QIO"
Flash Size: "4MB (32Mb)"
JTAG Adapter: "Disabled"
Arduino Runs On: "Core 1"
Partition Scheme: "Huge APP (3MB No OTA/1MB SPIFFS)"
PSRAM: "Disabled"
Upload Speed: "921600"
Zigbee Mode: "Disabled"

Default 4MB with spiffs (1.2MB APP/1.5MB SPIFFS)
Default 4MB with ffat (1.2MB APP/1.5MB FATFS)
8M with spiffs (3MB APP/1.5MB SPIFFS)
Minimal (1.3MB APP/700KB SPIFFS)
No FS 4MB (2MB APP x2)
No OTA (2MB APP/2MB SPIFFS)
No OTA (1MB APP/3MB SPIFFS)
No OTA (2MB APP/2MB FATFS)
No OTA (1MB APP/3MB FATFS)
✓ Huge APP (3MB No OTA/1MB SPIFFS)
Minimal SPIFFS (1.9MB APP with OTA/190KB SPIFFS)
16M Flash (2MB APP/12.5MB FATFS)

```
.h"          // Audio processing (I2S)
2DPSink.h"   // Bluetooth audio reception (A2DP)
.h"          // Fast Fourier Transform (FFT)

--------

--------

s
= 19;                 // Head forward / Tail backward
= 21;
uthOpen = 5;          // Motor opens mouth
uthClose = 18;        // Motor closes mouth

ervals (in milliseconds)
RD_TIME = 500;        // Head extension duration
N_TIME = 350;         // Head return duration
RD_TIME = 150;        // Tail hit duration
N_TIME = 150;         // Tail return duration
_TIME = 100;          // Mouth open duration
E_TIME = 100;         // Mouth closed duration

                      a power of 2)
                      (Hz)
```

Figure 13 - Arduino IDE Platform Scheme selection

2.3 Core Libraries Analysis

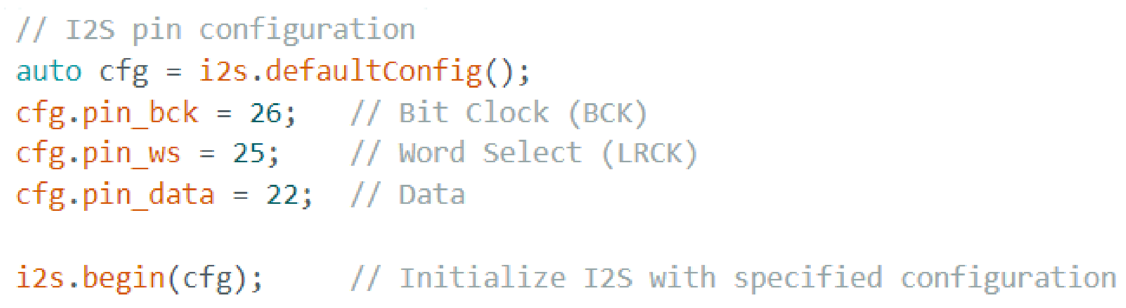
AudioTools is an extensive library designed to work with digital audio signals on microprocessors such as ESP32 and Arduino, it provides all the capabilities for capturing playback and processing audio using the I2S (Inter-IC-Sound) interface. Specifically, in our case, this library is used to read audio data in real time from the I2S interface connected to the Bluetooth module. The I2SStream class is used to receive an audio stream in 16-bit PCM format (Pulse Code Modulation) with a sampling frequency of 44100 Hz. This library was used due to its high compatibility with various devices such as amplifiers, microphones and speakers. The presence of a large number of materials reduces the chance of error and speeds up the process of writing code. It is worth noting that the BluetoothA2DPSink library cannot operate without AudioTools.

BluetoothA2DPSink is a library designed to receive audio via Bluetooth using the A2DP (Advanced Audio Distribution Profile) protocol. In the context of this project, it allows the microcontroller to act as a receiver of audio that is broadcast from an external device, such as a smartphone. This library is the optimal choice since it is designed to transmit high-quality audio, which is a critical point in this project. It is also worth noting the great popularity of this library and, as a result, the presence of a large number of working code examples.

ArduinoFFT is designed to implement FFT on microcontrollers. In the project, this library is used to highlight bass and vocals based on the analysis of low (60-200 Hz) and mid (300-3000 Hz) frequencies. It is a key component of the electric motor movement trigger. It was chosen due to the optimal ratio of the consumed microcontroller resources and the quality of the audio stream analysis.

2.4 Audio Processing Implementation

The I2S interface in the context of the project is used to transmit a digital audio stream from the ESP32 Bluetooth module forward to FFT analysis. In this implementation, the RX mode is used since the goal is to analyze the incoming audio stream and not to transmit it.
In the AudioTools library, the I2S configuration is specified using the structure:



```
// I2S pin configuration
auto cfg = i2s.defaultConfig();
cfg.pin_bck = 26;    // Bit Clock (BCK)
cfg.pin_ws = 25;     // Word Select (LRCK)
cfg.pin_data = 22;   // Data

i2s.begin(cfg);      // Initialize I2S with specified configuration
```

Figure 14 - I2S pin configuration

pin_bck — Bit Clock (BCLK): synchronizes individual bits in each audio sample.

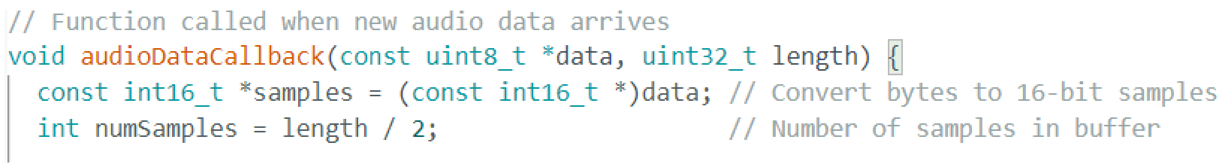
pin_ws — Word Select (WS или LRCLK): Indicates which audio channel (left or right) the current bit stream belongs to.

pin_data — Serial Data (DOUT/DIN): transmits the audio data.

These pins, namely GPIO 26, GPIO 25, GPIO 22, were selected based on the following criteria: no conflict with other peripheral functions; not used anywhere else in the project; physically located next to each other, which simplifies installation and simplifies access to the physical pins of the microcontroller.

This project uses the audioDataCallback callback function to handle audio, which is called every time a new block of audio data is received from Bluetooth via the BluetoothA2DPSink library. This allows the A2DP stream to be intercepted in real time and passed to the frequency analysis module.

The signature of the audioDataCallback function is:



```
// Function called when new audio data arrives
void audioDataCallback(const uint8_t *data, uint32_t length) {
  const int16_t *samples = (const int16_t *)data; // Convert bytes to 16-bit samples
  int numSamples = length / 2;                    // Number of samples in buffer
```

Figure 15 - audioDataCallback function signature

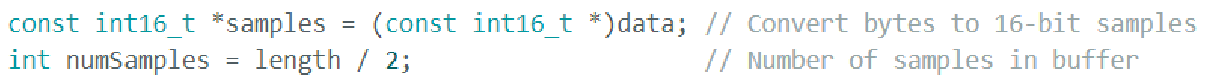
data — pointer to a byte array containing raw PCM audio data obtained after decoding the Bluetooth stream.

length — number of bytes in the buffer.

AudioDataCallback function is called automatically when audio data is received via Bluetooth and is responsible for processing the audio stream before performing the analysis (FFT).

When an external device (for example, a smartphone) transmits an audio signal via Bluetooth, the ESP32 microcontroller receives it as an array of bytes. This data is a 16-bit PCM stream, in which every two bytes correspond to one audio sample.

The first step in processing is to convert the incoming bytes into 16-bit integer values (int16_t) to interpret them as the amplitude of the audio signal. To do this, the pointer to the byte array is redefined as a pointer to an array of 16-bit integers, and the total number of audio samples is determined by dividing the length of the input array by two.



```
  const int16_t *samples = (const int16_t *)data; // Convert bytes to 16-bit samples
  int numSamples = length / 2;                    // Number of samples in buffer
```

Figure 16 - Convernig to 16-bit samples

The extracted values are then placed in the vReal[] array, which stores the amplitudes of the real part of the signal. At the same time, the vImag[] array, which is responsible for the imaginary part, is filled with zeros. This is necessary because the Fast Fourier Transform (FFT) algorithm requires complex input values, despite the fact that the original audio signal is purely real.

```
// Store audio samples into array for FFT
for (int i = 0; i < numSamples && sampleIndex < SAMPLES; i++) {
  vReal[sampleIndex] = (double)samples[i]; // Store amplitude
  vImag[sampleIndex] = 0.0;                 // Imaginary part = 0
  sampleIndex++;
}
```

Figure 17 - Storing audio samples in an array

When the buffer accumulates a specified number of samples (for example, 256 values), the spectral analysis procedure is launched. Before applying the FFT, the Hamming window function is applied to the data, which allows reducing spectral distortions at the sample boundaries. Then the Fourier transform itself is performed, as a result of which the vReal[] array contains the amplitude spectrum of the input signal. This is done using a sequence of method calls: windowing(), compute() and complexToMagnitude().

```
FFT.windowing(FFT_WIN_TYP_HAMMING, FFT_FORWARD); // Apply Hamming window
FFT.compute(FFT_FORWARD);                        // Compute FFT
FFT.complexToMagnitude();                        // Compute magnitudes
```

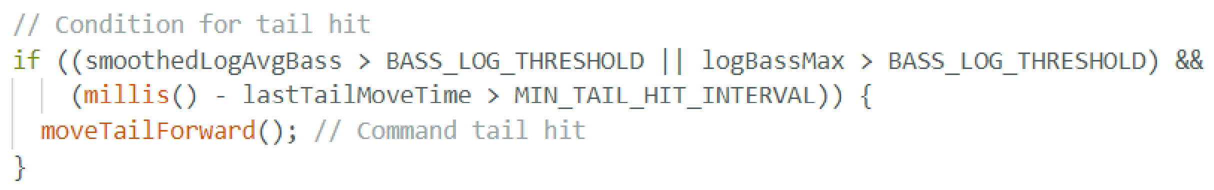
Figure 18 - FFT processing

The obtained values represent the intensity of various frequency components. This data is then used to isolate sound components - in particular, low frequencies (bass) and the range of vocal frequencies (vocals).

```
// Calculate average and max values in bass range
for (int i = bassIndexStart; i <= bassIndexEnd; i++) {
  bassSum += vReal[i];
  if (vReal[i] > bassMax) bassMax = vReal[i];
}

double avgBass = bassSum / (bassIndexEnd - bassIndexStart + 1);
double logAvgBass = log10(avgBass + 1);     // Logarithmic scale
double logBassMax = log10(bassMax + 1);

// Smooth values
smoothedLogAvgBass = (SMOOTHING_FACTOR * logAvgBass) + ((1 - SMOOTHING_FACTOR) * smoothedLogAvgBass);
```

Figure 19 - Isolating sound components

Depending on the energy in the corresponding frequency ranges, a decision is made about which DC motor should move and in which direction.



```
// Condition for tail hit
if ((smoothedLogAvgBass > BASS_LOG_THRESHOLD || logBassMax > BASS_LOG_THRESHOLD) &&
    (millis() - lastTailMoveTime > MIN_TAIL_HIT_INTERVAL)) {
  moveTailForward(); // Command tail hit
}
```

Figure 20 - Decision making

Thus, the audioDataCallback function plays the role of a real-time audio processing module: it receives a digital audio stream, converts it into a numerical form, performs a conversion into the frequency domain and transmits the result to the analyzer, which determines the behavior of the drives.

There is a flowchart for audioDataCallback function which describes the sequence of all processes which process in the function:

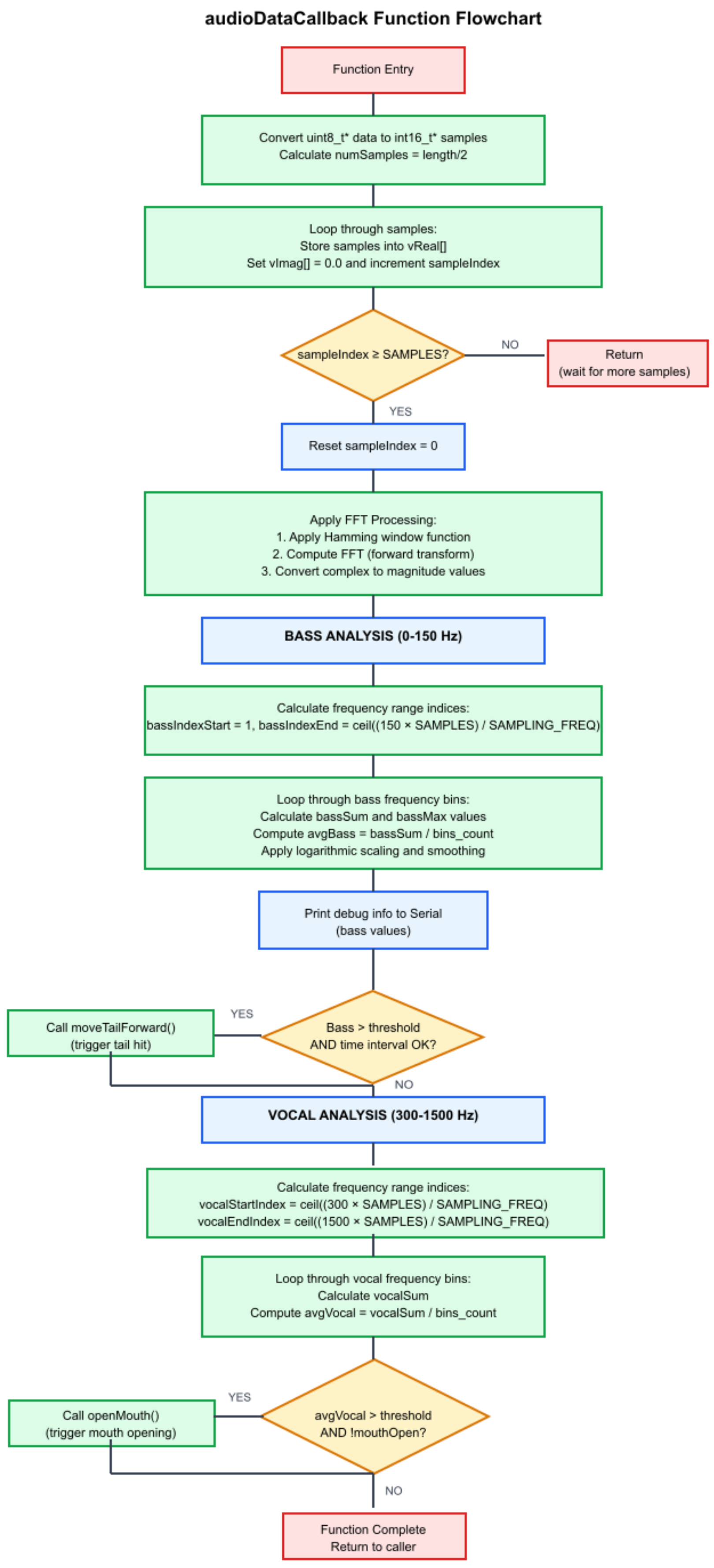
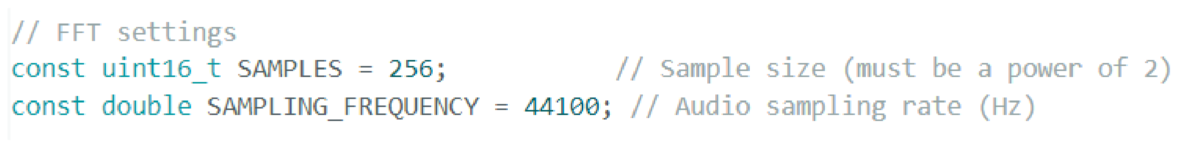

audioDataCallback Function Flowchart
Function Entry
Convert uint8_t* data to int16_t* samples
Calculate numSamples = length/2
Loop through samples:
Store samples into vReal[]
Set vImag[] = 0.0 and increment sampleIndex
sampleIndex ≥ SAMPLES?
NO
Return
(wait for more samples)
YES
Reset sampleIndex = 0
Apply FFT Processing:
1. Apply Hamming window function
2. Compute FFT (forward transform)
3. Convert complex to magnitude values
BASS ANALYSIS (0-150 Hz)
Calculate frequency range indices:
bassIndexStart = 1, bassIndexEnd = ceil((150 × SAMPLES) / SAMPLING_FREQ)
Loop through bass frequency bins:
Calculate bassSum and bassMax values
Compute avgBass = bassSum / bins_count
Apply logarithmic scaling and smoothing
Print debug info to Serial
(bass values)
Bass > threshold
AND time interval OK?
YES
Call moveTailForward()
(trigger tail hit)
NO
VOCAL ANALYSIS (300-1500 Hz)
Calculate frequency range indices:
vocalStartIndex = ceil((300 × SAMPLES) / SAMPLING_FREQ)
vocalEndIndex = ceil((1500 × SAMPLES) / SAMPLING_FREQ)
Loop through vocal frequency bins:
Calculate vocalSum
Compute avgVocal = vocalSum / bins_count
avgVocal > threshold
AND !mouthOpen?
YES
Call openMouth()
(trigger mouth opening)
NO
Function Complete
Return to caller

Figure 21 - audioDataCallback function flowchart

2.5 FFT Algorithm Implementation

This project uses a buffer size of 256 audio samples. With a sampling rate of 44100 Hz, there is 1 sample 44100 times per second, so 5.8ms audio per FFT window, which is enough to for rough definition of vocals or bass.



```
// FFT settings
const uint16_t SAMPLES = 256;             // Sample size (must be a power of 2)
const double SAMPLING_FREQUENCY = 44100; // Audio sampling rate (Hz)
```

Figure 22 - Key FFT settings

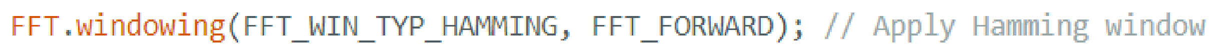
This number of samples and sampling frequency were chosen based on the power of the microcontroller, which at 256 samples provides low latency and the ability to roughly calculate bass and vocals. Increasing the number of samples, for example, to 512 naturally increases the resolution and accuracy of audio analysis, but has an extremely negative effect on the device performance, which is critical in the context of a microcontrollers and real-time sound playback.

It is also worth noting that the popular FTT processing library such as ArduinoFFT is optimized for 2^n sizes, where 256 is one of the well-tested sizes.

In addition, other sample sizes were tested, namely 128 and 512. In the first case, exceptional performance was achieved, but the accuracy of the analysis was lost, false positives occurred, or there was no response at all on the required segments. In the second case, the accuracy of the analysis was excellent, but when playing audio and generating movements, critical delays of more than 1000 ms occurred, which is critical for this type of project.

The frequency of 44100 Hz was chosen due to the widespread use of this frequency as the standard frequency of the A2DPBluetoothAudio library, as well as the fact that this frequency is a standard in audio CD and provides good sound quality for most sources (smartphones, laptops).

It is important to note that this project uses the Hamming window function, the use of which is appropriate in the conditions of our project due to the sampling frequency of the segmented audio, which leads to the creation of many segments with “cut” edges. The use of this function is justified due to its ability to increase the accuracy of the localization frequencies and also to avoid false detections, generally increasing the accuracy of the analysis and increasing the probability of correct operation of the logic and control of the electric motor.



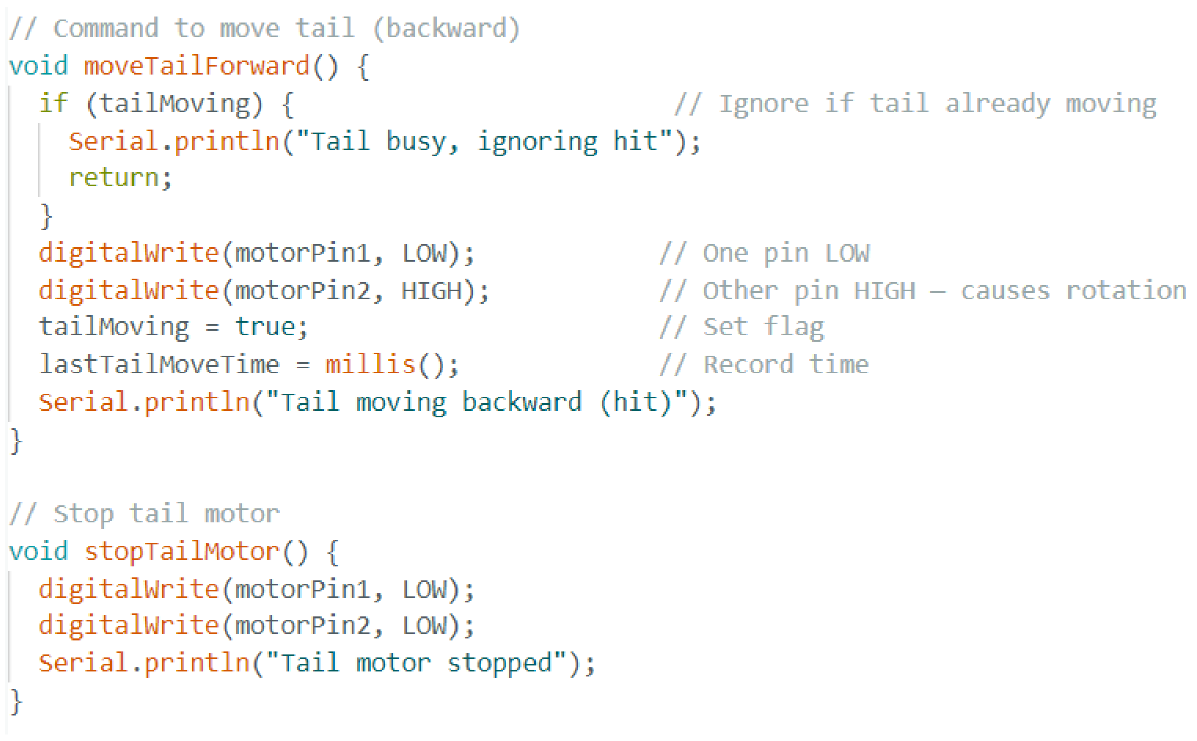
```
FFT.windowing(FFT_WIN_TYP_HAMMING, FFT_FORWARD); // Apply Hamming window
```

Figure 23 - Hamming windowing

2.6 Motor Control Logic

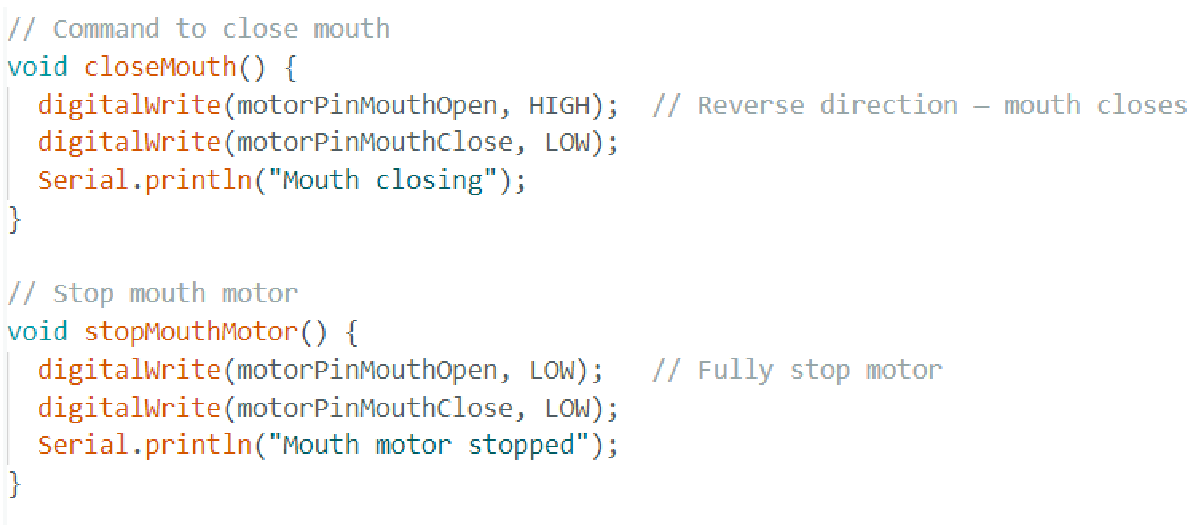
The device uses two DC motors that can move in both directions. These motors are controlled by a dual-channel DRV8833 motor driver, which controls only the direction of rotation and the fact of rotation. The first DC motor is responsible for the movement of the head and tail as follows: when the motor moves forward, the head extends; when the motor moves backward, the tail extends; when the motor power is turned off, these elements are returned to their original position by springs that return the parts to their original positions when there is no power supply to the motors. The second DC motor controls the movement of the mouth: forward rotation opens the mouth and backward closes it. It is worth noting that the mouth transmission design also uses a spring that, by analogy with the head motor, returns it to its original position. However, it is important to say that the frequency of use of the mouth is much higher than that of the tail, so there is a need to quickly open and close the mouth, so the mouth transmission is not involved, giving preference to manually control the direction of movement of the motors to imitate speech.

To implement the movement of electric motors, several functions were written that supply voltage to electric motors or de-energize them. Also, these functions contain flags that eliminate the possibility of chattering or repeated operation of motors if they are already activated.



```
// Command to move tail (backward)
void moveTailForward() {
  if (tailMoving) {                          // Ignore if tail already moving
    Serial.println("Tail busy, ignoring hit");
    return;
  }
  digitalWrite(motorPin1, LOW);            // One pin LOW
  digitalWrite(motorPin2, HIGH);           // Other pin HIGH – causes rotation
  tailMoving = true;                       // Set flag
  lastTailMoveTime = millis();             // Record time
  Serial.println("Tail moving backward (hit)");
}

// Stop tail motor
void stopTailMotor() {
  digitalWrite(motorPin1, LOW);
  digitalWrite(motorPin2, LOW);
  Serial.println("Tail motor stopped");
}
```

Figure 24 - Tail motor control function



```
// Command to close mouth
void closeMouth() {
  digitalWrite(motorPinMouthOpen, HIGH);  // Reverse direction – mouth closes
  digitalWrite(motorPinMouthClose, LOW);
  Serial.println("Mouth closing");
}

// Stop mouth motor
void stopMouthMotor() {
  digitalWrite(motorPinMouthOpen, LOW);   // Fully stop motor
  digitalWrite(motorPinMouthClose, LOW);
  Serial.println("Mouth motor stopped");
}
```

Figure 25 - Mouth motor control function

Since the project does not feature servo motors but conventional DC motors and transmissions with springs, the movement is controlled exclusively by the time of voltage supply to these DC motors. It is worth notice that the time during which one or another part of the device reaches its physical stop and is unable to move further (for example, from the extreme initial position of the tail to the extreme extended position of the tail), thus the timings of the movement of all parts of the device were experimentally established.

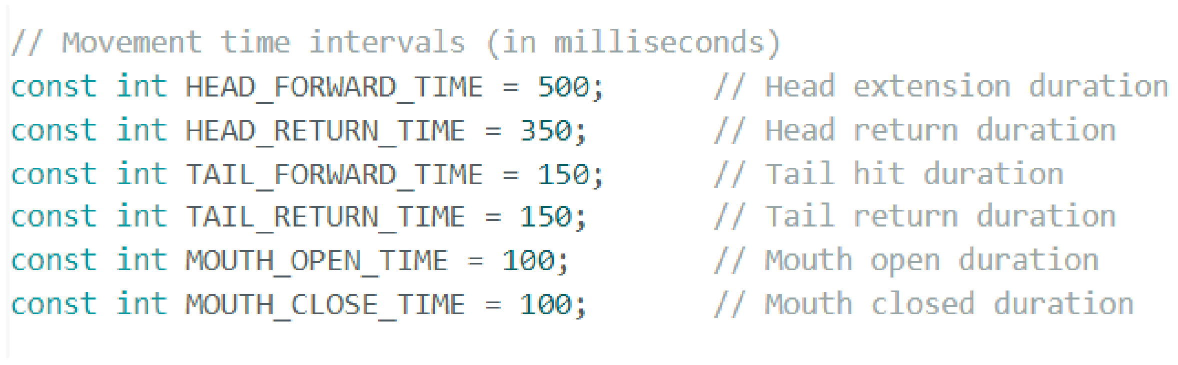
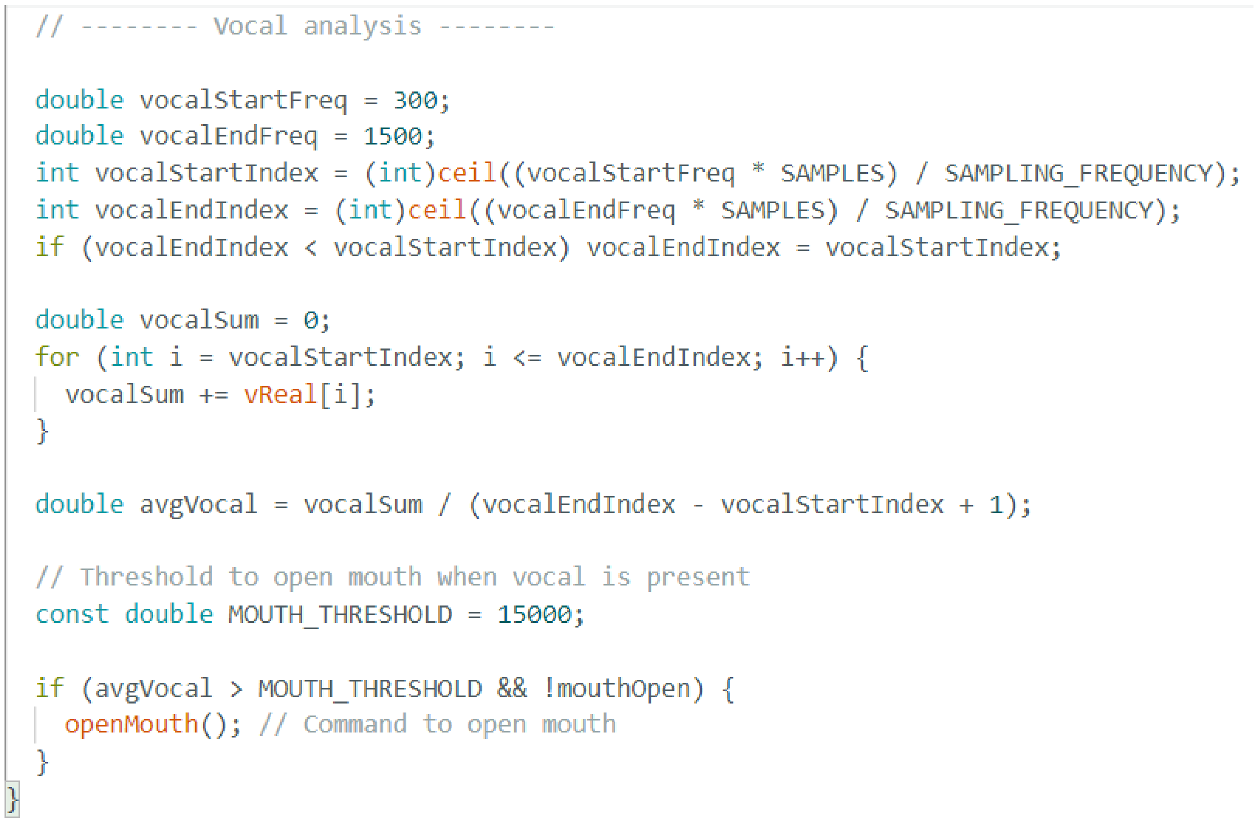

```
// Movement time intervals (in milliseconds)
const int HEAD_FORWARD_TIME = 500;      // Head extension duration
const int HEAD_RETURN_TIME = 350;       // Head return duration
const int TAIL_FORWARD_TIME = 150;      // Tail hit duration
const int TAIL_RETURN_TIME = 150;       // Tail return duration
const int MOUTH_OPEN_TIME = 100;        // Mouth open duration
const int MOUTH_CLOSE_TIME = 100;       // Mouth closed duration
```

Figure 26 - Movement time intervals

An important point in the logic of the electric motors' behavior is the fact that two motors can work simultaneously, providing simultaneous movement of both the head and tail to create a greater interactive effect.

2.7 Audio Analysis and Decision Making

One of the most important parts of the program is the logic of controlling the movements of electric motors, which is based on the analysis of the audio signal using FFT. The sound received via Bluetooth is processed by the FFT algorithm, after which the program determines whether it is necessary to activate a particular electric motor and in which direction.

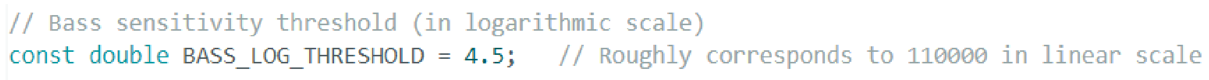


```
  // -------- Vocal analysis --------

  double vocalStartFreq = 300;
  double vocalEndFreq = 1500;
  int vocalStartIndex = (int)ceil((vocalStartFreq * SAMPLES) / SAMPLING_FREQUENCY);
  int vocalEndIndex = (int)ceil((vocalEndFreq * SAMPLES) / SAMPLING_FREQUENCY);
  if (vocalEndIndex < vocalStartIndex) vocalEndIndex = vocalStartIndex;

  double vocalSum = 0;
  for (int i = vocalStartIndex; i <= vocalEndIndex; i++) {
    vocalSum += vReal[i];
  }

  double avgVocal = vocalSum / (vocalEndIndex - vocalStartIndex + 1);

  // Threshold to open mouth when vocal is present
  const double MOUTH_THRESHOLD = 15000;

  if (avgVocal > MOUTH_THRESHOLD && !mouthOpen) {
    openMouth(); // Command to open mouth
  }
}
```

Figure 27 - Vocals analysis

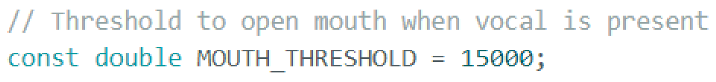
Vocal frequency analysis covers the main frequencies that a human uses when singing or speaking (300-1500 Hz). It is used to control the mouth DC motor that imitates speech or singing. Another DC motor, which controls the tail activates with bass (0-150 Hz). The algorithm tracks the amplitude and uses the Peak Detection Algorithm so that the motors are activated only when there are clear signs of vocals. This approach eliminates chatter and monotonous opening of the mouth in the presence of any sound, which has a positive effect on the visual component of the device.

The thresholds BASS_THRESHOLD and VOCAL_THRESHOLD are determined empirically, taking into account many tests on different genres of music. Since these thresholds are determined manually by experimentation, they provide a balance between the sensitivity of the algorithm and the resistance to false detections.



```
// Bass sensitivity threshold (in logarithmic scale)
const double BASS_LOG_THRESHOLD = 4.5;    // Roughly corresponds to 110000 in linear scale
```

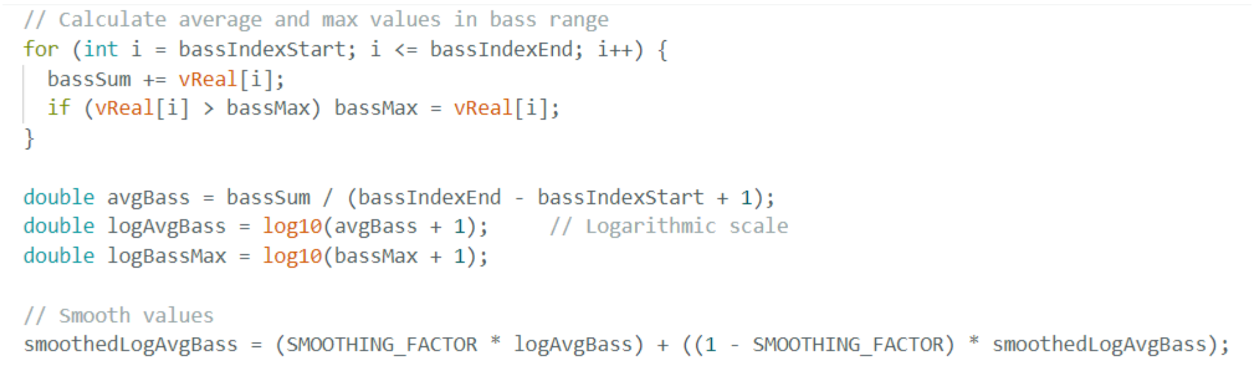
Figure 28 - Bass threshold



```
// Threshold to open mouth when vocal is present
const double MOUTH_THRESHOLD = 15000;
```

Figure 29 - vocals threshold

To improve the stability of audio analysis, methods such as Moving Averages and Peak Detection are used, which smooth out amplitudes by selecting sharp jumps and also allow one to identify sharp and sudden sound events such as the beginning of a word.

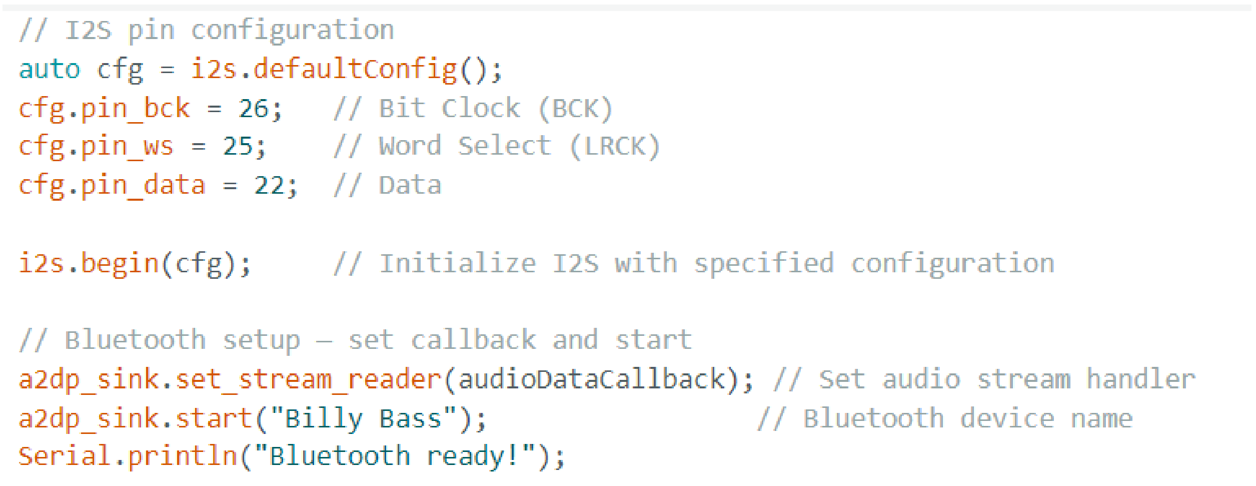


```
// Calculate average and max values in bass range
for (int i = bassIndexStart; i <= bassIndexEnd; i++) {
  bassSum += vReal[i];
  if (vReal[i] > bassMax) bassMax = vReal[i];
}

double avgBass = bassSum / (bassIndexEnd - bassIndexStart + 1);
double logAvgBass = log10(avgBass + 1);     // Logarithmic scale
double logBassMax = log10(bassMax + 1);

// Smooth values
smoothedLogAvgBass = (SMOOTHING_FACTOR * logAvgBass) + ((1 - SMOOTHING_FACTOR) * smoothedLogAvgBass);
```

Figure 30 - Improving the audio analysis

2.8 Bluetooth Integration

There is a BluetoothA2DPSink class is used to receive an audio stream via Bluetooth in the project. This allows the ESP32 microcontroller to act as a receiver for a Bluetooth signal transmitted from a mobile phone or other source.



```
// I2S pin configuration
auto cfg = i2s.defaultConfig();
cfg.pin_bck = 26;    // Bit Clock (BCK)
cfg.pin_ws = 25;     // Word Select (LRCK)
cfg.pin_data = 22;   // Data

i2s.begin(cfg);      // Initialize I2S with specified configuration

// Bluetooth setup — set callback and start
a2dp_sink.set_stream_reader(audioDataCallback); // Set audio stream handler
a2dp_sink.start("Billy Bass");                  // Bluetooth device name
Serial.println("Bluetooth ready!");
```

Figure 31 - I2S integration

First, an I2SStream stream is created, configured to receive an audio signal with certain pins (GPIO22, GPIO25, GPIO26). Then the BluetoothA2DPSink object is connected to this stream, gaining the ability to receive downstream via Bluetooth. After this, the **start** method is called on the BluetoothA2DPSink object, which turns on the Bluetooth module and sets the device name (“Billy Bass”) visible to other devices. Thus, the device is ready to receive an audio stream via Bluetooth, To handle the connection of Bluetooth devices and the audio stream, a callback function is assigned, which is called when audio data is received:

```
a2dp_sink.set_stream_reader(audioDataCallback); // Set audio stream handler
```

Figure 32 - Setting audio stream handler

The audioDataCallback function receives streaming data, converts it into 16-bit audio samples and saves it in a buffer for further FFT analysis.

2.9 System Timing and Performance

One of the most important requirements for the system is to minimize delays between receiving an audio signal via Bluetooth, analyzing and playing the audio signal, and also the reaction of the DC motors. Among all this, the main sources of delays can be identified: audio transmission via Bluetooth, which also includes buffering of audio data, audio processing, which includes computational costs for FFT and frequency analysis, as well as the delay associated with the supply of voltage or disconnection of electric motors.

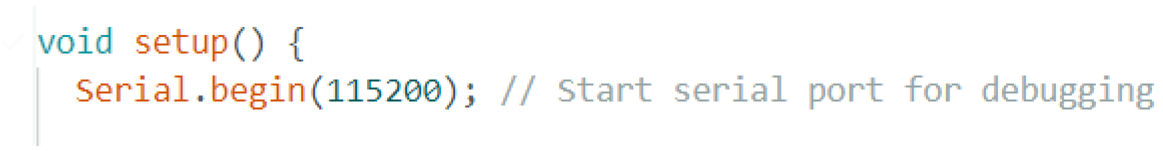
It is worth notice that as a solution to reduce delays, small sample sizes (SAMPLES = 256) are used in the code, which allows obtaining FFT results quickly and with good accuracy. To maintain smooth operation and timely analysis of audio data, buffering of incoming audio samples is implemented. The use of smoothing (smoothedLogAvgBass) allows avoiding sharp jumps and false positives, which reduces the load on motor control and eliminates unnecessary commands. All processing is performed in the callback function, which is called upon receipt of audio data, which minimizes delays and guarantees a prompt response. Also, static arrays of a fixed size are allocated for working with audio and FFT, which eliminates dynamic memory allocation during operation and minimizes memory fragmentation.

```
// Smooth values
smoothedLogAvgBass = (SMOOTHING_FACTOR * logAvgBass) + ((1 - SMOOTHING_FACTOR) * smoothedLogAvgBass);
```

Figure 33 - Smoothing operation

2.10 Testing and Debugging

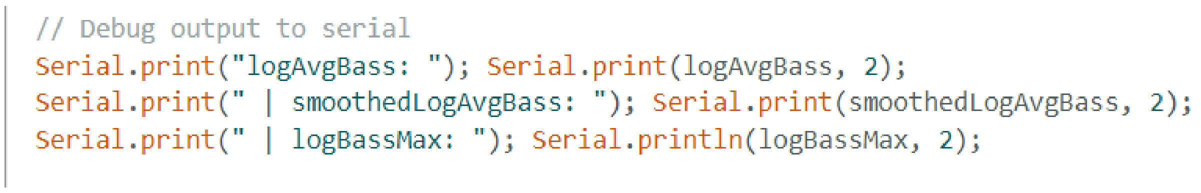
During the final debugging and manual adjustment of the bass and vocal thresholds, the Arduino IDE Serial Monitor console was mainly used.
The initialization of the Arduino IDE Serial Monitor placed in the very beginning of the setup section.



```
void setup() {
  Serial.begin(115200); // Start serial port for debugging
```

Figure 34 - Serial monitor setup

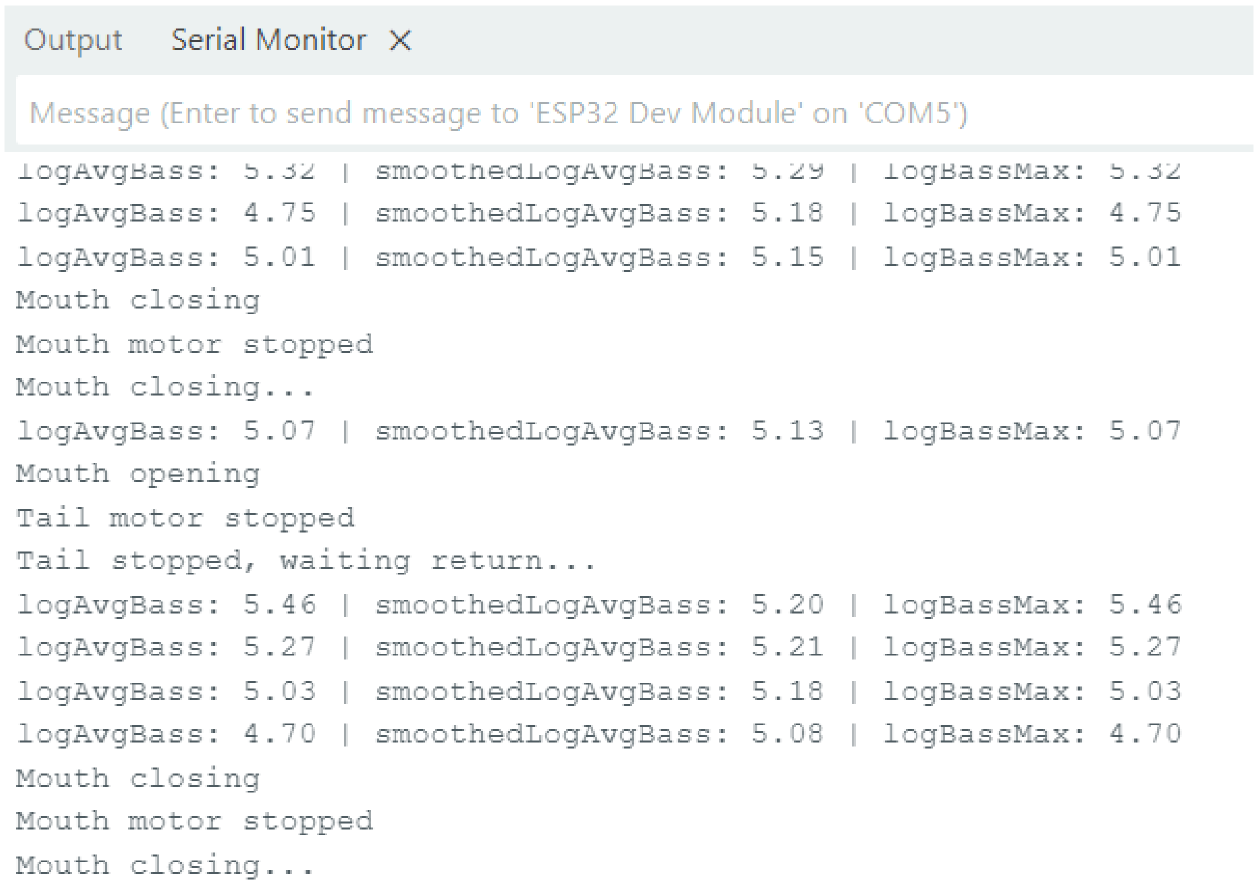
An output was deeply involved into the final adjustment of the system; running in the following way:



```
// Debug output to serial
Serial.print("logAvgBass: "); Serial.print(logAvgBass, 2);
Serial.print(" | smoothedLogAvgBass: "); Serial.print(smoothedLogAvgBass, 2);
Serial.print(" | logBassMax: "); Serial.println(logBassMax, 2);
```

Figure 35 - Debug information output

It took a main part into final calibration and debugging of the system.



Output
Serial Monitor ×
Message (Enter to send message to 'ESP32 Dev Module' on 'COM5')

```
logAvgBass: 5.32 | smoothedLogAvgBass: 5.29 | logBassMax: 5.32
logAvgBass: 4.75 | smoothedLogAvgBass: 5.18 | logBassMax: 4.75
logAvgBass: 5.01 | smoothedLogAvgBass: 5.15 | logBassMax: 5.01
Mouth closing
Mouth motor stopped
Mouth closing...
logAvgBass: 5.07 | smoothedLogAvgBass: 5.13 | logBassMax: 5.07
Mouth opening
Tail motor stopped
Tail stopped, waiting return...
logAvgBass: 5.46 | smoothedLogAvgBass: 5.20 | logBassMax: 5.46
logAvgBass: 5.27 | smoothedLogAvgBass: 5.21 | logBassMax: 5.27
logAvgBass: 5.03 | smoothedLogAvgBass: 5.18 | logBassMax: 5.03
logAvgBass: 4.70 | smoothedLogAvgBass: 5.08 | logBassMax: 4.70
Mouth closing
Mouth motor stopped
Mouth closing...
```

Figure 36 - Serial monitor output

Using the console output made it possible to solve such problems as incorrectly working conditions for starting electric motors, as well as to correct too high or low sensitivity to frequencies that were selected manually by testing. Also, various audio signals were tested, for example, tracks with dominant bass or compositions that were rich in vocals. Thus, testing and debugging made it possible to calibrate the algorithm so that all parts of the mechanism worked correctly without delays and without false alarms. It was also possible to identify musical styles that are most suitable for adaptation to this device due to the limited computer-mechanical capabilities.

Conclusion

The result of this project was the modernization of an interactive toy based on the ESP32 microcontroller. The main goal of the project was to create a physical device and develop software code with functions that allow audio playback via Bluetooth, as well as audio stream analysis and synchronization of the toy's movements with the musical rhythm in real time.

To achieve this, the following tasks were completed: audio transmission via the A2DP protocol; implementation of an audio stream analysis algorithm using FFT to extract key parameters from the audio; development, testing, and calibration of algorithms for controlling electric motors to drive the toy’s mechanisms; and synchronization of all software components under a unified principle to ensure an interactive result.

In the process of software development, the main attention was paid to the performance and memory management of the microcontroller due to the limitations inherent in the principle of embedded systems. The resulting system demonstrates stable operation, ensuring the implementation of all key tasks and requirements of the project.

In the process of debugging and testing various musical styles, the following patterns were identified: the best synchronization is shown when playing hip-hop and pop music due to the clear rhythm, strong bass and well-defined vocals. It is also worth noting that such styles as classical music, jazz and metal are not suitable for playback on this device due to the lack of a clear rhythm and the overload of frequencies, which makes it difficult to determine the rhythm and synchronize movements.
The completed system serves as a successful demonstration of how modern embedded technologies—specifically real-time Bluetooth audio processing and frequency domain analysis—can transform legacy consumer electronics into

adaptive, interactive smart devices. This work highlights not only the feasibility but also the potential of integrating signal analysis and actuator control on resource-constrained hardware platforms, paving the way for more accessible and innovative user interfaces in embedded products.

In the future, it would be possible to enhance the system with more complex logic that reacts not only to rhythm and volume, but also to specific voice patterns or music styles. This could further improve how well the toy imitates singing or responds to different genres. Overall, the project demonstrates how even a basic microcontroller, when used efficiently, can bring simple objects to life and create a much more engaging and personalized user experience.

References

Espressif Systems. *ESP32 Technical Reference Manual* [Electronic resource]. — Available at: https://www.espressif.com/en/support/documents/technical-documents

Espressif Systems. *ESP-IDF Programming Guide*. — https://docs.espressif.com/projects/esp-idf/en/latest/

Bluetooth SIG. *Bluetooth Core Specification v5.0*. — https://www.bluetooth.com/specifications/specs

Maksim Prasolov. *ESP32-A2DP: A2DP Sink and Source library for ESP32* [GitHub repository]. — https://github.com/pschatzmann/ESP32-A2DP

Makuna. *FFT Library for Arduino and ESP32* [GitHub repository]. — https://github.com/kosme/arduinoFFT

Adafruit. *Adafruit MAX98357 I2S Amp Guide* [Electronic resource]. — https://learn.adafruit.com/adafruit-max98357-i2s-class-d-mono-amp

Random Tutorials. *ESP32 with I2S – Playing Audio Using MAX98357A*. — https://randomnerdtutorials.com/esp32-i2s-max98357a-audio/