CircuitPython Tutorial: MP3 Playback on RP2040 with CircuitPython

I2S, or Inter-IC Sound, is a standard for transmitting digital audio data. It requires at least three connections. The first connection is a clock, called **bit clock** (**BCLK**, or sometimes written as serial clock or SCK). The second connection, which determines the channel (left or right) being sent, is called **word select** (**WS**). When stereo data is sent, WS is toggled so that the left and right channels are sent alternately, one data word at a time. The third connection, which transmits the data, is called **serial data** (**SD**).

Typically, there is a **transmitter** device which generates the bit clock, word select signal, and the data, and sends them to a **receiver** device. In this case, your microcontroller acts as the transmitter, and an I2S breakout acts as the receiver. The UDA1334A is an example of an I2S breakout that provides line-level output as well as output to a headphone jack. The MAX98357A is an example of an I2S class D amplifier that allows you to connect directly to a speaker.

I2S and CircuitPython

CircuitPython supports sending I2S audio signals using the `audiobusio` module, making it simple to use the I2S interface with your microcontroller.

In this section, you'll learn how to use CircuitPython to play different types of audio using I2S, including tones, WAV files and MP3 files.

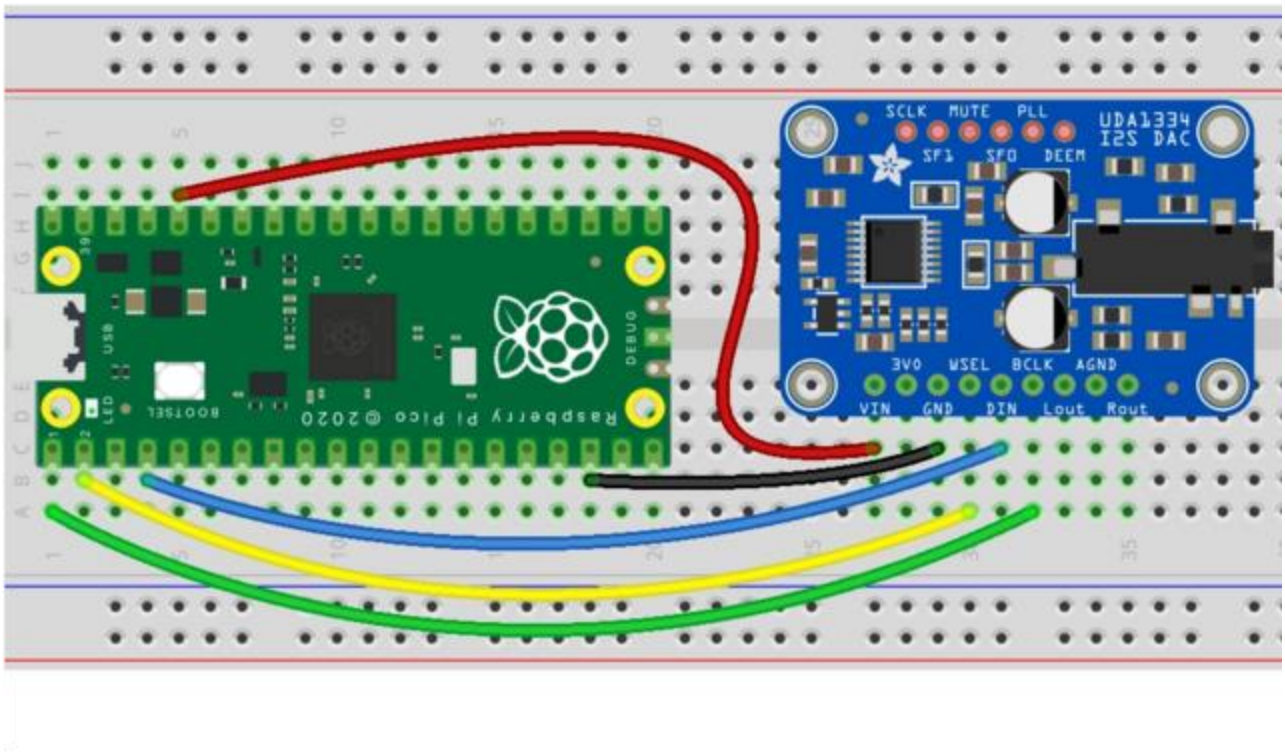
Necessary Hardware

You'll need the following additional hardware to complete the examples on this page.

- **A pair of headphones or speakers with a headphone jack**

Wiring the UDA1334A

Connect the UDA1334A breakout to your microcontroller as follows.
The bit clock and word select pins must be on consecutive pins! They can be on any pins you like, but they must be in consecutive order, for example, A0 for bit clock and A1 for word select.



- **Pico 3V3** to **breakout VIN**
- **Pico GND** to **breakout GND**
- **Pico GP0** to **breakout BCLK**
- **Pico GP1** to **breakout WSEL**
- **Pico GP2** to **breakout DIN**

I2S Tone Playback

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 frequency (in Hz) of the tone by changing the associated variables. Inside the loop, you play the tone for one second and stop it for one second.

Update your **code.py** to the following.

Click the **Download Project Bundle** button below to download the necessary libraries and the **code.py** file in a zip file. Extract the contents of the zip file, open the folder that matches your CircuitPython version, and copy the **code.py** file to your **CIRCUITPY** drive.

Download Project Bundle
Copy Code

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

"""
CircuitPython I2S Tone playback example.
Plays a tone for one second on, one
second off, in a loop.
"""
import time
import array
import math
import audiocore
import board
import audiobusio

audio = audiobusio.I2SOut(board.GP0, board.GP1, board.GP2)

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))
sine_wave_sample = audiocore.RawSample(sine_wave)

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

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

You can try changing the `440` Hz of the tone to produce a tone of a different pitch. Try changing the number of seconds in `time.sleep()` to produce longer or shorter tones.

I2S WAV File Playback

The second example plays a WAV file. You open the file in a readable format. Then, you play the file and, once finished, print `Done playing!` to the serial console.

Update your **code.py** to the following.

Click the **Download Project Bundle** button below to download the necessary libraries and the **code.py** file in a zip file. Extract the contents of the zip file, open the folder that matches your CircuitPython version, and copy the **StreetChicken.wav** file and the **code.py** file to your **CIRCUITPY** drive.

Download Project Bundle

Copy Code

```
# SPDX-FileCopyrightText: 2021 Kattni Rembor for Adafruit Industries
#
# SPDX-License-Identifier: MIT

"""
CircuitPython I2S WAV file playback.
Plays a WAV file once.
"""
import audiocore
import board
import audiobusio

audio = audiobusio.I2SOut(board.GP0, board.GP1, board.GP2)

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

print("Playing wav file!")
audio.play(wav)
while audio.playing:
    pass
print("Done!")
```

Now you'll hear the wave file play, and on completion, print `Done Playing!` to the serial console.

You can play a different WAV file by updating `"StreetChicken.wav"` to be the name of your CircuitPython-compatible WAV file.

You can do other things while the WAV file plays! There is a `pass` in this example where you can include other code, such as code to blink an LED.

I2S MP3 File Playback

The third example plays an MP3 file. First, you open the file in a readable format. Then you play the MP3 and, once finished, print `Done playing!` to the serial console.

CircuitPython supports any MP3 file, as long as it is the right bit rate and sample rate for your board.
Mono and **stereo** files less than **64kbit/s** work, with sample rates from **8kHz** to **24kHz**. The RP2040 has a PWM output with 10 bits, so there's not much point in using high bit rates.
Update your **code.py** to the following.

Click the **Download Project Bundle** button below to download the necessary libraries and the **code.py** file in a zip file. Extract the contents of the zip file, open the folder that matches your CircuitPython version, and copy the **slow.mp3** file and the **code.py** file to your **CIRCUITPY** drive.

Download Project Bundle

Copy Code

```
# SPDX-FileCopyrightText: 2021 Kattni Rembor for Adafruit Industries
#
# SPDX-License-Identifier: MIT

"""
CircuitPython I2S MP3 playback example.
Plays a single MP3 once.
"""
import board
import audiomp3
import audiobusio

audio = audiobusio.I2SOut(board.GP0, board.GP1, board.GP2)

mp3 = audiomp3.MP3Decoder(open("slow.mp3", "rb"))

audio.play(mp3)
while audio.playing:
    pass

print("Done playing!")
```

Now you'll hear the MP3 play, and on completion, print `Done Playing!` to the serial console.

You can play a different CircuitPython-compatible MP3 by updating `"slow.mp3"` to the name of your MP3 file.

CircuitPython I2S-Compatible Pin Combinations

I2S audio is supported on specific pins. The good news is, there's a simple way to find out which pins support audio playback.

Save the following file as **code.py** on your **CIRCUITPY** drive. Then, connect to the serial console to see a list of pins printed out. This file runs only once, so if you do not see anything in the output, press CTRL+D to reload and run the code again.

Download Project Bundle

Copy Code

```
# SPDX-FileCopyrightText: 2021 Kattni Rembor for Adafruit Industries
# SPDX-License-Identifier: MIT
"""
CircuitPython I2S Pin Combination Identification Script
"""
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 = [
        getattr(board, p)
        for p in [
            # This is not an exhaustive list of unexposed pins. Your results
            # may include other pins that you cannot easily connect to.
            "NEOPIXEL",
            "DOTSTAR_CLOCK",
            "DOTSTAR_DATA",
            "APA102_SCK",
            "APA102_MOSI",
            "LED",
            "SWITCH",
            "BUTTON",
        ]
        if p in dir(board)
    ]
    pins = [
        pin
        for pin in [getattr(board, p) for p in dir(board)]
        if isinstance(pin, Pin) and pin not in exclude
    ]
    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
```

`I2SOut` – Output an I2S audio signal

I2S is used to output an audio signal on an I2S bus.

class **audiobusio.I2SOut**(*bit_clock, word_select, data, *, left_justified*)

Create a I2SOut object associated with the given pins.

Parameters:

- **bit_clock** (*Pin*) – The bit clock (or serial clock) pin
- **word_select** (*Pin*) – The word select (or left/right clock) pin
- **data** (*Pin*) – The data pin
- **left_justified** (*bool*) – True when data bits are aligned with the word select clock. False when they are shifted by one to match classic I2S protocol.

Simple 8ksps 440 Hz sine wave on Metro M0 Express using UDA1334 Breakout:

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

# Generate one period of sine wave.
length = 8000 // 440
sine_wave = array.array("H", [0] * length)
for i in range(length):
    sine_wave[i] = int(math.sin(math.pi * 2 * i / 18) * (2 ** 15) + 2 ** 15)

sine_wave = audiocore.RawSample(sine_wave, sample_rate=8000)
i2s = audiobusio.I2SOut(board.D1, board.D0, board.D9)
i2s.play(sine_wave, loop=True)
time.sleep(1)
i2s.stop()
```

Playing a wave file from flash:

```
import board
import audioio
import audiocore
import audiobusio
import digitalio


f = open("cplay-5.1-16bit-16khz.wav", "rb")
wav = audiocore.WaveFile(f)

a = audiobusio.I2SOut(board.D1, board.D0, board.D9)

print("playing")
a.play(wav)
while a.playing:
  pass
print("stopped")
```

deinit()

Deinitialises the I2SOut and releases any hardware resources for reuse.

__enter__()

No-op used by Context Managers.

__exit__()

Automatically deinitializes the hardware when exiting a context. See Lifetime and ContextManagers for more info.

play(*sample*, *, *loop=False*)

Plays the sample once when loop=False and continuously when loop=True. Does not block. Use `playing` to block.

Sample must be an `audiocore.WaveFile`, `audiocore.RawSample`, or `audiomixer.Mixer`.

The sample itself should consist of 8 bit or 16 bit samples.

stop()

Stops playback.

playing

True when the audio sample is being output. (read-only)

pause()

Stops playback temporarily while remembering the position. Use `resume` to resume playback.

resume()

Resumes sample playback after `pause()`.

paused

True when playback is paused. (read-only)