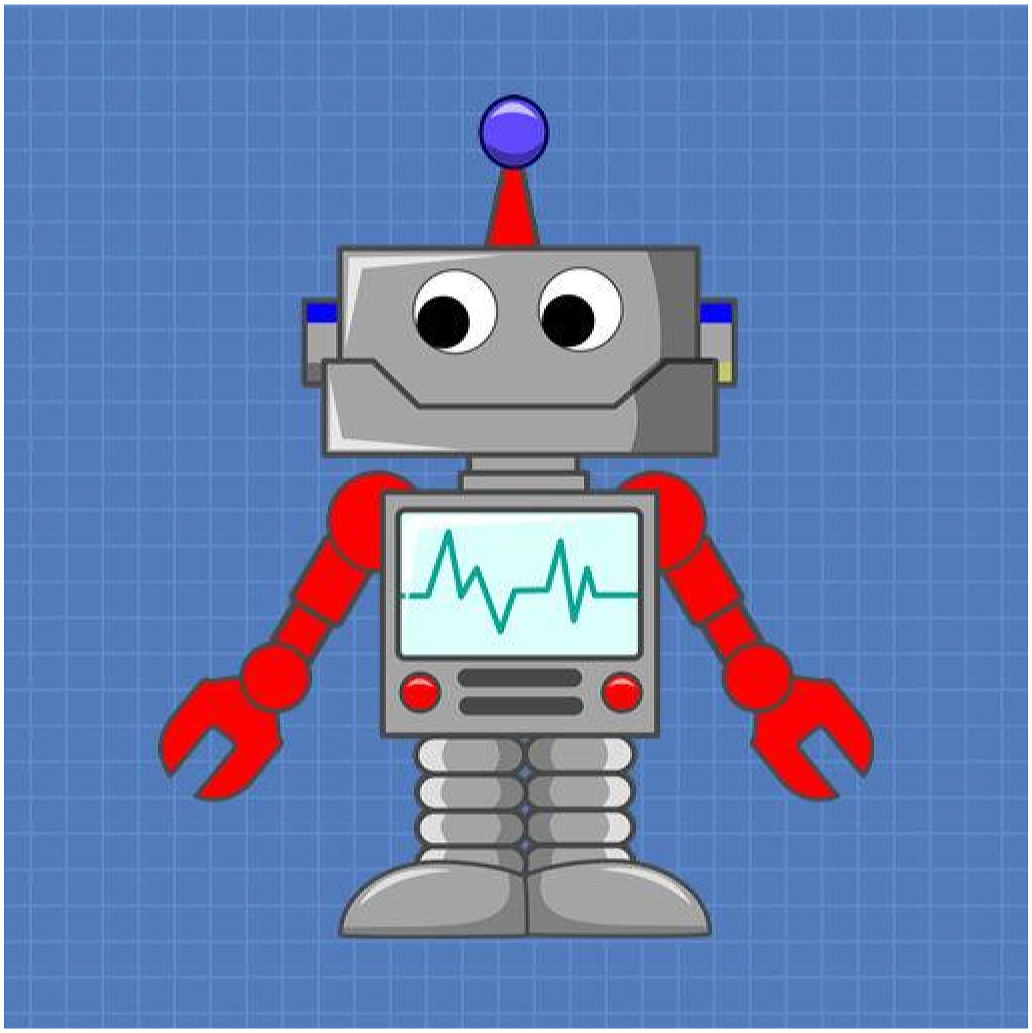
Sound with ESP32 – I2S Protocol



DroneBot Workshop Tutorial

https://dronebotworkshop.com

Audio meets ESP32 today, as we examine the I2S protocol for digital audio.

Introduction

If you are browsing through the specification sheet for an ESP32 device, you might run across the term “I2S”. At a glance, you may just think that it’s another form of I2C, and the “I2” does indeed stand for the same thing, “Inter-Integrated Circuit”. But that’s where the similarity ends.

I2S is a protocol for transferring digital audio. The audio quality can range from telephone-grade to ultra-high fidelity, and you can have one or two channels.

Today we will be exploring the use of I2S with the ESP32, and we'll build a few projects that use the I2S protocol.

Let's get going!

I2S & Digital Audio

The Inter-Integrated Circuit Sound Protocol, or I2S, was developed by Phillips Semiconductors in 1986. As you might recall, Phillips also developed the I2C protocol, and both protocols were built to serve a similar need.

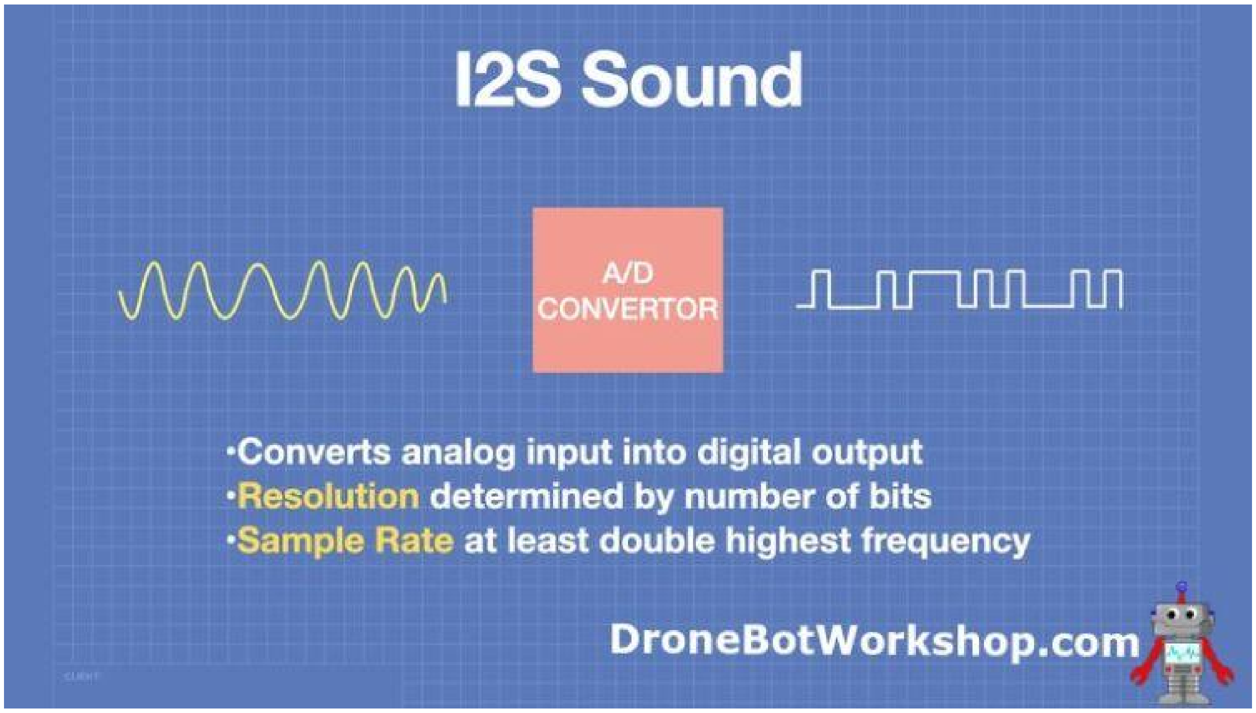
Both I2C and I2S addressed the need for compatibility between integrated circuits that handled data and sound information. Standardized protocols for transferring data and sound would allow designs using ICs from different manufacturers, which is a good thing for everybody.

In order to understand I2S, it's a good idea to also understand how digital audio works. Here's a quick refresher in case you aren't familiar with the concept, if you are then feel free to skip it!

Digital Audio

Sound by its own nature is analog, and, prior to the development of digital sound, audio equipment was also analog. The vibrations of sound on a transducer like a microphone can be amplified and then sent to a speaker, whose cone reproduces those vibrations.

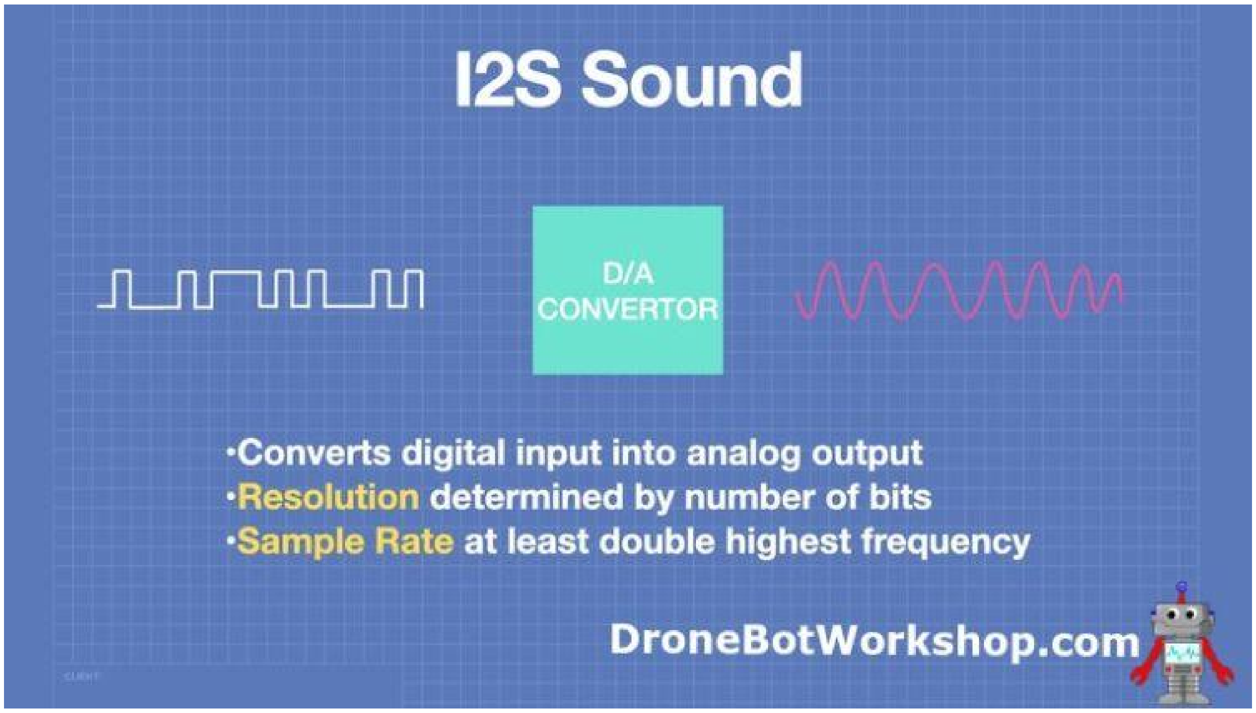
Getting the sound from the microphone to the speaker, especially if you wanted to record it and play it back on the speaker later, involved a lot of analog electronics. Even the best analog electronics will induce electrical noise and distortion in the signal, although modern designs are impressively clean.



I2S Sound

- Converts analog input into digital output
- Resolution determined by number of bits
- Sample Rate at least double highest frequency

Digital audio was touted as a way to eliminate those distortions of the sound. With digital audio, the sound is sent to an analog to digital converter (ADC) to create a digital representation of it. This can then be stored or transmitted without any degradation. On the other end, the digital signal goes through an equivalent digital to analog converter (DAC), which recreates the analog input and is then amplified to drive a speaker.



•Converts digital input into analog output
•Resolution determined by number of bits
•Sample Rate at least double highest frequency

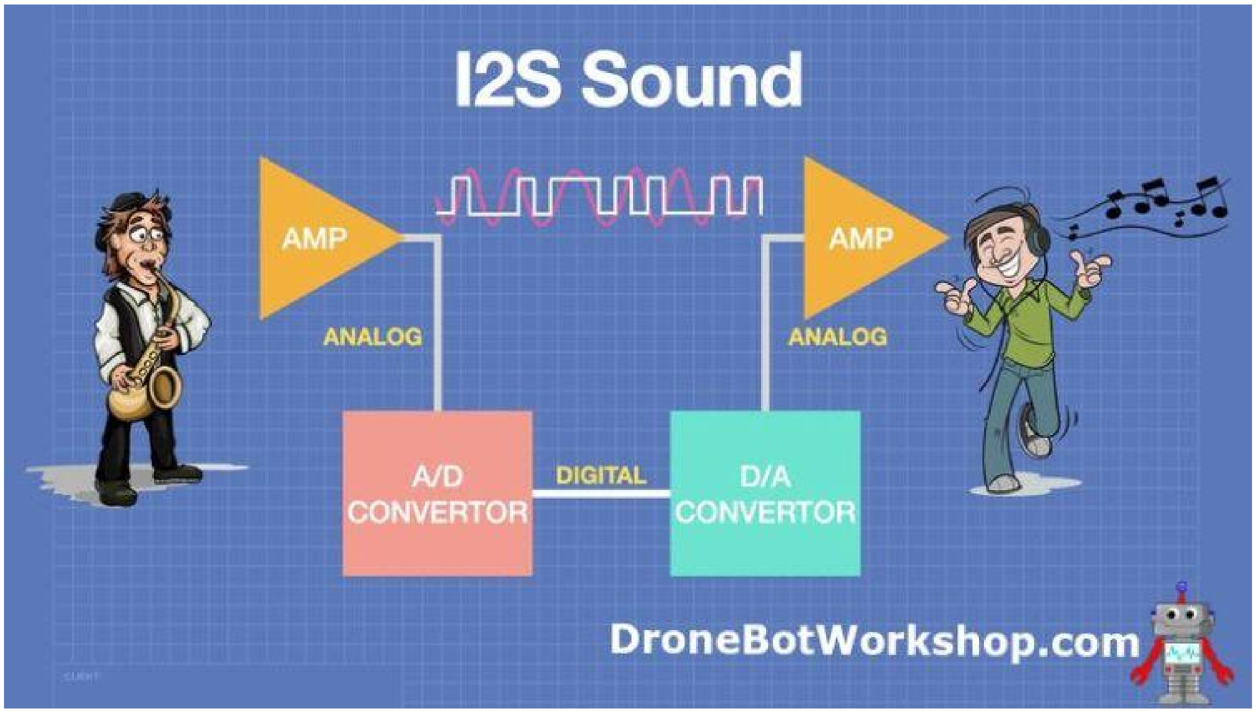
A digital audio signal is not, of course, a perfect representation of the original signal. The quality of the signal is dependent upon two parameters that apply to both the ADC and DAC:

- **Resolution** – The number of bits used in the sample. More bits equal better quality.
- **Sample Rate** – How many samples per second are we taking. This needs to be at least twice as high as the highest frequency we want to sample.

Obviously, the higher the resolutions and sample rate, the larger the resulting digital data file will be.

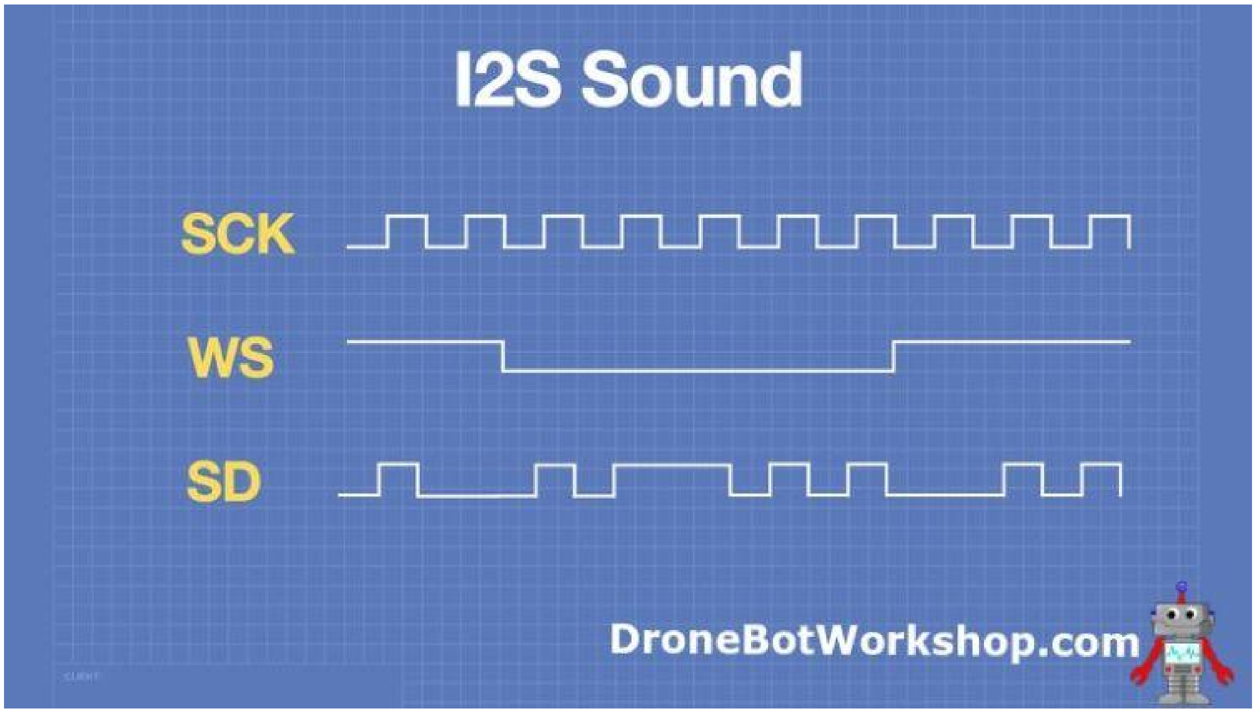
CD-quality digital audio has a resolution of 16-bits and a sample rate of 44.1 kHz, whereas telephone-quality digital audio is 8-bits and is sampled at 8 kHz.

When digital audio is transmitted, either around the world or between integrated circuits, it is done in a serial format. There are several formats, a common one is *Pulse Code Modulation* or PCM.



I2C works with digital PCM data, using any resolution and sample rate. It can be used to route the sound from source to destination, through various signal processors and equalizers in some cases.

I2S Protocol

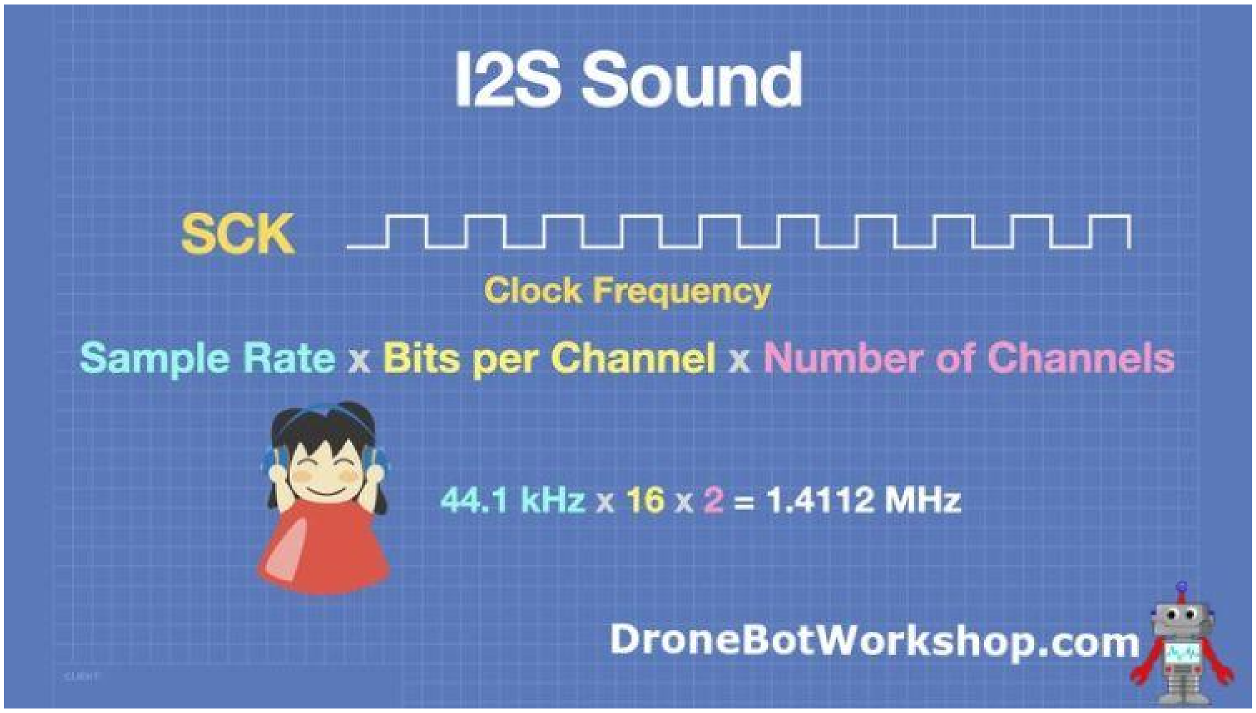


The I2S protocol manages PCM data on a bus that consists of at least the following three connection lines:

- **SCK** – The Serial Clock Line, sometimes referred to as the "bit clock line".
- **WS** – Word Select, which selects between the Left and Right audio channels.
- **SD** – Serial Data, the PCM audio data.

The quality of the audio signal determines the Serial Clock rate, and is determined with the following formula:

Clock Frequency = Sample Rate x Bits Per Channel x Number of Channels



I2S Sound

SCK

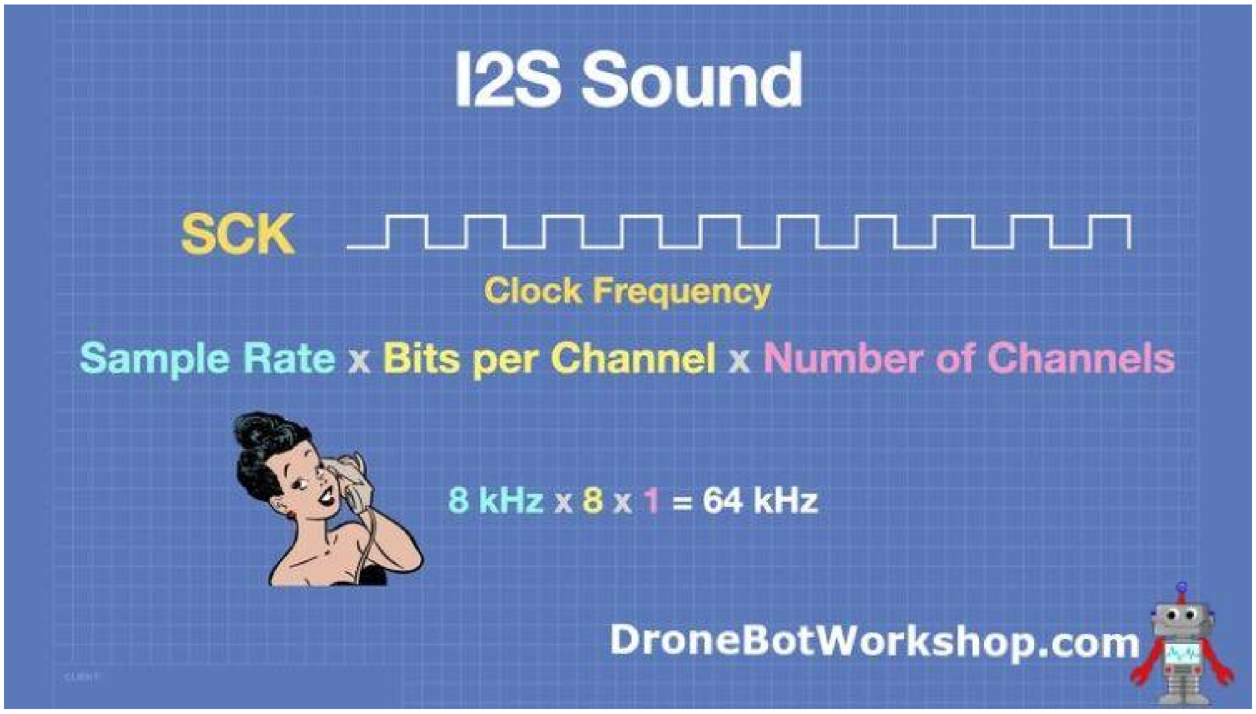
Clock Frequency

Sample Rate x Bits per Channel x Number of Channels

44.1 kHz x 16 x 2 = 1.4112 MHz

DroneBotWorkshop.com

So if we want to send two channels of high-quality audio, we would need a clock rate of 1.4112 MHz.



I2S Sound

SCK

Clock Frequency

Sample Rate x Bits per Channel x Number of Channels

8 kHz x 8 x 1 = 64 kHz

DroneBotWorkshop.com

Sending a single channel of telephone-quality audio would require a clock of 64 kHz.

Controllers & Targets

The devices connected to an I2S bus can be divided into two categories:

- **Controller** – Controls the SCK and WS signals.
- **Target** – Receives the SCK and WS signals.

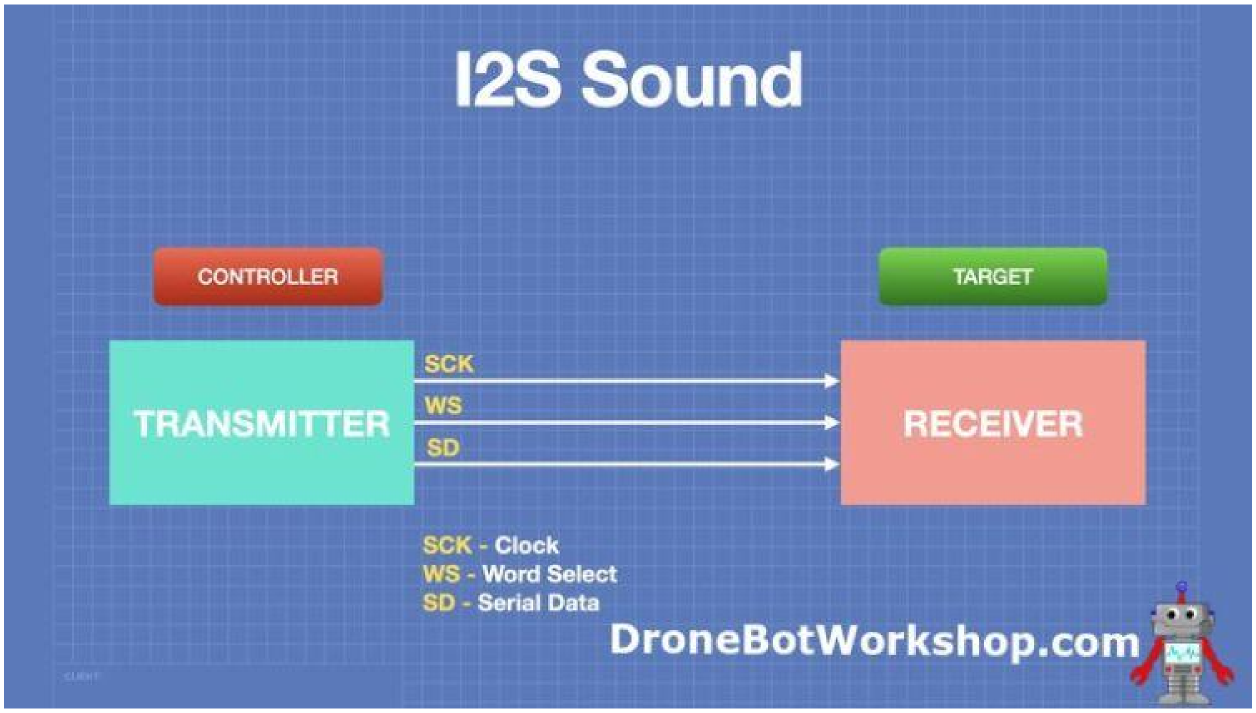
There can only be one controller on the bus, however, the bus can have multiple targets.

As for audio devices, they can be divided into three categories:

- **Transmitters** – Send audio signals.
- **Receivers** – Receive audio signals.
- **Controllers** – Control the audio signals

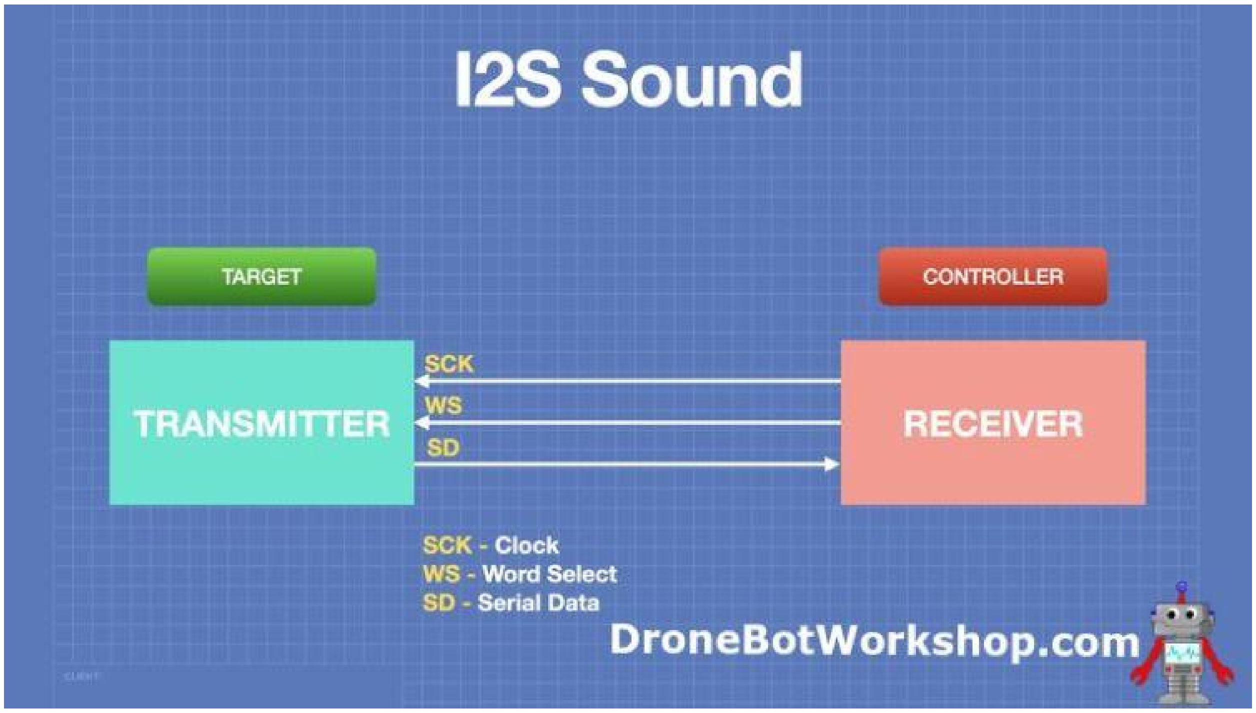
At a minimum, we need a Transmitter and Receiver, the Controller is optional.

A simple topology is illustrated here:

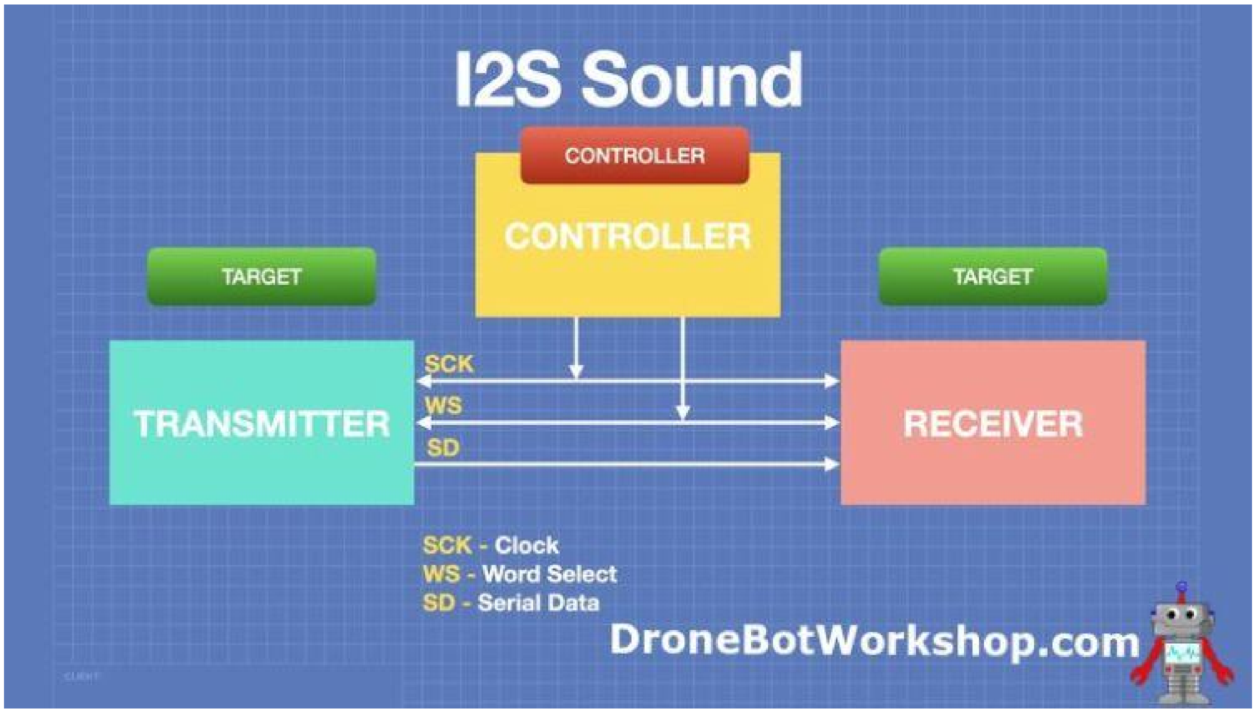


In this layout, the Audio Transmitter is also the I2S Controller, and it provides the SCK, WS, and SD signals. The Receiver is the I2S Target.

While that is a pretty standard arrangement it doesn't have to be the only one, the following layout is just as valid:



Here, the I2S roles are reversed, with the audio Receiver also acting as I2S Controller. It provides the SCK and WS signals. However, the Transmitter, which is the Target in this arrangement, still provides the SD (Serial Data). The audio Transmitter always provides the SD, which makes sense if you think about it!



And here is an arrangement with the Transmitter and Receiver both being I2S Targets. A separate Controller device is an I2S controller, and it provides SCK and WS to both Targets. Once again, the Transmitter supplies the SD signal.

I2S and ESP32

The ESP32 has two I2S peripherals, *I2S0,* and *I2S1*. Each one can be configured as a Controller or Target, and each one can be an audio Transmitter or Receiver.

Each I2S controller can operate in half-duplex communication mode. Thus, the two controllers can be combined to establish full-duplex communication.

The devices also have a DMA (Direct Memory Access) mode, this mode allows for streaming sample data without requiring the CPU to copy each data sample, and can be useful when streaming high-quality audio.

There is also a mode that allows the output of I2S0 to be internally routed to the input of the ESP32 DAC to produce direct analog output without involving any external I2S codecs.

The I2S peripherals also support an advanced mode called “LCD mode” for communicating data over a parallel bus. This is used by some LCDs and camera modules. LCD mode can be operated in the following modes:

- LCD master transmitting mode
- Camera slave receiving mode
- ADC/DAC mode

Espressif has their usual excellent documentation for I2S on the ESP32, it can link you to more information.

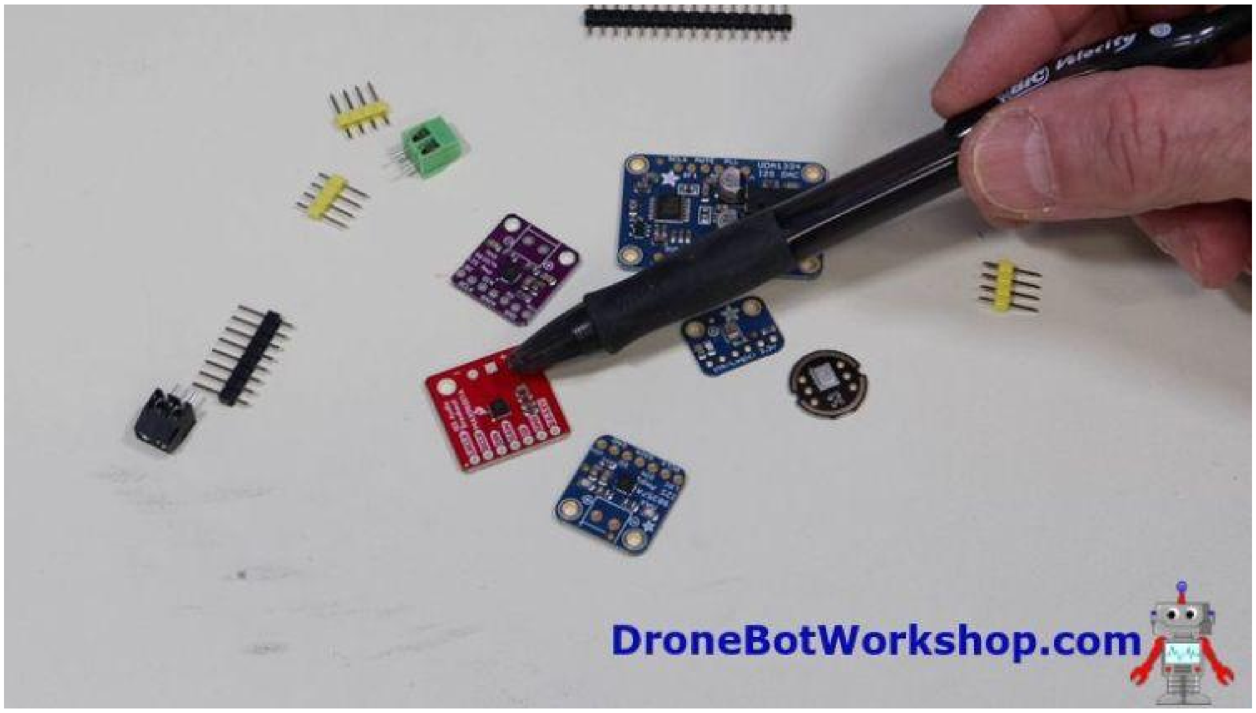
I2S Peripherals

I2S is a standard and peripherals range from small amplifier and microphone modules to complete audio systems with I2S connectivity.

In our experiments, we will be using a couple of I2S peripherals:

- INMP441 Microphone Module
- MAX98357A I2S Amplifier Module

In the video, I also show you a few other I2S microphones and amplifier modules



The microphone and amplifier are available from a variety of vendors, including Amazon and eBay. Sparkfun and Adafruit also have a number of I2S breakout boards for you to play with.

I2S Microphone with ESP32

We will begin our I2S experiments with an I2S microphone module.

There are a number of these modules available, I used a common INMP441 module, but you could substitute another I2S microphone module.

In our experiment, we will display the audio waveforms from the microphone using the Serial Plotter in the Arduino IDE.

INMP441 Microphone Module

The INMP441 is a common and inexpensive I2S microphone module. It uses a MEMS (*Micro-ElectroMechanical Systems)* Microphone and has an internal 24-bit A/D converter and I2S interface.

The INMP441 Microphone Module has the following specifications:

- Omnidirectional response.
- 24-bit I2S Interface.
- Signal to Noise Ratio of 61 dBA.
- Frequency response of 60 Hz – 15 kHz.

It has the following connections:

- **SD** – The I2S Serial Data connection.
- **VDD** – The Input voltage, from 3 to 6 volts.
- **GND** – Ground.
- **L/R** – Channel selection.
- **WS** – Word Select.
- **SCK** – Serial Clock.

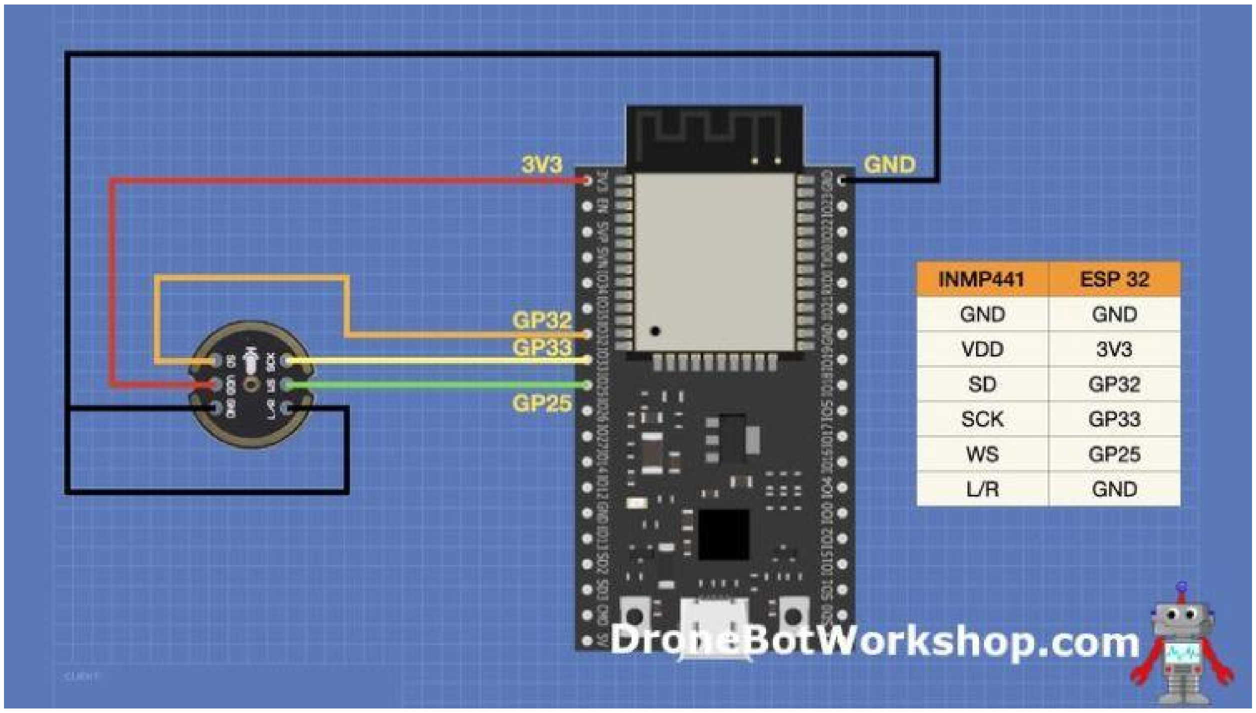
The Channel Selection (L/R pin) works as follows:

- **LEFT** – L/R connected to GND.
- **RIGHT** – L/R connected to VDD.

One thing about this, and most I2S MEMS microphone modules, is that the sound enters the microphone from the bottom of the circuit board. You'll see a small microphone icon beside a little hole, this is where the sound enters the microphone. Make sure you mount your module so that sound can enter the microphone, this usually involves soldering the pins "upside-down".

INMP441 Microphone Module Hookup

Here is how we will be hooking up our microphone module and ESP32. Note that your ESP32 may have a different pinout from the one illustrated here, use the GPIO numbers instead of physical pins to connect your module.



INMP441	ESP 32
GND	GND
VDD	3V3
SD	GP32
SCK	GP33
WS	GP25
L/R	GND

You'll note that the microphone module L/R pin is grounded, as we will be using it as the LEFT channel.

INMP441 Microphone Module Code

In our first experiment, we will be using the I2S Library that is installed in your Arduino IDE when you install the ESP32 Boards Manager files. Here is what the code looks like:

```
/*
  ESP32 I2S Microphone Sample
  esp32-i2s-mic-sample.ino
  Sample sound from I2S microphone, display on Serial Plotter
  Requires INMP441 I2S microphone

  DroneBot Workshop 2022
  https://dronebotworkshop.com
*/

// Include I2S driver
#include <driver/i2s.h>

// Connections to INMP441 I2S microphone
#define I2S_WS 25
#define I2S_SD 33
#define I2S_SCK 32

// Use I2S Processor 0
#define I2S_PORT I2S_NUM_0

// Define input buffer length
#define bufferLen 64
int16_t sBuffer[bufferLen];

void i2s_install() {
  // Set up I2S Processor configuration
  const i2s_config_t i2s_config = {
    .mode = i2s_mode_t(I2S_MODE_MASTER | I2S_MODE_RX),
    .sample_rate = 44100,
    .bits_per_sample = i2s_bits_per_sample_t(16),
    .channel_format = I2S_CHANNEL_FMT_ONLY_LEFT,
    .communication_format = i2s_comm_format_t(I2S_COMM_FORMAT_STAND_I2S),
    .intr_alloc_flags = 0,
    .dma_buf_count = 8,
    .dma_buf_len = bufferLen,
    .use_apll = false
  };

  i2s_driver_install(I2S_PORT, &i2s_config, 0, NULL);
}

void i2s_setpin() {
  // Set I2S pin configuration
  const i2s_pin_config_t pin_config = {
    .bck_io_num = I2S_SCK,
    .ws_io_num = I2S_WS,
    .data_out_num = -1,
    .data_in_num = I2S_SD
  };

  i2s_set_pin(I2S_PORT, &pin_config);
}

void setup() {

  // Set up Serial Monitor
  Serial.begin(115200);
  Serial.println(" ");

  delay(1000);

  // Set up I2S
  i2s_install();
  i2s_setpin();
  i2s_start(I2S_PORT);


  delay(500);
}

void loop() {

  // False print statements to "lock range" on serial plotter display
  // Change rangelimit value to adjust "sensitivity"
  int rangelimit = 3000;
  Serial.print(rangelimit * -1);
  Serial.print(" ");
  Serial.print(rangelimit);
  Serial.print(" ");

  // Get I2S data and place in data buffer
  size_t bytesIn = 0;
  esp_err_t result = i2s_read(I2S_PORT, &sBuffer, bufferLen, &bytesIn,
portMAX_DELAY);

  if (result == ESP_OK)
  {
    // Read I2S data buffer
    int16_t samples_read = bytesIn / 8;
    if (samples_read > 0) {
      float mean = 0;
      for (int16_t i = 0; i < samples_read; ++i) {
        mean += (sBuffer[i]);
      }

      // Average the data reading
      mean /= samples_read;

      // Print to serial plotter
      Serial.println(mean);
    }
  }
}
```

We start by including the ESP32 I2S driver.

We then define the connections to our microphone. If you wish, you can rewire the microphone and change the code here.

The ESP32 has two internal I2S processors. We will be using the first one, I2S Port 0. We also define the length of an input data buffer.

Next, we have a function called *i2s_install*, which sets up the I2S port parameters.

A second function, *i2s_setpin*, sets up the physical connection to the I2S device, which in our case is the microphone module.

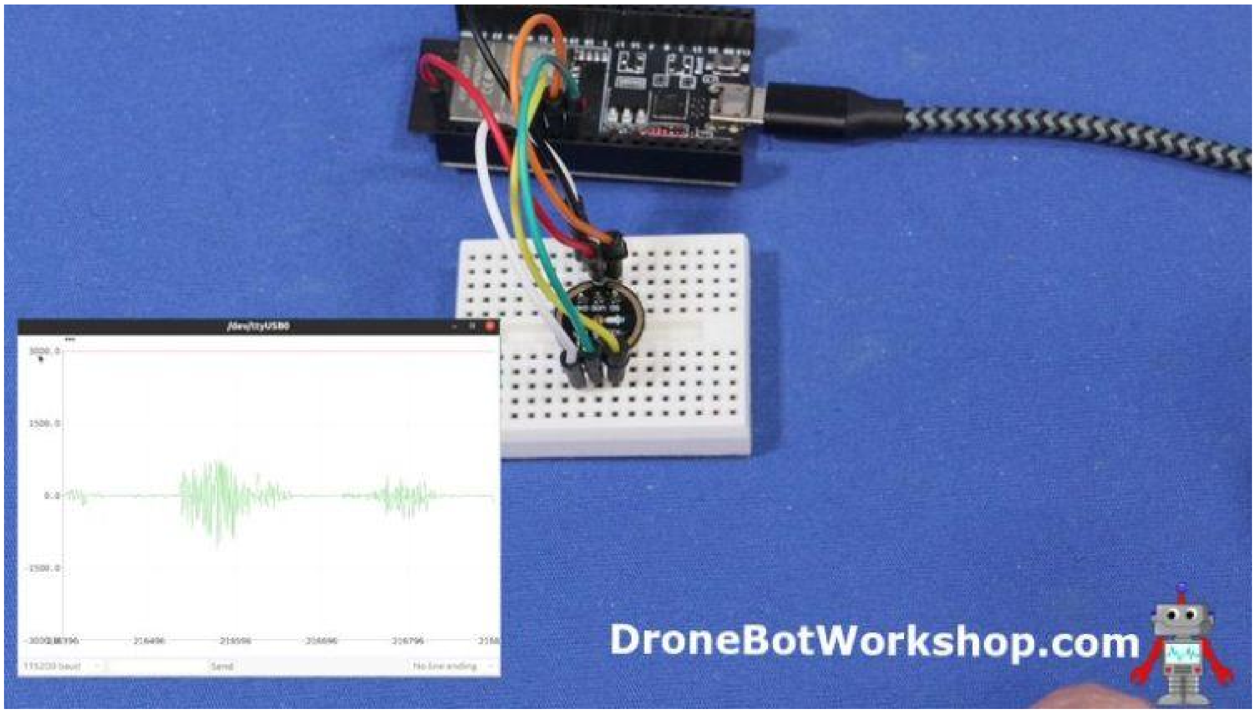
In the Setup, we set up our serial connection, as we will be using the Serial Plotter to display our audio waveforms. We then cal our two functions to set up the I2S port, and then start it with a third built-in function.

Our Loop starts with a "false" print statement, this just causes two constants to be printed to steady the reading on the Serial Plotter, which otherwise will dynamically change its Y-axis scale.

We then read data from the module and place it in our data buffer. If the data is good, we read it out and display it on the Serial Plotter

Testing the Microphone

Hook everything up, load the sketch and open the Serial Plotter.



You should see a representation of the sound that the microphone is getting. You can adjust the sensitivity by altering the *rangelimit* variable in the Loop.

ESP32 MP3 Player

For our next experiment, we will be using an I2S amplifier module. The sound source will be an MP3 file that is stored on a MicroSD card.

This is an extremely basic MP3 player, for practical use you would need to make a system for navigating the MicroSD to play more than one selection. It's just to illustrate how to use the I2S amplifier, as well as a library that makes working with I2S audio applications a bit easier.

MP3 Player Components

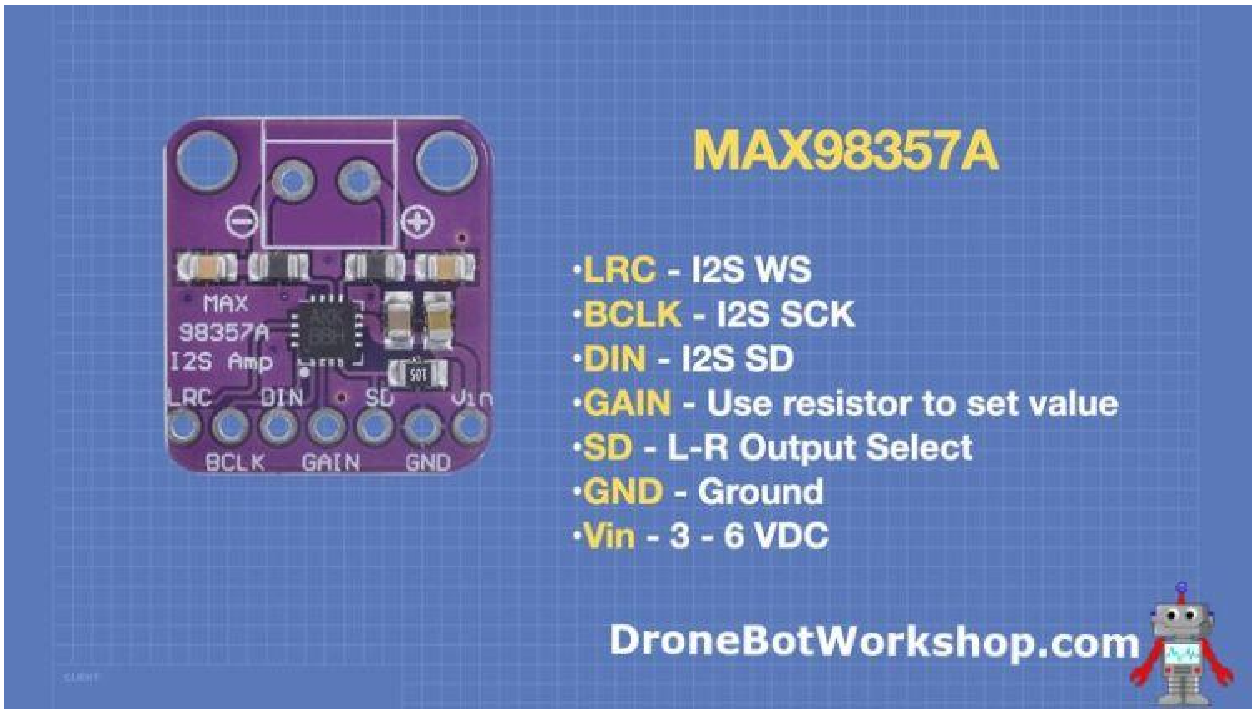
We will be adding two components to our ESP32 to construct our MP3 player, specifically an aI2S amplifier and a MicroSD card breakout box. Of course, we will also need a MicroSD card, a small one formatted with FAT32. Put an MP3 file onto the card, in our experiment we will only be playing one selection.

MAX98357A I2S Amplifier Module

The MAX98357A I2S amplifier module is an inexpensive yet surprisingly powerful audio amplifier module with an I2S input.

The device can output up to 3 watts into a 4-ohm load. Note that you must use a speaker with a 4 to 8-ohm voice coils, and you can't use a dynamic speaker, as the voice coil is actually part of the output filtering circuit in the module.

The device has the following pinout:



Note that the data is input on the DIN line, not the SD line, which is used for output channel selection. We will go into detail about output channel selection in a bit.

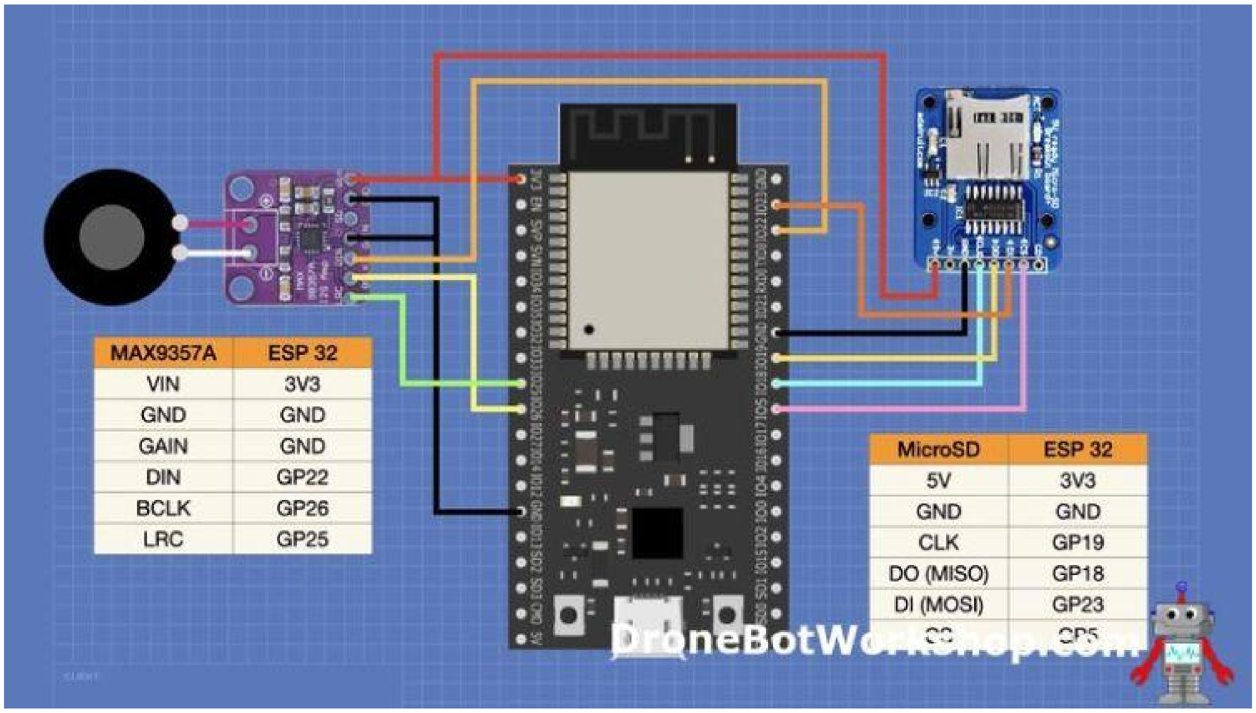
Adafruit MicroSD Breakout Board

We need a MicroSD card breakout board and I selected one from Adafruit. The reason for getting this module was its ability to work with both 3.3-volt and 5-volt logic. If you use another board, make sure it can operate on 3.3-volts, as many popular modules are for 5-volt logic and would damage the ESP32.

The MicroSD card module uses the SPI bus for communications.

MP3 Player Hookup

Here is the hookup for our MP3 player:



MAX9357A	ESP 32
VIN	3V3
GND	GND
GAIN	GND
DIN	GP22
BCLK	GP26
LRC	GP25

MicroSD	ESP 32
5V	3V3
GND	GND
CLK	GP19
DO (MISO)	GP18
DI (MOSI)	GP23
CS	GP5

If you wish, you can move the I2S amplifier to different GPIO pins, as nothing is special about them. Just be sure to make the appropriate changes in the sketch. The MicroSD card requires SPI bus connections, there are several on the ESP32 to choose from.

MP3 Player Code

We will be using a library that makes working with I2S a lot easier. It isn't in your Library Manager, so you'll need to download it.

The ESP32-AudioI2S Library can be found on GitHub, you can either clone it into your Arduino *Libraries* folder or just download it as a ZIP file. If you grab the ZIP file, you can add it to your Arduino IDE using the *Add ZIP Library* item on the *Sketch* menu.

This library will simplify working with I2S, you create an “audio” object that you can then manipulate in code.

Here is the sketch that we will be using:

```
/*
  ESP32 SD I2S Music Player
  esp32-i2s-sd-player.ino
  Plays MP3 file from microSD card
  Uses MAX98357 I2S Amplifier Module
  Uses ESP32-audioI2S Library - https://github.com/schreibfaul1/ESP32-audioI2S
  *
  DroneBot Workshop 2022
  https://dronebotworkshop.com
*/

// Include required libraries
#include "Arduino.h"
#include "Audio.h"
#include "SD.h"
#include "FS.h"
// microSD Card Reader connections
#define SD_CS          5
#define SPI_MOSI      23
#define SPI_MISO      19
#define SPI_SCK     18
// I2S Connections
#define I2S_DOUT      22
#define I2S_BCLK      26
#define I2S_LRC     25
// Create Audio object
Audio audio;
void setup() {
    // Set microSD Card CS as OUTPUT and set HIGH
    pinMode(SD_CS, OUTPUT);
    digitalWrite(SD_CS, HIGH);

    // Initialize SPI bus for microSD Card
    SPI.begin(SPI_SCK, SPI_MISO, SPI_MOSI);

    // Start Serial Port
    Serial.begin(115200);

    // Start microSD Card
    if(!SD.begin(SD_CS))
    {
      Serial.println("Error accessing microSD card!");
      while(true);
    }

    // Setup I2S
    audio.setPinout(I2S_BCLK, I2S_LRC, I2S_DOUT);

    // Set Volume
    audio.setVolume(5);

    // Open music file
    audio.connecttoFS(SD,"/MYMUSIC.mp3");

}
void loop()
{
    audio.loop();
}
```

We start by loading the new library we just installed, along with the SD and SPI libraries that we will need to work with the MicroSD card breakout.

We then define the connection to both the MicroSD module and the I2S amplifier module.

After that, we use our new library to create an “audio” object.

In Setup, we set up the CS (Chip Select) lead on the MicroSD so that it is always selected. We start the serial port and also start the MicroSD card.

Assuming that the MicroSD is OK, we then set up our I2C port. Note how much easier the library makes this compared to using just the ESP32 I2S library (upon which this library is dependent).

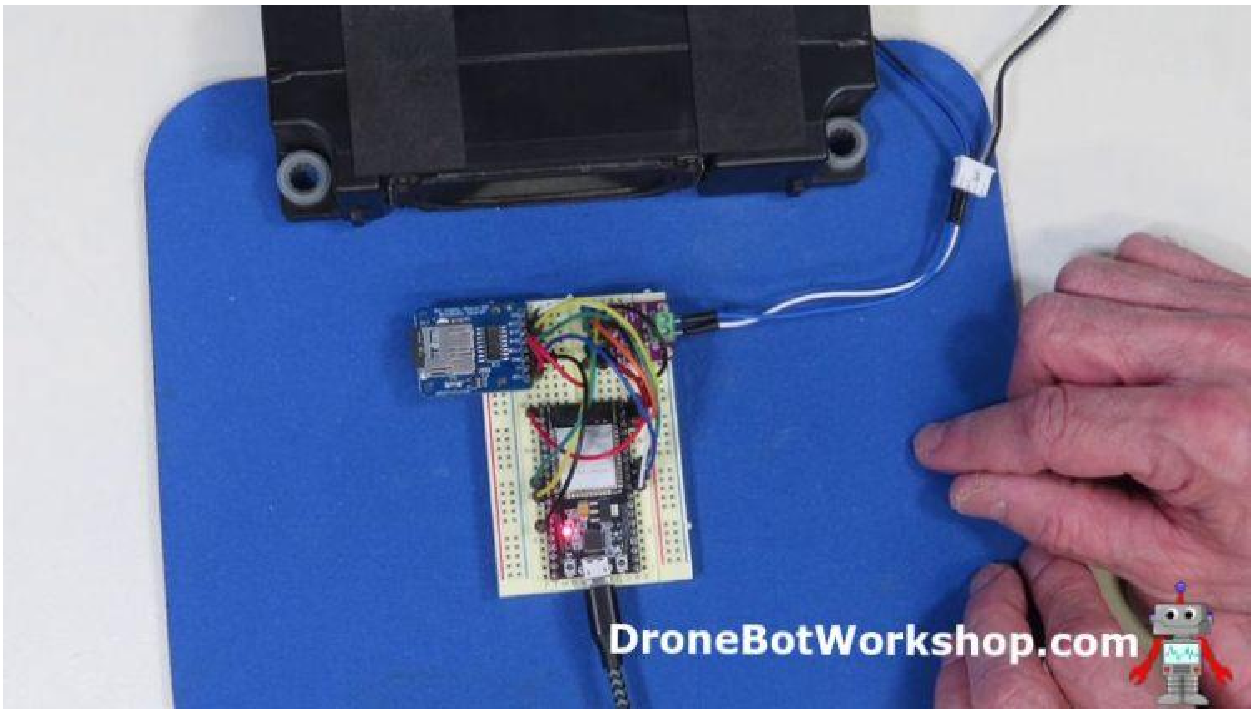
We also set the audio level, any number from 0 (no audio) to 21 will work here. Best to start with a small number (I chose 5) as the amplifier is pretty efficient.

Finally, we open up the MP3 file, using a “connect to FS (connect to File System) property of the audio object. Again, the library makes grabbing an audio source very easy.

The Loop couldn’t be simpler. We just cal the Loop method of the audio object repeatedly.

Test the MP3 Player

Place a MicroSD card that has the file you wish to play into the breakout board. Load the code onto the ESP32.



When it has done uploading, press the Reset button on the ESP32. You should be greeted b the contents of your MP3 file.

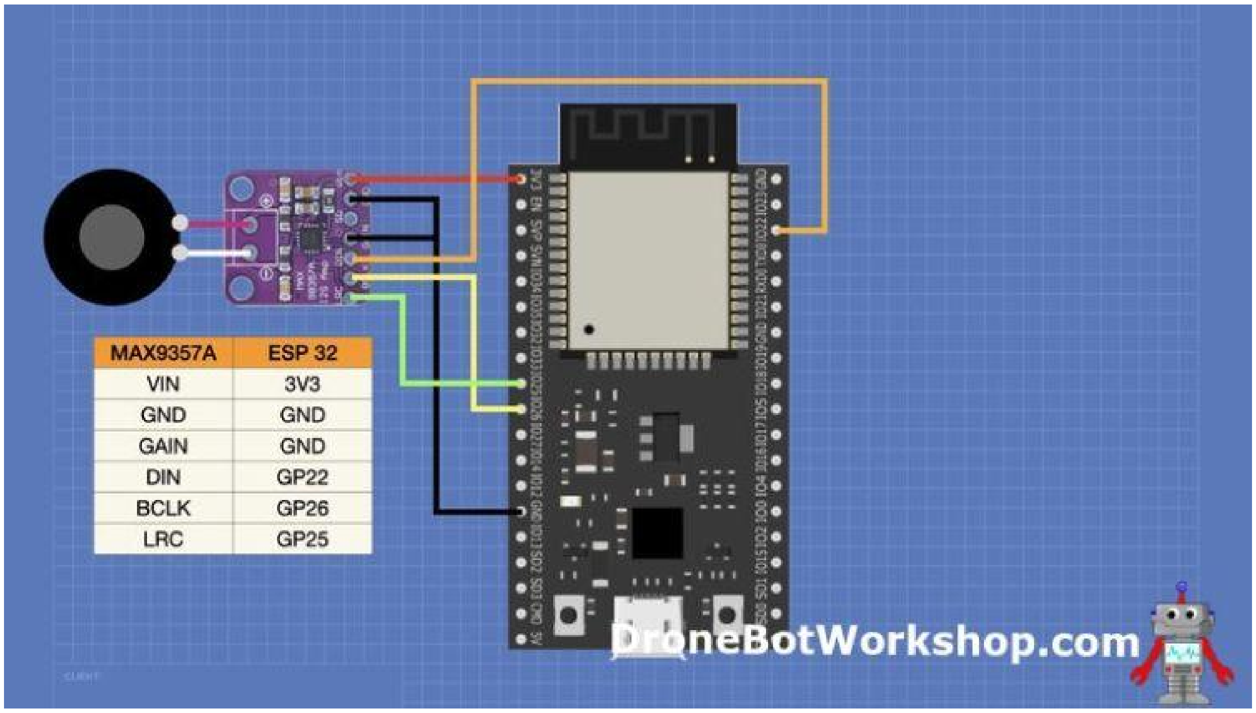
Make sure you chose something you like to listen to, as you have no volume control and can only play one selection once. But, as far as demo code goes, it shows you how simple it is to build the foundation of an I2S audio player.

ESP32 Internet Radio

In this experiment, we will take advantage of the WiFi capabilities of the ESP32 and build an internet radio, which will use the same I2S amplifier module that we previously used.

Internet Radio Hookup

The hookup of our Internet Radio is essentially the same as the hookup of the MP3 p[layer, minus the MicroSD module. If you have already wired up the MP3 player, then you can just leave it as it is, as the MicroSD module will just be ignored.



MAX9357A	ESP 32
VIN	3V3
GND	GND
GAIN	GND
DIN	GP22
BCLK	GP26
LRC	GP25

Internet Radio Code

We will be using the same library we used for the MP3 player, as it makes it very easy to build an Internet Radio since you can just specify a URL as a sound source.

```
/*
  Simple Internet Radio Demo
  esp32-i2s-simple-radio.ino
  Simple ESP32 I2S radio
  Uses MAX98357 I2S Amplifier Module
  Uses ESP32-audioI2S Library - https://github.com/schreibfaul1/ESP32-audioI2S

  DroneBot Workshop 2022
  https://dronebotworkshop.com
*/

// Include required libraries
#include "Arduino.h"
#include "WiFi.h"
#include "Audio.h"

// Define I2S connections
#define I2S_DOUT  22
#define I2S_BCLK  26
#define I2S_LRC   25

// Create audio object
Audio audio;

// Wifi Credentials
String ssid =    "YOURSSID";
String password = "YOURPASSWORD";

void setup() {

  // Start Serial Monitor
  Serial.begin(115200);

  // Setup WiFi in Station mode
  WiFi.disconnect();
  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid.c_str(), password.c_str());

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  // WiFi Connected, print IP to serial monitor
  Serial.println("");
  Serial.println("WiFi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
  Serial.println("");

  // Connect MAX98357 I2S Amplifier Module
  audio.setPinout(I2S_BCLK, I2S_LRC, I2S_DOUT);

  // Set thevolume (0-100)
  audio.setVolume(10);

  // Connect to an Internet radio station (select one as desired)
  //audio.connecttohost("http://vis.media-ice.musicradio.com/CapitalMP3");
  //audio.connecttohost("mediaserv30.live-nect MAX98357 I2S Amplifier Module
  //audio.connecttohost("www.surfmusic.de/m3u/100-5-das-hitradio,4529.m3u");
  //audio.connecttohost("stream.1a-webradio.de/deutsch/mp3-128/vtuner-1a");
  //audio.connecttohost("www.antenne.de/webradio/antenne.m3u");
  audio.connecttohost("0n-80s.radionetz.de:8000/0n-70s.mp3");

}

void loop()

{
  // Run audio player
  audio.loop();

}

// Audio status functions

void audio_info(const char *info) {
  Serial.print("info        "); Serial.println(info);
}
void audio_id3data(const char *info) { //id3 metadata
  Serial.print("id3data    "); Serial.println(info);
}
void audio_eof_mp3(const char *info) { //end of file
  Serial.print("eof_mp3    "); Serial.println(info);
}
void audio_showstation(const char *info) {
  Serial.print("station    "); Serial.println(info);
}
void audio_showstreaminfo(const char *info) {
  Serial.print("streaminfo  "); Serial.println(info);
}
void audio_showstreamtitle(const char *info) {
  Serial.print("streamtitle "); Serial.println(info);
}
void audio_bitrate(const char *info) {
  Serial.print("bitrate    "); Serial.println(info);
}
void audio_commercial(const char *info) { //duration in sec
  Serial.print("commercial  "); Serial.println(info);
}
void audio_icyurl(const char *info) { //homepage
  Serial.print("icyurl      "); Serial.println(info);
}
void audio_lasthost(const char *info) { //stream URL played
  Serial.print("lasthost    "); Serial.println(info);
}
void audio_eof_speech(const char *info) {
  Serial.print("eof_speech  "); Serial.println(info);
}
```

Our code is quite similar to the MP3 player, except we don't have the MicroSD breakout board to deal with.

After defining the I2S amplifier connections, we create an audio object, just as we did before.

Then we grab the WiFi credentials, as our Internet Radio will (obviously) need to connect to the Internet!

Setup starts the Serial Monitor, which we will use to display program information, and then connects to the WiFi.

We then define the connections to the amplifier module and set the volume, again as we did in the last sketch.

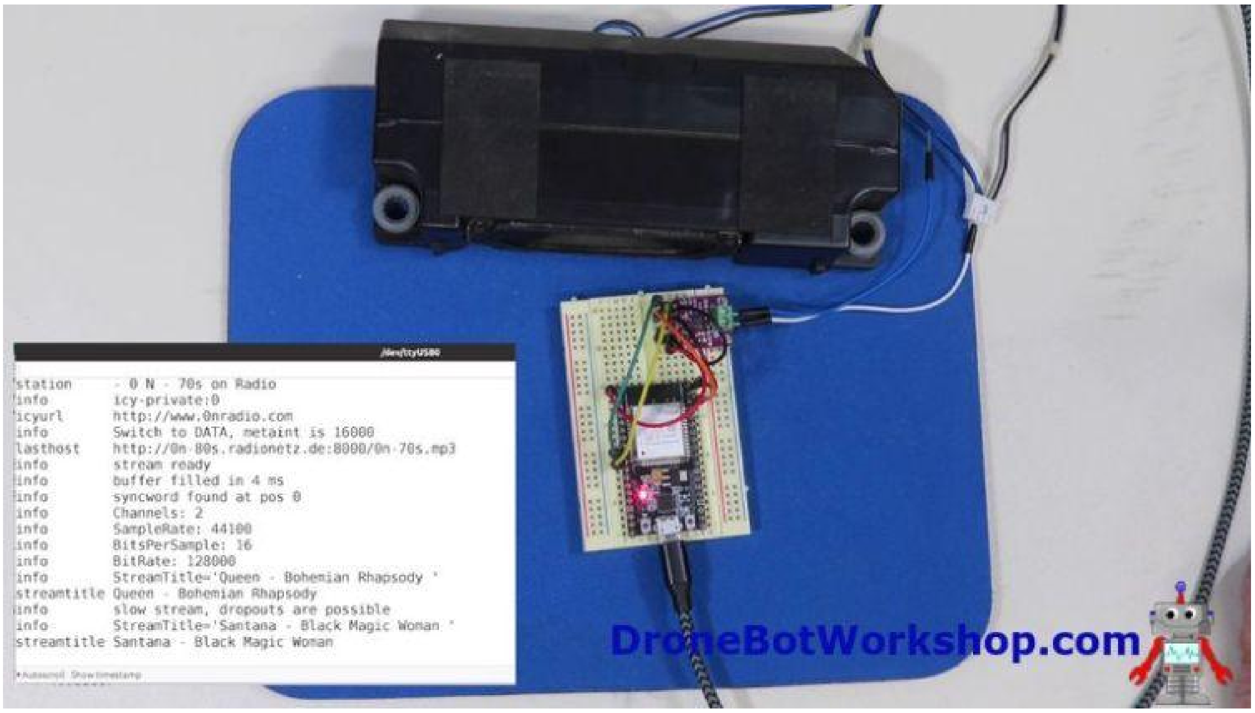
Finally, we use the libraries *connecttohost* function to connect to a URL, I have listed a few here and there are many more you can use.

Once again, the Loop just uses the audio object *Loop* method to play music.

We also have a number of other functions below the Loop. These functions will display status information on the serial monitor.

Testing the Internet Radio

Load the code onto the ESP32 and press reset. Observe the serial monitor.



```
station     - 0 N - 70s on Radio
info        icy-private:0
icyurl      http://www.0nradio.com
info        Switch to DATA, metaint is 16000
lasthost    http://0n-80s.radionetz.de:8000/0n-70s.mp3
info        stream ready
info        buffer filled in 4 ms
info        syncword found at pos 0
info        Channels: 2
info        SampleRate: 44100
info        BitsPerSample: 16
info        BitRate: 128000
info        StreamTitle='Queen - Bohemian Rhapsody '
streamtitle Queen - Bohemian Rhapsody
info        slow stream, dropouts are possible
info        StreamTitle='Santana - Black Magic Woman '
streamtitle Santana - Black Magic Woman
```

You should first see the WiFi connection information, which hopefully will show that you have connected to a network. Assuming you have, you will then start to load the URL for the radio station.

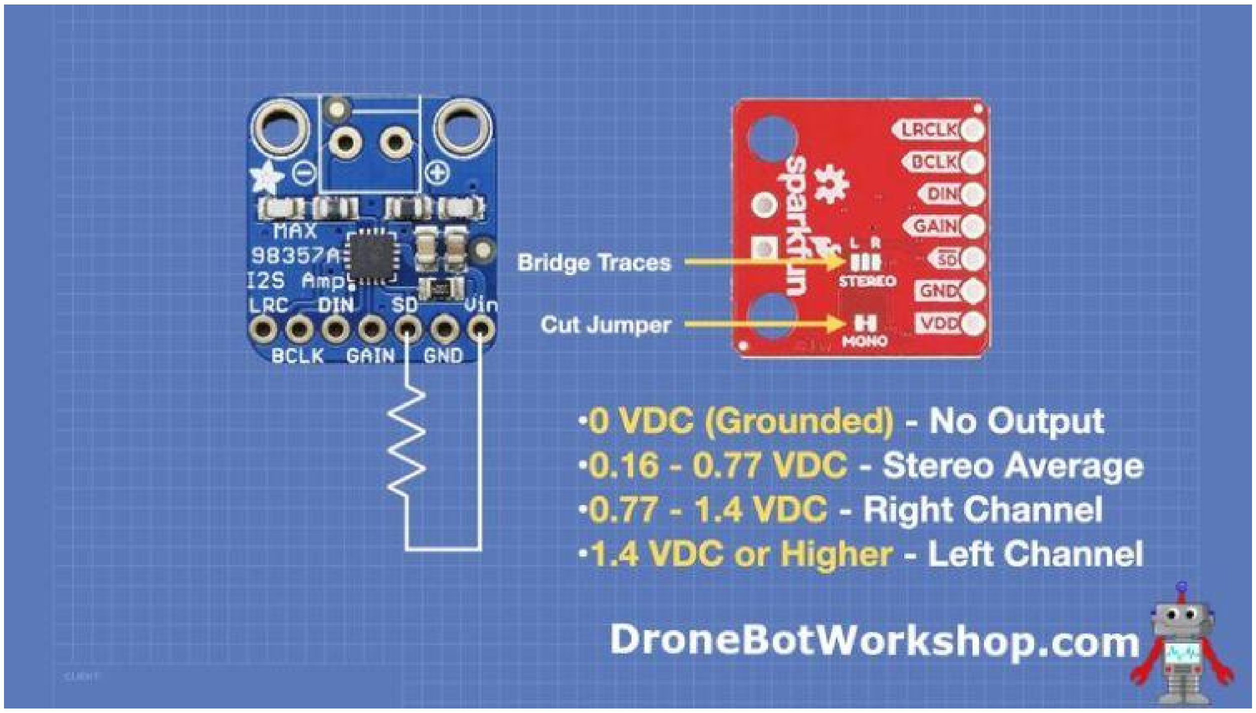
If you connect successfully, you will see the audio stream name and status displayed in the serial monitor.

Running in Stereo

We can modify our Internet Radio to produce a stereo output by simply adding another I2S amplifier module. We will also add a potentiometer to use as a volume control, a basic but essential control for any audio device.

MAX98357A Channel Selection

The MAX98357A I2S amplifier module that we have been using in our experiments can output either the left audio channel, the right audio channel, or a mix of both.


MAX
98357A
I2S Amp
LRC
DIN
SD
Vin
BCLK
GAIN
GND
sparkfun
LRCLK
BCLK
DIN
GAIN
SD
GND
VDD
L R
STEREO
MONO
Bridge Traces
Cut Jumper
•0 VDC (Grounded) - No Output
•0.16 - 0.77 VDC - Stereo Average
•0.77 - 1.4 VDC - Right Channel
•1.4 VDC or Higher - Left Channel
DroneBotWorkshop.com

By default, the device outputs a mix, which is essentially monophonic sound. But we can modify it to output just the left or just the right channel, so we can use two devices for stereo output.

The SD pin on the module is the key to selecting the channel output. Despite its confusing label, this is not the Serial Data input, instead, it is the channel selection pin.

The pin works with a control voltage, the voltage on this pin determines which channel the module will output.

By default, an internal pull-down resistor keeps the level between 0.16 and 0.77 volts, so the amplifier module will output a mixture of both channels. Note that you can also mute the amplifier by grounding the SD pin.

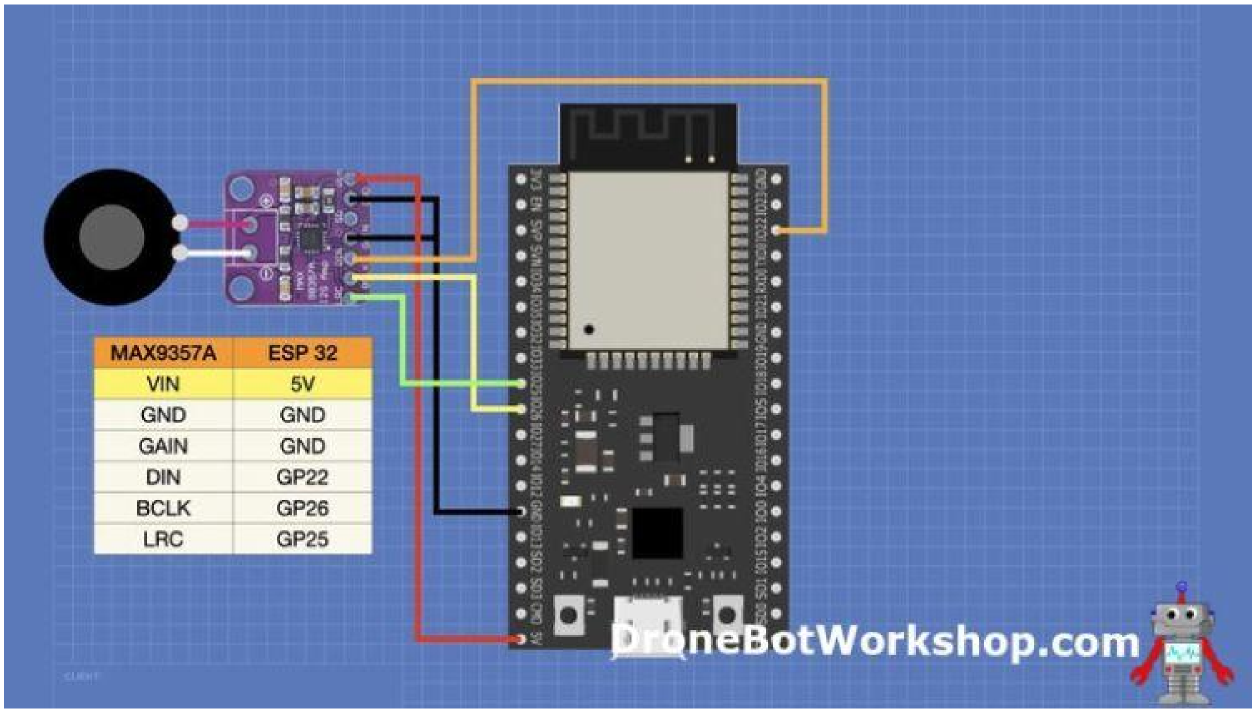
You can change the voltage on the pin using a pull-up resistor. The value will need to be determined by experimentation, as it is affected by the supply voltage.

I suggest using a 100k trimpot to set the voltage. You can then remove the trimpot, measure its resistance, and replace it with the standard resistor value that is closest.

If you purchase the Sparkfun version of the MAX98357A I2S amplifier module it is much easier, as the resistors are already there for you. You set the channel configuration by cutting the MONO jumper and bridging the STEREO pads to either L or R, depending upon which channel you want the module to output.

Stereo Radio Hookup

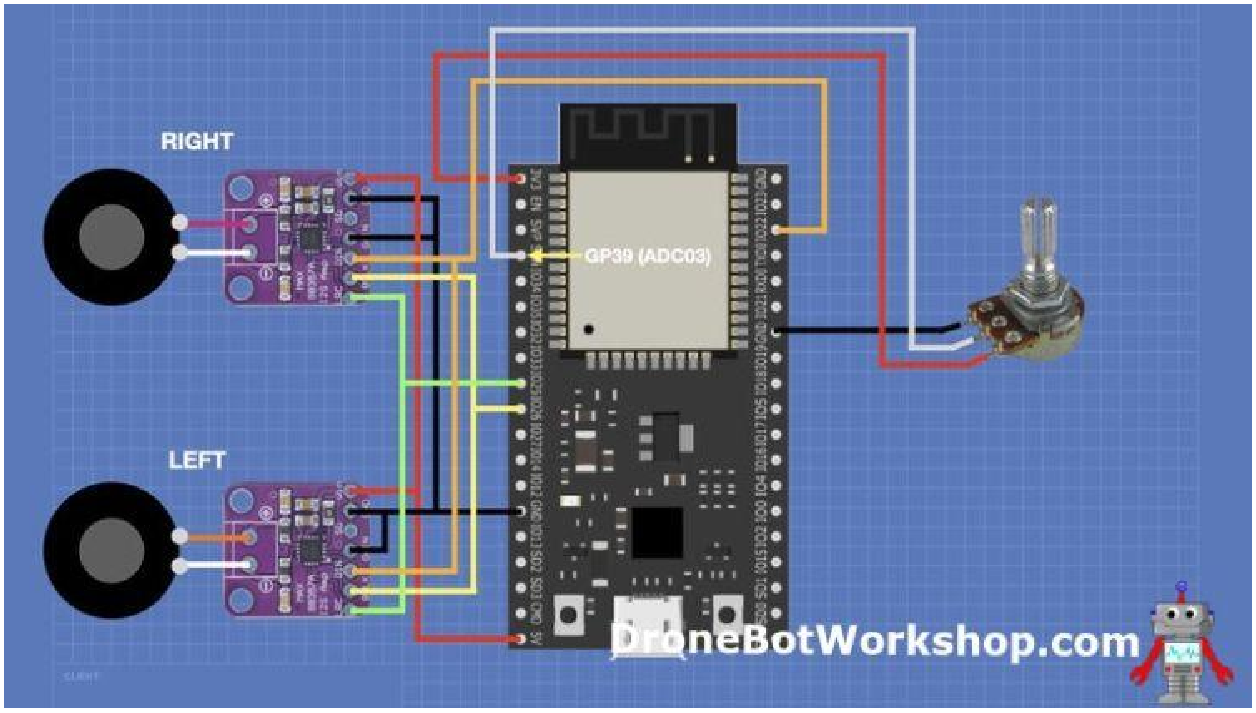
The hookup starts out using the same GPIO pins as our earlier Internet Radio, so you can just expand the wiring on that if you have built it already.



MAX9357A	ESP 32
VIN	5V
GND	GND
GAIN	GND
DIN	GP22
BCLK	GP26
LRC	GP25

One difference is that I am now using the 5-volt line on the ESP32 to power the amplifiers. I'm concerned that powering two 3-watt amplifiers with the ESP32 internal regulator may overload it. 5-volts will also give you more output power.

Otherwise, the wiring is identical for both amplifier modules, just wire them in parallel.



We also have a potentiometer connected to GPIO pin 39, which is also ADC03. We need to use an analog to digital converter in the lower group, as we are using WiFi and when you do, you can't use the upper group.

Also, make sure you wire the stereo speakers in phase!

Stereo Radio Code

There is nothing special about the code for stereo, as the previous code was actually producing stereo output. We were just using a mono amplifier to listen to it.

The code differences here are just for the volume control.

```
/*
  Internet Radio with Volume Demo
  esp32-i2s-radio-volume.ino
  ESP32 I2S radio with volume control
  Uses two MAX98357 I2S Amplifier Modules, strapped for Left and Right channel
  Uses ESP32-audioI2S Library - https://github.com/schreibfaul1/ESP32-audioI2S

  DroneBot Workshop 2022
  https://dronebotworkshop.com
*/

// Include required libraries
#include "Arduino.h"
#include "WiFi.h"
#include "Audio.h"

// Define I2S connections
#define I2S_DOUT  22
#define I2S_BCLK  26
#define I2S_LRC   25

// Define volume control pot connection
// ADC3 is GPIO 39
const int volControl = 39;

// Integer for volume level
int volume = 10;

// Create audio object
```

```
Audio audio;

// Wifi Credentials
String ssid =    "YOUR_SSID";
String password = "YOUR_PASSWORD";

void setup() {

  // Start Serial Monitor
  Serial.begin(115200);

  // Setup WiFi in Station mode
  WiFi.disconnect();
  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid.c_str(), password.c_str());

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  // WiFi Connected, print IP to serial monitor
  Serial.println("");
  Serial.println("WiFi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
  Serial.println("");

  // Connect MAX98357 I2S Amplifier Module
  audio.setPinout(I2S_BCLK, I2S_LRC, I2S_DOUT);

  // Set the volume
  audio.setVolume(volume);

  // Connect to an Internet radio station (select one as desired)
  //audio.connecttohost("http://vis.media-ice.musicradio.com/CapitalMP3");
  //audio.connecttohost("mediaserv30.live-nect MAX98357 I2S Amplifier Module
  //audio.connecttohost("www.surfmusic.de/m3u/100-5-das-hitradio,4529.m3u");
  //audio.connecttohost("stream.1a-webradio.de/deutsch/mp3-128/vtuner-1a");
  //audio.connecttohost("www.antenne.de/webradio/antenne.m3u");
  audio.connecttohost("0n-80s.radionetz.de:8000/0n-70s.mp3");

}

void loop()

{
  // Run audio player
  audio.loop();

  // Get the volume level
  volume = map ((analogRead(volControl)), 0, 4095, 0 , 20);

  // Set the volume
  audio.setVolume(volume);

}

```

```
// Audio status functions

void audio_info(const char *info) {
  Serial.print("info        "); Serial.println(info);
}
void audio_id3data(const char *info) { //id3 metadata
  Serial.print("id3data    "); Serial.println(info);
}
void audio_eof_mp3(const char *info) { //end of file
  Serial.print("eof_mp3    "); Serial.println(info);
}
void audio_showstation(const char *info) {
  Serial.print("station    "); Serial.println(info);
}
void audio_showstreaminfo(const char *info) {
  Serial.print("streaminfo  "); Serial.println(info);
}
void audio_showstreamtitle(const char *info) {
  Serial.print("streamtitle "); Serial.println(info);
}
void audio_bitrate(const char *info) {
  Serial.print("bitrate    "); Serial.println(info);
}
void audio_commercial(const char *info) { //duration in sec
  Serial.print("commercial  "); Serial.println(info);
}
void audio_icyurl(const char *info) { //homepage
  Serial.print("icyurl      "); Serial.println(info);
}
```

```
void audio_lasthost(const char *info) { //stream URL played
  Serial.print("lasthost    "); Serial.println(info);
}
void audio_eof_speech(const char *info) {
  Serial.print("eof_speech  "); Serial.println(info);
}
```

Note how once again the use of the library makes this very easy.

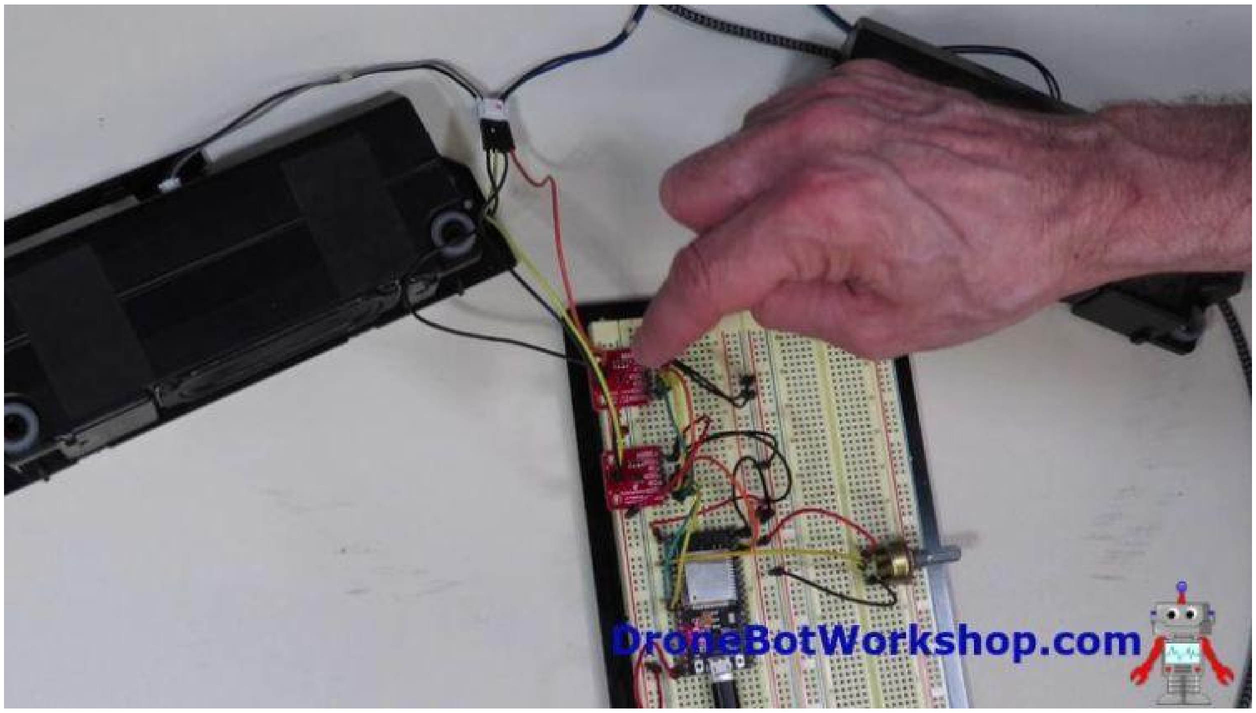
We have added a couple of integers to handle the potentiometer connection and the volume level.

Most of the code in the Setup is identical to that of the earlier Internet radio.

The only real change is in the Loop. We still run the audio object Loop method, but we also grab the input from the potentiometer and Map it to a range of 0 to 20. We then use that to control the volume.

Testing the Stereo Radio

Testing the radio is essentially the same as it was with the earlier version, load the code, press reset, and watch the serial monitor. Once you connect to a station, you should hear the program, this time in wonderful stereo.



Conclusion

Being able to manipulate digital audio increases the growing list of applications for the ESP32. You can probably envision a number of cool projects using the boards' I2S capabilities, and the peripheral boards are inexpensive and pretty easy to work with.

So no, that's not a misspelling of "I2C" on the ESP32 spec sheet. I2S is just another great feature of what is quickly becoming the world's most versatile microcontroller.