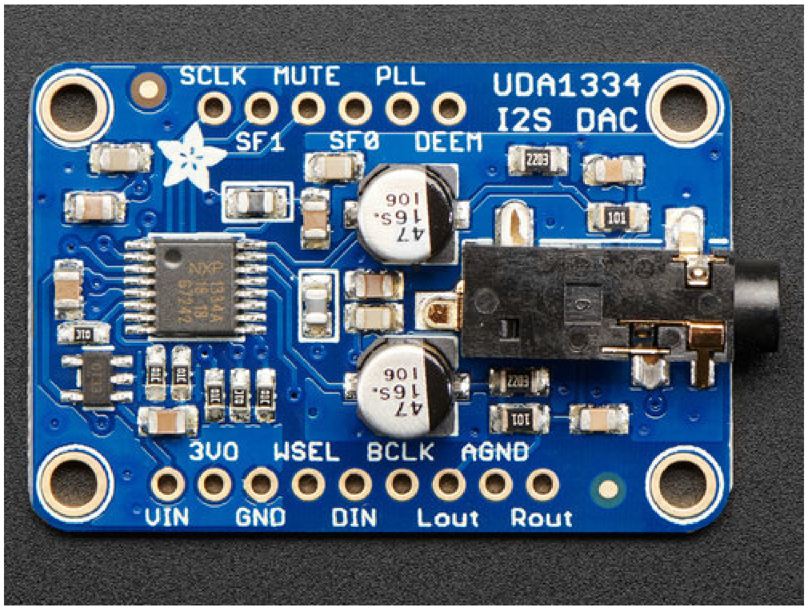
adafruit learning system

Adafruit I2S Stereo Decoder - UDA1334A

Created by lady ada



Last updated on 2019-01-30 06:56:56 PM UTC

Guide Contents

Guide Contents 2
Overview 4
Pinouts 7
Power Pins 7
I2S Pins 7
Audio Outputs 8
Optional Control Pins 8
Assembly 10
Installing Standard Headers 10
Raspberry Pi Wiring 14
Raspberry Pi Setup 15
Fast Install 15
Detailed Install 17
- Update /etc/modprobe.d (if it exists) 17
- Disable headphone audio (if it's set) 18
- Create asound.conf file 19
- Add Device Tree Overlay 21

Raspberry Pi Test 23
Speaker Tests! 23
- Simple white noise speaker test 23
- Simple WAV speaker test 23
- Simple MP3 speaker test 23

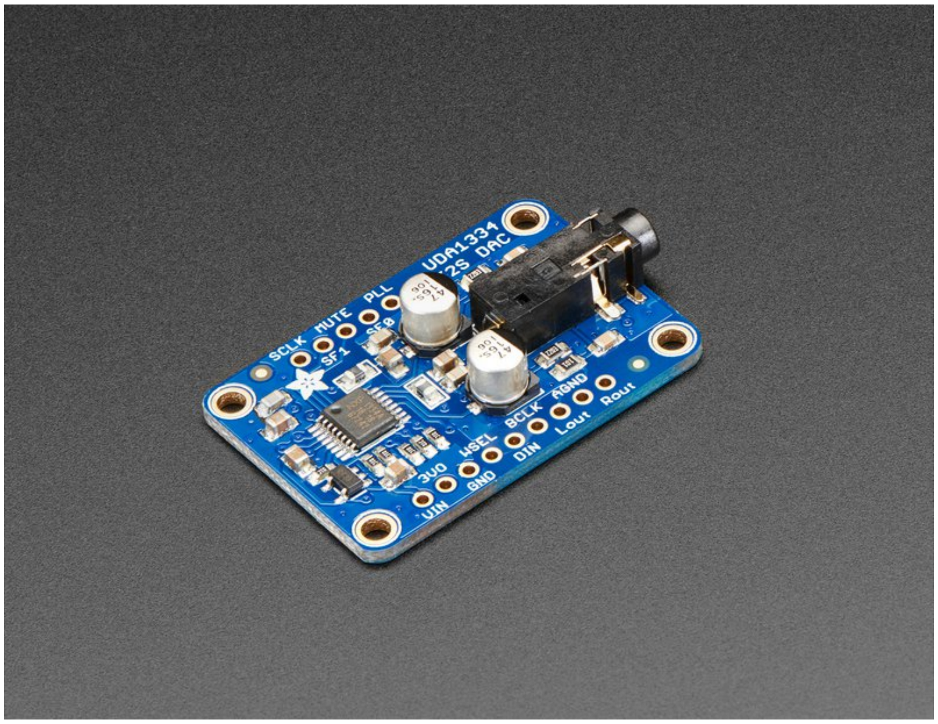
Volume adjustment 23
Pi I2S Tweaks 25
Reducing popping 25
- Step 1 25

Add software volume control 27
Play Audio with PyGame 30
Install PyGame 30
Run Demo 30
Arduino Wiring & Test 33
- Wiring 33
- Basic Test 33
- DMA Test 36

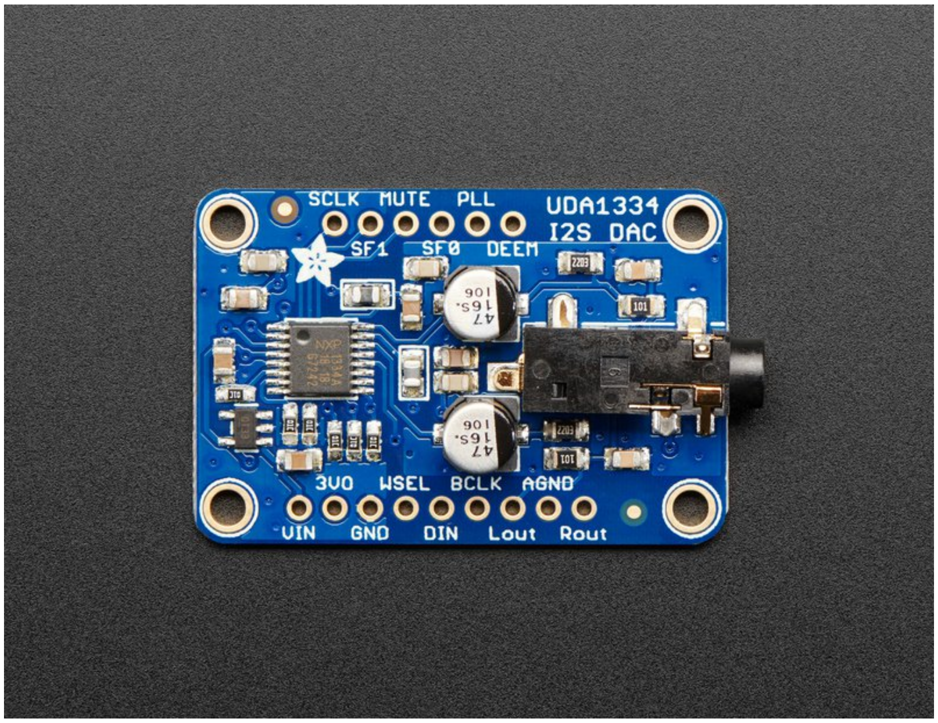
CircuitPython Wiring & Test 39
- Wiring 39
- Code Examples 41
- Tone Generation 41
- Wave File 42
- Where's my I2S? 43

Downloads 45
Files 45
Schematic & Fabrication Print 45

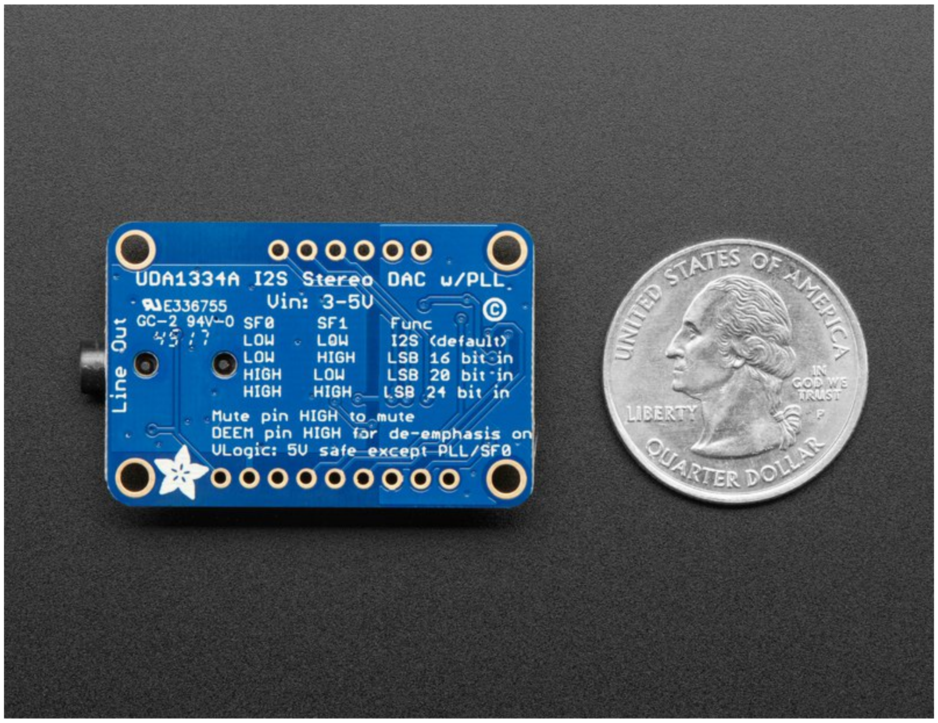
Overview



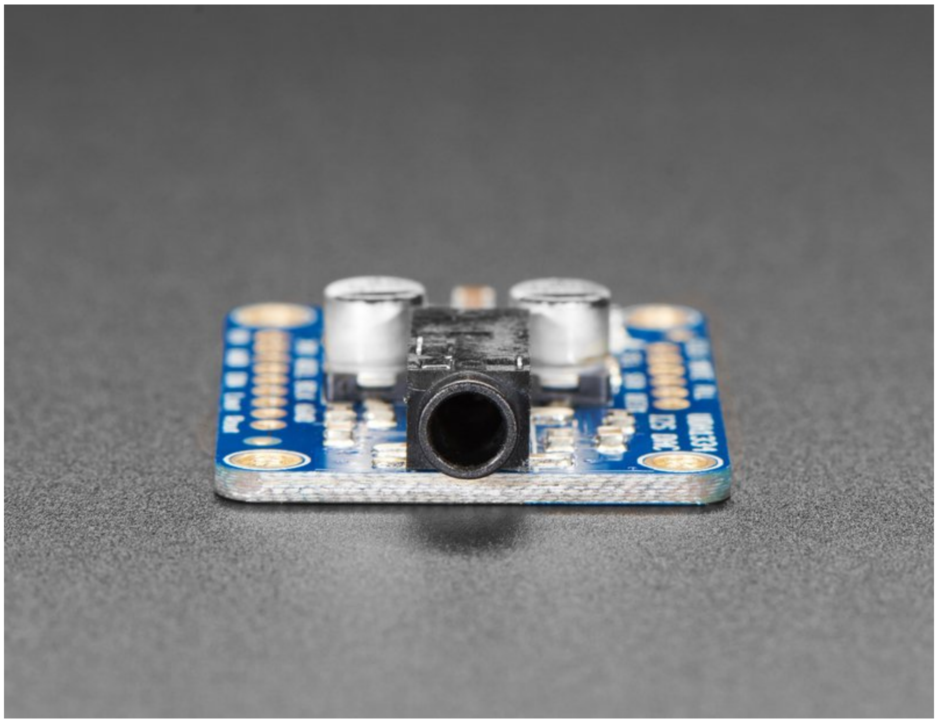
This fully-featured UDA1334A I2S Stereo DAC breakout is a perfect match for any I2S-output audio interface. It's affordable but sounds great! The NXP UDA1334A is a jack-of-all-I2S-trades: you can use 3.3V - 5V logic levels (a rarity), and can process multiple different formats by setting two pins to high or low. The DAC will process data immediately, and give you a clear, analog, stereo line level output. It's even cool with MCLK-less I2S interfaces such as the Raspberry Pi (which it's ideal for) - a built in PLL will generate the proper clock from the incoming signal.



For inputs, you can use classic I2S (the default) or 16-bit, 20-bit or 24-bit left justified data. You can set it up to take an input system/master clock but we default-set it to just generate it for you, so you only need to connect Data In, Word Select (Left/Right Clock) and Bit Clock lines. If you want, there's a mute pin and a de-emphasis filter you can turn on.

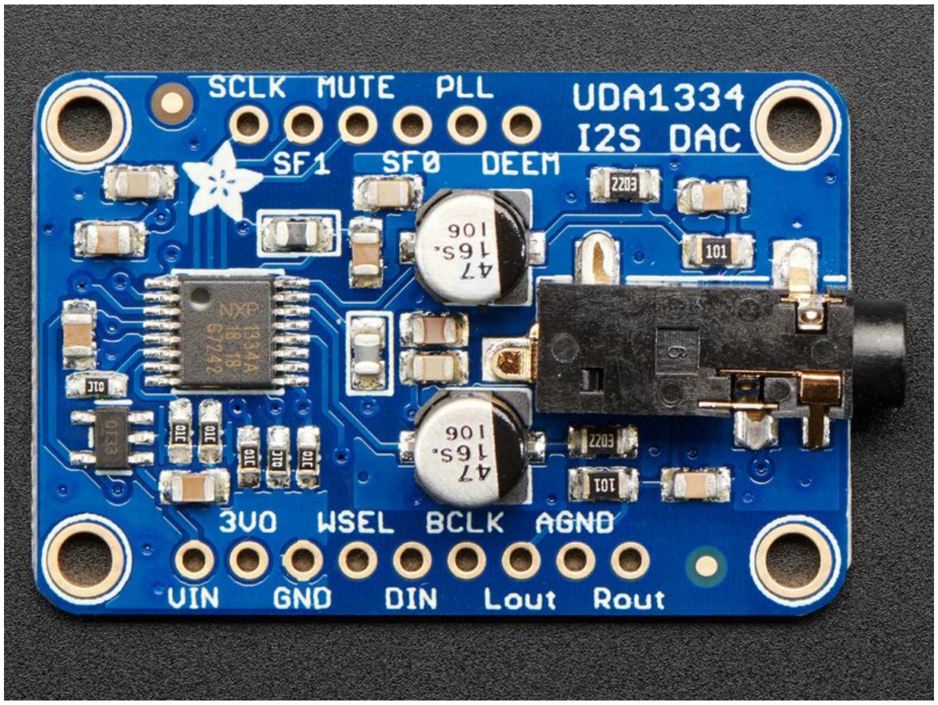


We put in plenty of ferrite beads, a low-dropout regulator, and the recommended band-pass filter so you get a very nice clean output. With a sine-wave generator we swept through 20-20KHz and saw no attenuation or distortion. Plug into either the 3.5mm stereo headphone jack or the breadboard-friendly pads. We think you'll be pleased with this DAC!



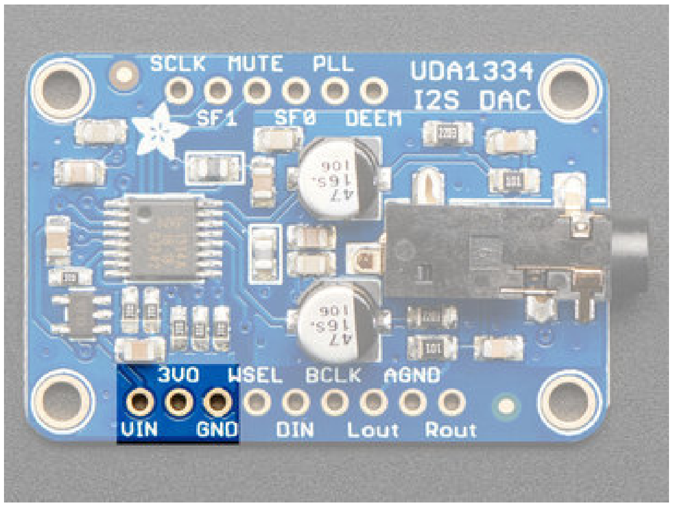
Each order comes with one I2S Stereo DAC breakout and some header you can solder on.

Pinouts



The UDA1334A 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.

Power Pins

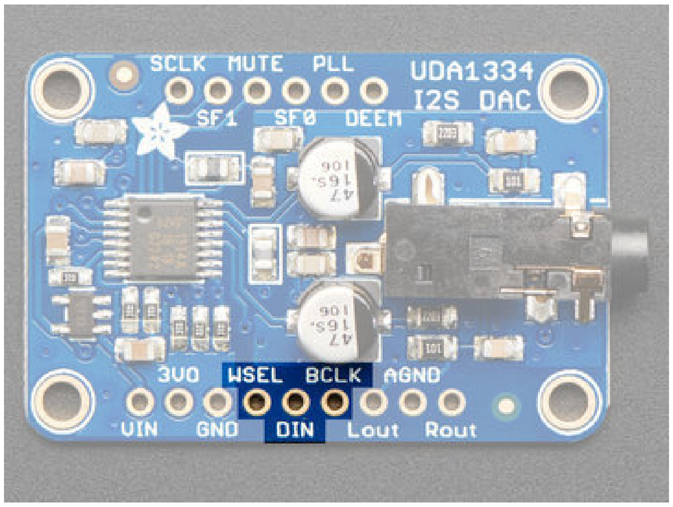


The UDA1334A requires 3.3V power but can take 3-5V level logic on nearly all pins.

You can provide 3-5V power on the **VIN** pin and **GND** and the built in regulator will generate a nice clean 3.3V supplier on **3VO**ut.

Use the quietest power supply for Vin, we do filter the power supply, but the quieter the better!

I2S Pins



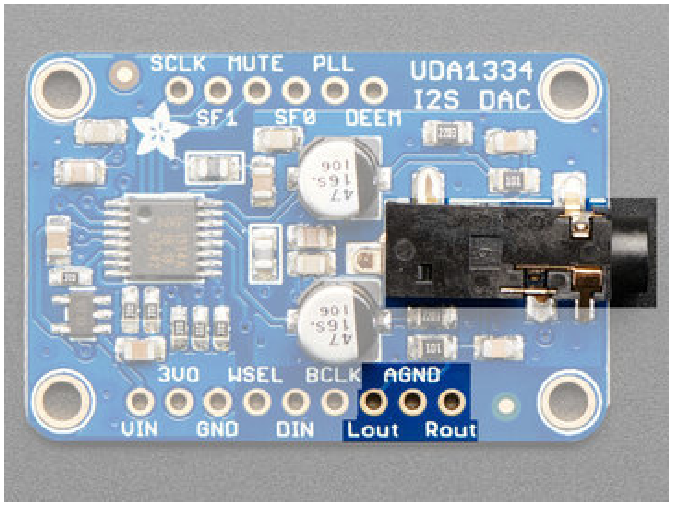
Three pins are used for stereo I2S data in. These pins are required!

These can be 3.3-5V logic

- **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 its 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
- **BCLK** (Bit Clock) - This is the pin that tells the amplifier when to read data on the data pin.

MCLK is not required to use this DAC, if you have an MCLK pin on your audio source, leave it disconnected.

Audio Outputs



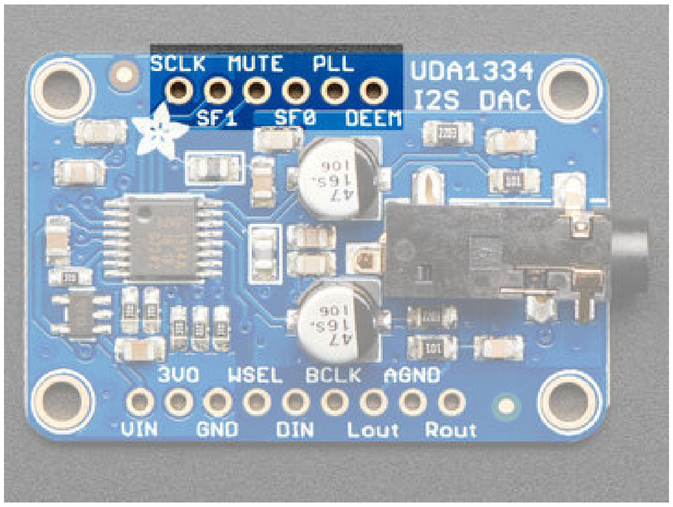
The exciting part! This is where your line level audio comes out. We put big 47uF blocking capacitors on the output so you can connect this to any stereo system. **AGND** is a clean analog ground signal that we recommend using as your analog reference, you'll get a cleaner signal.

Note that this DAC was intended for use with a separate amplifier and is rated for a 3 KΩ load. However, we've found you *can* plug in 32Ω headphones and the output is current-limited so it won't damage the DAC but you will get distortions. (Powered headphones won't have this issue)

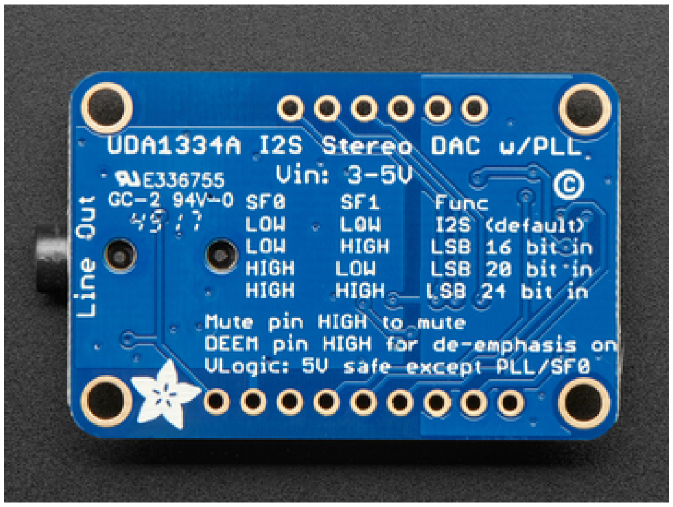
Optional Control Pins

There are some extra configuration pins if you want to use them. They are not required for 99% of usage with an Arduino or Teensy or Raspberry Pi. But you never know! So they are there for you. **PLL** and **SF0** are 3.3V logic only, the other pins are 3-5V safe.

Most of the pins have to do with changing the setup from audio mode to video mode. If you happen to want video-mode, for synchronizing with NTSC/PAL, check the datasheet - we haven't used it for that purpose.



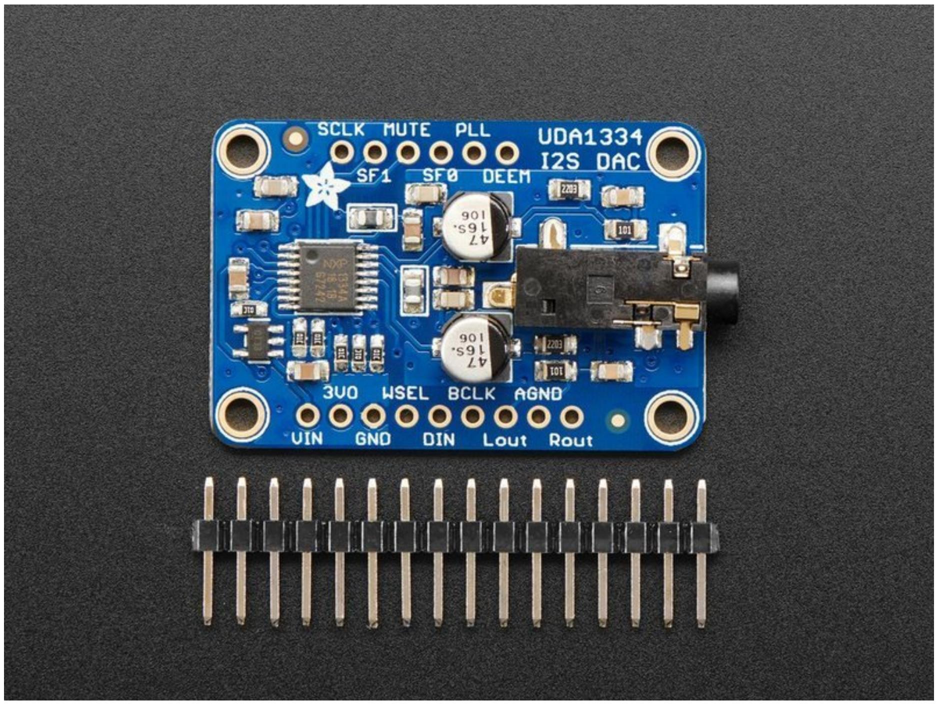
- **SCLK (Sys Clock)** - Optional 27 MHz 'video mode' ssytem clock input - by default we generate the sysclock from the WS clock in 'audio mode' But the UDA can also take a oscillator input on this pin
- **Mute** - Setting this pin High will mute the output
- **De-Em**phasis - In audio mode (which is the default), can be used to add a de-emphasis filter. In video mode, where the system clock is generated from an oscillator, this is the clock output.
- **PLL** - sets the PLL mode, by default pulled low for Audio. Can be pulled high or set to ~1.6V to set PAL or NTSC video frequency



SF0 and **SF1** are used to set the input data format. By default both are pulled Low for I2S but you can change them around for alternate formats.

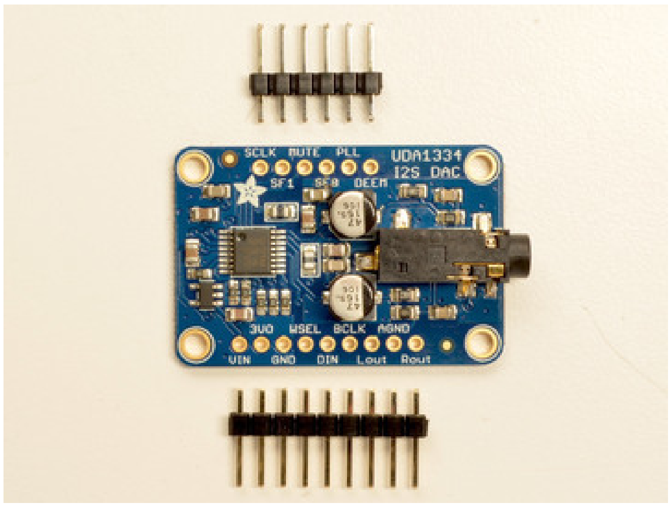
See the back of the PCB for a quick reference

Assembly

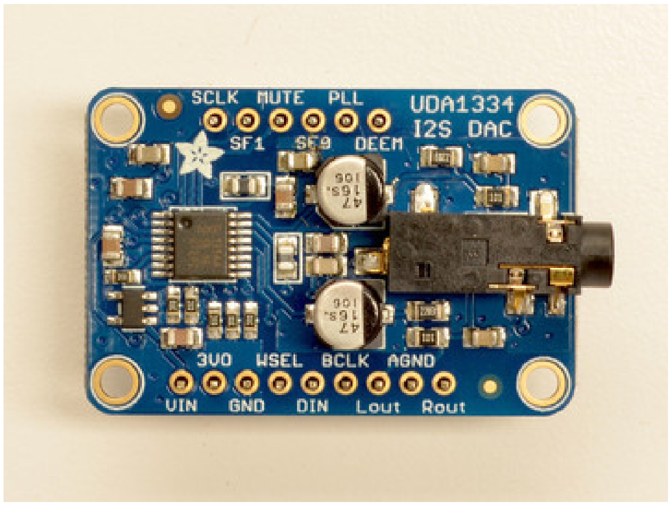


Installing Standard Headers

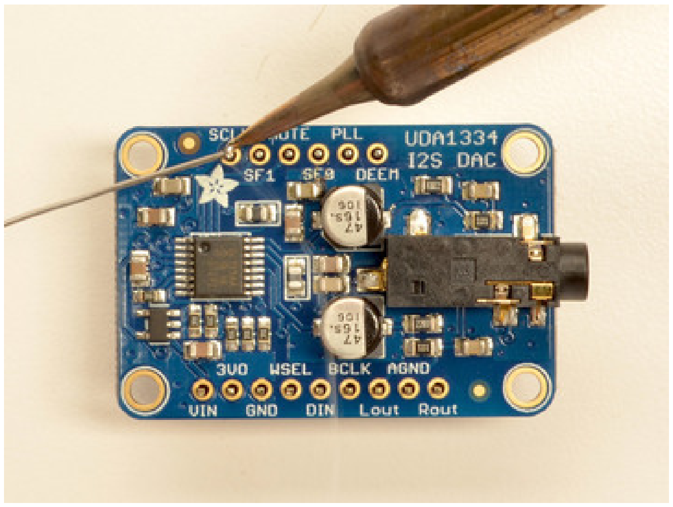
The shield comes with 0.1" standard header.



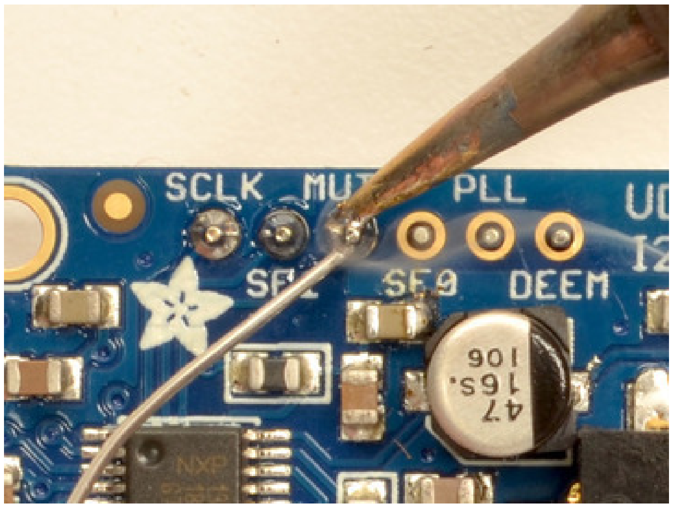
Break apart the 0.1" header into 6 and 9-pin long pieces and slip the short ends into the holes in the board

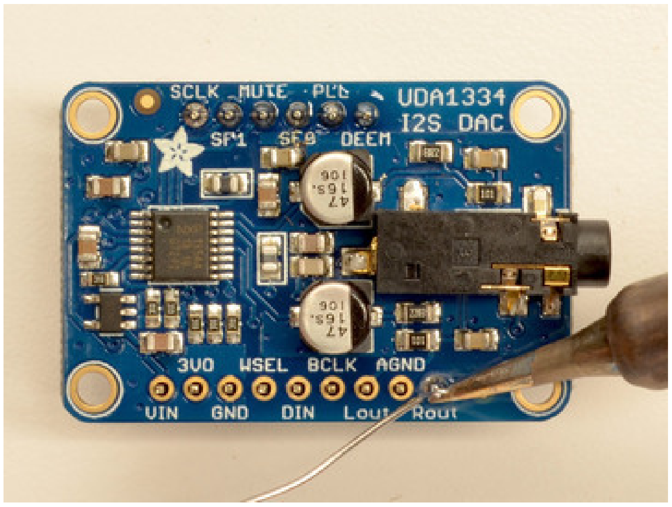


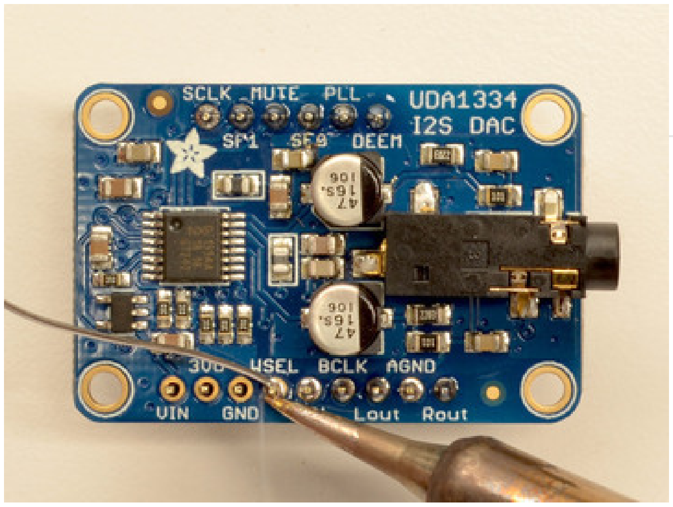
Make sure that all of the short parts of the header are sticking through the two sets of pads on either side of the board

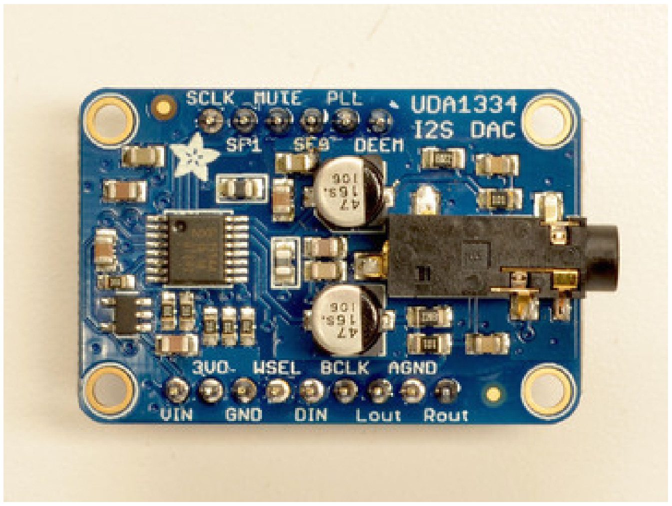


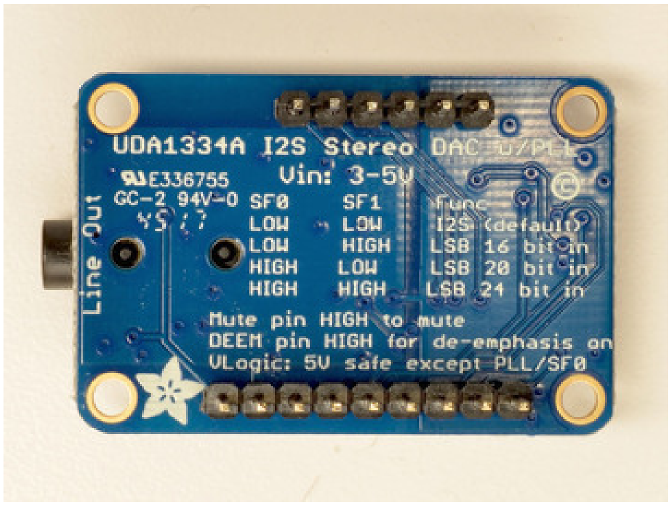
Solder each one of the pins into the board to make a secure connection


SCLK
MUTE
PLL
SF1
SF0
DEEM


SCLK
MUTE
PLL
UDA1334
I2S DAC
SF1
SF0
DEEM
3V0
WSEL
BCLK
AGND
VIN
GND
DIN
Lout
Rout


SCLK
MUTE
PLL
UDA1334
I2S DAC
SF1
SF0
DEEM
3V0
WSEL
BCLK
AGND
VIN
GND
Lout
Rout





That's it! Move on to next page for wiring information

Raspberry Pi Wiring

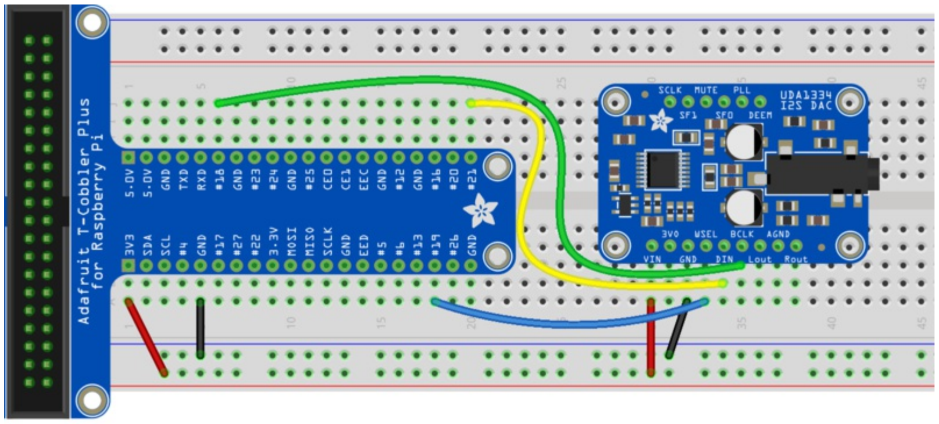
if you have a Raspberry Pi and you want higher quality audio than the headphone jack can provide, I2S is a good option! You only use 3 pins, and since its a pure-digital output, there can be less noise and interference.

This board works very well with boards that *don't* have audio like the Pi Zero and is the easiest way to get quality audio out

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 **3V** or **5V**
- **Amp GND** to Raspbery Pi **GND**
- **Amp DIN** to Raspbery Pi **#21**
- **Amp BCLK** to Raspbery Pi **#18**
- **Amp LRCLK** to Raspbery Pi **#19**



https://adafru.it/A9T

https://adafru.it/A9T

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 :)

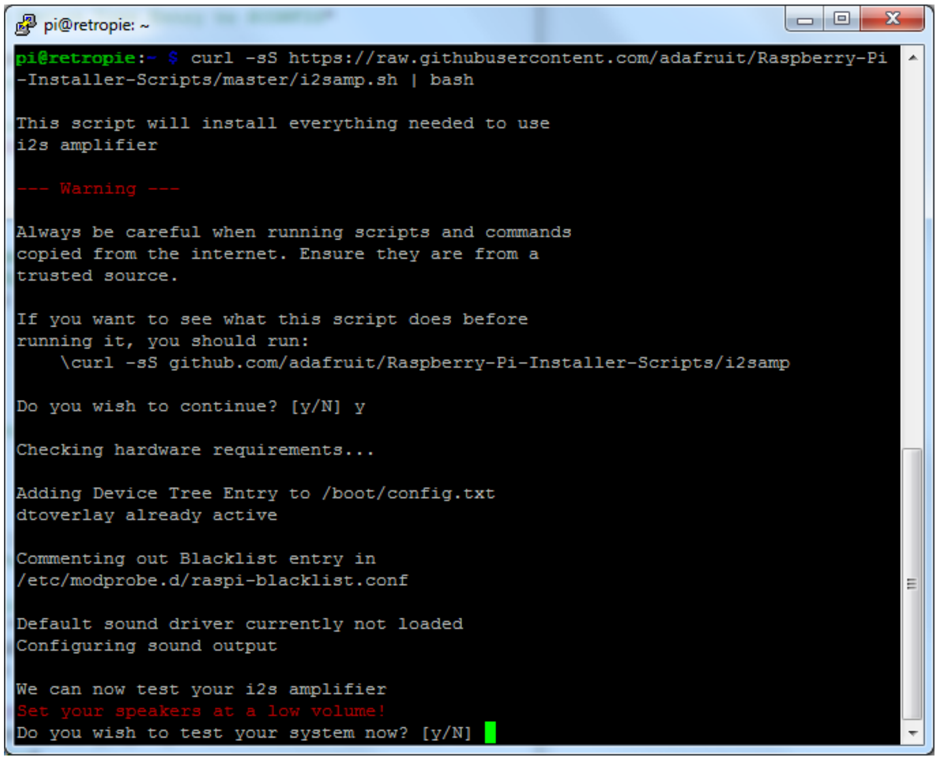
Fast Install

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)

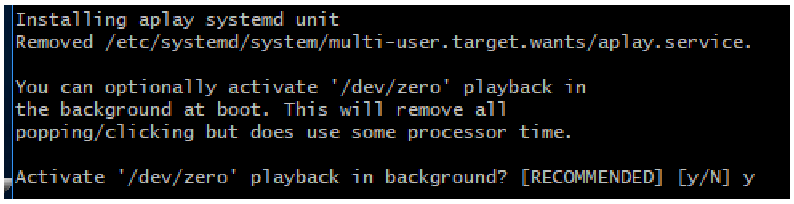
Run the following from your Raspberry Pi with Internet connectivity:

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



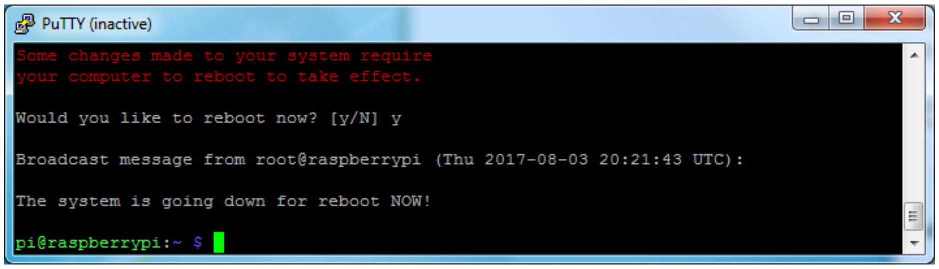
```
pi@retropie:~ $ curl -sS https://raw.githubusercontent.com/adafruit/Raspberry-Pi
-Installer-Scripts/master/i2samp.sh | bash

This script will install everything needed to use
i2s amplifier

--- Warning ---

Always be careful when running scripts and commands
copied from the internet. Ensure they are from a
trusted source.

If you want to see what this script does before
running it, you should run:
    \curl -sS github.com/adafruit/Raspberry-Pi-Installer-Scripts/i2samp

Do you wish to continue? [y/N] y

Checking hardware requirements...

Adding Device Tree Entry to /boot/config.txt
dtoverlay already active

Commenting out Blacklist entry in
/etc/modprobe.d/raspi-blacklist.conf

Default sound driver currently not loaded
Configuring sound output

We can now test your i2s amplifier
Set your speakers at a low volume!
Do you wish to test your system now? [y/N]
```

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.



```
Installing aplay systemd unit
Removed /etc/systemd/system/multi-user.target.wants/aplay.service.

You can optionally activate '/dev/zero' playback in
the background at boot. This will remove all
popping/clicking but does use some processor time.

Activate '/dev/zero' playback in background? [RECOMMENDED] [y/N] y
```

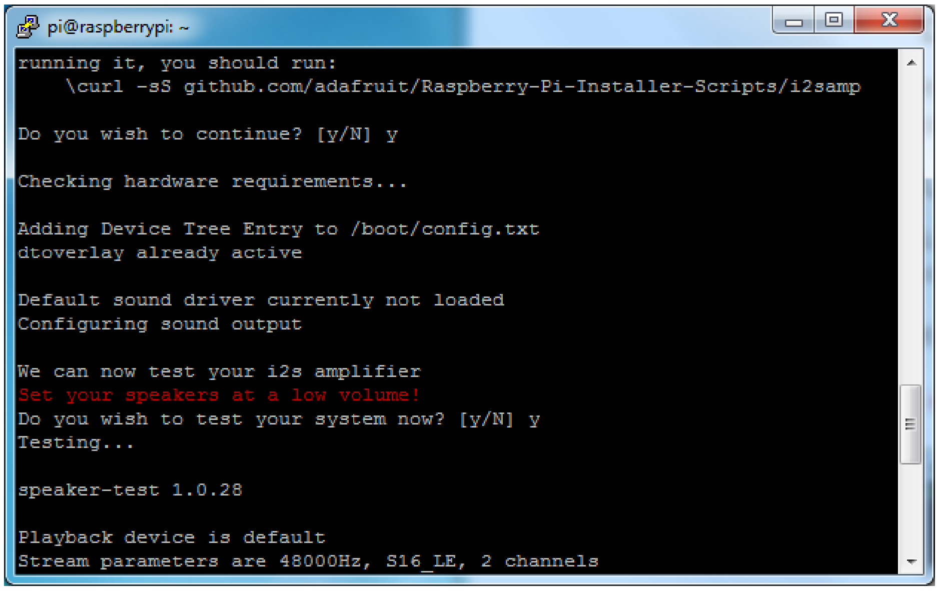
You will need to reboot once installed.



```
Some changes made to your system require
your computer to reboot to take effect.

Would you like to reboot now? [y/N] y

Broadcast message from root@raspberrypi (Thu 2017-08-03 20:21:43 UTC):

The system is going down for reboot NOW!

pi@raspberrypi:~ $
```

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...



```
running it, you should run:
    \curl -sS github.com/adafruit/Raspberry-Pi-Installer-Scripts/i2samp

Do you wish to continue? [y/N] y

Checking hardware requirements...

Adding Device Tree Entry to /boot/config.txt
dtoverlay already active

Default sound driver currently not loaded
Configuring sound output

We can now test your i2s amplifier
Set your speakers at a low volume!
Do you wish to test your system now? [y/N] y
Testing...

speaker-test 1.0.28

Playback device is default
Stream parameters are 48000Hz, S16_LE, 2 channels
```

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

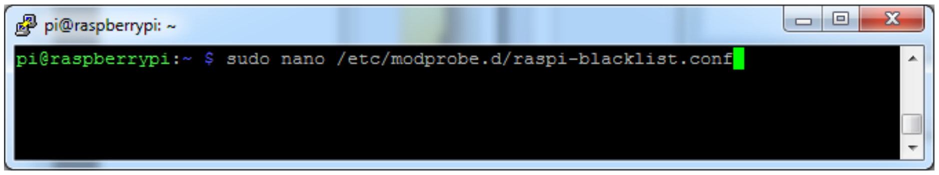
If, for some reason, you can't just run the script and you want to go through the install by hand - here's all the steps!

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`



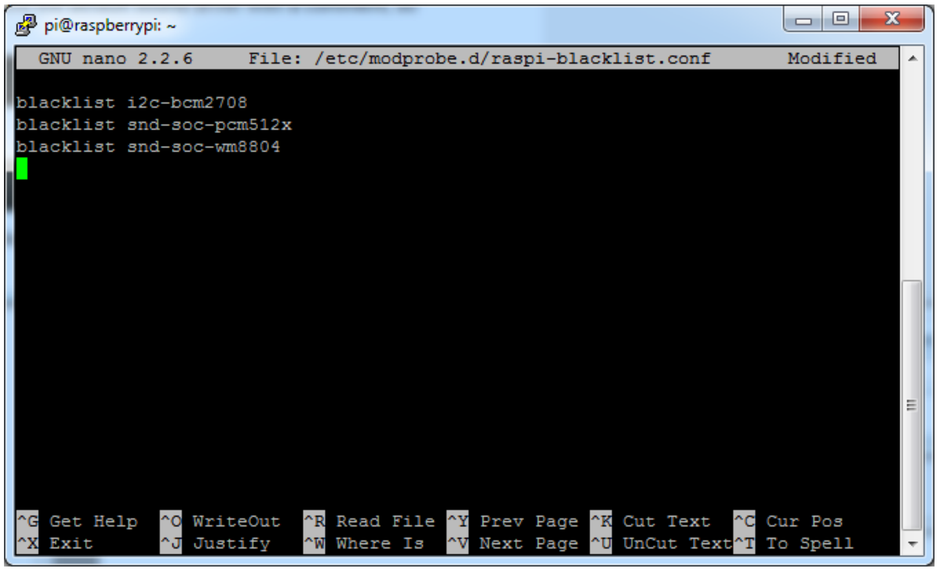
pi@raspberrypi: ~

```
pi@raspberrypi:~ $ sudo nano /etc/modprobe.d/raspi-blacklist.conf
```

If the file is empty, just skip this step

However, if you see the following lines:

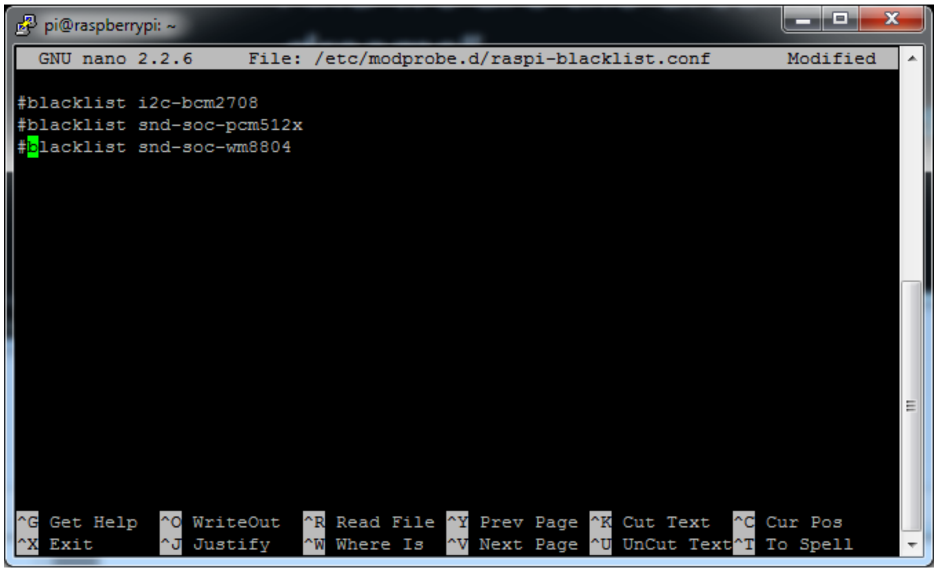
```
blacklist i2c-bcm2708
blacklist snd-soc-pcm512x
blacklist snd-soc-wm8804
```



pi@raspberrypi: ~

```
  GNU nano 2.2.6      File: /etc/modprobe.d/raspi-blacklist.conf        Modified

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



^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Page ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is  ^V Next Page ^U UnCut Text^T To Spell
```

Update the lines by putting a # before each line



```
pi@raspberrypi: ~
  GNU nano 2.2.6        File: /etc/modprobe.d/raspi-blacklist.conf        Modified

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

^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Page ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is  ^V Next Page ^U UnCut Text^T To Spell
```

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

Disable headphone audio (if it's set)

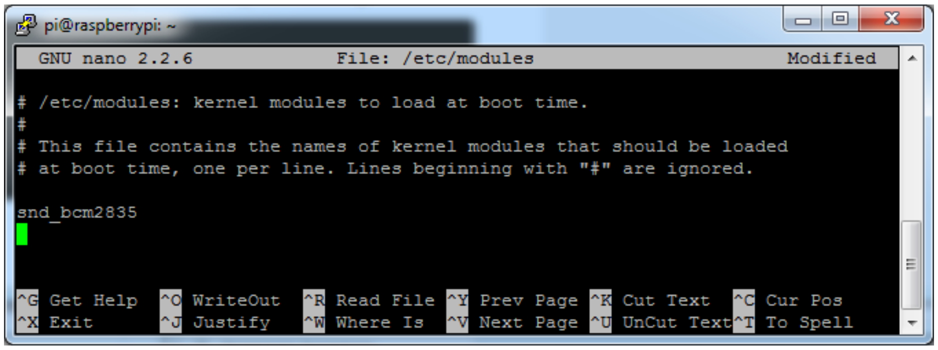
Edit the raspi modules list with

`sudo nano /etc/modules`

If the file is empty, just skip this step

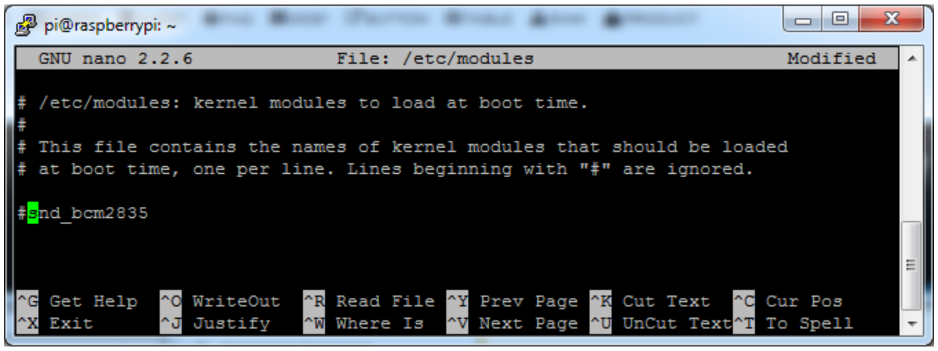
However, if you see the following line:

`snd_bcm2835`



```
pi@raspberrypi: ~
  GNU nano 2.2.6              File: /etc/modules                          Modified

# /etc/modules: kernel modules to load at boot time.
#
# This file contains the names of kernel modules that should be loaded
# at boot time, one per line. Lines beginning with "#" are ignored.

snd_bcm2835


^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Page ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is  ^V Next Page ^U UnCut Text^T To Spell
```

Put a # in front of it



```
pi@raspberrypi: ~
  GNU nano 2.2.6              File: /etc/modules                    Modified

# /etc/modules: kernel modules to load at boot time.
#
# This file contains the names of kernel modules that should be loaded
# at boot time, one per line. Lines beginning with "#" are ignored.

#snd_bcm2835


^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Page ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is  ^V Next Page ^U UnCut Text^T To Spell
```

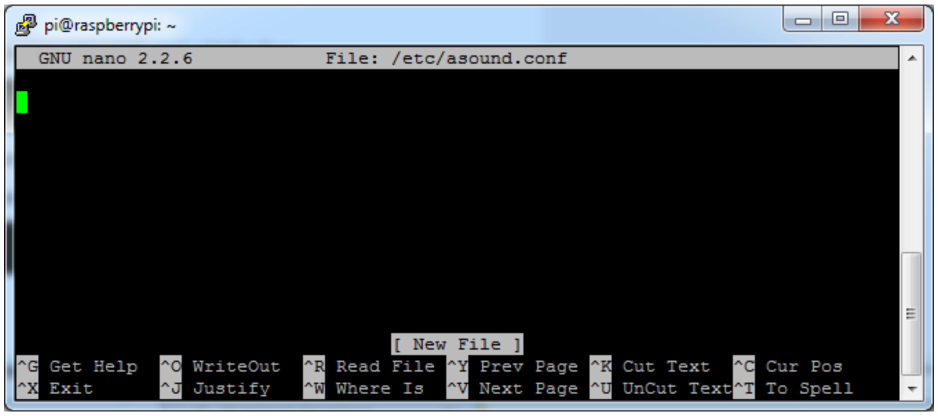
and save with **Control-X Y <return>**

Create asound.conf file

Edit the raspi modules list with

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

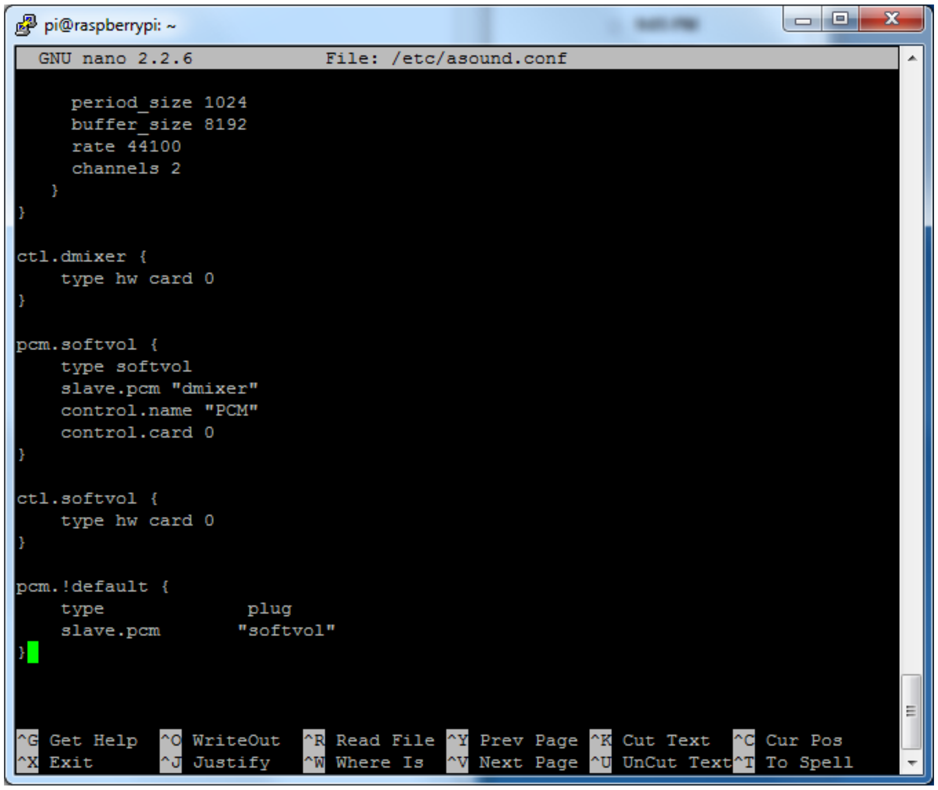
This file ought to be blank!



```
pi@raspberrypi: ~
  GNU nano 2.2.6             File: /etc/asound.conf


                                 [ New File ]
^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Page ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is  ^V Next Page ^U UnCut Text^T To Spell
```

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"
}
```



```
pi@raspberrypi: ~
  GNU nano 2.2.6              File: /etc/asound.conf

     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"
}

^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Page ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is  ^V Next Page ^U UnCut Text^T To Spell
```

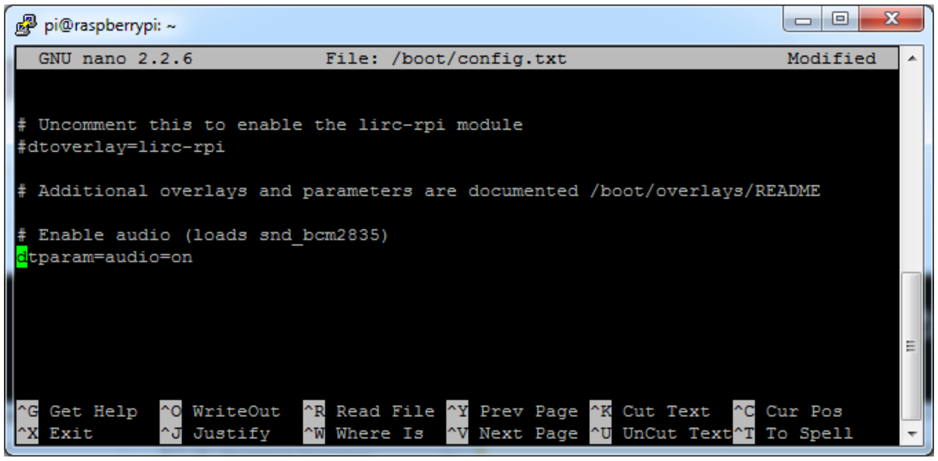
Save the file as usual

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`



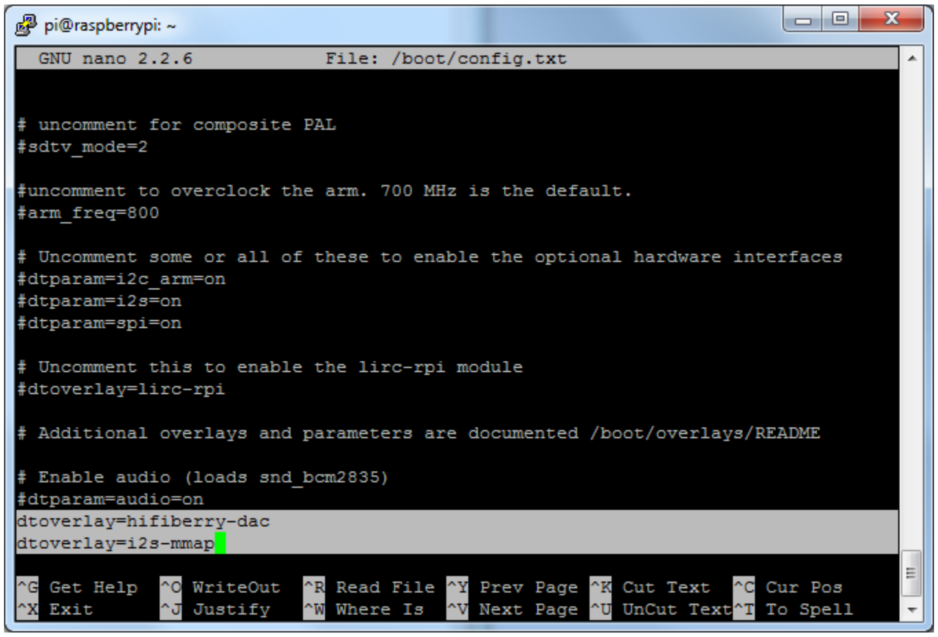
```
pi@raspberrypi: ~
  GNU nano 2.2.6              File: /boot/config.txt                   Modified


# Uncomment this to enable the lirc-rpi module
#dtoverlay=lirc-rpi

# Additional overlays and parameters are documented /boot/overlays/README

# Enable audio (loads snd_bcm2835)
dtparam=audio=on





^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Page ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is  ^V Next Page ^U UnCut Text^T To Spell
```

Disable it by putting a # in front.

Then add:

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

on the next line. Save the file.



```
  GNU nano 2.2.6                File: /boot/config.txt


# uncomment for composite PAL
#sdtv_mode=2

#uncomment to overclock the arm. 700 MHz is the default.
#arm_freq=800

# Uncomment some or all of these to enable the optional hardware interfaces
#dtparam=i2c_arm=on
#dtparam=i2s=on
#dtparam=spi=on

# Uncomment this to enable the lirc-rpi module
#dtoverlay=lirc-rpi

# Additional overlays and parameters are documented /boot/overlays/README

# Enable audio (loads snd_bcm2835)
#dtparam=audio=on
dtoverlay=hifiberry-dac
dtoverlay=i2s-mmap

^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Page ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is  ^V Next Page ^U UnCut Text^T To Spell
```

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

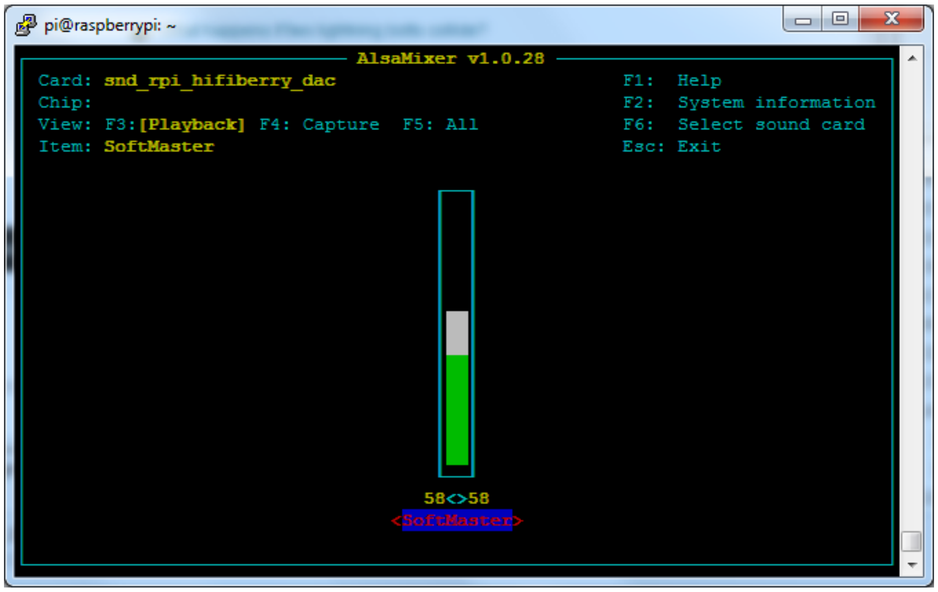
```
sudo apt-get install -y mpg123
mpg123 http://ice1.somafm.com/u80s-128-mp3
```

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!

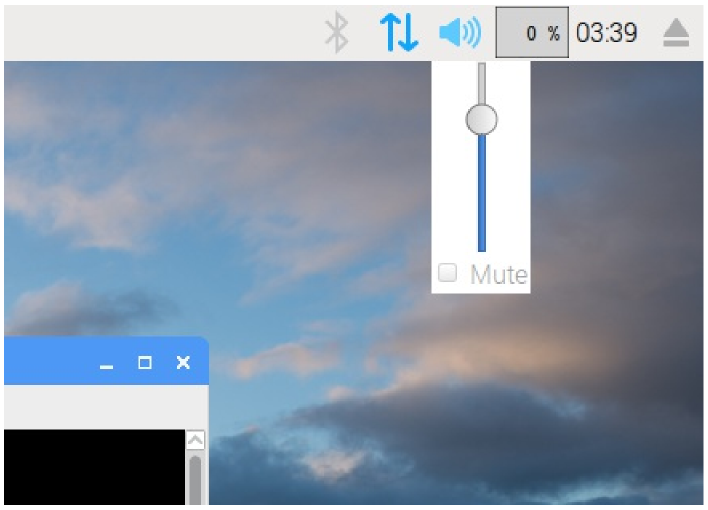
Volume adjustment

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



```
AlsaMixer v1.0.28
Card: snd_rpi_hifiberry_dac                         F1:  Help
Chip:                                               F2:  System information
View: F3:[Playback] F4: Capture  F5: All            F6:  Select sound card
Item: SoftMaster                                    Esc: Exit

                                  58<>58
                               <SoftMaster>
```

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

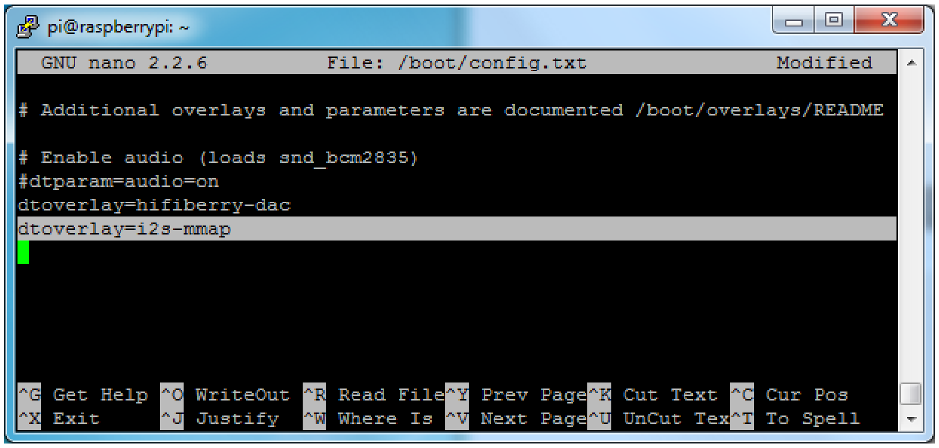
For people who followed our original installation instructions with the simple alsa config, they may find that the I2S audio pops when playing new audio.

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/config.txt** to add `dtoverlay=i2s-mmap`

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



```
  GNU nano 2.2.6            File: /boot/config.txt                  Modified

# Additional overlays and parameters are documented /boot/overlays/README

# Enable audio (loads snd_bcm2835)
#dtparam=audio=on
dtoverlay=hifiberry-dac
dtoverlay=i2s-mmap



^G Get Help  ^O WriteOut  ^R Read File^Y Prev Page^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is ^V Next Page^U UnCut Tex^T To Spell
```

Save and exit.

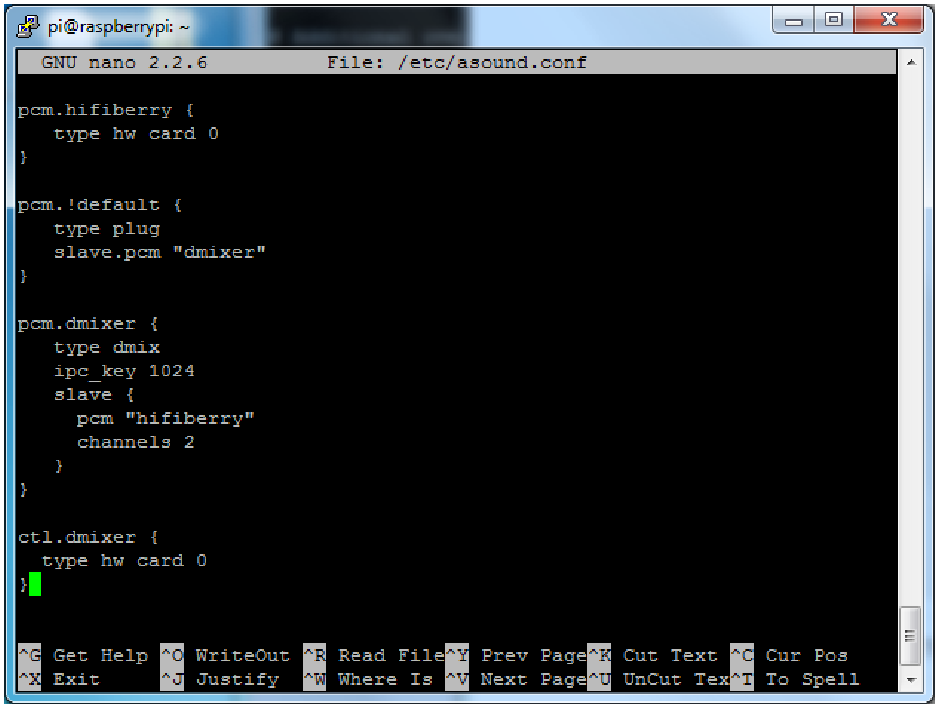
Then change **/etc/asound.conf** to:

```
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 }`



```
  GNU nano 2.2.6            File: /etc/asound.conf

pcm.hifiberry {
   type hw card 0
}

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

pcm.dmixer {
   type dmix
   ipc_key 1024
   slave {
     pcm "hifiberry"
     channels 2
   }
}

ctl.dmixer {
  type hw card 0
}


^G Get Help  ^O WriteOut  ^R Read File^Y Prev Page^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is ^V Next Page^U UnCut Tex^T To Spell
```

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!

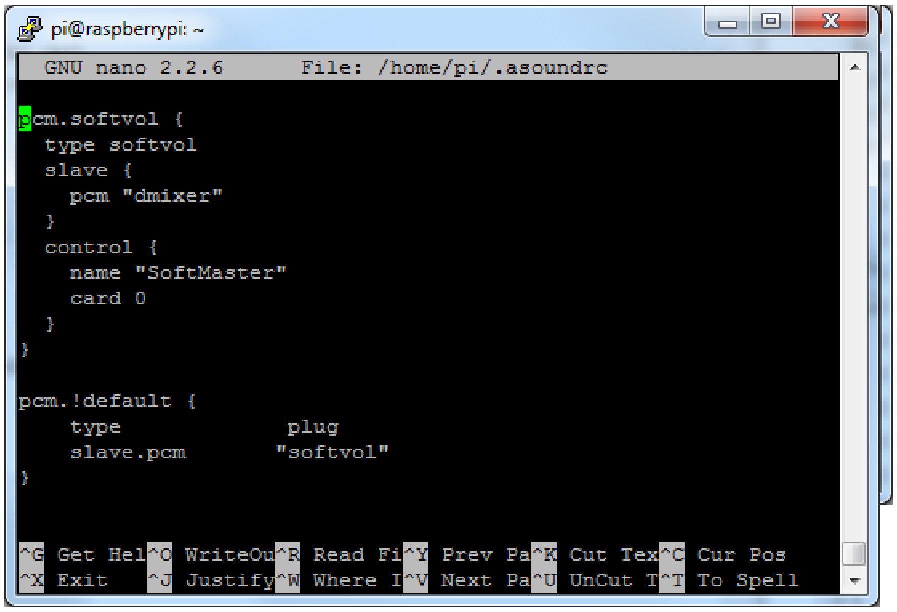
Add software volume control

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!

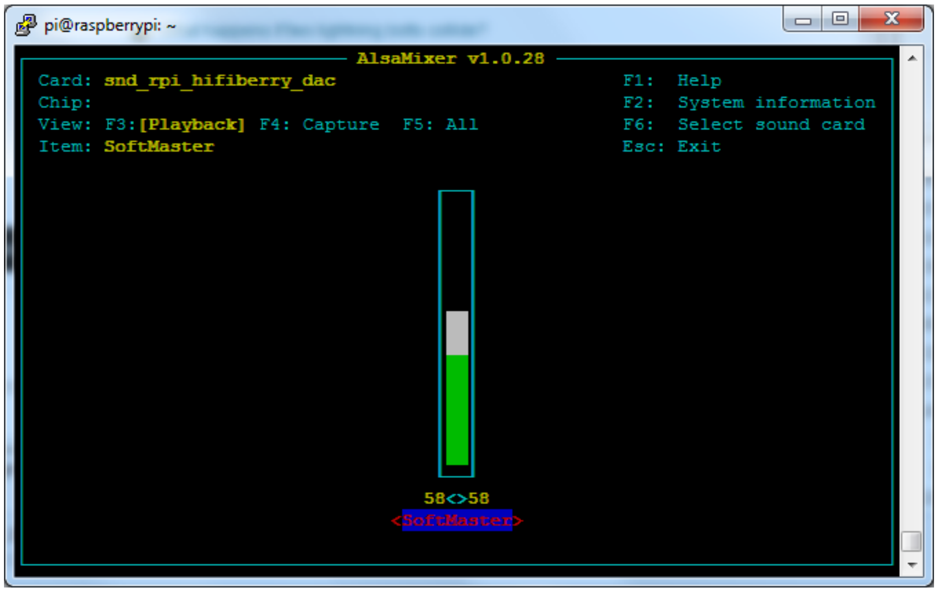

```
pi@raspberrypi: ~
  GNU nano 2.2.6          File: /home/pi/.asoundrc

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

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


^G Get Hel^O WriteOu^R Read Fi^Y Prev Pa^K Cut Tex^C Cur Pos
^X Exit   ^J Justify^W Where I^V Next Pa^U UnCut T^T To Spell
```

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


```
pi@raspberrypi: ~
                        AlsaMixer v1.0.28
Card: snd_rpi_hifiberry_dac                      F1:  Help
Chip:                                            F2:  System information
View: F3:[Playback] F4: Capture  F5: All         F6:  Select sound card
Item: SoftMaster                                 Esc: Exit

                              58<>58
                           <SoftMaster>
```

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 python-pygame
```

Next, download this pygame example zip to your Pi

https://adafru.it/wbp

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/4


#!/usr/bin/python

import sys
import pygame as pg
import os
import time
```

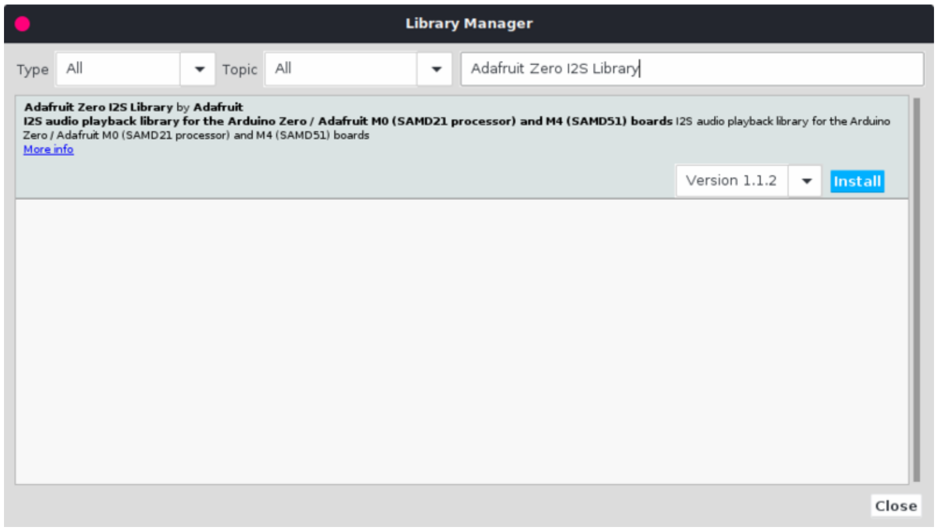
```
def play_music(music_file):
    '''
    stream music with mixer.music module in blocking manner
    this will stream the sound from disk while playing
    '''
    clock = pg.time.Clock()
    try:
        pg.mixer.music.load(music_file)
        print("Music file {} loaded!".format(music_file))
    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) > 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.



Adafruit Zero I2S Library by **Adafruit**
I2S audio playback library for the Arduino Zero / Adafruit M0 (SAMD21 processor) and M4 (SAMD51) boards I2S audio playback library for the Arduino Zero / Adafruit M0 (SAMD21 processor) and M4 (SAMD51) boards

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;
```

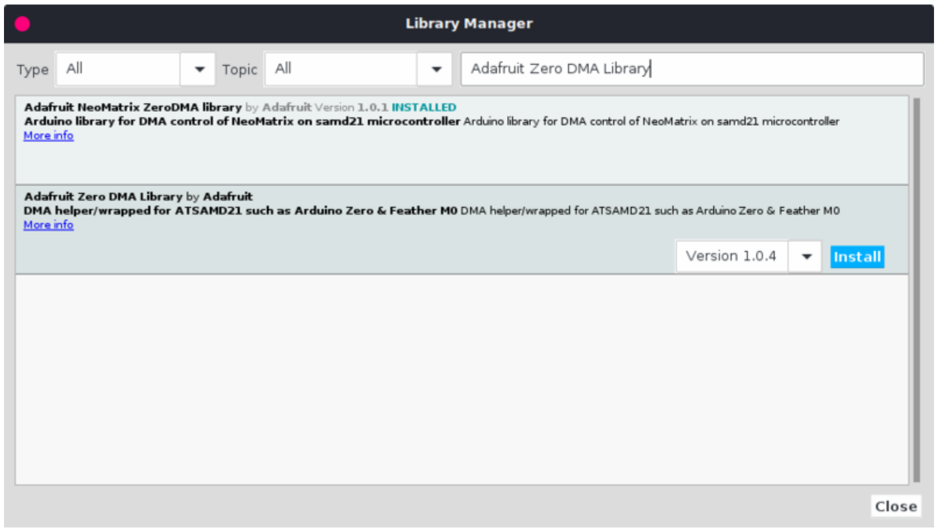
```
// Arduino Zero / Feather M0 I2S audio tone generation example.
// Author: Tony DiCola
//
```

```
// Connect an I2S DAC or amp (like the UDA1334A) to the Arduino Zero
// and play back simple sine, sawtooth, triangle, and square waves.
// Makes your Zero sound like a NES!
//
// NOTE: The I2S signal generated by the Zero does NOT have a MCLK /
// master clock signal.  You must use an I2S receiver that can operate
// without a MCLK signal (like the UDA1334A).
//
// For an Arduino Zero / Feather M0 connect it to you I2S hardware as follows:
// - Digital 0 -> I2S LRCLK / FS (left/right / frame select clock)
// - Digital 1 -> I2S BCLK / SCLK (bit / serial clock)
// - Digital 9 -> I2S DIN / SD (data output)
// - Ground
//
// Released under a MIT license: https://opensource.org/licenses/MIT
#include "Adafruit_ZeroI2S.h"


#define SAMPLERATE_HZ 44100  // The sample rate of the audio.  Higher sample rates have better fidelity,
                             // but these tones are so simple it won't make a difference.  44.1khz is
                             // standard CD quality sound.

#define AMPLITUDE     ((1<<29)-1)   // Set the amplitude of generated waveforms.  This controls how loud
                             // the signals are, and can be any value from 0 to 2**31 - 1.  Start with
                             // a low value to prevent damaging speakers!

#define WAV_SIZE      256    // The size of each generated waveform.  The larger the size the higher
                             // quality the signal.  A size of 256 is more than enough for these simple
                             // waveforms.


// Define the frequency of music notes (from http://www.phy.mtu.edu/~suits/notefreqs.html):
#define C4_HZ      261.63
#define D4_HZ      293.66
#define E4_HZ      329.63
#define F4_HZ      349.23
#define G4_HZ      392.00
#define A4_HZ      440.00
#define B4_HZ      493.88

// Define a C-major scale to play all the notes up and down.
float scale[] = { C4_HZ, D4_HZ, E4_HZ, F4_HZ, G4_HZ, A4_HZ, B4_HZ, A4_HZ, G4_HZ, F4_HZ, E4_HZ, D4_HZ, C4_

// Store basic waveforms in memory.
int32_t sine[WAV_SIZE]     = {0};
int32_t sawtooth[WAV_SIZE] = {0};
int32_t triangle[WAV_SIZE] = {0};
int32_t square[WAV_SIZE]   = {0};

// Create I2S audio transmitter object.
Adafruit_ZeroI2S i2s;

#define Serial Serial

void generateSine(int32_t amplitude, int32_t* buffer, uint16_t length) {
  // Generate a sine wave signal with the provided amplitude and store it in
  // the provided buffer of size length.
  for (int i=0; i<length; ++i) {
    buffer[i] = int32_t(float(amplitude)*sin(2.0*PI*(1.0/length)*i));
  }
  }
}
void generateSawtooth(int32_t amplitude, int32_t* buffer, uint16_t length) {
  // Generate a sawtooth signal that goes from -amplitude/2 to amplitude/2
  // and store it in the provided buffer of size length.
  float delta = float(amplitude)/float(length);
  for (int i=0; i<length; ++i) {
    buffer[i] = -(amplitude/2)+delta*i;
  }
}

void generateTriangle(int32_t amplitude, int32_t* buffer, uint16_t length) {
  // Generate a triangle wave signal with the provided amplitude and store it in
  // the provided buffer of size length.
  float delta = float(amplitude)/float(length);
  for (int i=0; i<length/2; ++i) {
    buffer[i] = -(amplitude/2)+delta*i;
  }
    for (int i=length/2; i<length; ++i) {
    buffer[i] = (amplitude/2)-delta*(i-length/2);
  }
}

void generateSquare(int32_t amplitude, int32_t* buffer, uint16_t length) {
  // Generate a square wave signal with the provided amplitude and store it in
  // the provided buffer of size length.
  for (int i=0; i<length/2; ++i) {
    buffer[i] = -(amplitude/2);
  }
    for (int i=length/2; i<length; ++i) {
    buffer[i] = (amplitude/2);
  }
}

void playWave(int32_t* buffer, uint16_t length, float frequency, float seconds) {
  // Play back the provided waveform buffer for the specified
  // amount of seconds.
  // First calculate how many samples need to play back to run
  // for the desired amount of seconds.
  uint32_t iterations = seconds*SAMPLERATE_HZ;
  // Then calculate the 'speed' at which we move through the wave
  // buffer based on the frequency of the tone being played.
  float delta = (frequency*length)/float(SAMPLERATE_HZ);
  // Now loop through all the samples and play them, calculating the
  // position within the wave buffer for each moment in time.
  for (uint32_t i=0; i<iterations; ++i) {
    uint16_t pos = uint32_t(i*delta) % length;
    int32_t sample = buffer[pos];
    // Duplicate the sample so it's sent to both the left and right channel.
    // It appears the order is right channel, left channel if you want to write
    // stereo sound.
    i2s.write(sample, sample);
  }
}

void setup() {
  // Configure serial port.
  Serial.begin(115200);
  Serial.println("Zero I2S Audio Tone Generator");
  // Initialize the I2S transmitter.
  if (!i2s.begin(I2S_32_BIT, SAMPLERATE_HZ)) {
    Serial.println("Failed to initialize I2S transmitter!");
    while (1);
  }
  i2s.enableTx();

  // Generate waveforms.
  generateSine(AMPLITUDE, sine, WAV_SIZE);
  generateSawtooth(AMPLITUDE, sawtooth, WAV_SIZE);
  generateTriangle(AMPLITUDE, triangle, WAV_SIZE);
  generateSquare(AMPLITUDE, square, WAV_SIZE);
}

void loop() {
  Serial.println("Sine wave");
  for (int i=0; i<sizeof(scale)/sizeof(float); ++i) {
    // Play the note for a quarter of a second.
    playWave(sine, WAV_SIZE, scale[i], 0.25);
    // Pause for a tenth of a second between notes.
    delay(100);
  }
  Serial.println("Sawtooth wave");
  for (int i=0; i<sizeof(scale)/sizeof(float); ++i) {
    // Play the note for a quarter of a second.
    playWave(sawtooth, WAV_SIZE, scale[i], 0.25);
    // Pause for a tenth of a second between notes.
    delay(100);
  }
  Serial.println("Triangle wave");
  for (int i=0; i<sizeof(scale)/sizeof(float); ++i) {
    // Play the note for a quarter of a second.
    playWave(triangle, WAV_SIZE, scale[i], 0.25);
    // Pause for a tenth of a second between notes.
    delay(100);
  }
  Serial.println("Square wave");
  for (int i=0; i<sizeof(scale)/sizeof(float); ++i) {
    // Play the note for a quarter of a second.
    playWave(square, WAV_SIZE, scale[i], 0.25);
    // Pause for a tenth of a second between notes.
    delay(100);
  }
}
```

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:


Library Manager

Type All Topic All Adafruit Zero DMA Library

Adafruit NeoMatrix ZeroDMA library by Adafruit Version 1.0.1 INSTALLED
Arduino library for DMA control of NeoMatrix on samd21 microcontroller Arduino library for DMA control of NeoMatrix on samd21 microcontroller
More info

Adafruit Zero DMA Library by **Adafruit**
DMA helper/wrapped for ATSAMD21 such as Arduino Zero & Feather M0 DMA helper/wrapped for ATSAMD21 such as Arduino Zero & Feather M0
More info

Version 1.0.4 Install

Close

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.

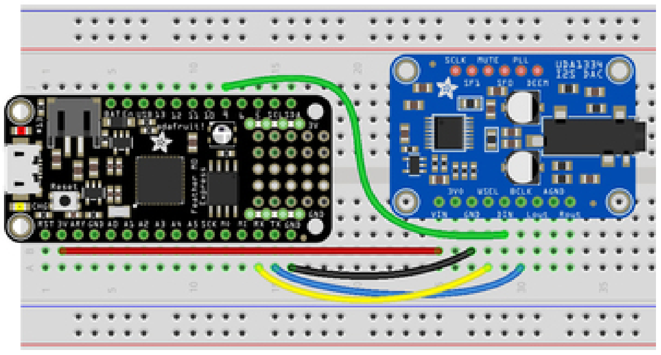
Note that Trinket M0, Gemma M0 and ItsyBitsy M4 do not support I2S (the last one is not a typo!)

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

Wiring

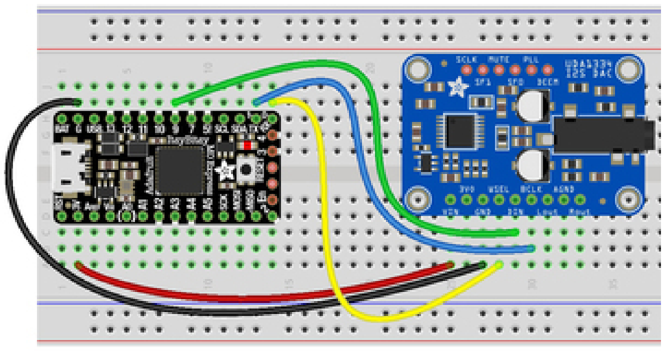
The following wiring diagrams show how to connect the UDA1334 breakout to your CircuitPython board. You'll be using voltage in, ground, bit clock, word select and data pins.

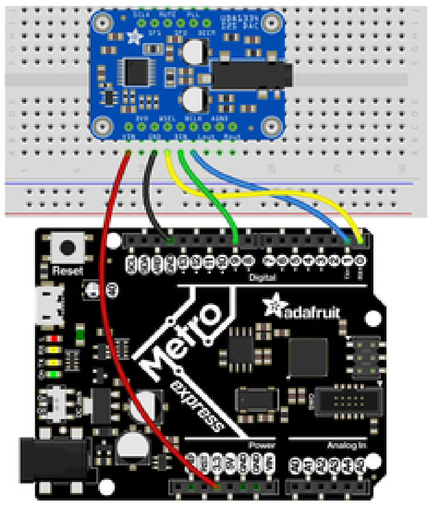
- **VIN** is the **red** wire.
- **GND** is the **black** wire.
- **BCLK** is the **blue** wire.
- **WSEL** is the **yellow** wire.
- **DIN** is the **green** wire.

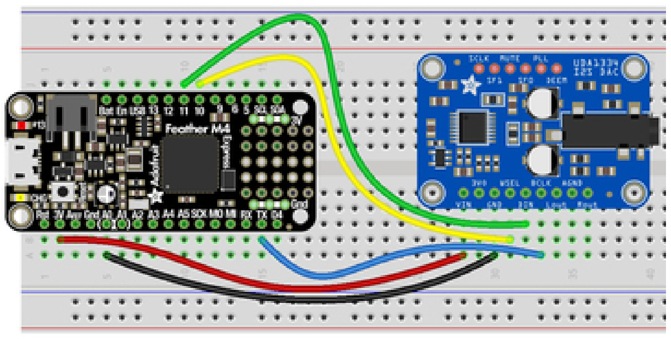


For Feather M0 Express, ItsyBitsy M0 Express and Metro M0 Express:

- 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 **WSEL on the breakout** to **D0/RX on the board**.
- Connect **DIN on the breakout** to **D9 on the board**.

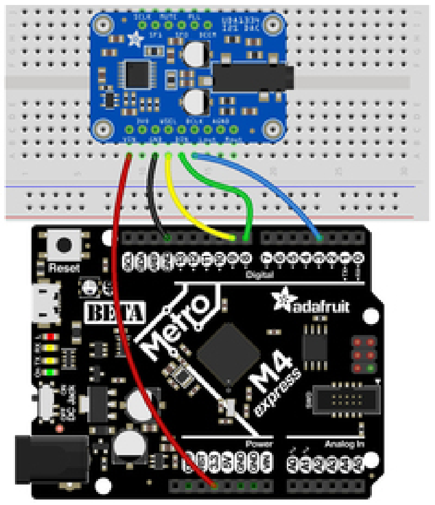






For Feather M4 Express:

- 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 **WSEL 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 **WSEL 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 audio jack on the breakout. The second plays a wave file. Let's take a look!

The default volume of the audio in the following code is very high. Do put on plugged in headpones before first running the code to check the volume.

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.

```
import time
import array
import math
import audioio
import board
import audiobusio

tone_volume = 0.1  # 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))

# 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 = audioio.RawSample(sine_wave)

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. Remember, listen for it without headphones on your ears first as the volume is quite high.

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.

https://adafru.it/BTM

https://adafru.it/BTM

```
import audioio
import board
import audiobusio

wave_file = open("StreetChicken.wav", "rb")
wave = audioio.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. Remember, listen for it without headphones on your ears first as the volume is quite high.

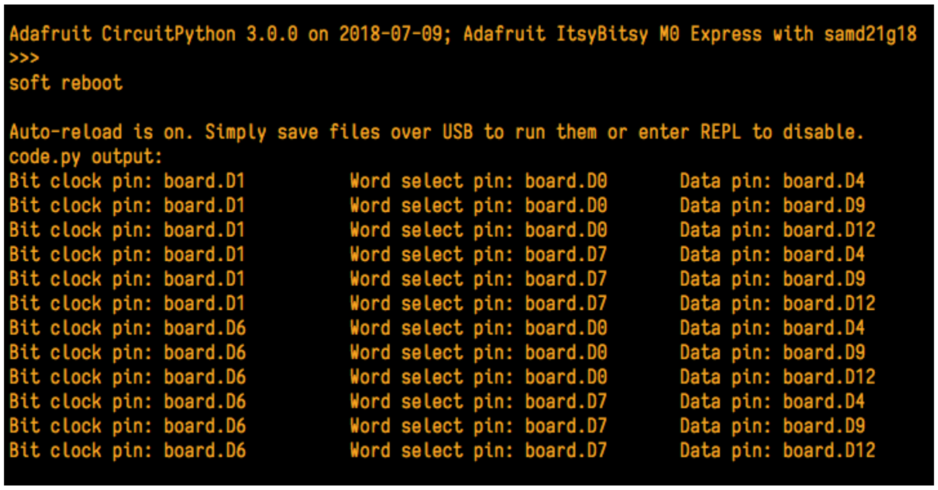
There's plenty you can do with this example. Try playing a different wave file, or, instead of including `while audio.playing: pass` , include a `time.sleep()` to have it play for a specified number of seconds. Check out the Audio Out page in the CircuitPython Essentials guide (https://adafru.it/BRj)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

For more information about I2SOut, check out https://circuitpython.readthedocs.io/en/latest/shared-bindings/audiobusio/I2SOut.html

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
```

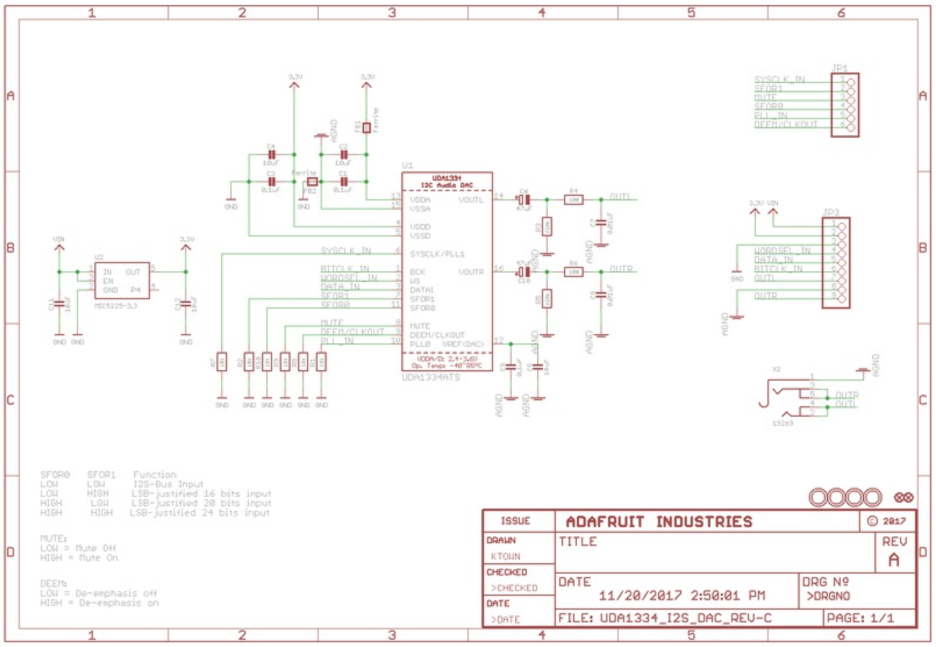
```python
import board
import audiobusio
from microcontroller import Pin


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
            else:
                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
```

Downloads

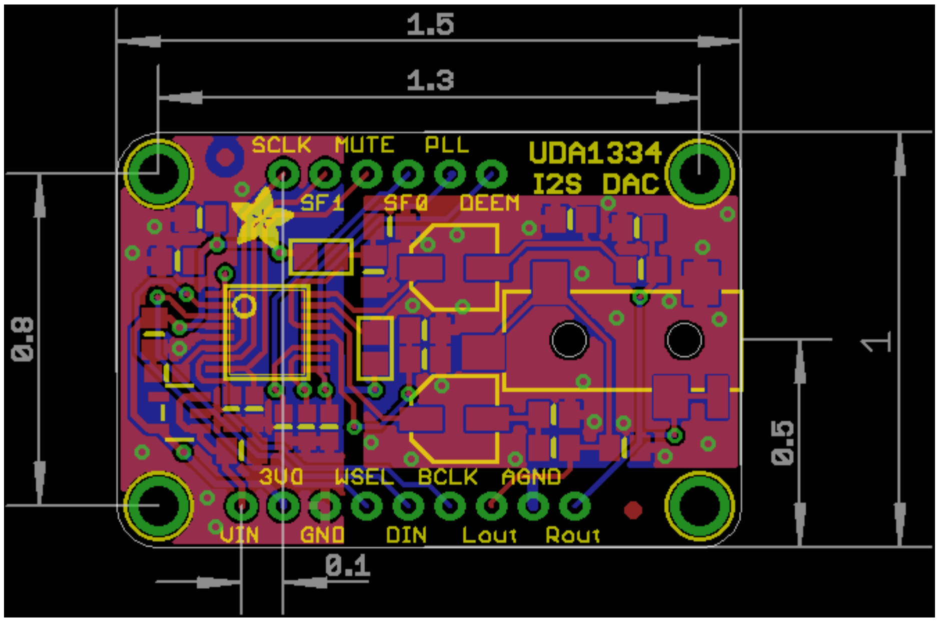
Files

- EagleCAD PCB Files (https://adafru.it/BtN)
- Fritzing object in Adafruit Fritzing library (https://adafru.it/aP3)
- UDA1334A Datasheet (https://adafru.it/BtO)

Schematic & Fabrication Print



UDA1334 I2S Audio DAC
SFOR0 SFOR1 Function
LOW LOW I2S-Bus Input
LOW HIGH LSB-justified 16 bits input
HIGH LOW LSB-justified 20 bits input
HIGH HIGH LSB-justified 24 bits input
MUTE:
LOW = Mute Off
HIGH = Mute On
DEEM:
LOW = De-emphasis off
HIGH = De-emphasis on
ADAFRUIT INDUSTRIES
TITLE
REV A
DATE 11/20/2017 2:50:01 PM
FILE: UDA1334_I2S_DAC_REV-C
PAGE: 1/1



1.5
1.3
SCLK MUTE PLL
UDA1334 I2S DAC
SF1 SF0 DEEM
3V0 WSEL BCLK AGND
VIN GND DIN Lout Rout
0.8
0.5
1
0.1