**🧠 Talking Skull Project (ESPHome + Audio-Reactive Jaw)**

**Interactive Halloween Skull with Moving Jaw Synced to TTS or Audio**

This project brings the classic “Talking Skull” prop to life using **ESP32 boards**, **ESPHome**, and **audio-reactive servo control**.  
It’s based on the original **JawDuino** concept by Mike (Arduino + KA2284 LED sound meter) and has been **modernized** for ESPHome, allowing wireless control, TTS integration, and synchronization with Home Assistant.

**🧩 Overview**

Two ESP32 devices work together:

| **Device** | **Role** | **Description** |
| --- | --- | --- |
| **ESP32 #1 (Audio)** | Audio/TTS playback | Plays TTS or MP3 files using a MAX98357A I²S amplifier. |
| **ESP32 #2 (Jaw Control)** | Servo motion | Moves the skull’s jaw in sync with sound amplitude captured from the audio line. |

Audio output and control signals are shared between them through a safe resistor divider.

**📁 Repository Structure**

/TalkingSkull/

│

├── JawDuino.ino # Original Arduino reference

├── SoftRcPulseOut.cpp/.h # Custom servo pulse library (used by JawDuino)

├── jawduino.pdf # Original build notes and wiring

│

├── esphome-web-3a75c8.yaml # ESPHome Skull TTS (Audio ESP32)

├── esphome-testdevice.yaml # ESPHome Jaw Servo (Motion ESP32)

│

└── README.md # This documentation

**⚙️ YAML File Descriptions**

**🎙️ esphome-web-3a75c8.yaml — Audio/TTS Controller**

Handles text-to-speech playback and streams audio to the MAX98357A amplifier.

**Key Features:**

* I²S audio output via GPIO26 (BCLK), GPIO27 (LRCLK), GPIO25 (DIN)
* Works as a Home Assistant media\_player
* Accepts local MP3/WAV playback or HA TTS commands
* Syncs audio level signal to the jaw controller via resistor divider line

**Relevant ESPHome Components:**

i2s\_audio:

i2s\_bclk\_pin: GPIO26

i2s\_lrclk\_pin: GPIO27

media\_player:

platform: i2s\_audio

name: "Talking Skull Speaker"

**Home Assistant Usage:**

service: media\_player.play\_media

target:

entity\_id: media\_player.talking\_skull\_speaker

data:

media\_content\_id: media-source://media\_source/local/phrase.mp3

media\_content\_type: music

**🦴 esphome-testdevice.yaml — Jaw Servo Controller**

Controls the jaw servo based on the incoming audio amplitude.

**Key Features:**

* Reads analog input (ADC) corresponding to audio amplitude
* Smoothly moves the jaw servo proportionally to loudness
* Allows testing and calibration modes through switches
* Integrates with Home Assistant for manual control and diagnostics

**Core Logic:**

servo:

id: jaw\_servo

output: jaw\_pwm

output:

- platform: ledc

id: jaw\_pwm

pin: GPIO23

frequency: 50Hz

sensor:

- platform: adc

pin: GPIO34

id: jaw\_audio\_level

update\_interval: 20ms

filters:

- sliding\_window\_moving\_average:

window\_size: 10

send\_every: 5

**Lambda Example:**

on\_value:

then:

- lambda: |-

float val = id(jaw\_audio\_level).state;

int angle = map(val \* 1000, 0, 800, 0, 100);

id(jaw\_servo).write(angle);

**🧰 Custom Component: SoftRcPulseOut**

Originally written by **RC Navy**, this library provides **software-based servo pulse control** with reduced jitter — an alternative to the standard Servo library.  
While the ESPHome version uses hardware PWM, the original Arduino version uses this for smooth analog servo movement.

**Features:**

* Microsecond precision for servo pulse width
* Handles multiple software-generated PWM outputs
* Compatible with low-resource microcontrollers like ATtiny85

**🔌 Wiring Diagram**

**ESP32 Audio (TTS / I²S Amplifier)**

| **Pin** | **Connection** | **Description** |
| --- | --- | --- |
| GPIO26 | BCLK → MAX98357A BCLK | I²S Bit Clock |
| GPIO27 | LRCLK → MAX98357A LRC | I²S Left/Right Clock |
| GPIO25 | DIN → MAX98357A DIN | Audio Data |
| 5V | → MAX98357A VIN | Power |
| GND | → MAX98357A GND | Ground |

**ESP32 Jaw Servo Controller**

| **Pin** | **Connection** | **Description** |
| --- | --- | --- |
| GPIO23 | → Servo Signal | PWM control |
| 5V | → Servo Power | Shared 5V Regulated |
| GND | → Common GND | Shared Ground |
| GPIO34 | → Audio Level Input | From Audio ESP32 via resistor divider |

**Audio Level Interconnect**

| **Signal** | **Connection** | **Note** |
| --- | --- | --- |
| Audio Out (Right/Left) | → 10kΩ →→ ADC Input (GPIO34) | via 2-resistor divider (e.g., 10k/4.7k) |
| GND | → GND | Common ground between both ESP32s |

**🔋 Powering Everything**

Use a **regulated 5V power supply** rated for at least **2A**:

* Servos can draw large transient currents.
* Both ESP32s and MAX98357A share the 5V rail.
* Use separate ground wires for audio and servo power to reduce noise.

If using a USB supply:

* Power Audio ESP32 via USB.
* Power Servo ESP32 via VIN with 5V directly.
* Ensure grounds are common.

**🧠 Calibration & Tuning**

1. **Start with the jaw servo disconnected.**
2. Use the ESPHome dashboard to monitor jaw\_audio\_level.
3. Adjust the resistor divider or amplifier output for a dynamic range between **0.1V and 1.5V**.
4. Connect the jaw servo and fine-tune the servo movement range:
   * Modify servo min\_angle / max\_angle in YAML.
   * Optionally create switches for **Jaw 25%, 50%, 75%, 100% test moves**.
5. Verify the jaw moves smoothly with speech and rests quietly between words.

**🧩 Integrations**

| **Integration** | **Purpose** |
| --- | --- |
| **Home Assistant** | Send TTS or MP3 files for playback |
| **ESPHome API** | Wireless control and synchronization |
| **Node-RED (optional)** | Create reactive flows for speech and motion |
| **Alexa Media Player** | Play TTS through the skull |

**⚠️ Troubleshooting**

| **Issue** | **Cause** | **Solution** |
| --- | --- | --- |
| Scratchy audio | Shared ground loop or missing resistor divider | Add isolation resistors, confirm GND wiring |
| Jaw buzzes at rest | Servo holding torque too high | Add dead-band zone or relax servo after idle |
| No jaw motion | Wrong ADC pin or scaling | Check analog wiring and calibration |
| Servo jitter | Power noise from audio amp | Use separate power filtering capacitors |

**📜 Credits**

* **Original Concept:** Mike’s *JawDuino* ([jawduino.pdf][10])
* **Servo Library:** *SoftRcPulseOut* by RC Navy ([soft.rc.pulseout][11])
* **ESPHome Conversion:** Adapted and modernized by Paul Goldstein
* **Audio Playback:** MAX98357A via I²S Audio integration