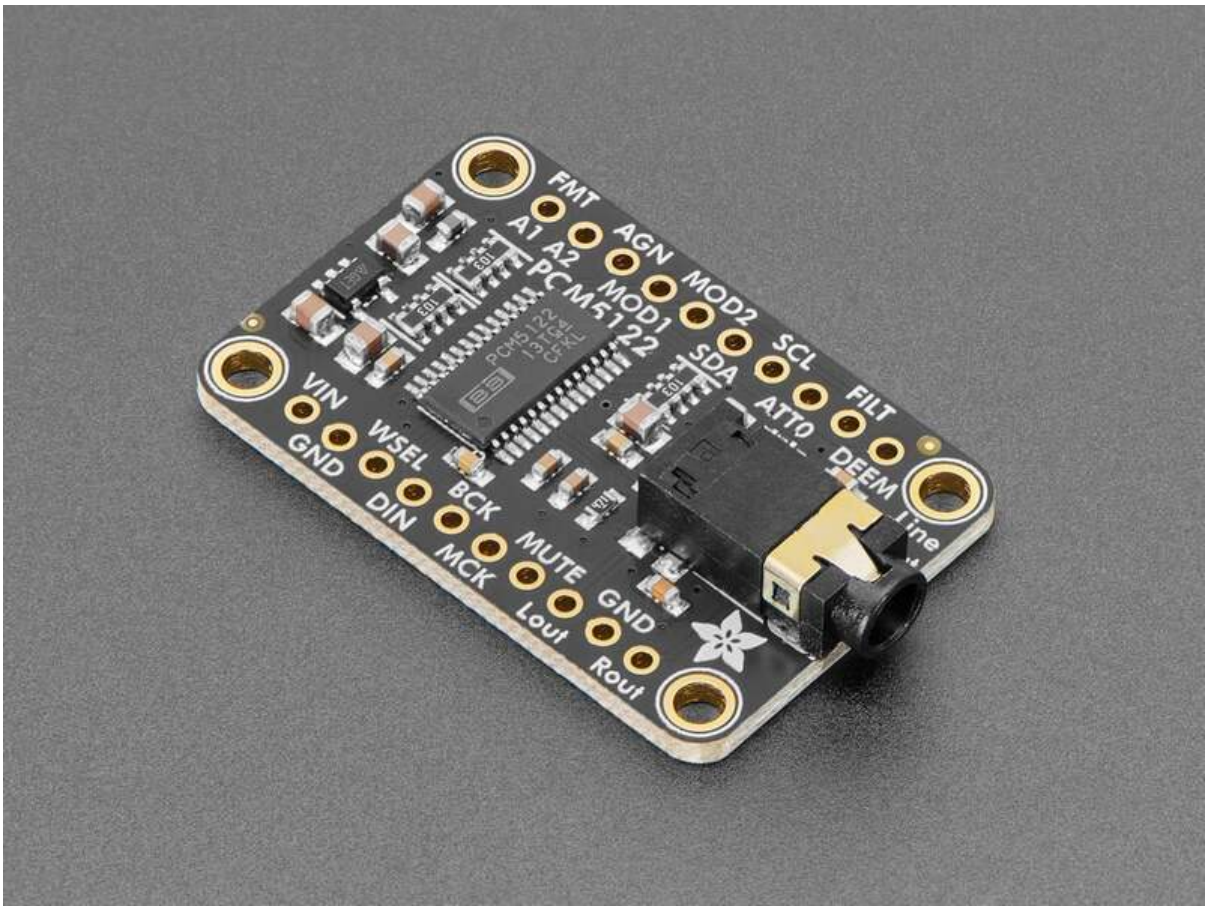
Adafruit PCM5122 I2S DAC

Created by Liz Clark

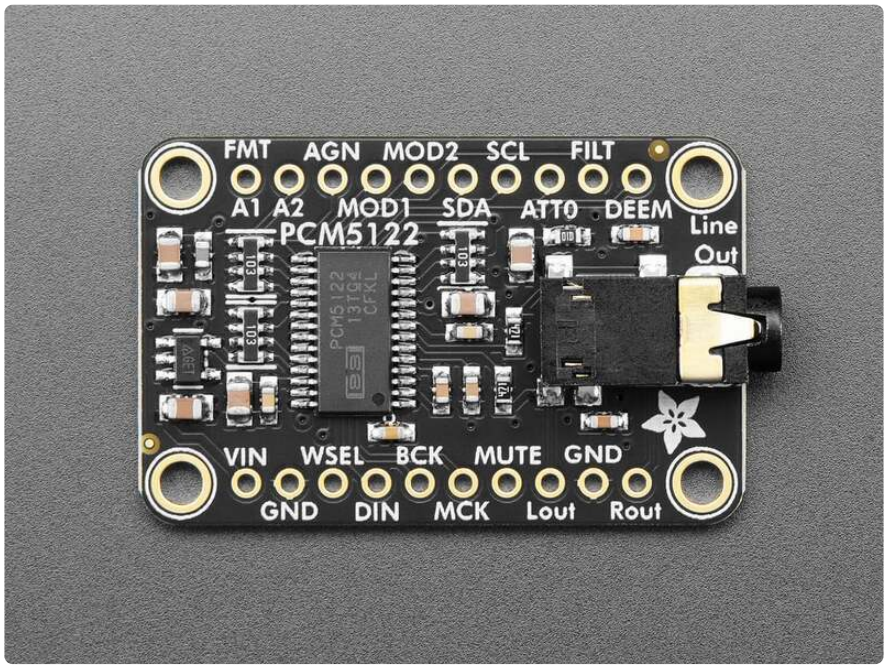


https://learn.adafruit.com/adafruit-pcm5122-i2s-dac

Last updated on 2025-09-26 02:27:19 PM EDT

Table of Contents

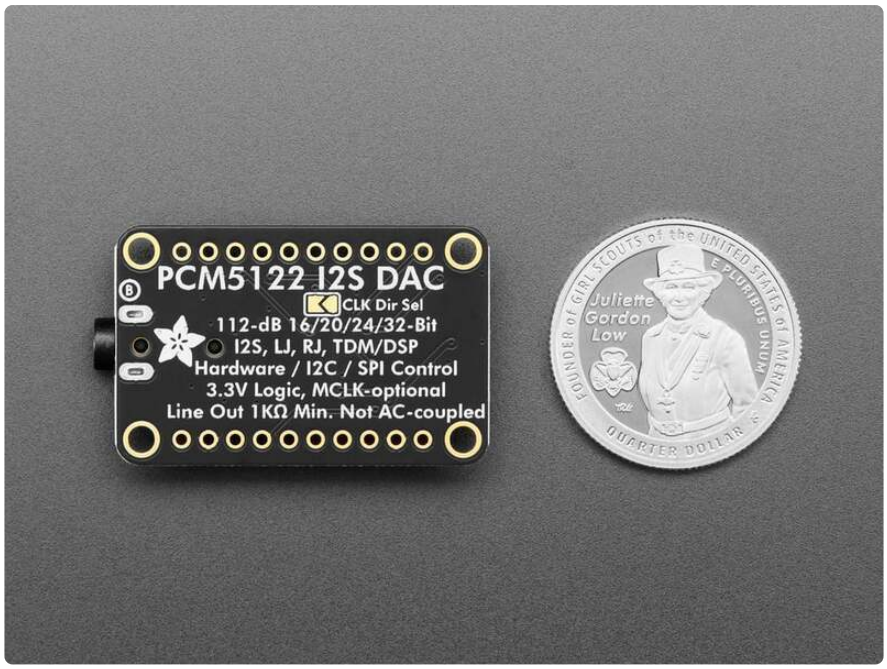
Overview



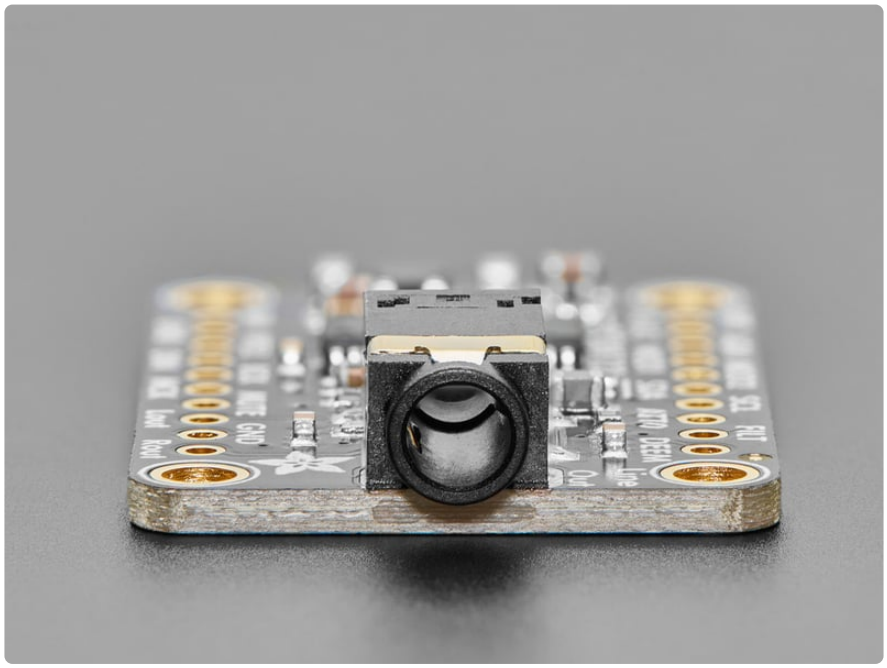
For converting digital I2S audio from your ESP32/RP2350 or Raspberry Pi, you'll need a digital-to-analog-converter (DAC). And the **Adafruit PCM5122 I2S DAC** is both powerful and easy to use - with excellent audio quality! It's got clean, high-quality, stereo audio and does not need any MCLK signal, or I2C configuration. Literally just pipe some I2S audio in and it'll just work.

The default hardware mode is excellent for quick starts, and, for those who do want configurability, such as volume control / software mute / EQ / filters, it's also easy to set up the chip for I2C or SPI interfacing with the two MODE pins.

The PCM5122 has excellent audio specs, with 112dB signal-to-noise/dynamic range, and -93 dB THD.



This breakout makes I2S digital audio easy: all you need to do is power it with 3~5VDC, and provide BCLK (bit clock), WSEL (left/right word select), and DIN (data in). The data lines are 3.3V logic only. By default it's configured for I2S but you can also do Left-Justified by toggling the Format pin. Audio can be 16, 24 or 32-bit wide, the chip will automagically determine the right format from the WSEL / BCLK ratio. No MCLK pin is needed, the chip will auto-generate it internally from the bit clock - or you can provide it on the MCLK input if you want.

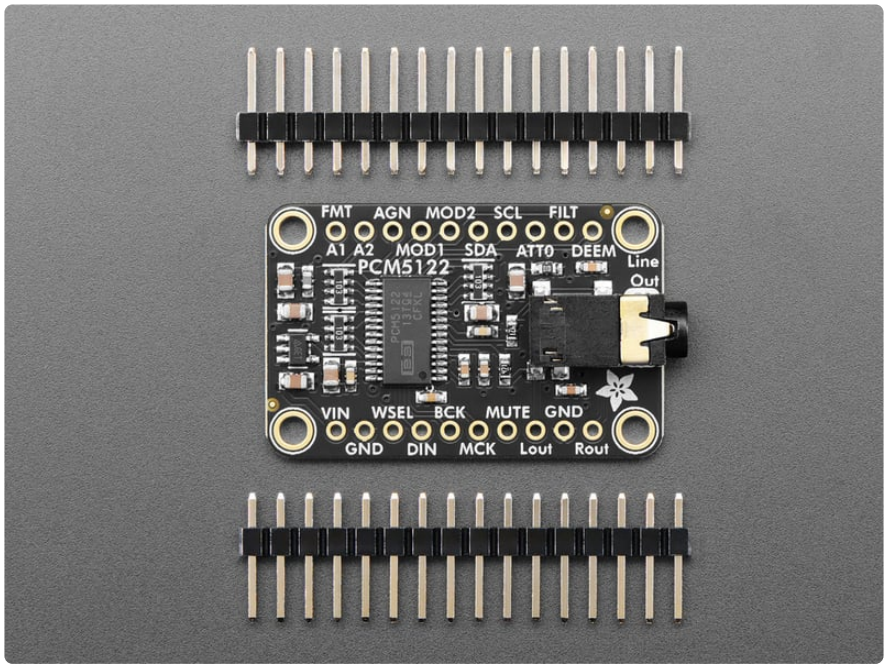


For hardware (not I2C/SPI config) mode, the other breakout pads provide:

- **Filtering** (change from normal to low-latency by pulling high)
- **De-emphasis**

- **Mute** (pull low to quickly set the outputs to ground), and de-emphasis for 44.1khz audio (default is off)
- Three **ATTenuation/gain** pins that can be used for changing the gain from -6dB to +15dB. See the datasheet's Table 3 for the pin-to-gain settings.

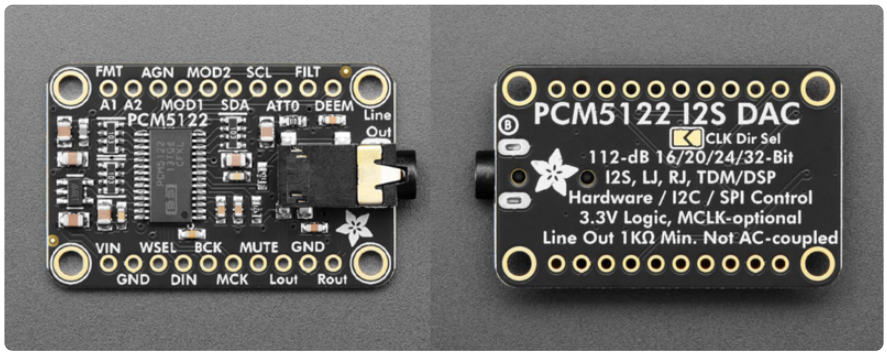
For I2C/SPI configuration mode, gain/volume, filtering and de-emphasis is done over digital register commands. The hardware Mute pin still works as expected.



The audio outputs are also available on breakout pads if you want to wire directly without using the 3.5mm jack. Audio output is not AC-coupled because it is centered on ground: you can plug it into anything that is either AC coupled or has the same ground reference. Note that this is a line-level output, it cannot drive headphones - the output is for no less than 1K ohm loads!

Each order comes with one I2S Stereo DAC breakout and some header you can solder on for breadboard usage.

Pinouts



The default I2C address is **0x4C**.

Power Pins

- **VIN** - this is the power pin. It can be powered with 3.3 to 5VDC, however, the data lines are 3.3V logic only.
- **GND** - common ground for power and logic

The data lines are 3.3V logic level only

I2S Pins

- **WSEL** (Word Select or Left/Right Clock) - this is the pin that tells the DAC when the data is for the left channel and when it's for the right channel.
- **DIN** (Data In) - This is the pin that has the actual data coming in, both left and right data are sent on this pin, the WSEL pin indicates when left or right is being transmitted.
- **BCK** (Bit Clock) - This is the pin that tells the amplifier when to read data on the data pin.
- **MCK** (Main clock, optional) - This pin is optional for the PCM5122 because it will auto-generate the main clock internally from BCK.

Audio Output

The audio output from this breakout is line-level. It is not AC-coupled because it is centered on ground. You can plug it into anything that is either AC coupled or has the same ground reference.

- **Lout** - This is the left channel audio output
- **Rout** - This is the right channel audio output
- **GND** - This is a clean analog ground signal for the audio output
- **3.5mm output jack** - This is the onboard output audio jack. Note that it cannot drive headphones - the output is for no less than 1K ohm loads!

Note that it cannot drive headphones - the output is for no less than 1K ohm loads!

Control Mode Selection

The PCM5122 can be controlled via hardware, I2C or SPI. The control mode is determined by pulling pins **MOD1** and **MOD2** high or low. By default, hardware mode is enabled with both **MOD1** and **MOD2** open. The table below shows the possible combinations to enable the different modes.

Note that in SPI mode, the **MOD2** pin is also the **CS** pin, so it does not need to be pulled specifically high or low.

MODE	MODE1	MODE2
HW	L	L
I2C	L	H
SPI	H	N/A

Hardware Control Pins

These pins can be used in hardware control mode (not I2C/SPI config mode) to affect the DAC.

- **DEEM** - This is the de-emphasis pin for 44.1khz audio. By default it is off.
- **FILT** - This is the filter pin. You can change the filter from normal to low-latency by pulling the pin high.
- **FMT** - This is the format pin. You can change the format from I2S to Left justified by pulling the pin high.
- **AGN** - This is the analog gain selector. When the pin is low, the gain is set to 0 dB. When it is pulled high, the gain is set to -6 dB.
- **ATT0, ATT1 (SCL)** and **ATT2 (SDA)** - These are the gain and attenuation control pins. Note that **ATT1** is labeled **SCL** on the board silk and **ATT2** is labeled **SDA** on the board silk. They set the output level by being pulled high or low. The table below shows the possible combinations to set the different gain levels:

ADDR	ATT0	ATT1	ATT2
0 dB	L	L	L
3 dB	H	L	L
6 dB	L	H	L
9 dB	H	H	L
12 dB	L	L	H
15 dB	H	L	H
-6 dB	L	H	H
-3 dB	H	H	H

The **MUTE pin** can be used to mute the DAC in all modes.

- **MUTE** - This is the mute pin. You can pull this pin low to set the outputs to ground.

I2C Control Pins

These pins are used to control the DAC via I2C.

- **SDA** - the I2C data pin, connect to your microcontroller's I2C data line.
- **SCL** - the I2C clock pin, connect to your microcontroller's I2C clock line.

I2C Address Pins

On the front of the board are **two address pins**, labeled **A1** and **A2**. These pins allow you to change the I2C address to connect multiple boards by connecting them to **VIN**.

The default I2C address is **0x4C**. The other address options can be calculated by "adding" the **A1/A2** to the base of **0x4C**.

A1 sets the lowest bit with a value of **1** and **A2** sets the next bit with a value of **2.** The final address is **0x4C** + **A2** + **A1** which would be **0x4F**.

If only **A1** is pulled high, the address is **0x4C** + **1** = **0x4D**

If only **A2** is pulled high, the address is **0x4C** + **2** = **0x4E**

The table below shows all possible addresses, and whether the pin(s) should be high or low.

ADDR	A1	A2
0x4C	L	L
0x4D	H	L
0x4E	L	H
0x4F	H	H

SPI Control Pins

These pins are used to control the DAC via SPI.

- **A1 (MISO)** - Labeled **A1** on the board silk, this is the SPI **M**icrocontroller **I**n **S**erial **O**ut pin.
- **SDA (MOSI)** - Labeled **SDA** on the board silk, this is the SPI **M**icrocontroller **O**ut **S**erial **I**n pin.
- **SCL (SCK)** - Labeled **SCL** on the board silk, this is the SPI clock input pin.
- **MOD2 (CS)** - Labeled **MOD2** on the board silk, this is the chip select pin.

Clock Direction Jumper

- **CLK Dir Set** - On the back of the board is the I2S clock mode jumper. By default, it is open and expects a clock input from your microcontroller (secondary mode). When soldered closed, it sets the pin high and generates its own clock with a main clock source (primary mode).

CircuitPython - Hardware Mode

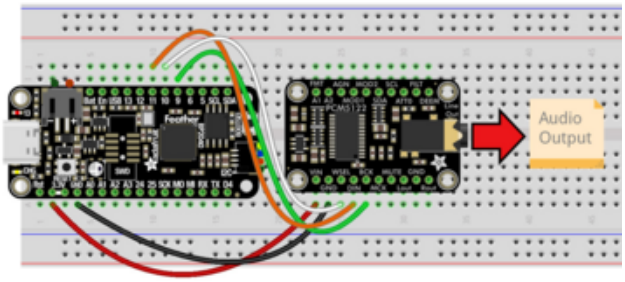
It's easy to use the **PCM5122** in hardware mode with CircuitPython and the the builtin `audiobusio` module. This module allows you to easily write Python code to play audio.

If you want to use this DAC with its I2C driver, you can reference the CircuitPython - I2C Mode (https://adafru.it/1asM) page.

No additional libraries are required for this example.

CircuitPython Microcontroller Wiring

First wire up the I2S DAC to your board exactly as follows. The following is the DAC wired to a Feather RP2040 with the headphone output:



Board 3.3V to **DAC VIN (red wire)**
Board GND to **DAC GND (black wire)**
Board D9 to **DAC BCK (green wire)**
Board D10 to **DAC WSEL (white wire)**
Board D11 to **DAC DIN (orange wire)**

Example Code

Click the **Download Project Bundle** button below to download the **code.py** file in a zip file. Extract the contents of the zip, and copy the **code.py** file to your **CIRCUITPY** drive.

```
# SPDX-FileCopyrightText: Copyright (c) 2025 Liz Clark for Adafruit Industries
#
# SPDX-License-Identifier: MIT

"""
Sine tone playback test for the PCM5122 I2S DAC in hardware mode.
"""

import array
import math
import time

import audiobusio
import audiocore
import board

audio = audiobusio.I2SOut(board.D9, board.D10, board.D11)

tone_volume = 0.5  # Increase this to increase the volume of the tone.
frequency = 440  # Set this to the Hz of the tone you want to generate.
```

```
length = 8000 // frequency
sine_wave = array.array("h", [0] * length)
for i in range(length):
    sine_wave[i] = int((math.sin(math.pi * 2 * i / length)) * tone_volume * (2**15 -
1))
sine_wave_sample = audiocore.RawSample(sine_wave)

while True:
    audio.play(sine_wave_sample, loop=True)
    time.sleep(1)
    audio.stop()
    time.sleep(1)
```

Once the code starts running, you'll begin hearing a one second 440Hz tone, every other second.

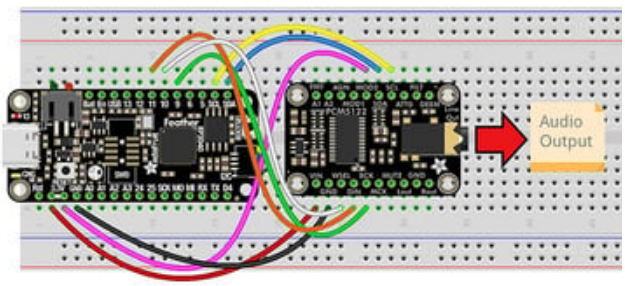
CircuitPython - I2C Mode

It's easy to use the **PCM5122** in I2C mode with CircuitPython, and the Adafruit_CircuitPython_PCM51xx (https://adafru.it/1asD) module. This module allows you to easily write Python code to configure this I2S DAC.

If you want to get up and running quickly without I2C, you can use this DAC in hardware mode as shown on the CircuitPython - Hardware Mode page (https://adafru.it/1asK).

CircuitPython Microcontroller Wiring

First wire up the I2S DAC to your board exactly as follows. The following is the DAC wired to a Feather RP2040 with the headphone output:



Board 3.3V to **DAC VIN (red wire)**
Board GND to **DAC GND (black wire)**
Board SCL to **DAC SCL (yellow wire)**
Board SDA to **DAC SDA (blue wire)**
Board D9 to **DAC BCK (green wire)**
Board D10 to **DAC WSEL (white wire)**
Board D11 to **DAC DIN (orange wire)**
Board 3.3V to **DAC MOD2 (pink wire)**

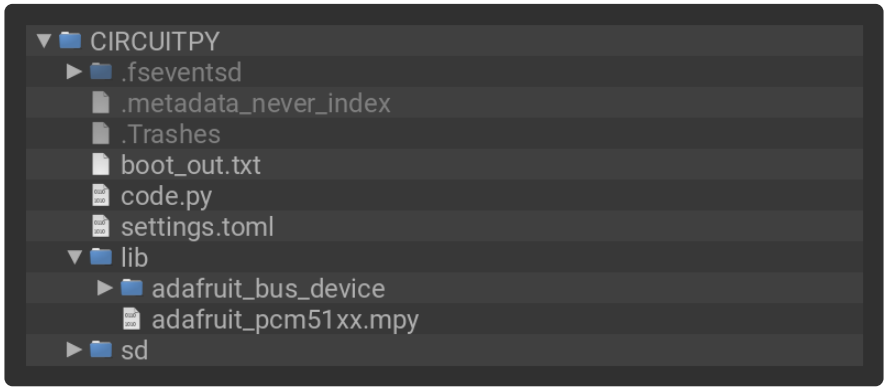
CircuitPython Usage

To use with CircuitPython, you need to first install the **Adafruit_CircuitPython_PCM51xx** library, and its dependencies, into the **lib** folder on your **CIRCUITPY** drive. Then you need to update **code.py** with the example script.

Thankfully, we can do this in one go. In the example below, click the **Download Project Bundle** button below to download the necessary libraries and the **code.py** file in a zip file. Extract the contents of the zip file, and copy the **entire lib folder** and the **code.py** file to your **CIRCUITPY** drive.

Your **CIRCUITPY/lib** folder should contain the following folder and file:

- **adafruit_bus_device/**
- **adafruit_pcm51xx.mpy**



Example Code

```
# SPDX-FileCopyrightText: Copyright (c) 2025 Liz Clark for Adafruit Industries
#
# SPDX-License-Identifier: MIT

"""
Sine tone playback test for the PCM5122 I2S DAC.
"""

import array
import math
import time

import audiobusio
import audiocore
import board
import busio

import adafruit_pcm51xx

# Initialize I2C
i2c = board.I2C()

# Initialize PCM5122
print("Initializing PCM5122...")
pcm = adafruit_pcm51xx.PCM51XX(i2c)
print("Found PCM5122!")

# Set volume to -5dB on both channels
print("\nSetting volume to -5dB")
pcm.volume_db = (-5.0, -5.0)
left_db, right_db = pcm.volume_db
print(f"Volume set to: L={left_db}dB, R={right_db}dB")

# Unmute the DAC
```

```
print("\nUnmuting DAC")
pcm.mute = False
print(f"Muted: {pcm.mute}")

audio = audiobusio.I2SOut(board.D9, board.D10, board.D11)

tone_volume = 0.5  # Increase this to increase the volume of the tone.
frequency = 440  # Set this to the Hz of the tone you want to generate.
length = 8000 // frequency
sine_wave = array.array("h", [0] * length)
for i in range(length):
    sine_wave[i] = int((math.sin(math.pi * 2 * i / length)) * tone_volume * (2**15 -
1))
sine_wave_sample = audiocore.RawSample(sine_wave)

while True:
    audio.play(sine_wave_sample, loop=True)
    time.sleep(1)
    audio.stop()
    time.sleep(1)
```

Once the code starts running, you'll begin hearing a one second 440Hz tone, every other second.

Python Docs

Python Docs (https://adafru.it/1asv)

Arduino - Hardware Mode

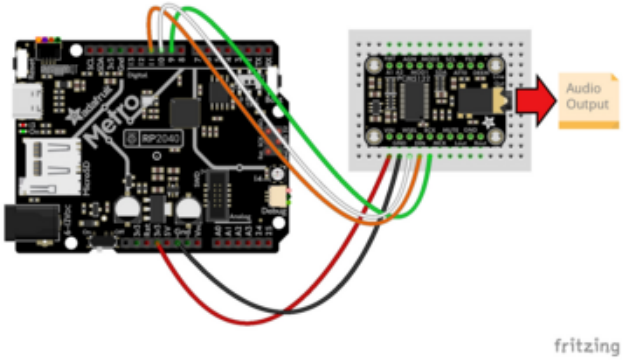
Using the PCM5122 breakout in hardware mode with Arduino involves wiring up the breakout to your Arduino-compatible microcontroller and running the provided example code.

If you want to use this DAC with its I2C driver, you can reference the Arduino - I2C Mode (https://adafru.it/1asN) page.

Wiring

You can power the I2S DAC with 3.3 to 5VDC, however, the data lines are 3.3V logic only. You'll want to use a 3.3V logic level board.

Here is an Adafruit Metro RP2040 wired up to the DAC for hardware control mode:



Board 3.3V to **DAC VIN (red wire)**
Board GND to **DAC GND (black wire)**
Board D9 to **DAC BCK (green wire)**
Board D10 to **DAC WSEL (white wire)**
Board D11 to **DAC DIN (orange wire)**

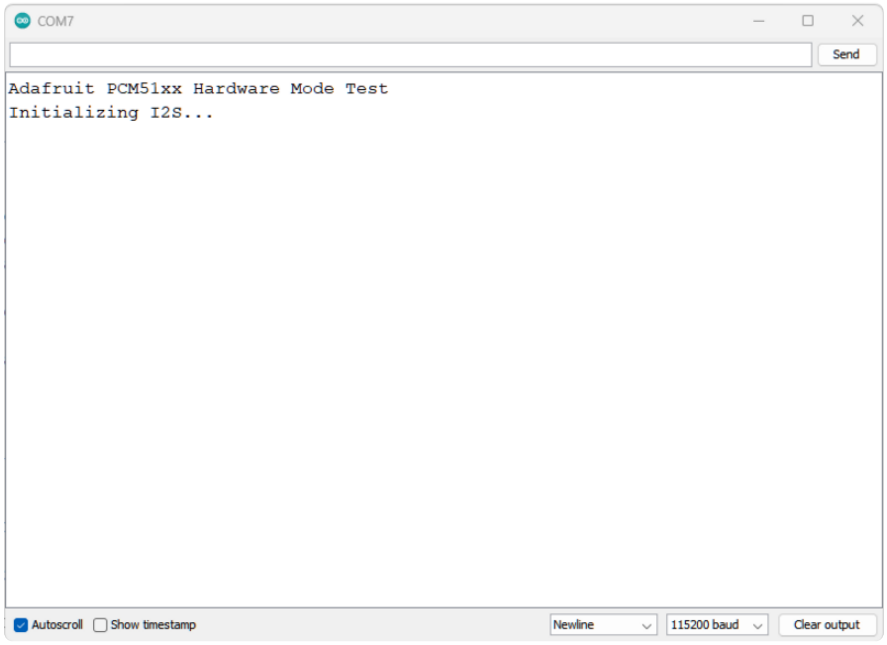
No additional libraries are needed for this example.

Example Code

```
// SPDX-FileCopyrightText: 2025 Ladyada for Adafruit Industries
//
// SPDX-License-Identifier: MIT

#include <I2S.h>
#include <math.h>

#define pBCLK D9   // BITCLOCK - I2S clock
#define pWS   D10  // LRCLOCK - Word select
#define pDOUT D11  // DATA - I2S data

// Create I2S port
I2S i2s(OUTPUT);

const int frequency = 440; // frequency of square wave in Hz
const int amplitude = 500; // amplitude of square wave
const int sampleRate = 16000; // 16 KHz is a good quality

const int halfWavelength = (sampleRate / frequency); // half wavelength of square
wave

int16_t sample = amplitude; // current sample value
int count = 0;

void setup() {
  Serial.begin(115200);
  while (!Serial) delay(10);

  Serial.println(F("Adafruit PCM51xx Hardware Mode Test"));

  // Initialize I2S peripheral
  Serial.println("Initializing I2S...");
  i2s.setBCLK(pBCLK);
  i2s.setDATA(pDOUT);
  i2s.setBitsPerSample(16);

  // Start I2S at the sample rate
```

```
  if (!i2s.begin(sampleRate)) {
    Serial.println("Failed to initialize I2S!");
  }

}

void loop() {
  if (count % halfWavelength == 0) {
    // invert the sample every half wavelength count multiple to generate square
wave
    sample = -1 * sample;
  }

  // write the same sample twice, once for left and once for the right channel
  i2s.write(sample);
  i2s.write(sample);

  // increment the counter for the next sample
  count++;
}
```

Upload the sketch to your board and open up the Serial Monitor (**Tools -> Serial Monitor**) at 115200 baud. You'll see the I2S peripheral initialized. In the loop, a sine tone will play through the 3.5 mm jack output.



Arduino - I2C Mode

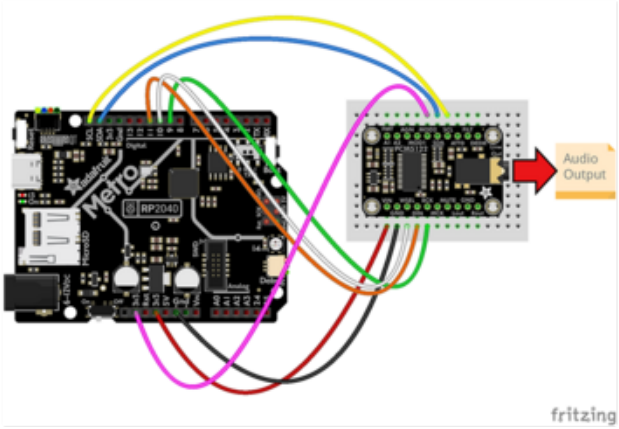
Using the PCM5122 breakout in I2C Mode with Arduino involves wiring up the breakout to your Arduino-compatible microcontroller, installing the Adafruit_PCM51xx (https://adafru.it/1asE) library, and running the provided example code.

If you want to get up and running quickly without I2C, you can use this DAC in hardware mode as shown on the Arduino - Hardware Mode page (https://adafru.it/1asL).

Wiring

You can power the I2S DAC with 3.3 to 5VDC, however, the data lines are 3.3V logic only. You'll want to use a 3.3V logic level board.

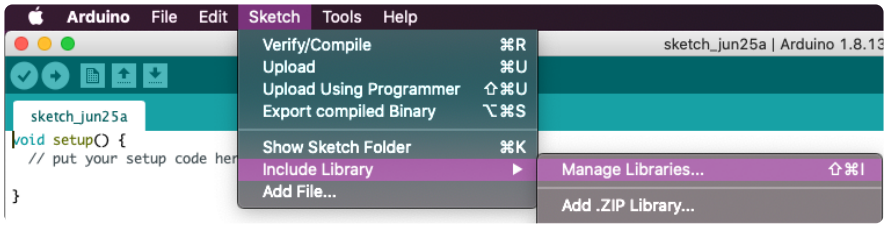
Here is an Adafruit Metro RP2040 wired up to the DAC for I2C control mode:



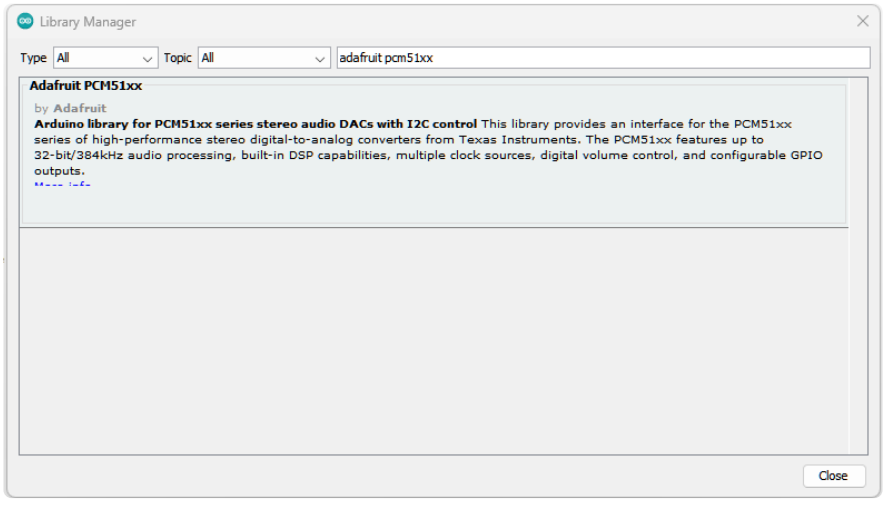
Board 3.3V to **DAC VIN (red wire)**
Board GND to **DAC GND (black wire)**
Board SCL to **DAC SCL (yellow wire)**
Board SDA to **DAC SDA (blue wire)**
Board D9 to **DAC BCK (green wire)**
Board D10 to **DAC WSEL (white wire)**
Board D11 to **DAC DIN (orange wire)**
Board 3.3V to **DAC MOD2 (pink wire)**

Library Installation

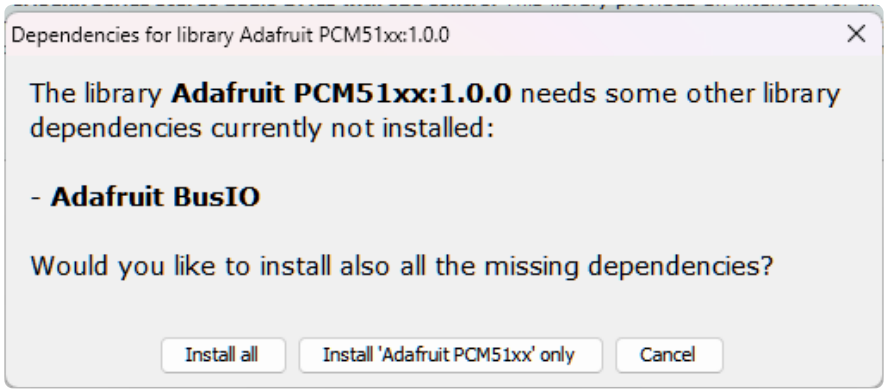
You can install the **Adafruit_PCM51xx** library for Arduino using the Library Manager in the Arduino IDE.



Click the **Manage Libraries ...** menu item, search for **Adafruit_PCM51xx**, and select the **Adafruit PCM51xx** library:



If asked about dependencies, click "Install all".



If the "Dependencies" window does not come up, then you already have the dependencies installed.

> If the dependencies are already installed, you must make sure you update them through the Arduino Library Manager before loading the example!

Example Code

```
// SPDX-FileCopyrightText: 2025 Ladyada for Adafruit Industries
//
// SPDX-License-Identifier: MIT

/*!
 *
 * Basic test example for the Adafruit PCM5122
 *
 * Written by Limor 'ladyada' Fried with assistance from Claude Code
 * for Adafruit Industries.
 *
 * MIT license, all text here must be included in any redistribution.
 */

#include <Adafruit_PCM51xx.h>
#include <I2S.h>
#include <math.h>

Adafruit_PCM51xx pcm;

#define pBCLK D9   // BITCLOCK - I2S clock
#define pWS   D10  // LRCLOCK - Word select
#define pDOUT D11  // DATA - I2S data

// Create I2S port
I2S i2s(OUTPUT);

const int frequency = 440; // frequency of square wave in Hz
const int amplitude = 500; // amplitude of square wave
const int sampleRate = 16000; // 16 KHz is a good quality

const int halfWavelength = (sampleRate / frequency); // half wavelength of square
wave
```

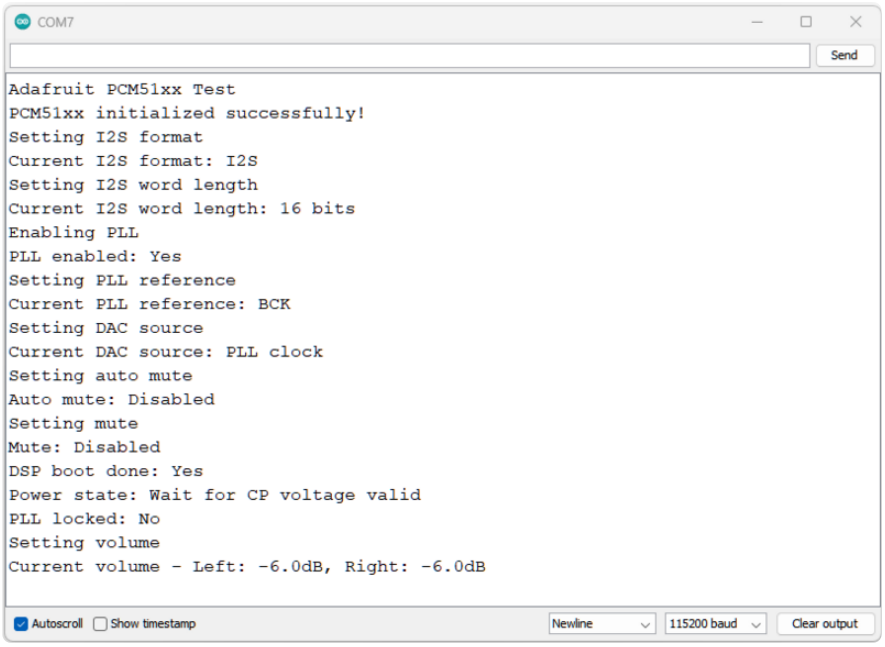
```
int16_t sample = amplitude; // current sample value
int count = 0;

void setup() {
  Serial.begin(115200);
  while (!Serial) delay(10);

  Serial.println(F("Adafruit PCM51xx Test"));

  // I2C mode (default)
  if (!pcm.begin()) {
    Serial.println(F("Could not find PCM51xx, check wiring!"));
    while (1) delay(10);
  }

  // Hardware SPI mode (uncomment to use)
  // if (!pcm.begin(10, &SPI)) {  // CS pin 10
  //   Serial.println(F("Could not find PCM51xx over SPI, check wiring!"));
  //   while (1) delay(10);
  // }

  // Software SPI mode (uncomment to use)
  // if (!pcm.begin(10, 11, 12, 13)) {  // CS, MOSI, MISO, SCLK
  //   Serial.println(F("Could not find PCM51xx over software SPI, check wiring!"));
  //   while (1) delay(10);
  // }

  Serial.println(F("PCM51xx initialized successfully!"));

  // Set I2S format to I2S
  Serial.println(F("Setting I2S format"));
  pcm.setI2SFormat(PCM51XX_I2S_FORMAT_I2S);

  // Read and display current format
  pcm51xx_i2s_format_t format = pcm.getI2SFormat();
  Serial.print(F("Current I2S format: "));
  switch (format) {
    case PCM51XX_I2S_FORMAT_I2S:
      Serial.println(F("I2S"));
      break;
    case PCM51XX_I2S_FORMAT_TDM:
      Serial.println(F("TDM/DSP"));
      break;
    case PCM51XX_I2S_FORMAT_RTJ:
      Serial.println(F("Right Justified"));
      break;
    case PCM51XX_I2S_FORMAT_LTJ:
      Serial.println(F("Left Justified"));
      break;
    default:
      Serial.println(F("Unknown"));
      break;
  }

  // Set I2S word length to 32-bit
  Serial.println(F("Setting I2S word length"));
  pcm.setI2SSize(PCM51XX_I2S_SIZE_16BIT);

  // Read and display current word length
  pcm51xx_i2s_size_t size = pcm.getI2SSize();
  Serial.print(F("Current I2S word length: "));
  switch (size) {
    case PCM51XX_I2S_SIZE_16BIT:
      Serial.println(F("16 bits"));
      break;
    case PCM51XX_I2S_SIZE_20BIT:
      Serial.println(F("20 bits"));
      break;
```

```
    case PCM51XX_I2S_SIZE_24BIT:
      Serial.println(F("24 bits"));
      break;
    case PCM51XX_I2S_SIZE_32BIT:
      Serial.println(F("32 bits"));
      break;
    default:
      Serial.println(F("Unknown"));
      break;
  }

  // Set error detection bits
  if (!pcm.ignoreFSDetect(true) || !pcm.ignoreBCKDetect(true) || !
pcm.ignoreSCKDetect(true) ||
      !pcm.ignoreClockHalt(true) || !pcm.ignoreClockMissing(true) || !
pcm.disableClockAutoset(false) ||
      !pcm.ignorePLLUnlock(true)) {
    Serial.println(F("Error detection failed to configure"));
  }

  // Enable PLL
  Serial.println(F("Enabling PLL"));
  pcm.enablePLL(true);

  // Check PLL status
  bool pllEnabled = pcm.isPLLEnabled();
  Serial.print(F("PLL enabled: "));
  Serial.println(pllEnabled ? F("Yes") : F("No"));

  // Set PLL reference to BCK
  Serial.println(F("Setting PLL reference"));
  pcm.setPLLReference(PCM51XX_PLL_REF_BCK);

  // Read and display current PLL reference
  pcm51xx_pll_ref_t pllRef = pcm.getPLLReference();
  Serial.print(F("Current PLL reference: "));
  switch (pllRef) {
    case PCM51XX_PLL_REF_SCK:
      Serial.println(F("SCK"));
      break;
    case PCM51XX_PLL_REF_BCK:
      Serial.println(F("BCK"));
      break;
    case PCM51XX_PLL_REF_GPIO:
      Serial.println(F("GPIO"));
      break;
    default:
      Serial.println(F("Unknown"));
      break;
  }

  // Set DAC clock source to PLL
  Serial.println(F("Setting DAC source"));
  pcm.setDACSource(PCM51XX_DAC_CLK_PLL);

  // Read and display current DAC source
  pcm51xx_dac_clk_src_t dacSource = pcm.getDACSource();
  Serial.print(F("Current DAC source: "));
  switch (dacSource) {
    case PCM51XX_DAC_CLK_MASTER:
      Serial.println(F("Master clock (auto-select)"));
      break;
    case PCM51XX_DAC_CLK_PLL:
      Serial.println(F("PLL clock"));
      break;
    case PCM51XX_DAC_CLK_SCK:
      Serial.println(F("SCK clock"));
      break;
    case PCM51XX_DAC_CLK_BCK:
```

```
    Serial.println(F("BCK clock"));
    break;
  default:
    Serial.println(F("Unknown"));
    break;
}

// Test auto mute (default turn off)
Serial.println(F("Setting auto mute"));
pcm.setAutoMute(false);

// Read and display current auto mute status
bool autoMuteEnabled = pcm.getAutoMute();
Serial.print(F("Auto mute: "));
Serial.println(autoMuteEnabled ? F("Enabled") : F("Disabled"));

// Test mute (default do not mute)
Serial.println(F("Setting mute"));
pcm.mute(false);

// Read and display current mute status
bool muteEnabled = pcm.isMuted();
Serial.print(F("Mute: "));
Serial.println(muteEnabled ? F("Enabled") : F("Disabled"));

// Check DSP boot status and power state
Serial.print(F("DSP boot done: "));
Serial.println(pcm.getDSPBootDone() ? F("Yes") : F("No"));

pcm51xx_power_state_t powerState = pcm.getPowerState();
Serial.print(F("Power state: "));
switch (powerState) {
  case PCM51XX_POWER_POWERDOWN:
    Serial.println(F("Powerdown"));
    break;
  case PCM51XX_POWER_WAIT_CP_VALID:
    Serial.println(F("Wait for CP voltage valid"));
    break;
  case PCM51XX_POWER_CALIBRATION_1:
  case PCM51XX_POWER_CALIBRATION_2:
    Serial.println(F("Calibration"));
    break;
  case PCM51XX_POWER_VOLUME_RAMP_UP:
    Serial.println(F("Volume ramp up"));
    break;
  case PCM51XX_POWER_RUN_PLAYING:
    Serial.println(F("Run (Playing)"));
    break;
  case PCM51XX_POWER_LINE_SHORT:
    Serial.println(F("Line output short / Low impedance"));
    break;
  case PCM51XX_POWER_VOLUME_RAMP_DOWN:
    Serial.println(F("Volume ramp down"));
    break;
  case PCM51XX_POWER_STANDBY:
    Serial.println(F("Standby"));
    break;
  default:
    Serial.println(F("Unknown"));
    break;
}

// Check PLL lock status
bool pllLocked = pcm.isPLLLocked();
Serial.print(F("PLL locked: "));
Serial.println(pllLocked ? F("Yes") : F("No"));

// Set volume to -6dB on both channels
Serial.println(F("Setting volume"));
```

```
  pcm.setVolumeDB(-6.0, -6.0);

  // Read and display current volume
  float leftVol, rightVol;
  pcm.getVolumeDB(&leftVol, &rightVol);
  Serial.print(F("Current volume - Left: "));
  Serial.print(leftVol, 1);
  Serial.print(F("dB, Right: "));
  Serial.print(rightVol, 1);
  Serial.println(F("dB"));

  // Initialize I2S peripheral
  Serial.println("Initializing I2S...");
  i2s.setBCLK(pBCLK);
  i2s.setDATA(pDOUT);
  i2s.setBitsPerSample(16);

  // Start I2S at the sample rate
  if (!i2s.begin(sampleRate)) {
    Serial.println("Failed to initialize I2S!");
  }

}

void loop() {
  if (count % halfWavelength == 0) {
    // invert the sample every half wavelength count multiple to generate square
wave
    sample = -1 * sample;
  }

  // write the same sample twice, once for left and once for the right channel
  i2s.write(sample);
  i2s.write(sample);

  // increment the counter for the next sample
  count++;
}
```

Upload the sketch to your board and open up the Serial Monitor (**Tools -> Serial Monitor**) at 115200 baud. You'll see the PCM5122 recognized over I2C. Then, you'll see the setup parameters print out to the Serial Monitor. In the loop, a sine tone will play through the 3.5 mm jack output.



Arduino Docs

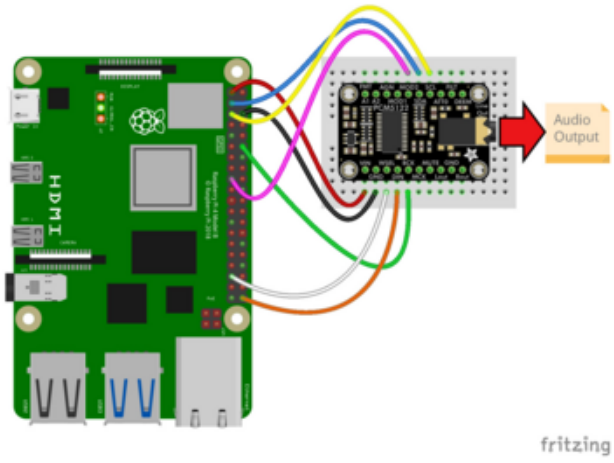
Arduino Docs (https://adafru.it/1asw)

Raspberry Pi

You can use a device tree overlay (`dtoverlay`) in Raspberry Pi OS to use the PCM5122 as the audio output device for your Raspberry Pi. The device tree overlay being used is the `iqaudio-dac` overlay (https://adafru.it/1asF), which uses the same PCM5122 DAC on the default I2C address of **0x4C**.

Wiring

Here's the Raspberry Pi wired up to the DAC:



Pi 3V to **DAC VIN (red wire)**
Pi GND to **DAC GND (black wire)**
Pi SDA to **DAC SDA (blue wire)**
Pi SCL to **DAC SCL (yellow wire)**
Pi GPIO19 to **DAC WSEL (white wire)**
Pi GPIO21 to **DAC DIN (orange wire)**
Pi GPIO18 to **DAC BCK (green wire)**
Pi 3V to **DAC MOD2 (pink wire)**

Prerequisite Pi Setup!

In this page, it's assumed that you have already gotten your Raspberry Pi up and running and can log into the command line.

Here's the quick-start for people with some experience:

1. Download the latest Raspberry Pi OS or Raspberry Pi OS Lite (https://adafru.it/Pf5) to your computer
2. Burn the OS image to your MicroSD card (https://adafru.it/dDL) using your computer
3. Re-plug the SD card into your computer (don't use your Pi yet!) and set up your wifi connection by editing supplicant.conf (https://adafru.it/yuD)
4. Activate SSH support (https://adafru.it/yuD)
5. Plug the SD card into the Pi
6. If you have an HDMI monitor we recommend connecting it so you can see that the Pi is booting OK
7. Plug in power to the Pi - you will see the green LED flicker a little. The Pi will reboot while it sets up so wait a good 10 minutes
8. If you are running Windows on your computer, install Bonjour support so you can use **.local** names, you'll need to reboot Windows after installation (https://adafru.it/IPE)
9. You can then **ssh** into **raspberrypi.local** (https://adafru.it/jvB)

The Pi Foundation has tons of guides as well (https://adafru.it/BJY).

Update the Raspberry Pi

After loading your fresh install, run the standard updates:

```
sudo apt-get update
sudo apt-get upgrade
sudo reboot
```

Add Device Tree Overlay

Edit your Pi configuration file with:

```
sudo nano /boot/firmware/config.txt
```

At the top of the file, uncomment these lines to enable I2C and I2S:

```
dtparam=i2c_arm=on
dtparam=i2s=on
```

Then, at the bottom of the file, add this line:

```
dtoverlay=iqaudio-dac
```

To save the config, press **Ctrl+X** and then **Y** to save your changes. Then press **Enter** to exit the file.

Reboot the Pi with:

```
sudo reboot
```

Testing the DAC

After rebooting, you can test the audio output in the terminal with:

```
speaker-test -c2
```

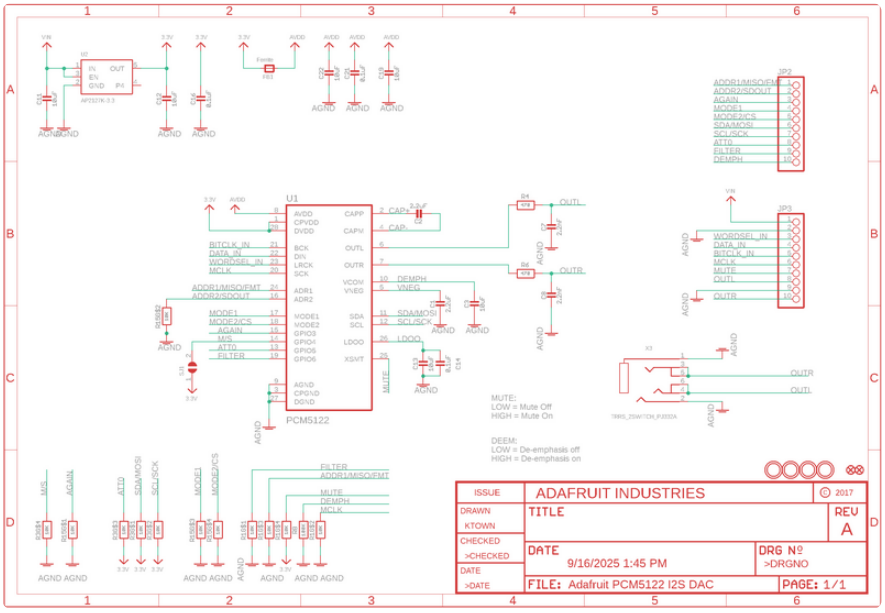
You should hear white noise coming from the output. If you do, then all of that setup worked and you can start to use the DAC as the audio output for your Raspberry Pi.

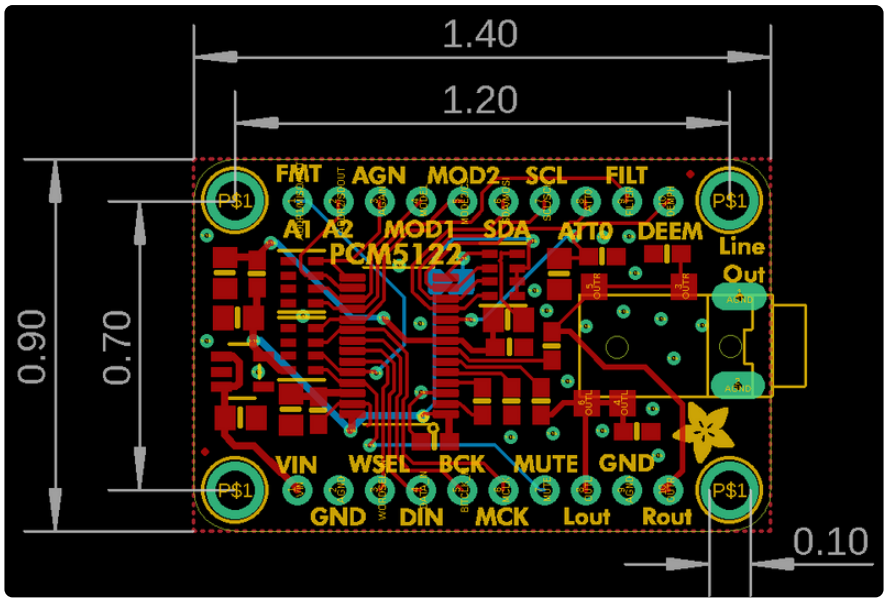
Downloads

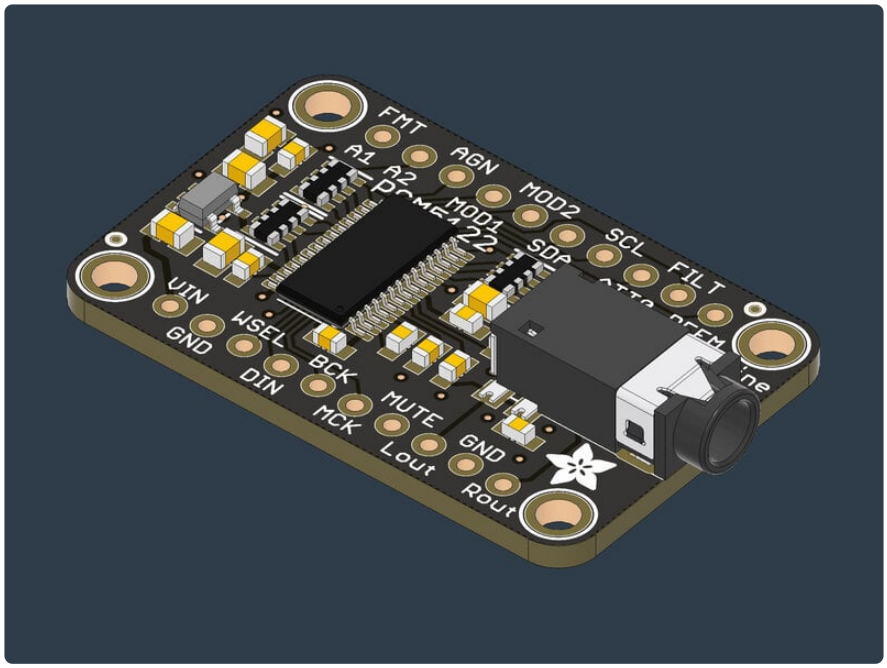
Files

- PCM5122 Datasheet (https://adafru.it/1asG)
- EagleCAD PCB Files on GitHub (https://adafru.it/1asH)
- 3D models on GitHub (https://adafru.it/1asI)
- Fritzing object in the Adafruit Fritzing Library (https://adafru.it/1asJ)

Schematic and Fab Print




1.40
1.20
0.90
0.70
0.10
FMT
AGN
MOD2
SCL
FILT
A1
A2
MOD1
SDA
ATT0
DEEM
PCM5122
Line
Out
VIN
WSEL
BCK
MUTE
GND
GND
DIN
MCK
Lout
Rout
P$1


FMT
A1
AGN
MOD2
SCL
FILT
VIN
WSEL
GND
BCK
DIN
MUTE
MCK
GND
Lout
Rout