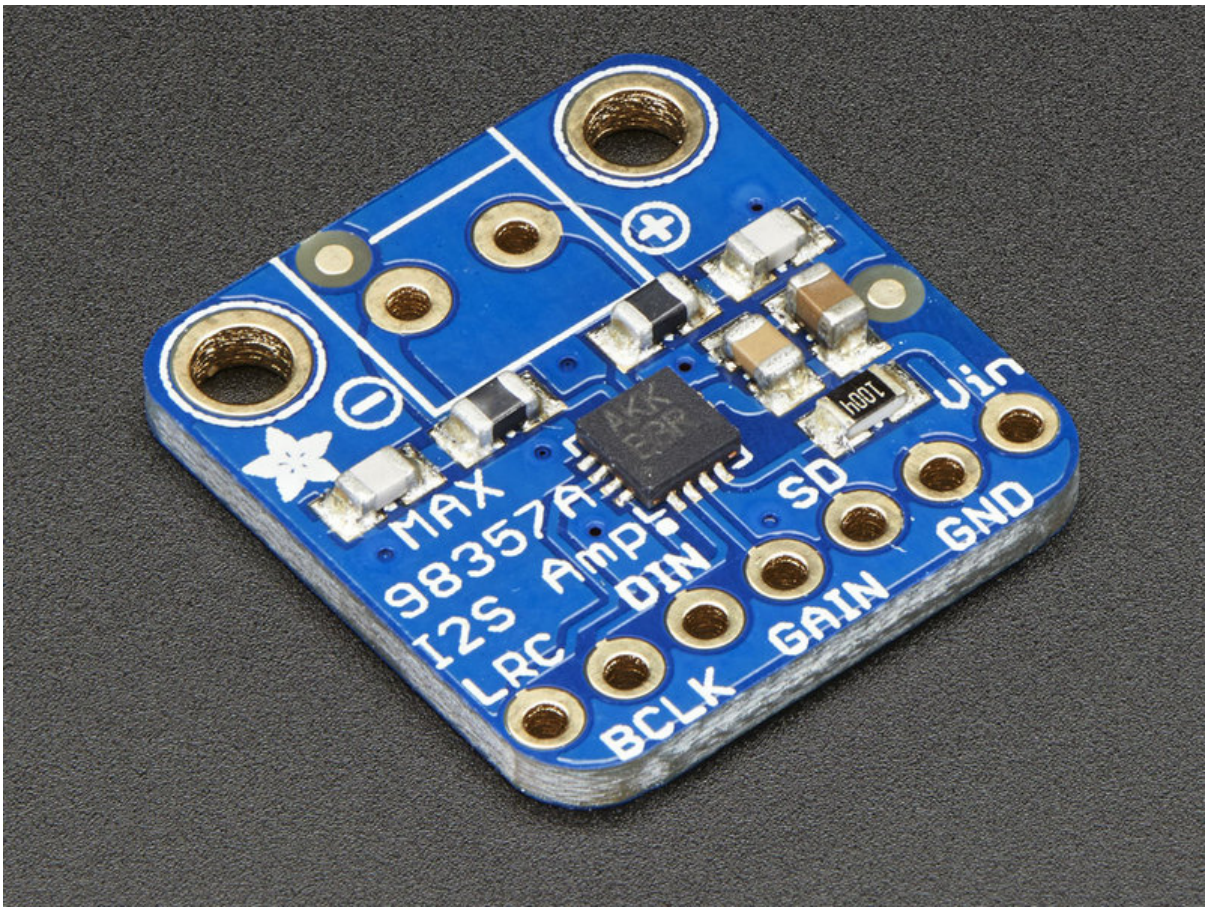
Adafruit MAX98357 I2S Class-D Mono Amp

Created by lady ada



https://learn.adafruit.com/adafruit-max98357-i2s-class-d-mono-amp

Last updated on 2024-03-08 02:26:59 PM EST

Table of Contents

Overview 5

Pinouts 7

- Speaker Output
- Power Pins
- I2S Pins
- Other Pins
- Gain
- SD / MODE

Assembly 12

- Prepare the header strip:
- Add the breakout board:
- And Solder!

Raspberry Pi Wiring 16

Raspberry Pi Setup 17

- Fast Install
- Detailed Install
- Update /etc/modprobe.d (if it exists)
- Disable headphone audio (if it's set)
- Create asound.conf file
- Add Device Tree Overlay

Raspberry Pi Test 24

- Speaker Tests!
- Simple white noise speaker test
- Simple WAV speaker test
- Simple MP3 speaker test
- Volume adjustment

Pi I2S Tweaks 27

- Reducing popping
- Step 1
- Add software volume control

Play Audio with PyGame 31

- Install PyGame
- Run Demo

Arduino Wiring & Test 33

- Wiring
- Basic Test
- DMA Test

CircuitPython Wiring & Test 37

- Wiring
- Code Examples
- Tone Generation
- Wave File

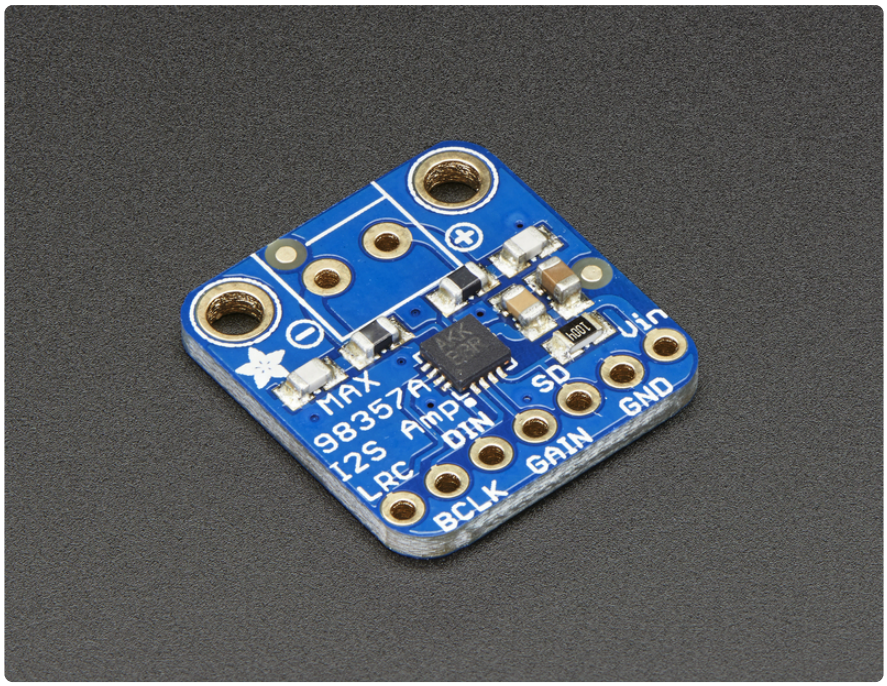
• Where's my I2S?

I2S Audio FAQ 44

Downloads 45

• Schematic
• Fabrication Print
• 3D Model

Overview

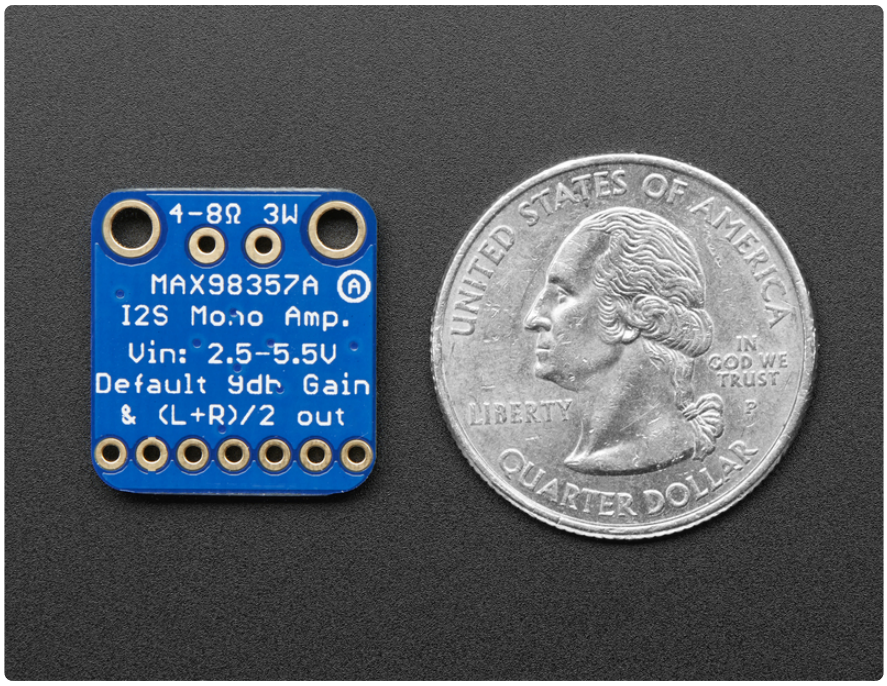


If your microcontroller or microcomputer has digital audio capability, this amp is for you! It takes standard I2S digital audio input and, not only decodes it into analog, but also amplifies it directly into a speaker. Perfect for adding compact amplified sound, it takes 2 breakouts (I2S DAC + Amp) and combines them into one.

I2S (not to be confused with I2C) is a digital sound protocol that is used on circuit boards to pass audio data around. Many high end chips and processors manage all of the audio in digital I2S format. Then, to input or output data, three or four pins are used (data in, data out, bit clock and left-right channel select). Usually, for audio devices, there's a DAC chip that will take I2S in and convert it to analog that can drive a headphone.

This small mono amplifier is surprisingly powerful - able to deliver 3.2 Watts of power into a 4 ohm impedance speaker (5V power @ 10% THD). Inside the miniature chip is a class D controller, able to run from 2.7V-5.5VDC. Since the amp is a class D, it's incredibly efficient - making it perfect for portable and battery-powered projects. It has built in thermal and over-current protection but we could barely tell it got hot.

The audio input is I2S standard, you can use 3.3V or 5V logic data. The outputs are "Bridge Tied" - that means they connect directly to the outputs, no connection to ground. The output is a ~300KHz square wave PWM that is then 'averaged out' by the speaker coil - the high frequencies are not heard. All the above means that you can't connect the output into another amplifier, it should drive the speakers directly.



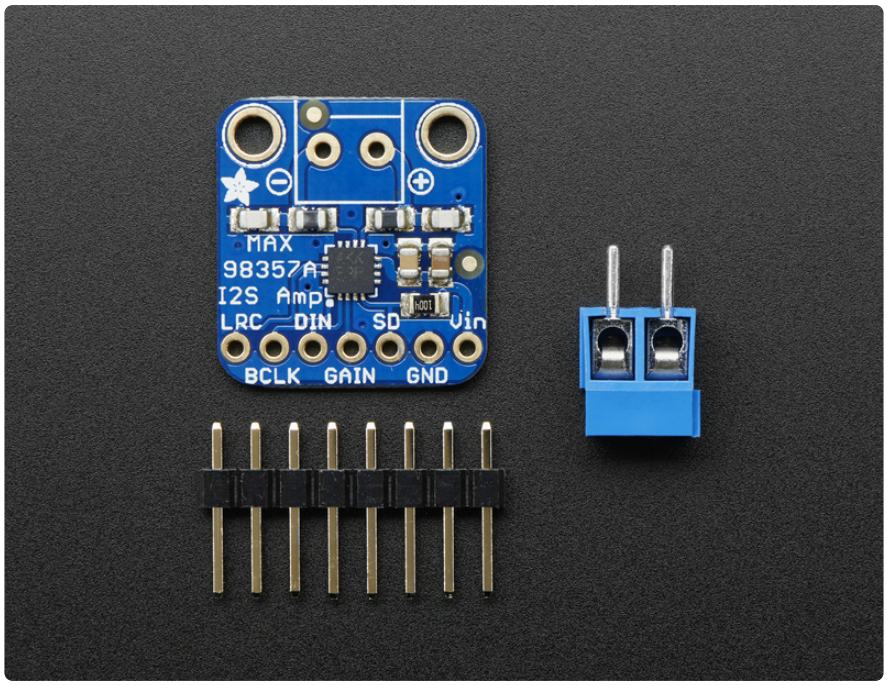
There's a Gain pin that can be manipulated to change the gain. By default, the amp will give you **9dB** of gain. By connecting a pullup or pull down resistor, or wiring directly, the Gain pin can be set up to give 3dB, 6dB, 9dB, 12dB or 15dB.

the ShutDown/Mode pin can be used to put the chip in shutdown or set up which I2S audio channel is piped to the speaker. By default, the amp will output (L+R)/2 stereo mix into mono out. By adding a resistor, you can change it to be just left or just right output

Works great with Raspberry Pi, Arduino Zero, and any other microcontroller or microcomputer with I2S audio outputs

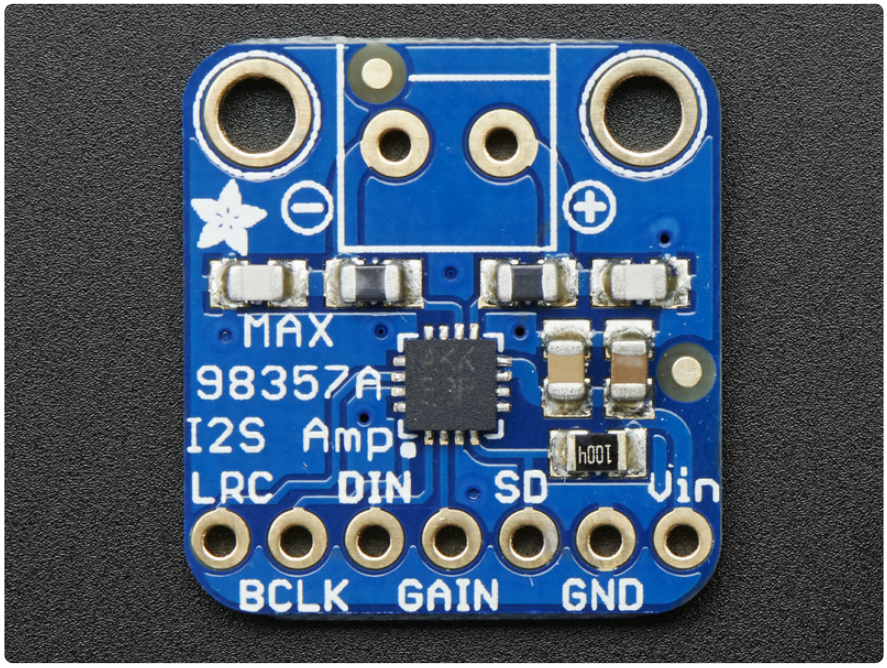
Specs:

- Output Power: 3.2W at 4Ω, 10% THD, 1.8W at 8Ω, 10% THD, with 5V supply
- PSRR: 77 dB typ @ 1KHz
- I2S sample rates from 8kHz to 96kHz
- No MCLK required
- Click + Pop reduction
- Five pin-selectable gains: 3dB, 6dB, 9dB, 12dB, 15dB
- Excellent click-and-pop suppression
- Thermal shutdown protection



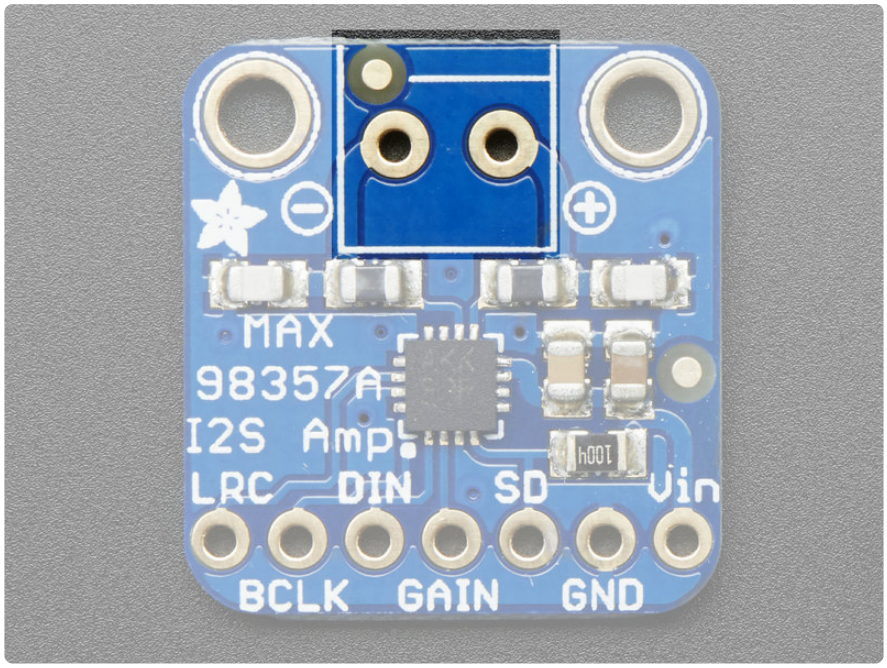
Comes as an assembled and tested breakout board, with a small piece of optional header and 3.5mm terminal block. Some soldering is required to attach the header and terminal block if those are desired.

Pinouts



The MAX98357A is an **I2S** amplifier - it does not use analog inputs, it only has digital audio input support! Don't confuse I2S with I2C, I2S is a sound protocol whereas I2C is for small amounts of data.

Speaker Output



This amplifier is designed to drive moving coil loudpeakers only. Speaker impedence must be 4Ω or more. The output signal is a 330KHz PWM square wave with a duty cycle proportional to the audio signal. The inductance of the speaker coil serves as a low-pass filter to average out the high-frequency components. Do not try to use this as a pre-amplifier.

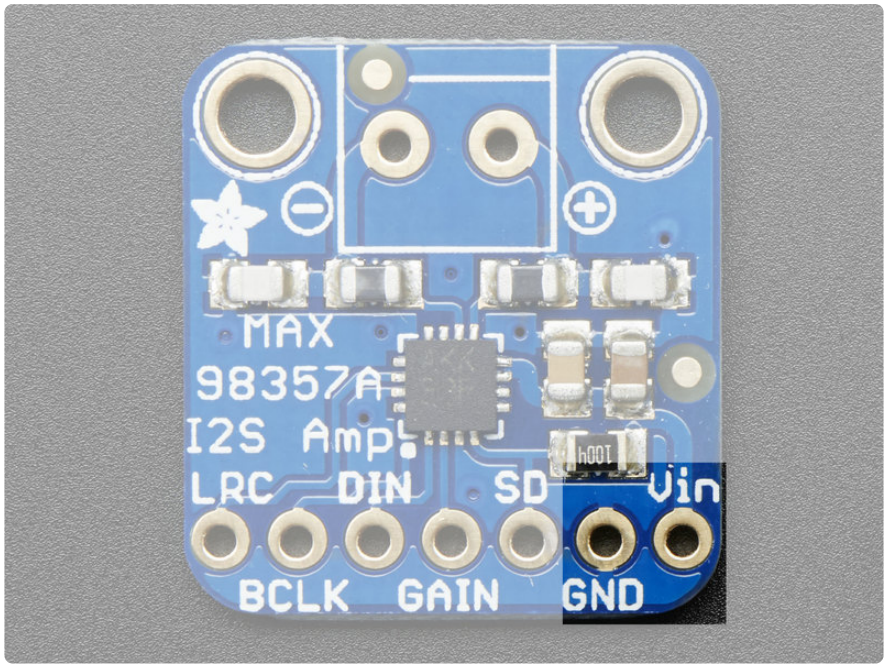
The outputs of each channel are "Bridge-Tied" with no connection to ground. This means that for each channels, the + and - alternate polarity to create a single channel amplifier with twice the available power.

Connect your speakers using the 3.5mm screw-terminal blocks.

- 5V into 4Ω @ 10% THD - 3W max
- 5V into 4Ω @ 1% THD - 2.5W max
- 3.3V into 4Ω @ 10% THD - 1.3W max
- 3.3V into 4Ω @ 1% THD - 1.0W max

- 5V into 8Ω @ 10% THD - 1.8W max
- 5V into 8Ω @ 1% THD - 1.4W max
- 3.3V into 8Ω @ 10% THD - 0.8W max
- 3.3V into 8Ω @ 1% THD - 0.6W max

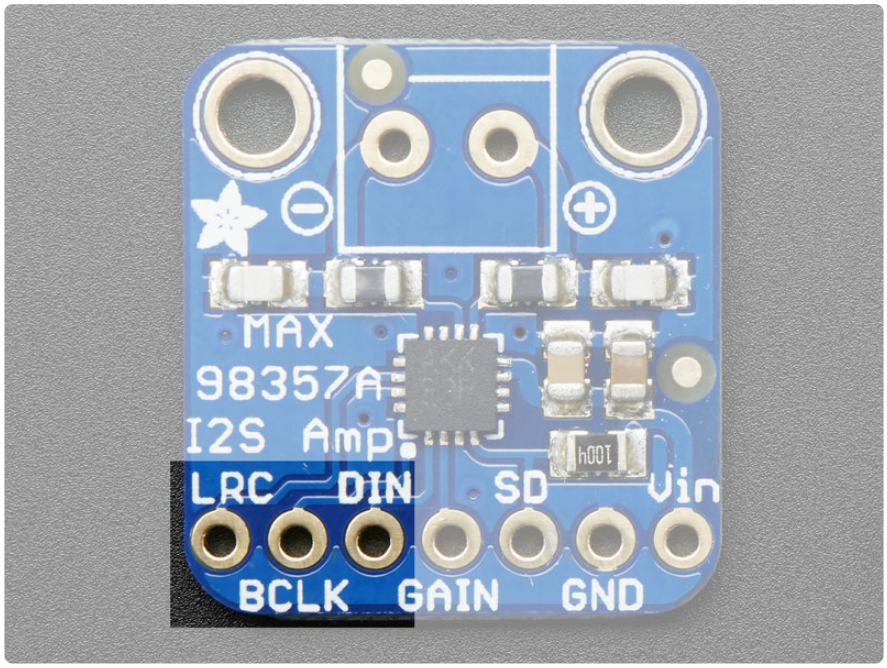
Power Pins



This is the power for the amplifier and logic of the amplifier. You can provide 2.5V up to 5.5V. Note that at 5V you can end up putting up to 2.8W into your speaker, so make sure your power supply can easily handle up to 650mA and we recommend a power supply spec'd for at least 800mA to give yourself some 'room'

If you have a 3.3V logic device, you can still power the amp from 5V, and that's recommended to get the most power output!

I2S Pins

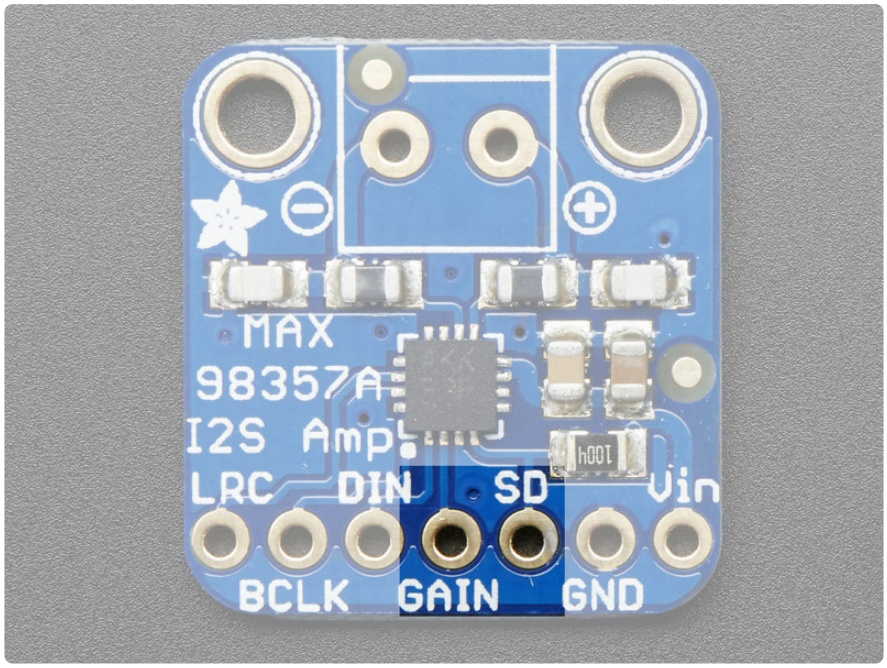


Three pins are used to receive audio data. These can be 3.3-5V logic

- **LRC** (Left/Right Clock) - this is the pin that tells the amplifier when the data is for the left channel and when its for the right channel
- **BCLK** (Bit Clock) - This is the pin that tells the amplifier when to read data on the data pin.
- **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 LRC pin indicates when left or right is being transmitted

Note that this amplifier does not require an **MCLK** pin, if you have an MCLK output, you can leave it disconnected!

Other Pins



The other settings are handled by **GAIN** and **SD**

Gain

GAIN is, well, the gain setting. You can have a gain of **3dB, 6dB, 9dB, 12dB** or **15dB**.

- **15dB** if a 100K resistor is connected between **GAIN** and **GND**
- **12dB** if **GAIN** is connected directly to **GND**
- **9dB** if **GAIN** is not connected to anything (this is the default)
- **6dB** if **GAIN** is connected directly to **Vin**
- **3dB** if a 100K resistor is connected between **GAIN** and **Vin**

This way, the default gain is 9dB but you can easily change it by tweaking the connection to the **GAIN** pin. Note you may need to perform a power reset to adjust the gain.

SD / MODE

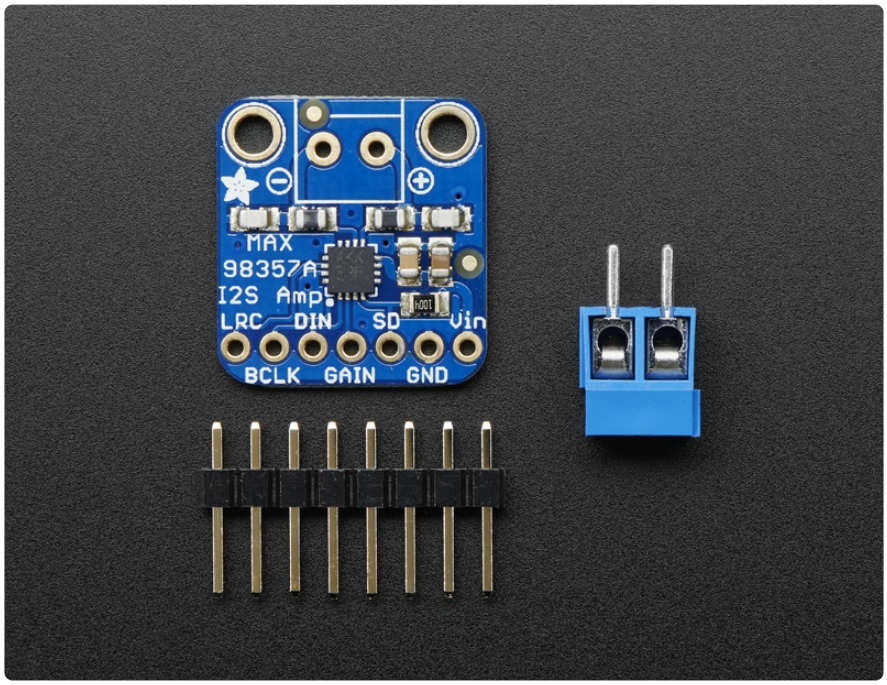
This pin is used for shutdown mode but is also used for setting which channel is output. It's a little confusing but essentially:

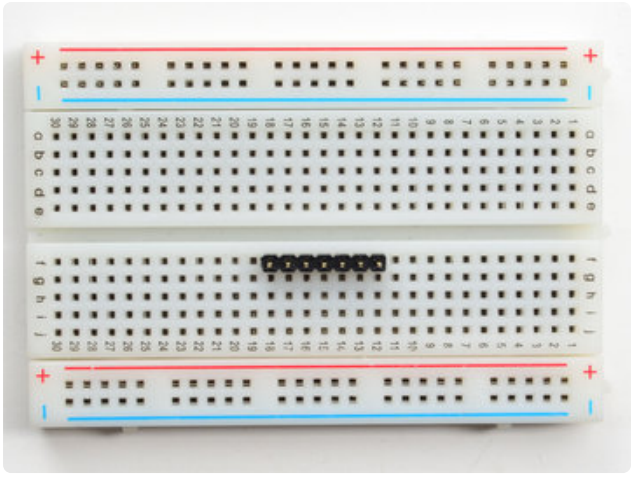
- If **SD** is connected to ground directly (voltage is under 0.16V) then the amp is **shut down**
- If the voltage on **SD** is between 0.16V and 0.77V then the output is (Left + Right)/2, that is the stereo average.
- If the voltage on **SD** is between 0.77V and 1.4V then the output is just the Right channel
- If the voltage on **SD** is higher than 1.4V then the output is the Left channel.

This is compounded by an internal 100K pulldown resistor on **SD** so you need to use a pullup resistor on SD to balance out the 100K internal pulldown.

For the breakout board, there's a 1Mohm resistor from **SD** to **Vin** which, when powering from 5V will give you the 'stereo average' output. If you want left or right channel only, or if you are powering from non-5V power, you may need to experiment with different resistors to get the desired voltage on **SD**

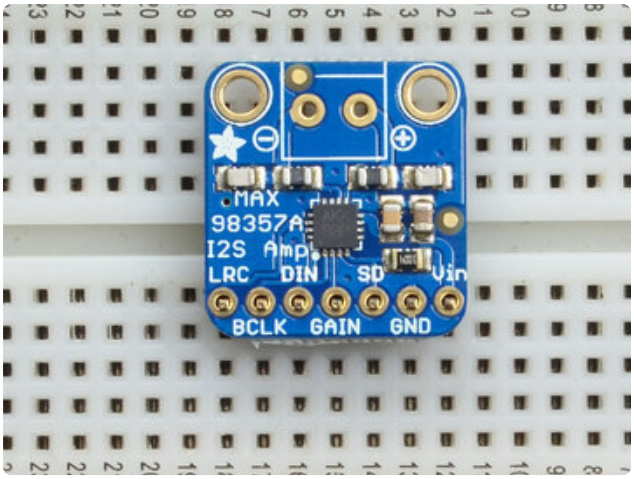
Assembly





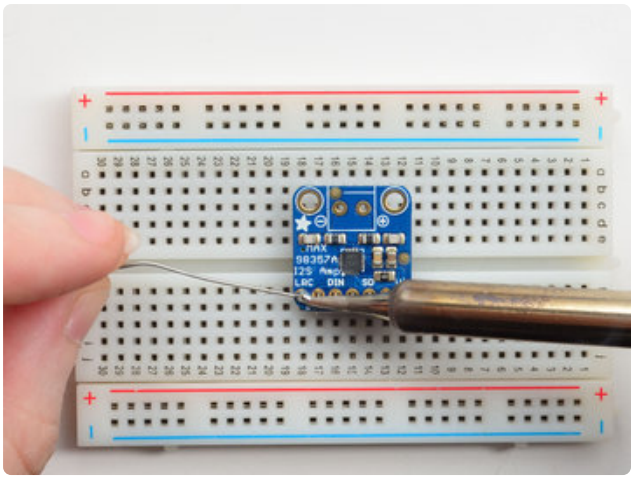
Prepare the header strip:

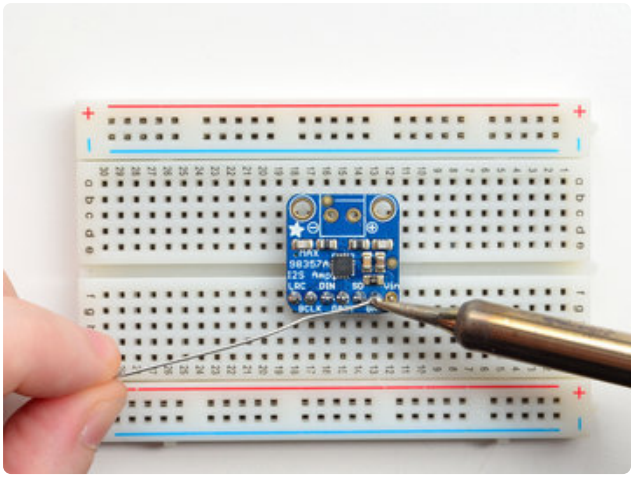
Cut the strip to length if necessary. It will be easier to solder if you insert it into a breadboard - **long pins down**

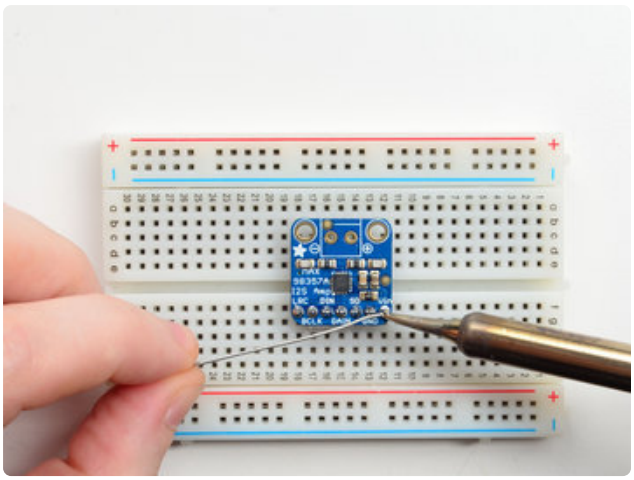


Add the breakout board:

Place the breakout board over the pins so that the short pins poke through the breakout pads



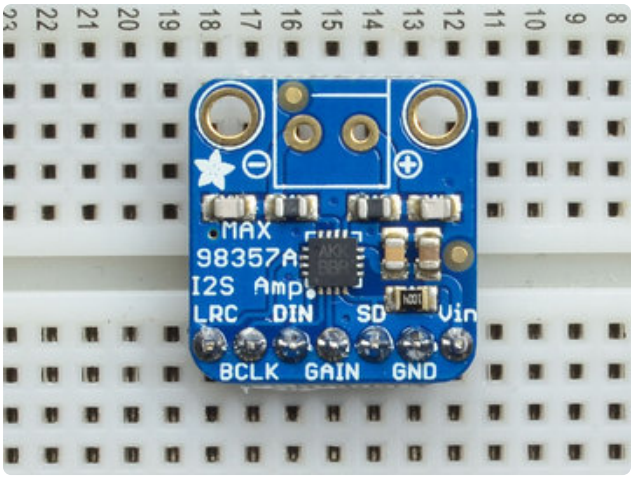




And Solder!

Be sure to solder all pins for reliable electrical contact.

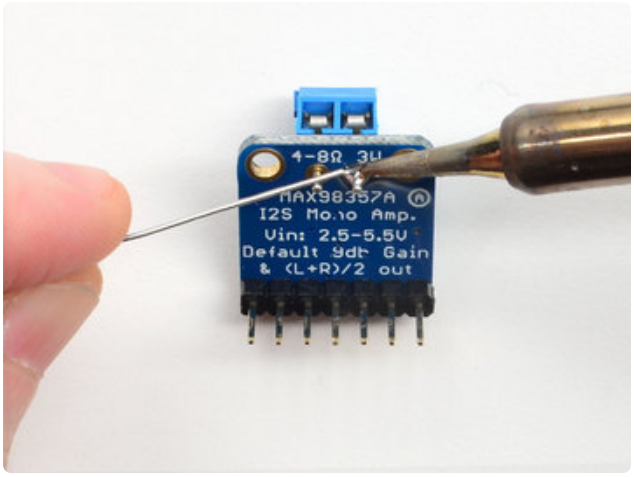
(For tips on soldering, be sure to check out our Guide to Excellent Soldering (https://adafru.it/aTk)).

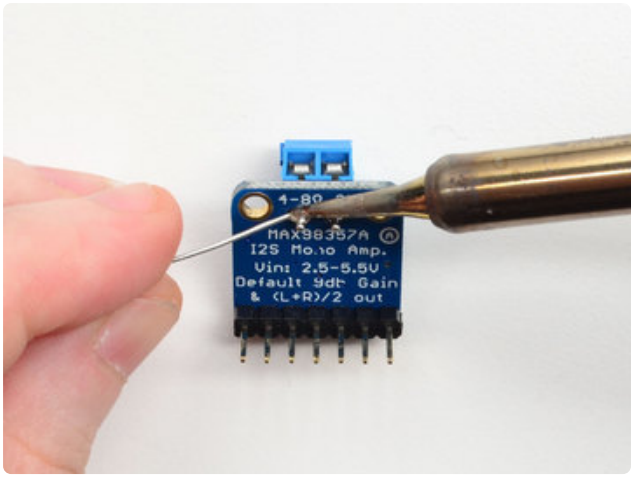


You're done! Check your solder joints visually and continue onto the next steps



If you want to use a terminal block for connecting a speaker, place the 3.5mm terminal so the mouthes point out.





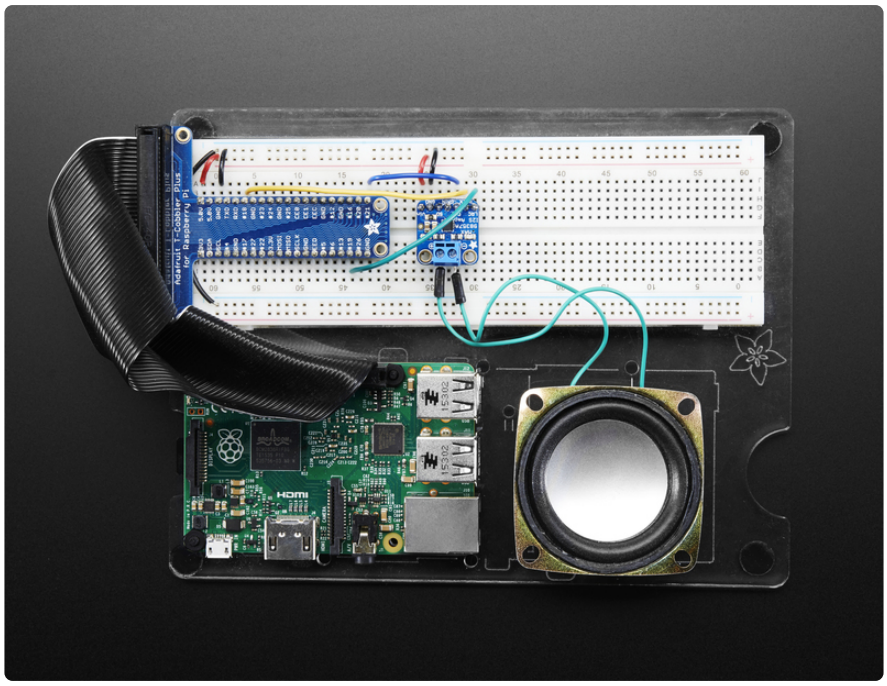
Solder in both pins with plenty of solder!

Raspberry Pi Wiring

if you have a Raspberry Pi and you want higher quality audio than the headphone jack can provide, I2S is a good option! You only use 3 pins, and since its a pure-digital output, there can be less noise and interference. Of course, you'll need to make sure that you have a nice strong 5V power supply so make sure to add 500mA or more to your power supply requirements!

This board also works very well with boards that don't have audio like the Pi Zero

This technique will work with any Raspberry Pi with the 2x20 connector. Older Pi 1's with a 2x13 connector do not bring out the I2S pins as easily



Connect:

- **Amp Vin** to Raspbery Pi **5V Power**
- **Amp GND** to Raspbery Pi **Ground**
- **Amp DIN** to Raspbery Pi **GPIO 21**
- **Amp BCLK** to Raspbery Pi **GPIO 18**
- **Amp LRCLK** to Raspbery Pi **GPIO 19**

Check Raspberry Pi documentation for which pins are GPIO 18, 19, 21, 5V and GND (https://adafru.it/112A)

Raspberry Pi Setup

At this time, Raspbery Pi linux kernel does not support mono audio out of the I2S interface, you can only play stereo, so any mono audio files may need conversion to stereo!

2017-11-2 Raspbian PIXEL ('full') has broken something in volume control. I2S works, but there's no software volume setup, if you need this, try Raspbian Lite - will try to fix as soon as we figure out why :)

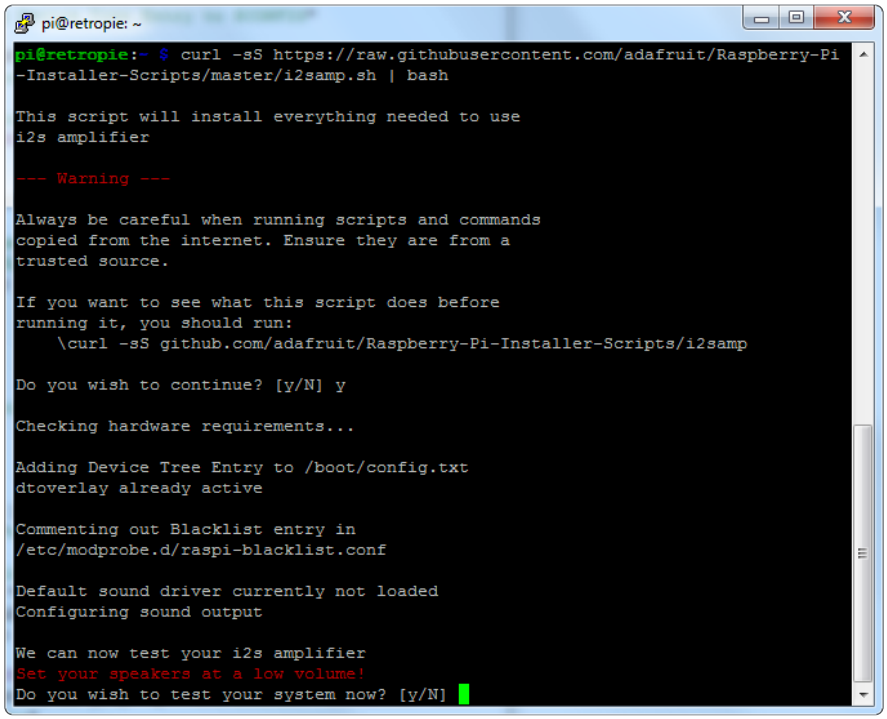
Fast Install

Luckily its quite easy to install support for I2S DACs on Raspbian.

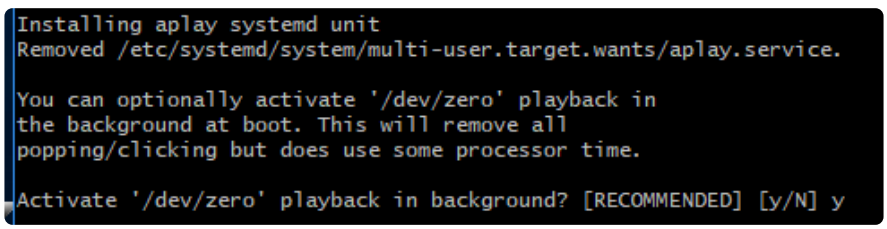
These instructions are totally cribbed from the PhatDAC instructions at the lovely folks at Pimoroni! (https://adafru.it/nFy)

Run the following from your Raspberry Pi with Internet connectivity:

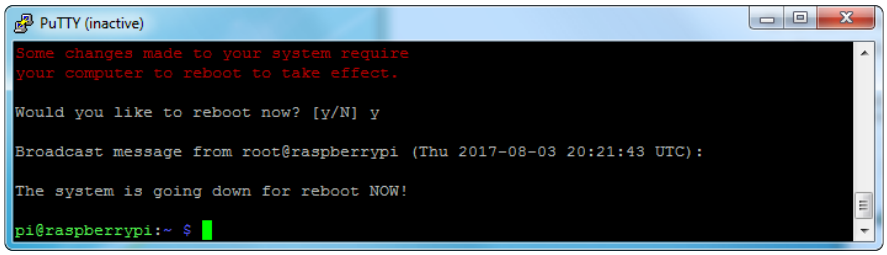
```
curl -sS https://raw.githubusercontent.com/adafruit/Raspberry-Pi-Installer-Scripts/main/i2samp.sh | bash
```



We've added an extra helper systemd script that will play quiet audio when the I2S peripheral isn't in use. This removes popping when playback starts or stops. It uses a tiny amount of CPU time (on a Pi Zero, 5%, on a Pi 2 or 3 its negligible). You don't need this on RetroPie because it never releases the I2S device, but it's great for Raspbian.

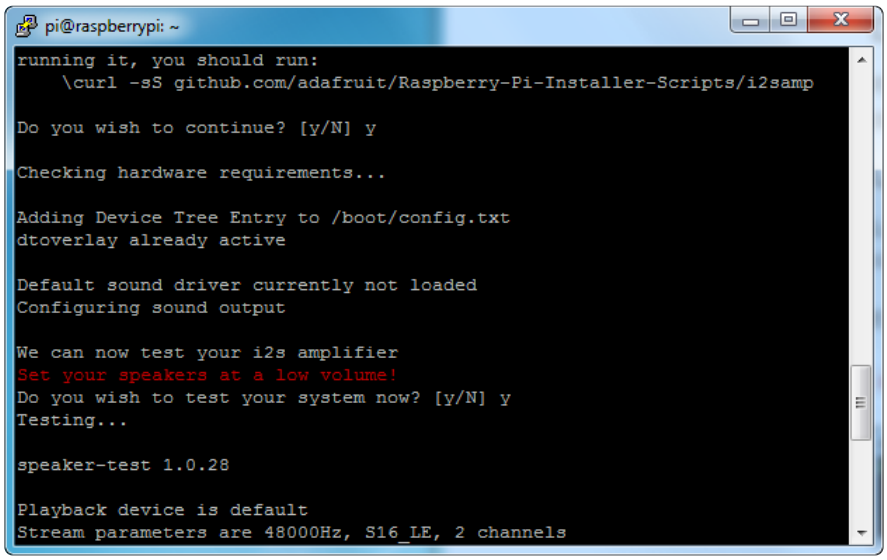


You will need to reboot once installed.



You must reboot to enable the speaker hardware!

After rebooting, log back in and re-run the script again...It will ask you if you want to test the speaker. Say **y**es and listen for audio to come out of your speakers...



If it sounds really distorted, it could be the volume is too high. However, in order to have volume control appear in Raspbian desktop or Retropie you must reboot a second time after doing the speaker test, with **sudo reboot**

You must reboot *twice* to enable alsamixer volume (really!)

Once rebooted, try running **alsamixer** and use arrow keys to lower the volume, 50% is a good place to start.

If you're still having audio problems, try re-running the script and saying **N** (disable) the `/dev/zero playback service`.

You can then go to the next page on testing and optimizing your setup. Skip the rest of this page on **Detailed Installation** if the script worked for you!

Detailed Install

If, for some reason, you can't just run the script and you want to go through the install by hand - here's all the steps!

Update /etc/modprobe.d (if it exists)

Log into your Pi and get into a serial console (either via a console cable, the TV console, RXVT, or what have you)

Edit the raspi blacklist with

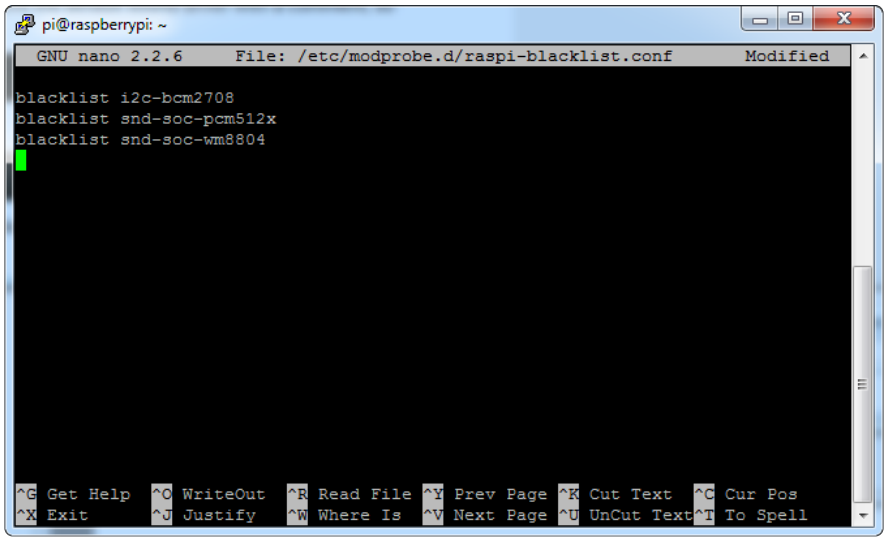
```
sudo nano /etc/modprobe.d/raspi-blacklist.conf
```



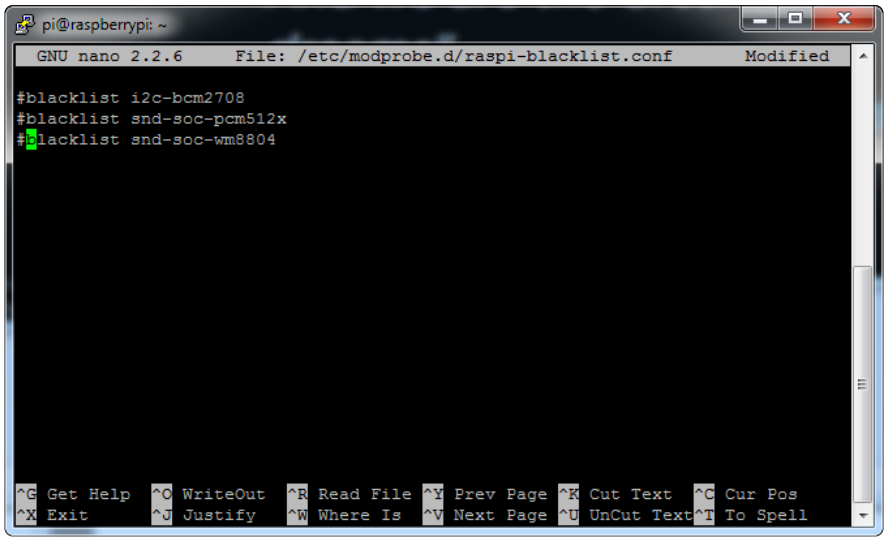
If the file is empty, just skip this step

However, if you see the following lines:

```
blacklist i2c-bcm2708
blacklist snd-soc-pcm512x
blacklist snd-soc-wm8804
```



Update the lines by putting a # before each line



Save by typing **Control-X Y <return>**

Disable headphone audio (if it's set)

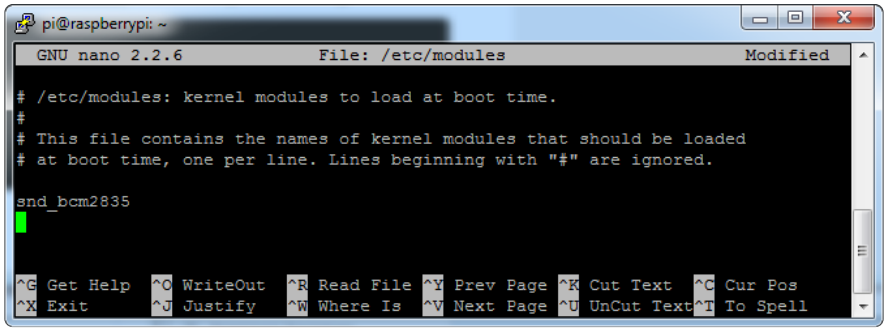
Edit the raspi modules list with

```
sudo nano /etc/modules
```

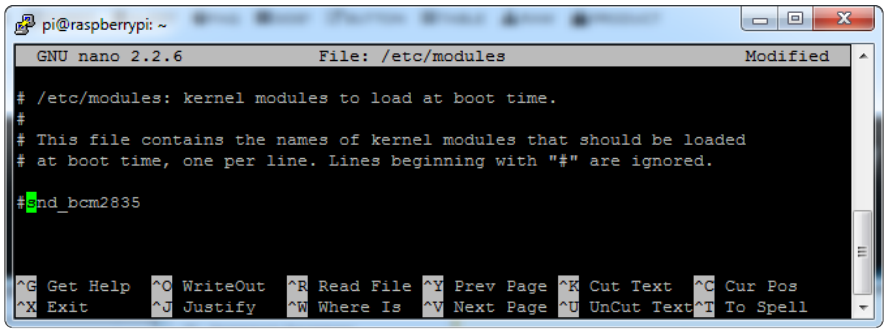
If the file is empty, just skip this step

However, if you see the following line:

```
snd_bcm2835
```



Put a # in front of it



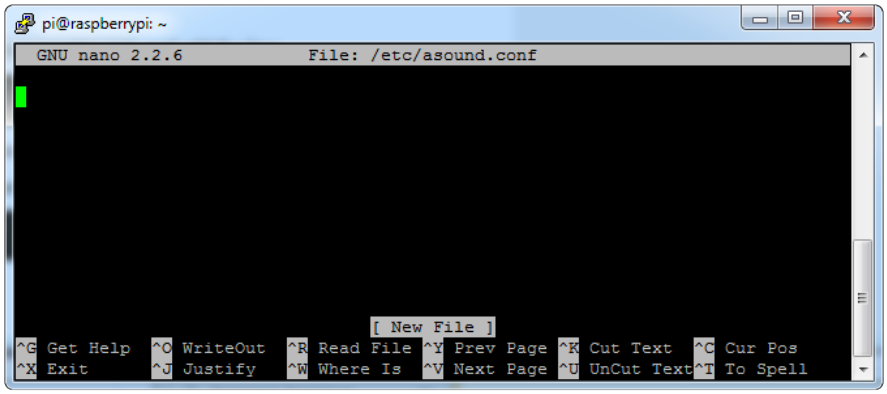
and save with **Control-X Y <return>**

Create asound.conf file

Edit the raspi modules list with

```
sudo nano /etc/asound.conf
```

This file ought to be blank!



Copy and paste the following text into the file

```
pcm.speakerbonnet {
   type hw card 0
}

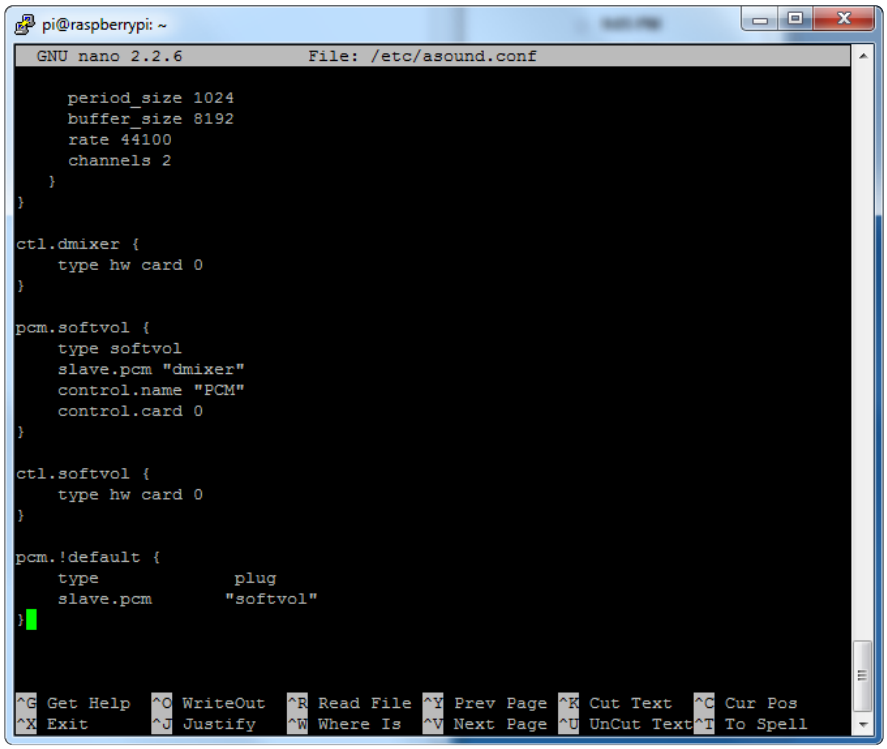
pcm.dmixer {
   type dmix
   ipc_key 1024
   ipc_perm 0666
   slave {
     pcm "speakerbonnet"
     period_time 0
     period_size 1024
     buffer_size 8192
     rate 44100
     channels 2
   }
}

ctl.dmixer {
    type hw card 0
}
```

```
pcm.softvol {
    type softvol
    slave.pcm "dmixer"
    control.name "PCM"
    control.card 0
}

ctl.softvol {
    type hw card 0
}

pcm.!default {
    type             plug
    slave.pcm       "softvol"
}
```



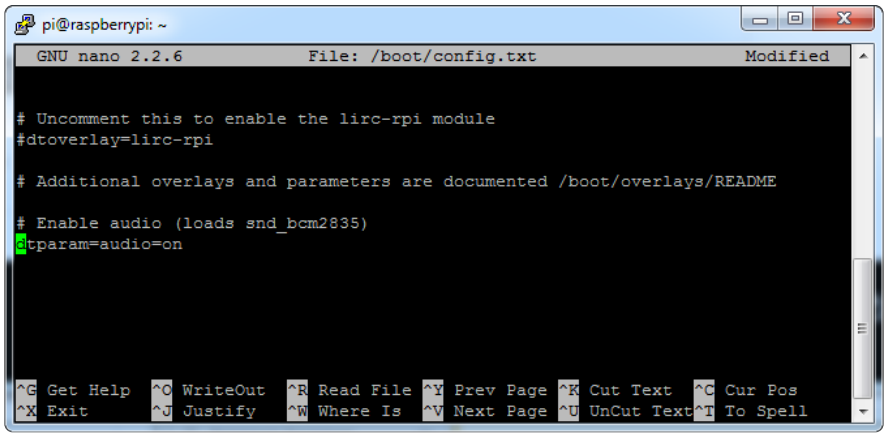
Save the file as usual

Add Device Tree Overlay

Edit your Pi configuration file with

`sudo nano /boot/config.txt`

And scroll down to the bottom. If you see a line that says: `dtparam=audio=on`

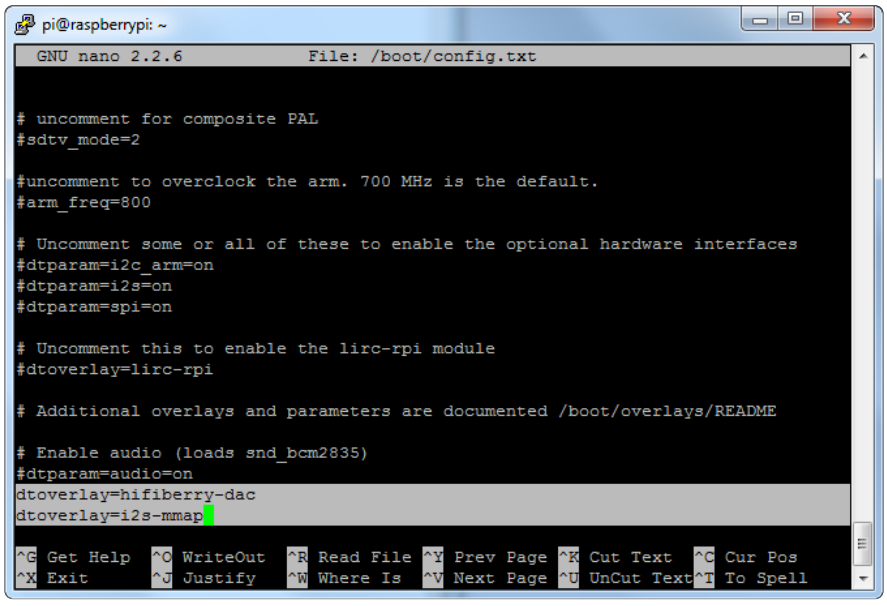


Disable it by putting a # in front.

Then add:

```
dtoverlay=hifiberry-dac
dtoverlay=i2s-mmap
```

on the next line. Save the file.



Reboot your Pi with **sudo reboot**

Raspberry Pi Test

Speaker Tests!

OK you can use whatever software you like to play audio but if you'd like to test the speaker output, here's some quick commands that will let you verify your amp and speaker are working as they should!

Simple white noise speaker test

Run `speaker-test -c2` to generate white noise out of the speaker, alternating left and right.

If you have a mono output amplifier, the I2S amp merges left and right channels, so you'll hear continuous white noise

Simple WAV speaker test

Once you've got something coming out, try to play an audio file with **speaker-test** (for WAV files, not MP3)

```
speaker-test -c2 --test=wav -w /usr/share/sounds/alsa/Front_Center.wav
```

You'll hear audio coming from left and right alternating speakers

Simple MP3 speaker test

If you want to play a stream of music, you can try

```
sudo apt-get install -y mpg123
mpg123 http://ice1.somafm.com/u80s-128-mp3
```

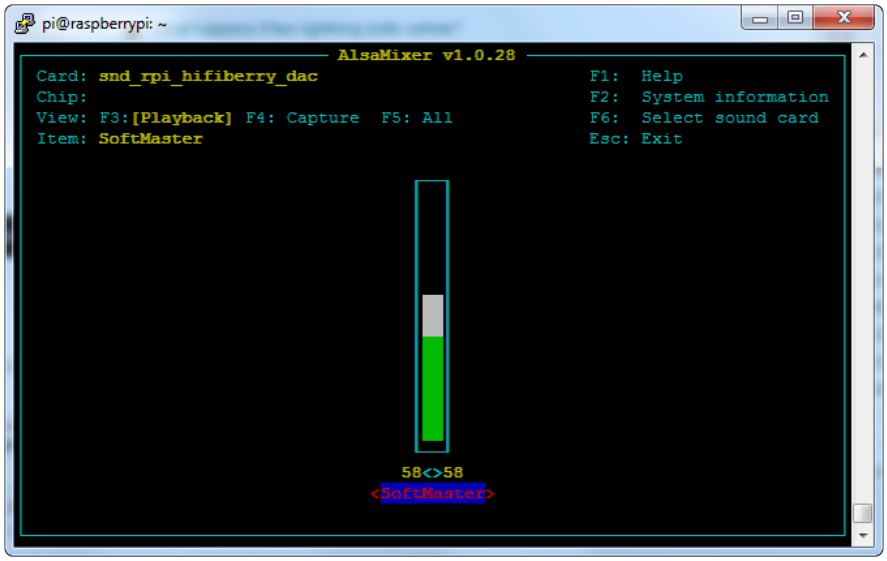
If you want to play MP3's on command, check out this tutorial which covers how to set that up (https://adafru.it/aTD)

At this time, Jessie Raspbery Pi kernel **does not support mono audio** out of the I2S interface, **you can only play stereo**, so any mono audio files may need conversion to stereo!

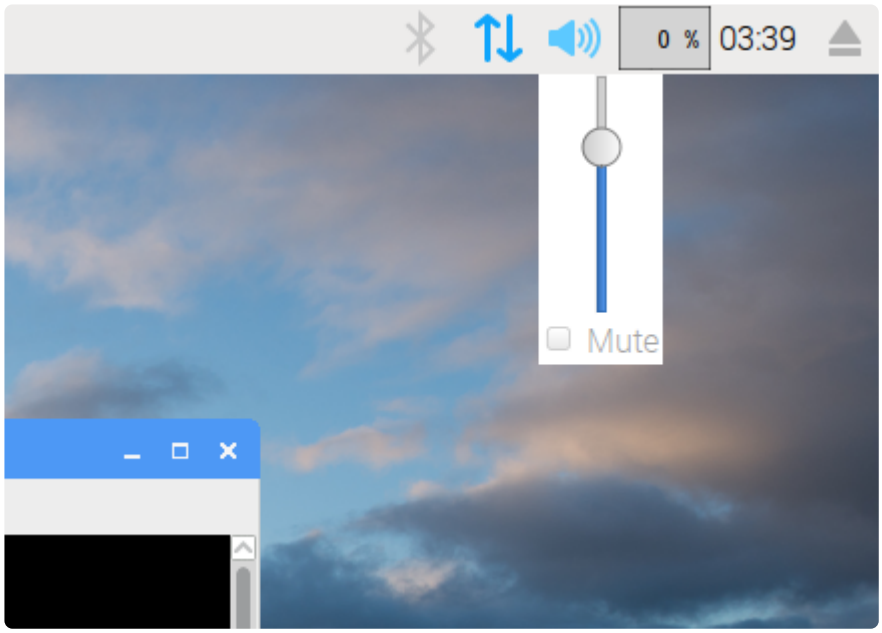
omxplayer does not seem use the I2S interface for audio - only HDMI - so you won't be able to use it

Volume adjustment

Many programs like PyGame and Sonic Pi have volume control within the application. For other programs you can set the volume using the command line tool called **alsamixer**. Just type alsamixer in and then use the up/down arrows to set the volume. Press Escape once its set



In Raspbian PIXEL you can set the volume using the menu item control. If it has an X through it, try restarting the Pi (you have to restart twice after install to get PIXEL to recognize the volume control



Pi I2S Tweaks

This page is deprecated, our installer already performs these steps for you, but we'll keep them here for archival use!

Reducing popping

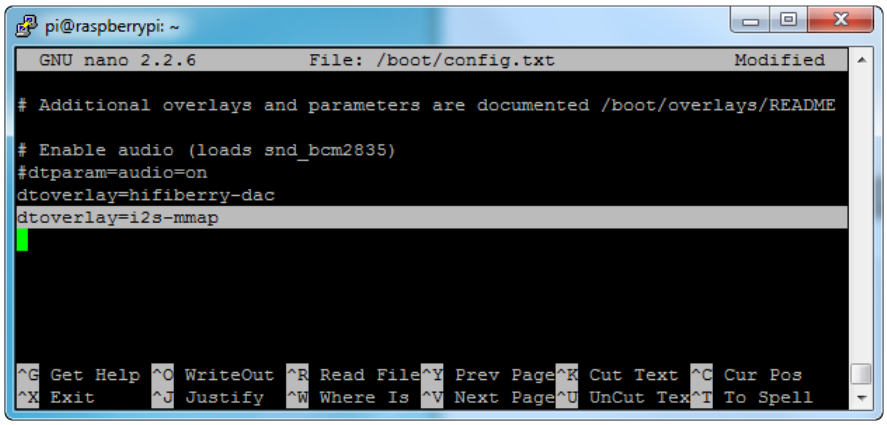
For people who followed our original installation instructions with the simple alsa config, they may find that the I2S audio pops when playing new audio.

The workaround is to use a software mixer to output a fixed sample rate to the I2S device so the bit clock does not change. I use ALSA so I configured **dmixer** and I no longer have any pops or clicks. Note that the RaspPi I2S driver does not support **dmixer** by default and you must follow these instructions provided (https://adafru.it/sHF) to add it. Continue on for step-by-step on how to enable it!

Step 1

Start by modify **/boot/firmware/config.txt** to add `dtoverlay=i2s-mmap`

Run **sudo nano /boot/firmware/config.txt** and add the text to the bottom like so:



If you are using an older version of Raspberry Pi OS and there is no firmware folder, you may need to edit the /boot/config.txt file.

Save and exit.

Then change **/etc/asound.conf** to:

```
pcm.speakerbonnet {
   type hw card 0
}

pcm.!default {
   type plug
   slave.pcm "dmixer"
}

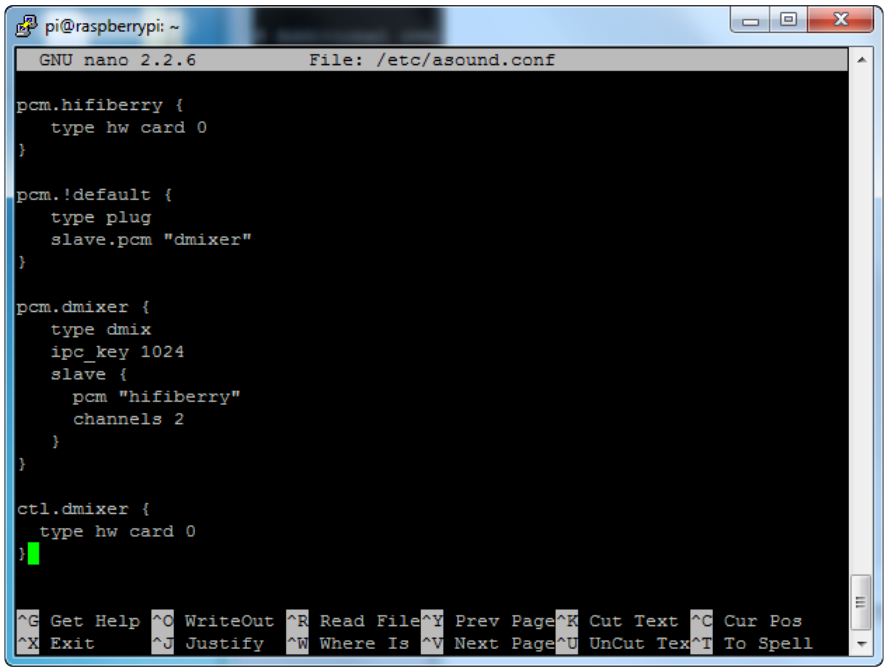
pcm.dmixer {
   type dmix
   ipc_key 1024
   ipc_perm 0666
   slave {
     pcm "speakerbonnet"
     period_time 0
     period_size 1024
     buffer_size 8192
     rate 44100
     channels 2
   }
}

ctl.dmixer {
  type hw card 0
}
```

By running `sudo nano /etc/asound.conf`

This creates a PCM device called speakerbonnet which is connected to the hardware I2S device. Then we make a new 'dmix' device (`type dmix`) called `pcm.dmixer` . We give it a unique Inter Process Communication key (`ipc_key 1024`) and permissions that are world-read-writeable `(ipc_perm 0666`) The mixer will control the hardware pcm device speakerbonnet (pcm "speakerbonnet") and has a buffer set up so its nice and fast. The communication buffer is set up so there's no delays (`period_time 0` , `period_size 1024` and `buffer_size 8192` work well). The default mixed rate is 44.1khz stereo (`rate 44100 channels 2`)

Finally we set up a control interface but it ended up working best to just put in the hardware device here - `ctl.dmixer { type hw card 0 }`



Save and exit. Then reboot the Pi to enable the mixer. Also, while it will greatly reduce popping, you still may get one once in a while - especially when first playing audio!

Add software volume control

The basic I2S chipset used here does not have software control built in. So we have to 'trick' the Pi into creating a software volume control. Luckily, its not hard once you know how to do it (https://adafru.it/ydQ).

Create a new audio config file in **~/.asoundrc** with `nano ~/.asoundrc` and inside put the following text:

```
pcm.speakerbonnet {
   type hw card 0
}

pcm.dmixer {
   type dmix
   ipc_key 1024
   ipc_perm 0666
   slave {
     pcm "speakerbonnet"
     period_time 0
     period_size 1024
     buffer_size 8192
     rate 44100
     channels 2
   }
}

ctl.dmixer {
    type hw card 0
}

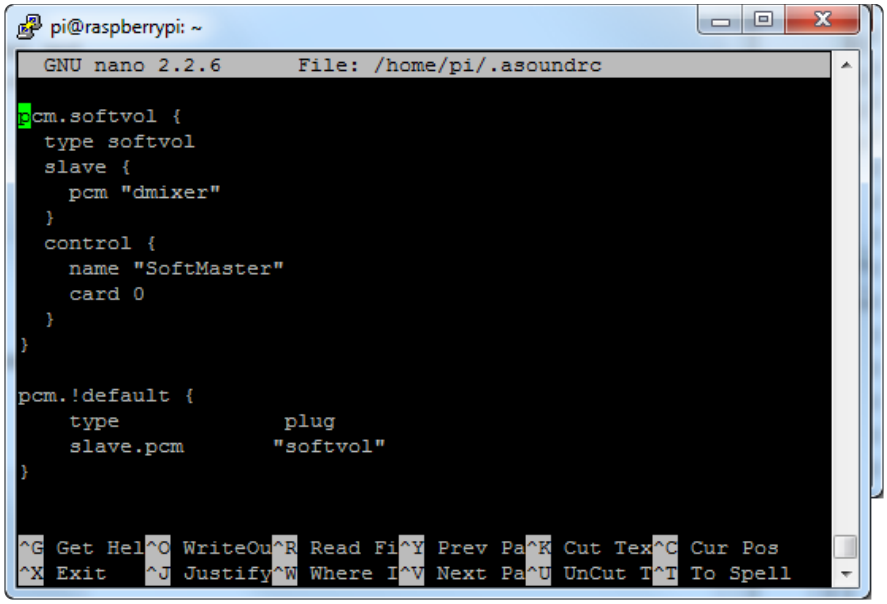
pcm.softvol {
```

```
    type softvol
    slave.pcm "dmixer"
    control.name "PCM"
    control.card 0
}

ctl.softvol {
    type hw card 0
}

pcm.!default {
    type             plug
    slave.pcm       "softvol"
}
```

This assumes you set up the dmixer for no-popping above!

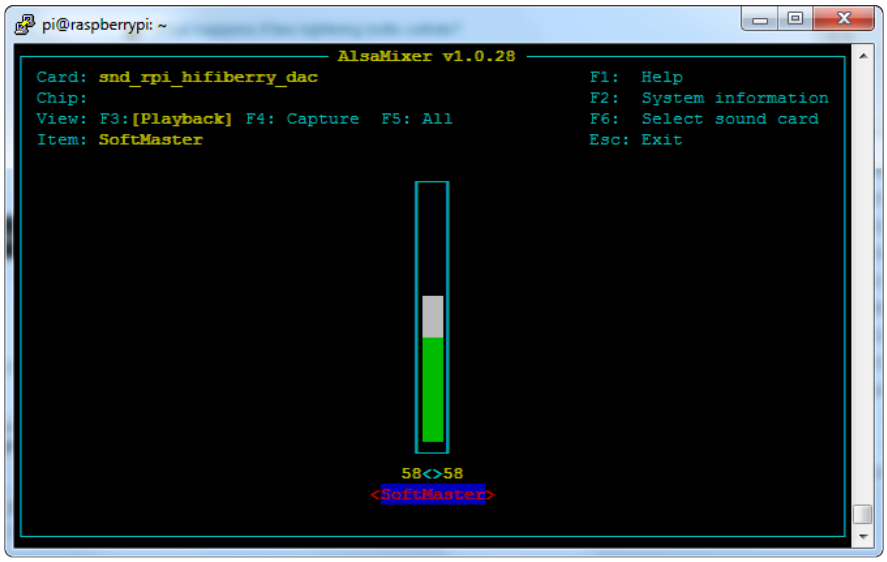


Save and exit

Now, here's the trick, you have to reboot, then play some audio through alsa, then reboot to get the alsamixer to sync up right:

`speaker-test -c2 --test=wav -w /usr/share/sounds/alsa/Front_Center.wav`

Then you can type **alsamixer** to control the volume with the 'classic' alsa mixing interface



Just press the up and down arrows to set the volume, and ESC to quit

---

Play Audio with PyGame

You can use **mpg123** for basic testing but it's a little clumsy for use where you want to dynamically change the volume or have an interactive program. For more powerful audio playback we suggest using PyGame to playback a variety of audio formats (MP3 included!)

Install PyGame

Start by installing pygame support, you'll need to open up a console on your Pi with network access and run:

```
sudo apt-get install python3-pygame
```

Next, download this pygame example zip to your Pi

Click to download PyGame example code & sample mp3s

https://adafru.it/wbp

On the command line, run

```
wget https://cdn-learn.adafruit.com/assets/assets/000/041/506/original/pygame_example.zip (https://adafru.it/wbq)
```

```
unzip pygame_example.zip (https://adafru.it/wbq)
```

Run Demo

Inside the zip is an example called **pygameMP3.py**

This example will playback all MP3's within the script's folder. To demonstrate that you can also adjust the volume within pygame, the second argument is the volume for playback. Specify a volume to playback with a command line argument between 0.0 and 1.0

For example here is how to play at 75% volume:

```
python pygameMP3.py 0.75
```

Here's the code if you have your own mp3s!

```
''' pg_midi_sound101.py
play midi music files (also mp3 files) using pygame
tested with Python273/331 and pygame192 by vegaseat
'''
#code modified by James DeVito from here: https://www.daniweb.com/programming/
software-development/code/454835/let-pygame-play-your-midi-or-mp3-files


#!/usr/bin/python

import sys
import pygame as pg
import os
import time


def play_music(music_file):
    '''
    stream music with mixer.music module in blocking manner
    this will stream the sound from disk while playing
    '''
    clock = pg.time.Clock()
    try:
        pg.mixer.music.load(music_file)
        print("Music file {} loaded!".format(music_file))
    except pygame.error:
        print("File {} not found! {}".format(music_file, pg.get_error()))
        return

    pg.mixer.music.play()

    # If you want to fade in the audio...
    # for x in range(0,100):
    #     pg.mixer.music.set_volume(float(x)/100.0)
    #     time.sleep(.0075)
    # # check if playback has finished
    while pg.mixer.music.get_busy():
        clock.tick(30)


freq = 44100    # audio CD quality
bitsize = -16   # unsigned 16 bit
channels = 2    # 1 is mono, 2 is stereo
```

```
buffer = 2048   # number of samples (experiment to get right sound)
pg.mixer.init(freq, bitsize, channels, buffer)


if len(sys.argv) &gt; 1:

    try:
        user_volume = float(sys.argv[1])
    except ValueError:
        print "Volume argument invalid. Please use a float (0.0 - 1.0)"
        pg.mixer.music.fadeout(1000)
        pg.mixer.music.stop()
        raise SystemExit

    print("Playing at volume: " + str(user_volume)+ "\n")
    pg.mixer.music.set_volume(user_volume)
    mp3s = []
    for file in os.listdir("."):
        if file.endswith(".mp3"):
            mp3s.append(file)

    print mp3s

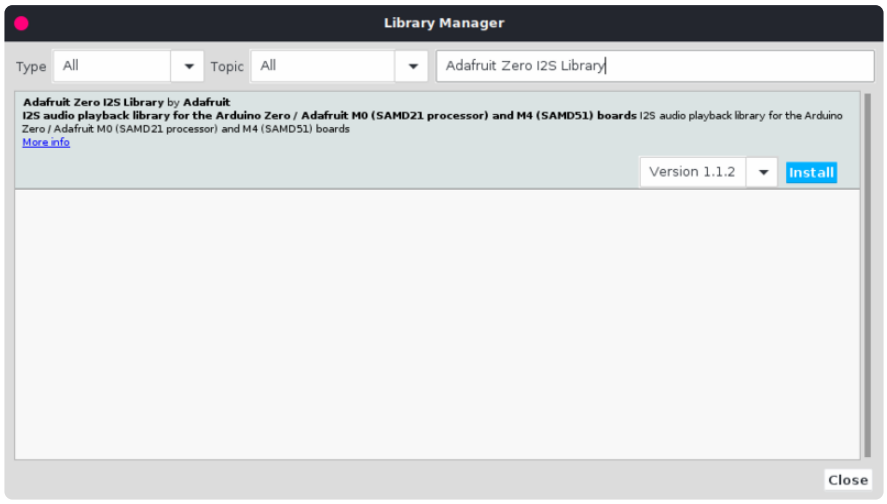
    for x in mp3s:
        try:
            play_music(x)
            time.sleep(.25)
        except KeyboardInterrupt:
            # if user hits Ctrl/C then exit
            # (works only in console mode)
            pg.mixer.music.fadeout(1000)
            pg.mixer.music.stop()
            raise SystemExit
else:
    print("Please specify volume as a float! (0.0 - 1.0)")
```

Arduino Wiring & Test

The classic ATmega328P-based Arduino's like the UNO and Metro 328 don't have I2S interfaces, so you can't use this breakout with them

But the newer ATSAMD21-based boards like the Zero, Metro M0, Feather M0 can! (Note, Gemma M0 & Trinket M0 do not have I2S pins available). And so can the even newer ATSAMD51-based boards like the Metro M4 and Feather M4.

To use I2S with M0 or M4 boards, you'll need to install the Adafruit Zero I2S library (https://adafru.it/DHD). It is available through the Library Manager. You can search for (see below) and then just click the install button.



Wiring

Wiring connections are the same as those used for CircuitPython. So go to the **CircuitPython Wiring & Test** page to see how to wire the breakout for your specific board.

Basic Test

To test things out, try running the demo below. It comes with the library installation, so you can find it by going to:

File -> Examples -> Adafruit Zero I2S Library -> basic

Be sure to change this line:

```
Adafruit_ZeroI2S i2s(0, 1, 9, 2);
```

to match the pins used for your setup. If you've wired as shown in this guide, then you can try using the default pins by changing that line to this:

```
Adafruit_ZeroI2S i2s;
```

```
#include <Arduino.h>

#include <Adafruit_ZeroI2S.h>
#include <math.h>

/* max volume for 32 bit data */
#define VOLUME ( (1UL << 31) - 1)

/* create a buffer for both the left and right channel data */
#define BUFSIZE 128
int left[BUFSIZE];
int right[BUFSIZE];
```

```
// Use default pins in board variant
Adafruit_ZeroI2S i2s = Adafruit_ZeroI2S();

void setup()
{
  while (!Serial) delay(10);

  Serial.println("I2S demo");

  for(int i=0; i<BUFSIZE; i++){
      /* create a sine wave on the left channel */
        left[i] = sin( (2*PI / (BUFSIZE) ) * i) * VOLUME;

        /* create a cosine wave on the right channel */
        right[i] = cos( (2*PI / (BUFSIZE) ) * i) * VOLUME;
  }

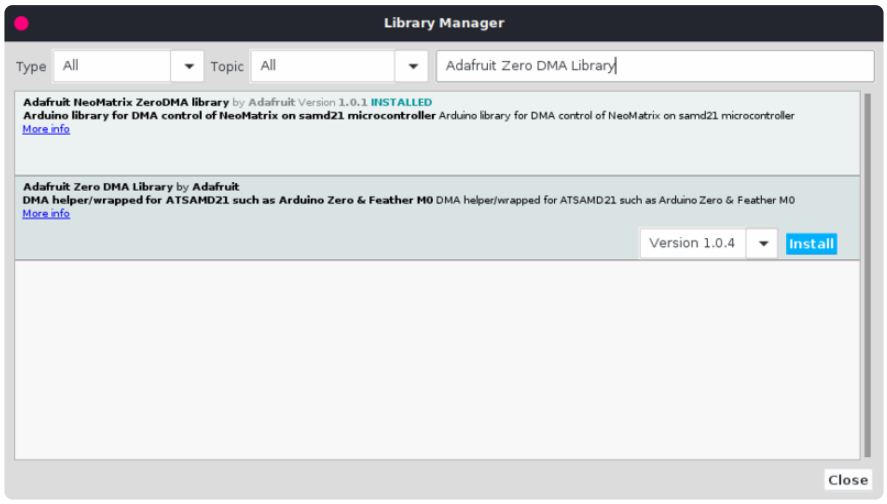
  /* begin I2S on the default pins. 24 bit depth at
   * 44100 samples per second
   */
  i2s.begin(I2S_32_BIT, 44100);
  i2s.enableTx();
}

void loop()
{
  /* write the output buffers
   * note that i2s.write() will block until both channels are written.
   */
  for(int i=0; i<BUFSIZE; i++){
    i2s.write(left[i], right[i]);
  }
}
```

DMA Test

The basic test above created the output directly by using the `i2s.write()` function in a loop. Another approach is to use DMA to generate the output. With this approach, you do some initial setup to configure the DMA engine for playback. It can then take care of generating the output in the background allowing you to do other things in your code.

To take this approach, you will need to install the Zero DMA library (https://adafru.it/lnb). You can do that through the Library Manager:



And then you can use the DMA example found in the Zero I2S library:

File -> Examples -> Adafruit Zero I2S Library -> dma

```
#include <Adafruit_ZeroI2S.h>
#include <Adafruit_ZeroDMA.h>
#include "utility/dma.h"
#include <math.h>

/* max volume for 32 bit data */
#define VOLUME ( (1UL << 31) - 1)

/* create a buffer for both the left and right channel data */
#define BUFSIZE 256
int data[BUFSIZE];

Adafruit_ZeroDMA myDMA;
ZeroDMAstatus    stat; // DMA status codes returned by some functions

Adafruit_ZeroI2S i2s;

void dma_callback(Adafruit_ZeroDMA *dma) {
  /* we don't need to do anything here */
}

void setup()
{
  Serial.begin(115200);
  //while(!Serial);                 // Wait for Serial monitor before continuing

  Serial.println("I2S output via DMA");

  int *ptr = data;

  /*the I2S module will be expecting data interleaved LRLR*/
  for(int i=0; i<BUFSIZE/2; i++){
      /* create a sine wave on the left channel */
        *ptr++ = sin( (2*PI / (BUFSIZE/2) ) * i) * VOLUME;

        /* create a cosine wave on the right channel */
        *ptr++ = cos( (2*PI / (BUFSIZE/2) ) * i) * VOLUME;
  }

  Serial.println("Configuring DMA trigger");
  myDMA.setTrigger(I2S_DMAC_ID_TX_0);
  myDMA.setAction(DMA_TRIGGER_ACTON_BEAT);
```

```
  Serial.print("Allocating DMA channel...");
  stat = myDMA.allocate();
  myDMA.printStatus(stat);

  Serial.println("Setting up transfer");
    myDMA.addDescriptor(
      data,                     // move data from here
#if defined(__SAMD51__)
      (void *)(&I2S->TXDATA.reg), // to here (M4)
#else
      (void *)(&I2S->DATA[0].reg), // to here (M0+)
#endif
    BUFSIZE,                      // this many...
      DMA_BEAT_SIZE_WORD,               // bytes/hword/words
      true,                             // increment source addr?
      false);
  myDMA.loop(true);
  Serial.println("Adding callback");
  myDMA.setCallback(dma_callback);

  /* begin I2S on the default pins. 24 bit depth at
   * 44100 samples per second
   */
  i2s.begin(I2S_32_BIT, 44100);
  i2s.enableTx();

  stat = myDMA.startJob();
}

void loop()
{
  Serial.println("do other things here while your DMA runs in the background.");
  delay(2000);
}
```

CircuitPython Wiring & Test

CircuitPython 3.0 and higher has I2S built in which means you can use this breakout super easily with the supported M0 and M4 Express CircuitPython boards! Supported boards are Feather M0 Express, Feather M4 Express, Metro M0 Express, Metro M4 Express, and ItsyBitsy M0 Express.

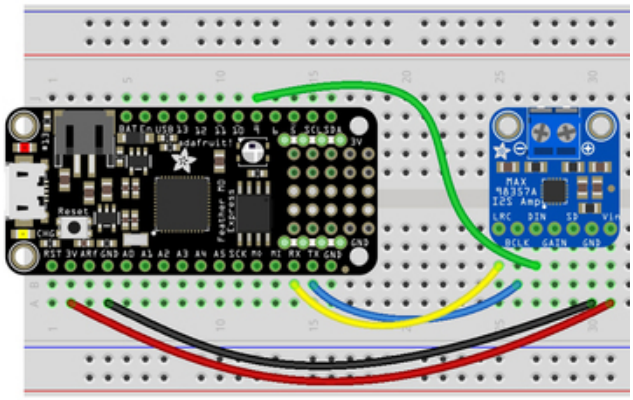
Note that Trinket M0, Gemma M0 and ItsyBitsy M4 do not support I2S (the last one is not a typo!)

The M0 boards have multiple I2S pin combinations available. We're going to demonstrate a single pin combination for each board.

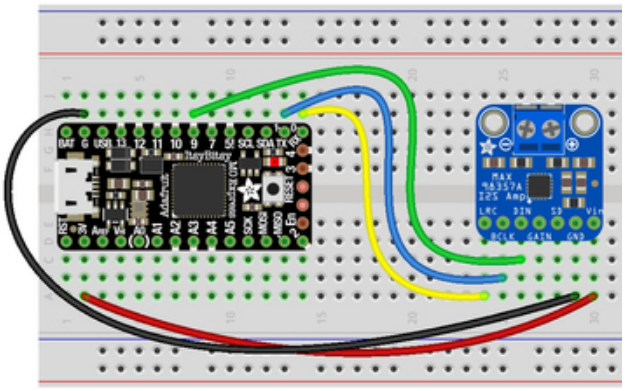
Wiring

The following wiring diagrams show how to connect the MAX98357 breakout to your CircuitPython board. You'll be using voltage in, ground, bit clock, left/right clock and data pins.

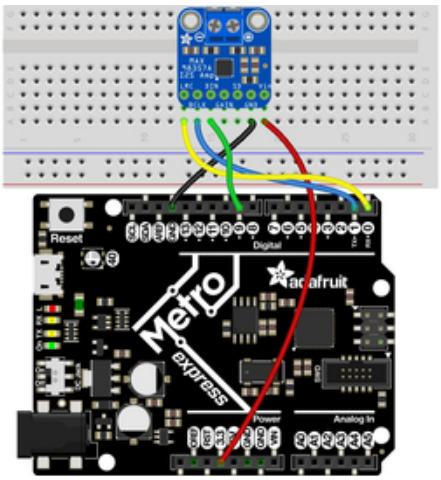
- **VIN** is the **red** wire.
- **GND** is the **black** wire.
- **BCLK** is the **blue** wire.
- **LRC** is the **yellow** wire.
- **DIN** is the **green** wire.



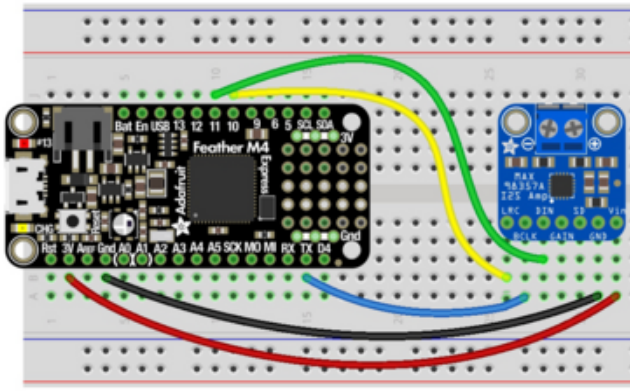
For Feather M0 Express, ItsyBitsy M0 Express and Metro M0 Express:



Connect **VIN on the breakout** to **3V/3.3 on the board**.
Connect **GND on the breakout** to **G/GND on the board**.
Connect **BCLK on the breakout** to **D1/TX on the board**.
Connect **LRC on the breakout** to **D0/RX on the board**.
Connect **DIN on the breakout** to **D9 on the board**.

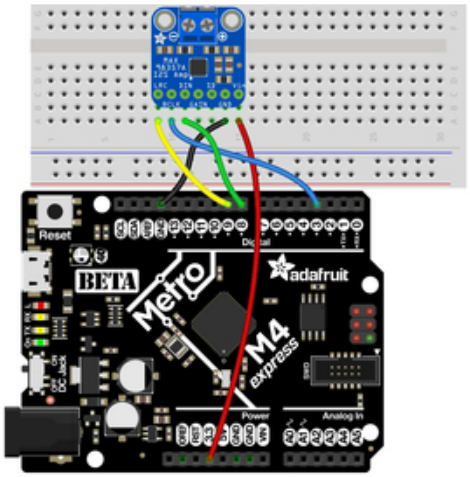


For Feather M4 Express:



Connect **VIN on the breakout** to **3V on the board**.
Connect **GND on the breakout** to **Gnd on the board**.
Connect **BCLK on the breakout** to **TX on the board**.
Connect **LRC on the breakout** to **D10 on the board**.
Connect **DIN on the breakout** to **D11 on the board**.

For Metro M4 Express:



Connect **VIN on the breakout** to **3.3 on the board**.
Connect **GND on the breakout** to **GND on the board**.
Connect **BCLK on the breakout** to **D3 on the board**.
Connect **LRC on the breakout** to **D9 on the board**.
Connect **DIN on the breakout** to **D8 on the board**.

Code Examples

We have two CircuitPython code examples. The first plays a generated tone through the speaker on the breakout. The second plays a wave file. Let's take a look!

Tone Generation

The first example generates one period of a sine wave and then loops it to generate a tone. You can change the volume and the Hz of the tone by changing the associated variables. Inside the loop, we play the tone for one second and stop it for one second.

```
# SPDX-FileCopyrightText: 2018 Kattni Rembor for Adafruit Industries
#
```

```
# SPDX-License-Identifier: MIT

import time
import array
import math
import audiocore
import board
import audiobusio

sample_rate = 8000
tone_volume = .1  # Increase or decrease this to adjust the volume of the tone.
frequency = 440  # Set this to the Hz of the tone you want to generate.
length = sample_rate // frequency  # One freqency period
sine_wave = array.array("H", [0] * length)
for i in range(length):
    sine_wave[i] = int((math.sin(math.pi * 2 * frequency * i / sample_rate) *
                        tone_volume + 1) * (2 ** 15 - 1))

# For Feather M0 Express, ItsyBitsy M0 Express, Metro M0 Express
audio = audiobusio.I2SOut(board.D1, board.D0, board.D9)
# For Feather M4 Express
# audio = audiobusio.I2SOut(board.D1, board.D10, board.D11)
# For Metro M4 Express
# audio = audiobusio.I2SOut(board.D3, board.D9, board.D8)
sine_wave_sample = audiocore.RawSample(sine_wave, sample_rate=sample_rate)

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

For **Feather M0 Express, ItsyBitsy M0 Express** and **Metro M0 Express**, no changes are needed for the code to work.

For **Feather M4 Express**, comment out `audio = audiobusio.I2SOut(board.D1, board.D0, board.D9)` and uncomment `# audio = audiobusio.I2SOut(board.D1, board.D10, board.D11)`.

For **Metro M4 Express**, comment out `audio = audiobusio.I2SOut(board.D1, board.D0, board.D9)` and uncomment `# audio = audiobusio.I2SOut(board.D3, board.D3, board.D8)`.

Now you'll hear one second of a 440Hz tone, and one second of silence.

You can try changing the Hz of the tone to produce different tones. Try changing the number of seconds in `time.sleep()` to produce longer or shorter tones.

Wave File

The second example plays a wave file. We open the file in a readable format. Then inside the loop, we play the file and tell the code to continue playing the file until it's

completed. You can use any supported wave file (https://adafru.it/BRj). We've included the wave file used in the code.

StreetChicken.wav

https://adafru.it/BUC

```
# SPDX-FileCopyrightText: 2018 Kattni Rembor for Adafruit Industries
#
# SPDX-License-Identifier: MIT

import audiocore
import board
import audiobusio

wave_file = open("StreetChicken.wav", "rb")
wave = audiocore.WaveFile(wave_file)

# For Feather M0 Express, ItsyBitsy M0 Express, Metro M0 Express
audio = audiobusio.I2SOut(board.D1, board.D0, board.D9)
# For Feather M4 Express
# audio = audiobusio.I2SOut(board.D1, board.D10, board.D11)
# For Metro M4 Express
# audio = audiobusio.I2SOut(board.D3, board.D9, board.D8)

while True:
    audio.play(wave)
    while audio.playing:
        pass
```

The object setup in the code is the same as above.

For **Feather M0 Express, ItsyBitsy M0 Express** and **Metro M0 Express**, no changes are needed for the code to work.

For **Feather M4 Express**, comment out `audio = audiobusio.I2SOut(board.D1, board.D0, board.D9)` and uncomment `# audio = audiobusio.I2SOut(board.D1, board.D10, board.D11)`.

For **Metro M4 Express**, comment out `audio = audiobusio.I2SOut(board.D1, board.D0, board.D9)` and uncomment `# audio = audiobusio.I2SOut(board.D3, board.D3, board.D8)`.

Now you'll hear the wave file play through and loop.

There's plenty you can do with this example. Try playing a different wave file, or, instead of including `while audio.playing: pass`, include a `time.sleep()` to have it play for a specified number of seconds. Check out the Audio Out page in the CircuitPython Essentials guide (https://adafru.it/BRj) for `pause` and `resume` features.

Where's my I2S?

We mentioned earlier that the supported M0 boards have multiple I2S pin combinations available to you. The M4 boards have one option. Either way, if you'd like to know what options are available to you, copy the following code into your **code.py**, **connect to the serial console**, and **check out the output**.

These are the results from the ItsyBitsy M0 Express.

```
Adafruit CircuitPython 3.0.0 on 2018-07-09; Adafruit ItsyBitsy M0 Express with samd21g18
>>>
soft reboot

Auto-reload is on. Simply save files over USB to run them or enter REPL to disable.
code.py output:
Bit clock pin: board.D1         Word select pin: board.D0      Data pin: board.D4
Bit clock pin: board.D1         Word select pin: board.D0      Data pin: board.D9
Bit clock pin: board.D1         Word select pin: board.D0      Data pin: board.D12
Bit clock pin: board.D1         Word select pin: board.D7      Data pin: board.D4
Bit clock pin: board.D1         Word select pin: board.D7      Data pin: board.D9
Bit clock pin: board.D1         Word select pin: board.D7      Data pin: board.D12
Bit clock pin: board.D6         Word select pin: board.D0      Data pin: board.D4
Bit clock pin: board.D6         Word select pin: board.D0      Data pin: board.D9
Bit clock pin: board.D6         Word select pin: board.D0      Data pin: board.D12
Bit clock pin: board.D6         Word select pin: board.D7      Data pin: board.D4
Bit clock pin: board.D6         Word select pin: board.D7      Data pin: board.D9
Bit clock pin: board.D6         Word select pin: board.D7      Data pin: board.D12
```

```
# SPDX-FileCopyrightText: 2018 Kattni Rembor for Adafruit Industries
#
# SPDX-License-Identifier: MIT

import board
import audiobusio
from microcontroller import Pin


def is_hardware_i2s(bit_clock, word_select, data):
    try:
        p = audiobusio.I2SOut(bit_clock, word_select, data)
        p.deinit()
        return True
    except ValueError:
        return False


def get_unique_pins():
    exclude = ['NEOPIXEL', 'APA102_MOSI', 'APA102_SCK']
    pins = [pin for pin in [
        getattr(board, p) for p in dir(board) if p not in exclude]
            if isinstance(pin, Pin)]
    unique = []
    for p in pins:
        if p not in unique:
            unique.append(p)
    return unique


for bit_clock_pin in get_unique_pins():
    for word_select_pin in get_unique_pins():
        for data_pin in get_unique_pins():
            if bit_clock_pin is word_select_pin or bit_clock_pin is data_pin or
word_select_pin\
```

```
                    is data_pin:
                continue
            if is_hardware_i2s(bit_clock_pin, word_select_pin, data_pin):
                print("Bit clock pin:", bit_clock_pin, "\t Word select pin:",
word_select_pin,
                      "\t Data pin:", data_pin)
            else:
                pass
```

I2S Audio FAQ

Hey in Raspbian Pixel desktop, the speaker icon is X'd out!

Try rebooting once after playing some audio. Also make sure you have our latest alsa configuration (check the detailed install page on the Raspberry Pi Setup page for the `/etc/asound.conf`!

If its still not working, you can still change the volume, just use **alsamixer** from a Terminal command prompt.

Even with dmixer enabled, I get a staticy-pop when the Pi first boots or when it first starts playing audio

Yep, this is a known Raspbian Linux thing. Yay Linux! We don't have a fix for it. If it makes you feel better, my fancy Windows development computer does the same thing with my desktop speakers.

You can reduce popping a lot with the '/dev/zero play' option in the i2s setup script. (We added it in October 2018) but you still might get that one initial pop

The audio on my DAC sounds really bad/distorted

Make sure you've lowered the volume. The default is 100% which is waaaay too high! Use **alsamixer** to reduce the volume to 50% or so

Does this work with my favorite software?

It will work with anything that has alsa audio support. There's thousands of linux programs so we can't guarantee all of them will work but here's what we found does for sure!

- **PyGame** - see our page on playing audio with PyGame (https://adafru.it/ydS) for example code. Volume can be controlled within pygame

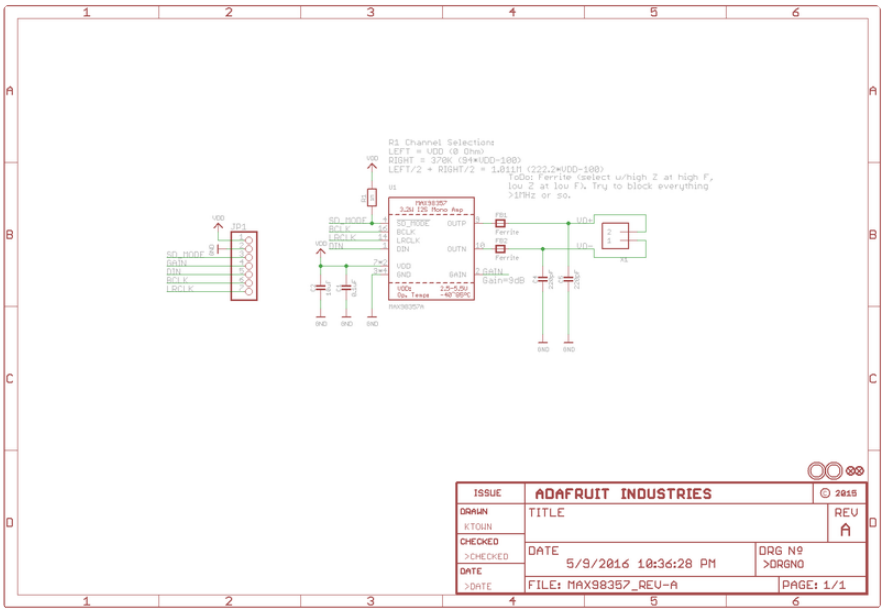
- **mpg123** - command line mp3 audio playback. use alsamixer to control the volume
- **aplay** - for playing wav files on the command line
- **Sonic Pi** - tested in the Pixel Desktop. Use the Sonic Pi settings panel to change the volume - it does not seem to care about what global audio volume you set!
- **Scratch 2** - tested in the Pixel Desktop. Works fine but may have a delay and make a popping sound the first time you play audio. You can set volume with alsamixer and also via the app by using the **set volume to nn%** block
- **Scratch 1** - doesn't work, something not set up with Scratch 1 to use alsa?
- **RetroPie/Emulation Station** - audio works within games (we tested NES and MAME libretro) but does not work in the 'main screen' (selecting which game to play interface)

Downloads

- MAX98357 Datasheet (http://adafru.it/30069835798357)
- GitHub with EagleCAD PCB Files (https://adafru.it/nFB)
- 3D Models on GitHub (https://adafru.it/10fp)
- Fritzing object in the Adafruit Fritzing library (https://adafru.it/aP3)

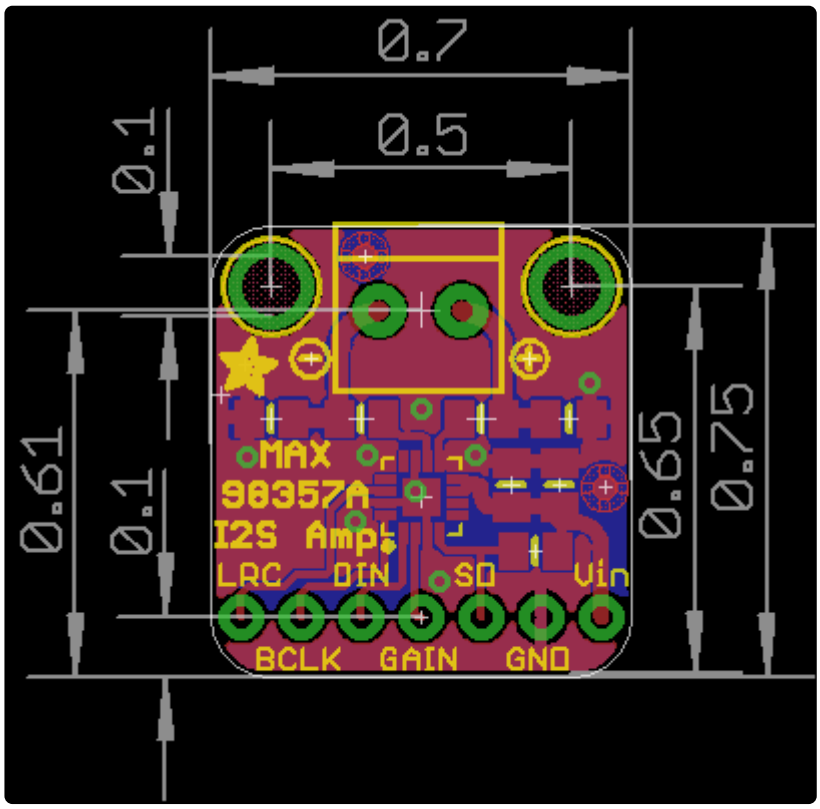
Schematic

Click to embiggen



Fabrication Print

Dimensions in Inches



3D Model

