Sending Audio Wirelessly: Building the Bluetooth Audio Pipeline

Author Maximilian Ardalan Bernhard

Institutions FH Joanneum, KUG

Course Design & Research 2

Date 2026-08-31

1. Choosing a Wireless Protocol

The two devices need two kinds of data: a continuous audio stream, and a stream of low-rate control commands. The search for a wireless protocol therefore began with this split between the two uses. My first candidate was ESP-NOW, a connectionless protocol from Espressif that runs directly over the WiFi hardware. ESP-NOW is fast and simple, because two devices exchange short packets without holding a connection open. It works well for control data, but it is not designed to stream audio. The protocol has no mechanism for the steady, continuous delivery that a microphone signal requires. So even though ESP-NOW was a plausible choice for the control channel, it could not carry the audio, and I set it aside.

Bluetooth offered both halves on a single radio. The ESP32 implements two complementary Bluetooth protocols: A2DP (Advanced Audio Distribution Profile) for continuous audio streams, and BLE (Bluetooth Low Energy) for low-rate control data. A2DP is the profile behind many Bluetooth headphones, and it provides a fixed streaming transport for the audio. BLE is built for small, occasional messages and low power, which suits the pan and tilt commands. Choosing Bluetooth therefore let one radio carry the audio and the control traffic at the same time.

The corresponding decision in software came down to library research. Phil Schatzmann's ESP32-A2DP library provides both a sink and a source implementation, so the same family of code can run on both the loudspeaker and the microphone side (Schatzmann 2026). The companion arduino-audio-tools library wraps the I2S interface, which turned out to be the cleanest way to move samples in and out of the controllers.

The decisive lesson of this phase was that library compatibility is everything. The libraries initialize the Bluetooth controller, so the order and mode of that initialization decide whether the stacks behave. Incompatible initialization sequences failed silently rather than with a clear error. I therefore had to read the library source code to understand how each one sets up the controller. That discovery and the debugging it triggered is described in depth in a later blogpost, where the two Bluetooth modes are finally made to coexist.

2. Microphone Input and the ESP32 ADC Problem

The audio pipeline begins at the microphone. My first microphone was an old module that my father had lying around, but it proved unusable for this purpose. Its sensitivity was too low and its noise floor too high for the signals this project works with. That limitation is a property of the hardware, so no amount of software could compensate for it. The module had to be replaced.

The replacement was a MAX9814 module, a microphone amplifier with a built-in preamplifier and automatic gain control ("MAX9814" 2020). The built-in preamp brings the weak capsule signal up to a usable level, and the automatic gain control could be used as a hardwired compressor if desired[1]. This solved the input-stage problem, but the path into the microcontroller introduced a new one.

The signal enters the ESP32 through its built-in ADC, which proved to be the weak point. The ESP32′s ADC has a non-linear transfer characteristic that varies from chip to chip ("ESP32 Datasheet, Version 5.3" 2026). Espressif offers a calibration driver to compensate for the offset, the gain and the nonlinearity, but the calibration code I tried did not work on the first attempt. The sound was adequate for the prototype, and the timeline was tight, so I postponed the calibration rather than let it block the project.

The practical workaround was to read the ADC through the I2S interface[2] at a steady sample rate of 44.1 kilohertz, then subtract the DC bias to center the signal. Reading through I2S lets the samples arrive on a stable clock and in a format that fits the audio chain directly. Subtracting the DC bias removes the constant offset so that the alternating sound signal is centered around zero.

Even with this workaround, the built-in ADC is at its limit for audio-grade sampling. An external I2S ADC module remains a possible upgrade for a later phase, if the prototype's sound quality ever needs to go beyond that limit.

[1] The compressor wasn't used in the end though, as it had an undesired ducking effect on the feedback loop.

[2] The I2S-ADC peripheral provides 12-bit samples in 16-bit frames.

3. First Audio Output

The audio path becomes a full loop once the microphone signal reaches the loudspeaker. The two devices take distinct roles in that path:

- The `MicGimbal` acts as the A2DP source: it captures the audio and sends it over Bluetooth.
- The `SpeakerGimbal` acts as the A2DP sink: it receives that audio and drives the loudspeaker.

On the loudspeaker side, the stream is routed over I2S to a MAX98357A, a tiny class-D amplifier with an integrated DAC ("MAX98357A/MAX98357B" 2026). The amplifier receives its samples over the I2S bus and drives the three-watt loudspeaker directly.

The first end-to-end test connected the microphone input straight to the Bluetooth output. There was a clear result: the signal showed noticeable correlation with the captured sound, which meant the path was fundamentally working. But it was also wrong in a characteristic way, because the buffers repeated and the signal distorted whenever the amplitude exceeded the noise floor. The repeated buffers pointed to a timing mismatch rather than a broken signal.

The root cause was that the microphone sample buffer and the Bluetooth frame buffer run at different rates. The ADC fills its buffer at the pace of the sample clock, while the Bluetooth stack sends its frames on its own schedule. Without any buffering between the two, samples were either dropped or repeated, which produced the stutter and the distortion.

The fix was a ring buffer placed between the two timing domains. A ring buffer decouples the sampling side from the transmission side, absorbing the jitter between the ADC read rate and the Bluetooth frame rate. Once the ring buffer was in place, the stutter disappeared and the signal ran smoothly.

Bibliography

"ESP32 Datasheet, Version 5.3." 2026. July. https://documentation.espressif.com/esp32_datasheet_en.pdf.

"MAX9814." 2020. February.

"MAX98357A/MAX98357B." 2026. February.

Schatzmann, Phil. 2026. "ESP32-A2DP.". https://github.com/pschatzmann/ESP32-A2DP.