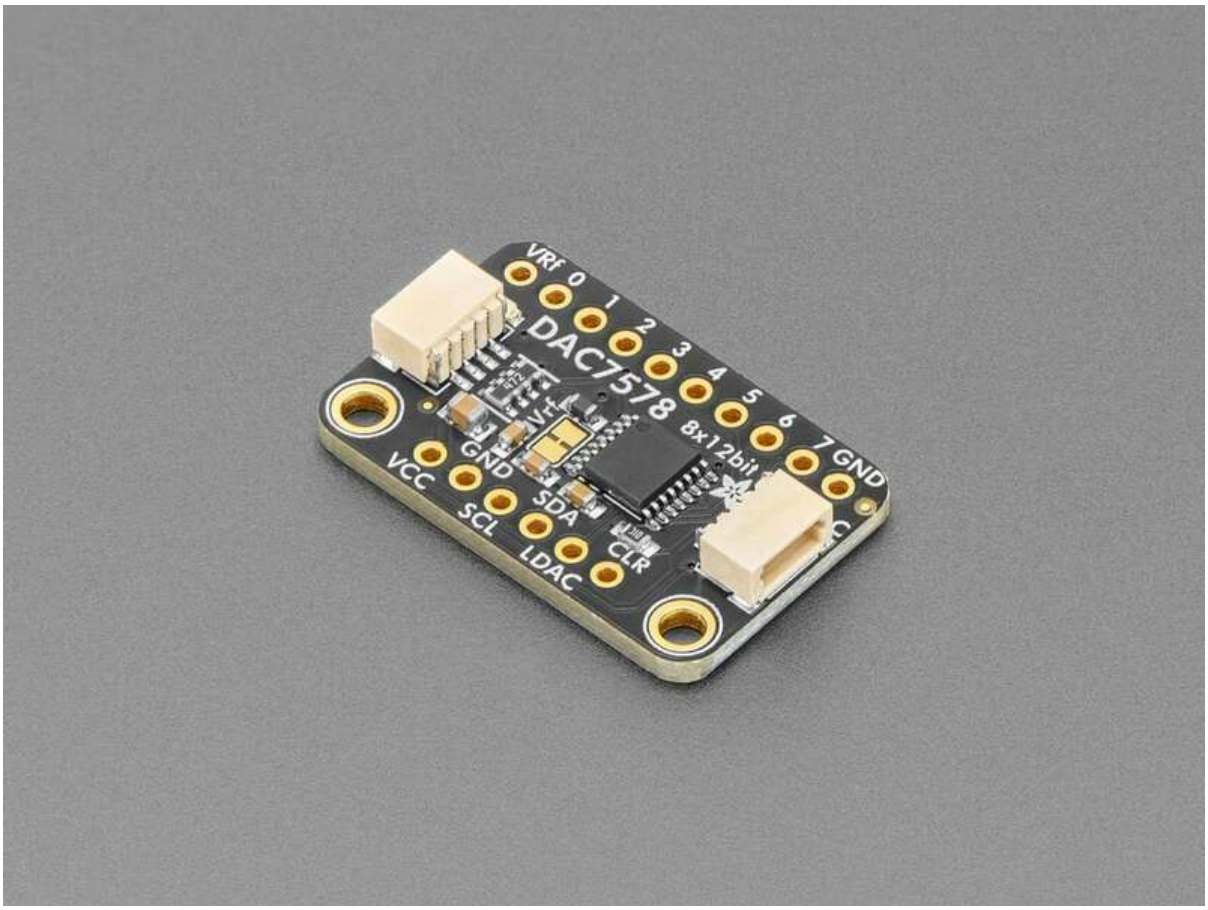
# Adafruit DACx578 - 8 x Channel I2C DAC

Created by Liz Clark

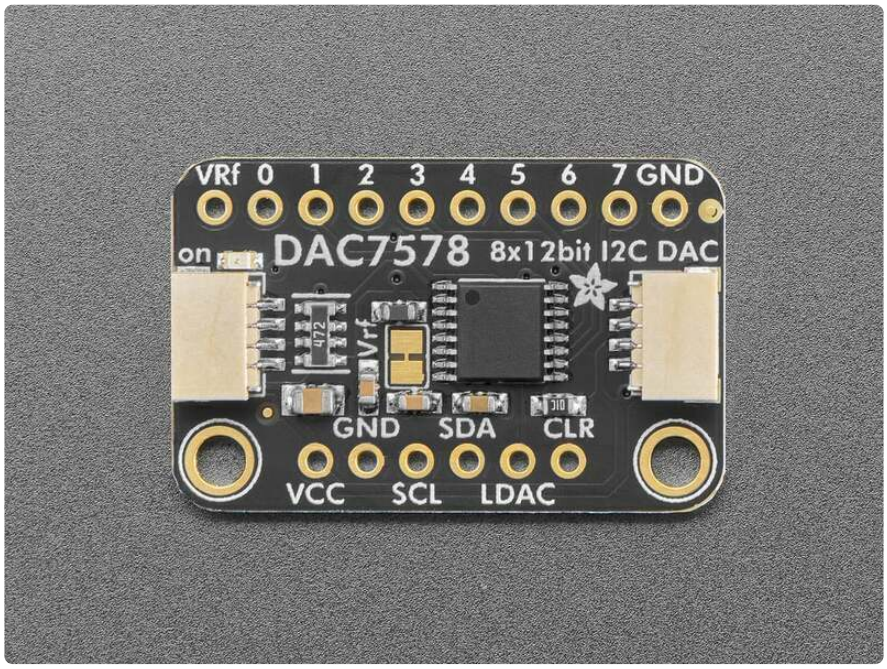



https://learn.adafruit.com/adafruit-dac7578-8-x-channel-12-bit-i2c-dac

Last updated on 2025-03-12 09:06:23 AM EDT

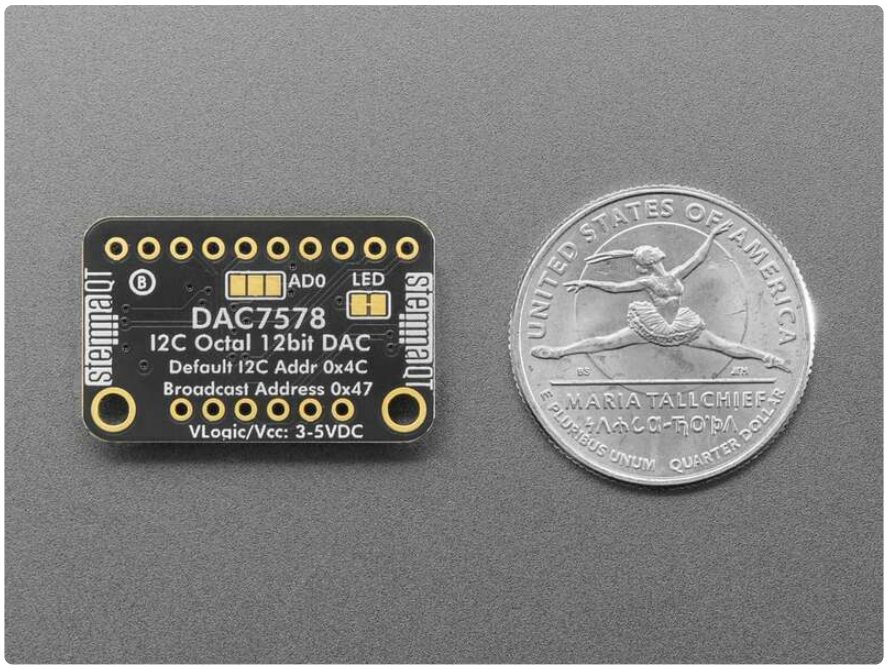
# Table of Contents

# Overview

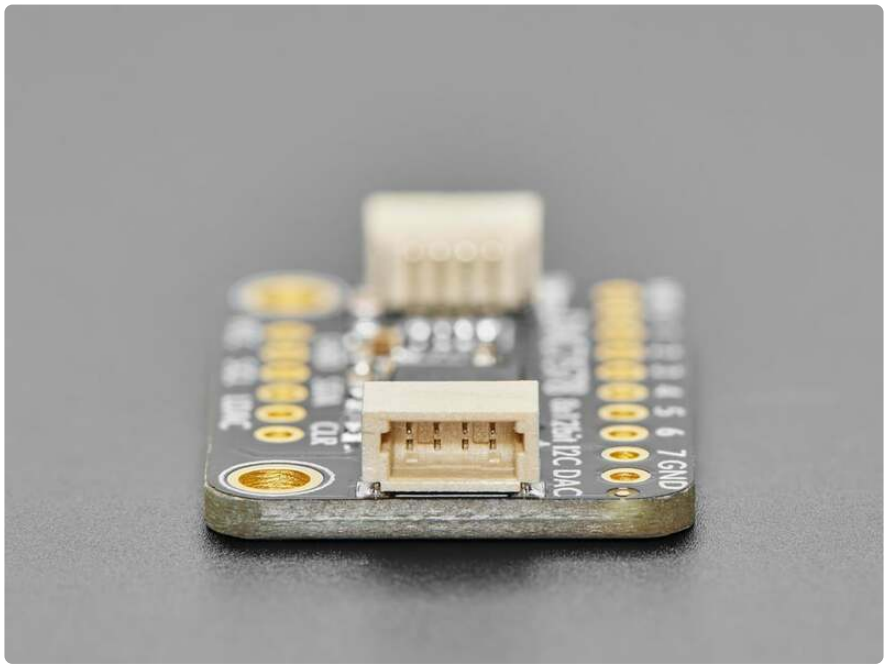



There are two variants of this DAC: the 10-bit DAC6578 and the 12-bit DAC7578.

If you've ever said to yourself, "Gee, I wish I had eight DACs that came in a single package", well I have good news. **The Adafruit DACx578 Breakouts are** the answer to your wishes! Within its little package, the DACx578 has eight DACs for whatever voltage setting needs you may have, easily controlled over I2C.

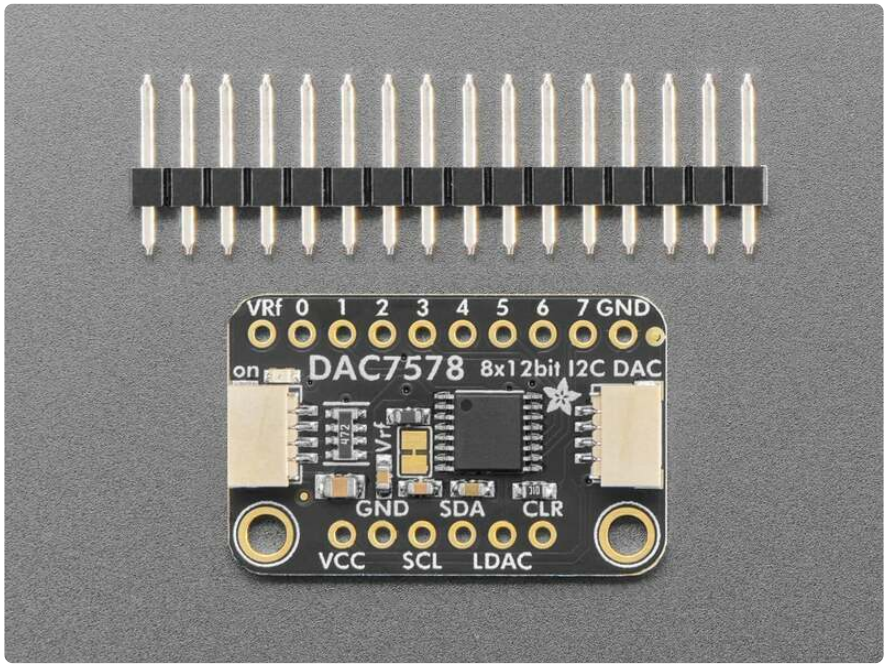



The DACx578 can easily work with 3.3V or 5V power and logic so no matter what microcontroller / microcomputer you're using, you'll get full range output. Note that this DAC is 'ratiometric' - that means that 0 is ground and 4095 is whatever the Vref voltage is. By default the Vref is tied to the power pin, but you could cut a jumper to set the top voltage to anything between 0 and the power pin. E.g. if you want a precision 4.095V reference, you'll need to power with 5V, and provide 4.095V on the VRef pin.




The breakout for the DACx578 is populated with the required support circuitry to use it with your microcontroller of choice or Blinka-supported computer. The SparkFun Qwiic (https://adafru.it/Fpw) compatible **STEMMA QT** (https://adafru.it/Ft4) JST SH connectors ease the process of connecting the DACx578 to your project and allows

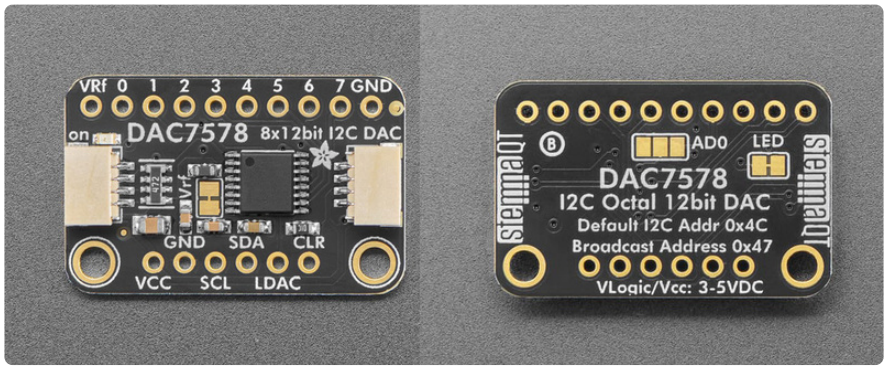
you to easily share an I2C bus with other STEMMA QT, Qwiic, Grove, or other compatible sensors. **QT Cable is not included**, but we have a variety in the shop (https://adafru.it/17VE)




Our drivers, wiring diagrams, and example code for Arduino, CircuitPython and Python make it easy to get started so you can get rolling with your project instead of figuring out how to wire things or get the code working.

---

# Pinouts




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

There are two variants of this DAC: the 10-bit DAC6578 and the 12-bit DAC7578. The pinout is the same for both.

## Power Pins

- **VCC** - this is the power pin. To power the board, give it the same power as the logic level of your microcontroller - e.g. for a 5V micro like Arduino, use 5V.
- **VRf** - this is the reference voltage pin. By default, it is tied to **VCC** and as a result will share the same voltage level.
- **Vrf Jumper** - this is the reference voltage jumper. It is located on the front of the board, to the left of the DACx578 chip. If you cut this jumper, the **VRf** pin is disconnected from **VCC** and can be connected to to any voltage between 0 and the **VCC** voltage level. For example, if you want a precision 4.095V reference, you'll need to cut the **Vrf jumper**, power with 5V to **VCC**, and provide 4.095V on the **VRf** pin.
- **GND** - common ground for power and logic

## I2C Logic Pins

- **SCL** - I2C clock pin, connect to your microcontroller's I2C clock line. This pin can use 3-5V logic, and there's a **10K pullup** on this pin.
- **SDA** - I2C data pin, connect to your microcontroller's I2C data line. This pin can use 3-5V logic, and there's a **10K pullup** on this pin.
- STEMMA QT **(https://adafru.it/Ft4) -** These connectors allow you to connect to dev boards with **STEMMA QT** (qwiic) connectors or to other things with various associated accessories (https://adafru.it/Ft6)

## DAC Output Pins

Along the top edge of the board are the DAC output pins. They are labeled **0**-**7**. Each pin is connected to one of the eight channel outputs from the DACx578.

## Other Pins

- **LDAC** - This is the load DAC pin. **LDAC** is used for synchronous or asynchronous updating of the DAC outputs. According to the datasheet, you should be able to toggle this pin to send the buffer to the output pins after using the write only command. However, in testing, this pin has not worked as described.
- **CLR** - This is the clear pin. It clears all DAC registers and buffers, replacing them with the value stored in the clear code register. The register can be set to zero-scale, mid-scale, full-scale, or disabled. The default setting of the clear code register sets the output of all DAC channels to 0V when the **CLR** pin is brought low.

In testing, the LDAC pin has not worked as described in the datasheet. Its believed to be a silicon bug in the DACx578.

## I2C Address Jumper

On the back of the board is a three section jumper labeled **AD0**, directly above the **DACx578 label** on the board silk. This jumper allow you to chain up to 3 of these boards on the same pair of I2C clock and data pins. To do so, you solder the jumpers "closed" by connecting the two pads.

The default I2C address is **0x4C**. This address is set by leaving the address pin floating. None of the jumper pads soldered closed.

If you solder the center jumper pad to the pad to its left, you connect the address pin to GND and set the I2C address to **0x48**.

If you solder the center jumper pad to the pad to its right, you connect the address pin to VCC and set the I2C address to **0x4A**.

The table below shows all possible addresses and the jumper positions.

| ADDR | AD0 |
|---|---|
| 0x48 | GND |
| 0x4A | VCC |
| 0x4C | FLOAT |

## Power LED and LED Jumper

- **Power LED -** In the upper left corner, above the STEMMA connector, on the front of the board, is the power LED, labeled **on**. It is a green LED.

- **LED jumper** - This jumper, labeled **LED** on the board silk, is located on the back of the board, on the right. Cut the trace on this jumper to cut power to the "on" LED.

## Broadcast Addressing

The DACx578 chips support broadcast addressing. This means that you can update DACx578 devices synchronously. This only supports write mode, you cannot perform reads with it. The broadcast address is **0x47**.

---

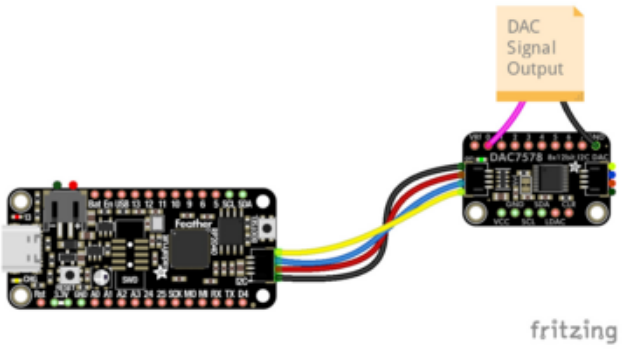
# CircuitPython and Python

It's easy to use the **DACx578** with Python or CircuitPython, and the Adafruit_CircuitPython_x578 (https://adafru.it/1aeQ) module. This module allows you to easily write Python code to write data to the DAC outputs.

You can use this driver with any CircuitPython microcontroller board or with a computer that has GPIO and Python thanks to Adafruit_Blinka, our CircuitPython-for-Python compatibility library (https://adafru.it/BSN).

## CircuitPython Microcontroller Wiring

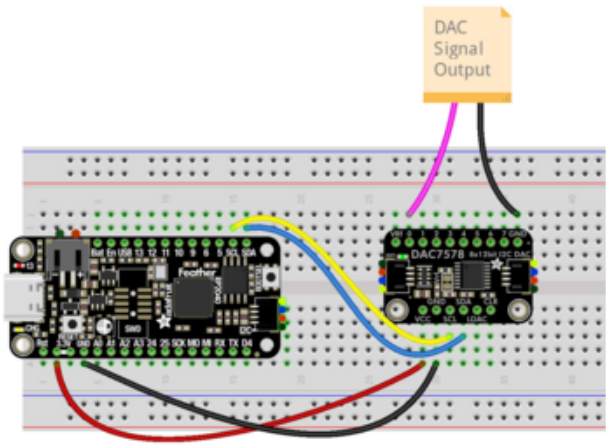
First wire up the DAC to your board exactly as follows. The following is the DAC wired to a Feather RP2040 using the STEMMA connector:




**Board STEMMA 3V** to **DAC VCC (red wire)**
**Board STEMMA GND** to **DAC GND (black wire)**
**Board STEMMA SCL** to **DAC SCL (yellow wire)**
**Board STEMMA SDA** to **DAC SDA (blue wire)**

You can output a signal from any of the 8 DAC channels.

The following is the DAC wired to a Feather RP2040 using a solderless breadboard:



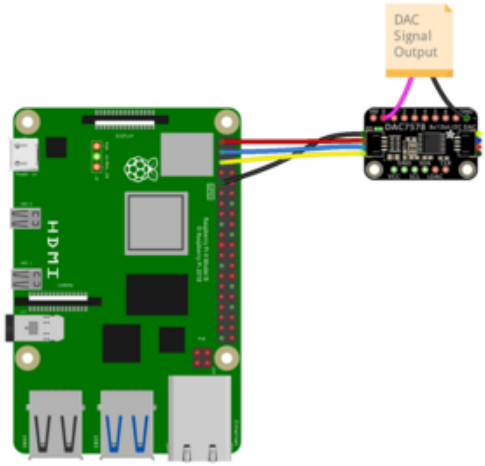

**Board 3V** to **DAC VCC (red wire)**
**Board GND** to **DAC GND (black wire)**
**Board SCL** to **DAC SCL (yellow wire)**
**Board SDA** to **DAC SDA (blue wire)**

You can output a signal from any of the 8 DAC channels.

## Python Computer Wiring

Since there are dozens of Linux computers/boards you can use, we will show wiring for Raspberry Pi. For other platforms, [please visit the guide for CircuitPython on Linux to see whether your platform is supported](https://adafru.it/BSN) (https://adafru.it/BSN).

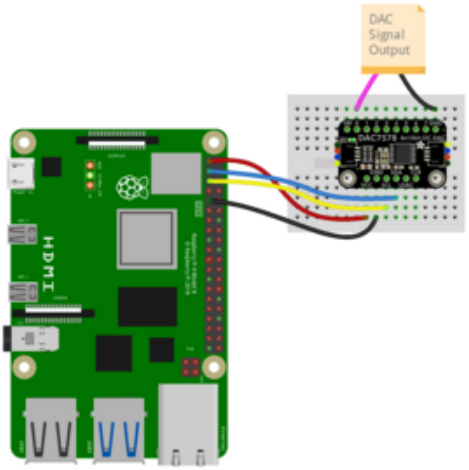
Here's the Raspberry Pi wired with I2C using the STEMMA connector:




**Pi 3V** to **DAC VCC (red wire)**
**Pi GND** to **DAC GND (black wire)**
**Pi SCL** to **DAC SCL (yellow wire)**
**Pi SDA** to **DAC SDA (blue wire)**

You can output a signal from any of the 8 DAC channels.

Here's the Raspberry Pi wired with I2C using a solderless breadboard:




**Pi 3V** to **DAC VCC (red wire)**
**Pi GND** to **DAC GND (black wire)**
**Pi SCL** to **DAC SCL (yellow wire)**
**Pi SDA** to **DAC SDA (blue wire)**

You can output a signal from any of the 8 DAC channels.

## Python Installation of DACx578 Library

You'll need to install the **Adafruit_Blinka** library that provides the CircuitPython support in Python. This may also require enabling I2C on your platform and verifying you are running Python 3. Since each platform is a little different, and Linux changes often, please visit the CircuitPython on Linux guide to get your computer ready (https://adafru.it/BSN)!

Once that's done, from your command line run the following command:

- `pip3 install adafruit-circuitpython-dacx578`

If your default Python is version 3 you may need to run 'pip' instead. Just make sure you aren't trying to use CircuitPython on Python 2.x, it isn't supported!

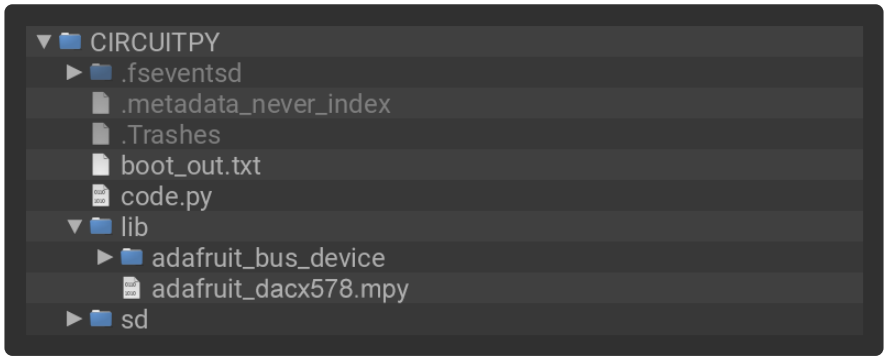
## CircuitPython Usage

To use with CircuitPython, you need to first install the **Adafruit_CircuitPython_DACx578** library, and its dependencies, into the **lib** folder on your **CIRCUITPY** drive. Then you need to update **code.py** with the example script.

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

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

- **adafruit_bus_device/**
- **adafruit_dacx578.mpy**



## Python Usage

Once you have the library `pip3` installed on your computer, copy or download the following example to your computer, and run the following, replacing **code.py** with whatever you named the file:

```
python3 code.py
```

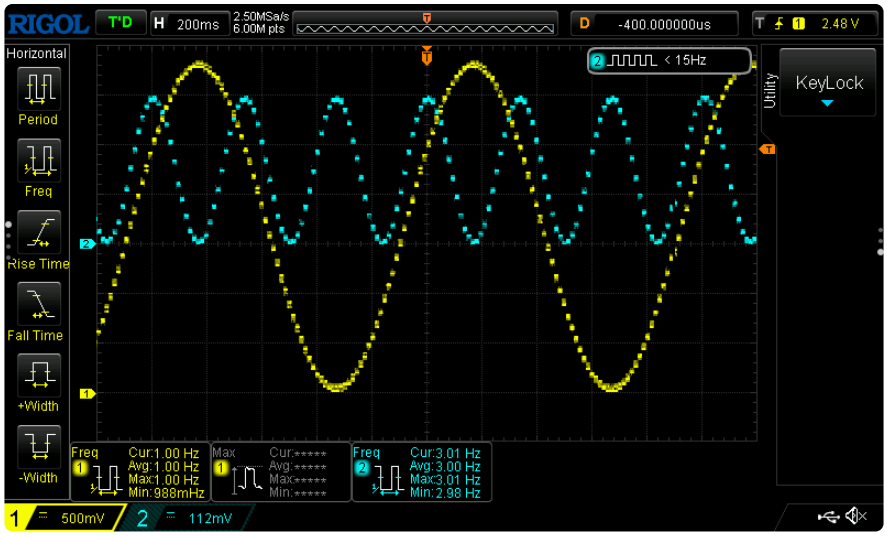
## Example Code

**If running CircuitPython:** Once everything is saved to the **CIRCUITPY** drive, connect to the serial console (https://adafru.it/Bec) to see the data printed out!

**If running Python:** The console output will appear wherever you are running Python.

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

import board

import adafruit_dacx578

# Initialize I2C and DAC
i2c = board.I2C()
dac = adafruit_dacx578.DACx578(i2c)

MAX_VALUE = 65535  # 16-bit value
BASE_FREQ = 1.0  # frequency in Hz
SAMPLE_RATE = 100  # samples per second

FREQ_MULTIPLIERS = [
    1.0,  # Channel 0: 1 Hz
    2.0,  # Channel 1: 2 Hz
    3.0,  # Channel 2: 3 Hz
    4.0,  # Channel 3: 4 Hz
    5.0,  # Channel 4: 5 Hz
    6.0,  # Channel 5: 6 Hz
    7.0,  # Channel 6: 7 Hz
    8.0,  # Channel 7: 8 Hz
]


def calculate_sinewave(frequency, time_point):
    angle = 2 * math.pi * frequency * time_point
    return int((math.sin(angle) + 1) * (MAX_VALUE / 2))


start_time = time.monotonic()

while True:
    current_time = time.monotonic() - start_time

    for channel_num in range(8):
        frequency = BASE_FREQ * FREQ_MULTIPLIERS[channel_num]
        value = calculate_sinewave(frequency, current_time)
        dac.channels[channel_num].value = value

    time.sleep(1 / SAMPLE_RATE)
```

First, the DAC is instantiated over I2C. Then, in the loop, the sinewaves are written to each of the 8 DAC outputs. Each sinewave is generated at a different frequency. You can connect the outputs to your oscilloscope to measure them. In the image below, channels 0 and 2 on the DAC are connected to the two inputs on an oscilloscope. Channel 0 is operating at 1 Hz and channel 2 is operating at 3 Hz.




# Python Docs

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

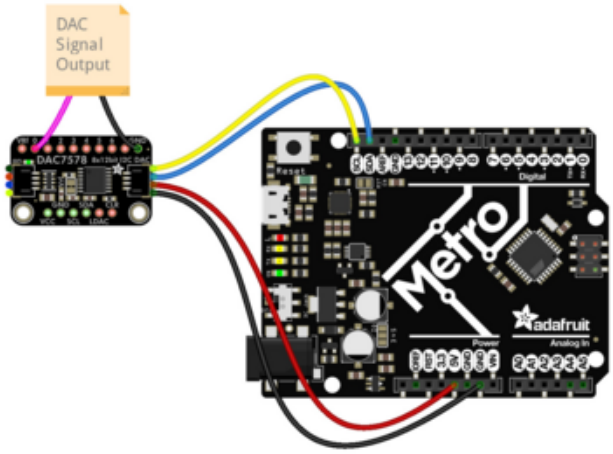
# Arduino

Using the DACx578 breakout with Arduino involves wiring up the breakout to your Arduino-compatible microcontroller, installing the Adafruit_DACX578 (https://adafru.it/1aeR) library, and running the provided example code.

## Wiring

Wire as shown for a **5V** board like an Uno. If you are using a **3V** board, like an Adafruit Feather, wire the board's 3V pin to the DAC VIN.

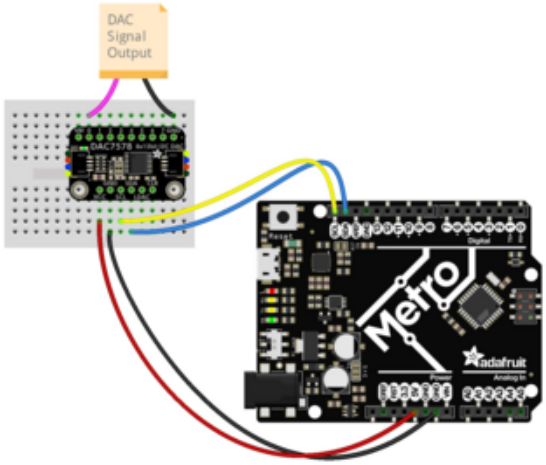
Here is an Adafruit Metro wired up to the DAC using the STEMMA QT connector:




**Board 5V** to **DAC VCC (red wire)**
**Board GND** to **DAC GND (black wire)**
**Board SCL** to **DAC SCL (yellow wire)**
**Board SDA** to **DAC SDA (blue wire)**

You can output a signal from any of the 8 DAC channels.

Here is an Adafruit Metro wired up using a solderless breadboard:

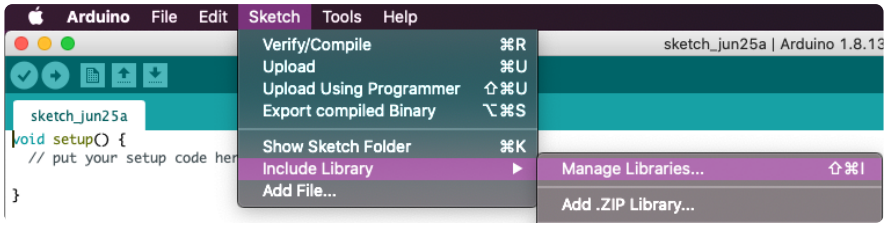



**Board 5V** to **DAC VCC (red wire)**
**Board GND** to **DAC GND (black wire)**
**Board SCL** to **DAC SCL (yellow wire)**
**Board SDA** to **DAC SDA (blue wire)**

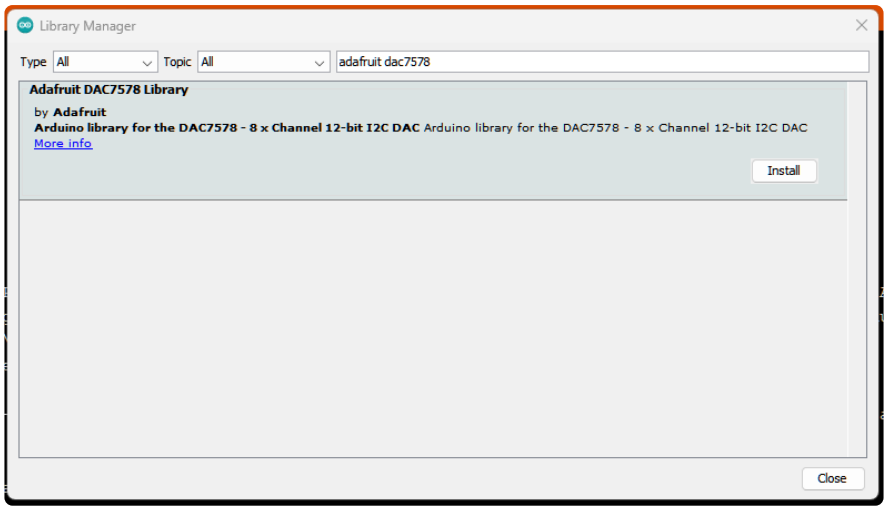
You can output a signal from any of the 8 DAC channels.

## Library Installation

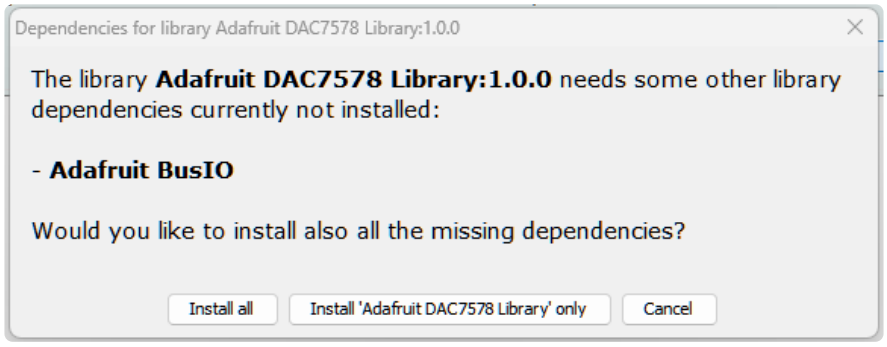
You can install the **Adafruit_DACX578** library for Arduino using the Library Manager in the Arduino IDE.




Click the **Manage Libraries ...** menu item, search for **Adafruit_DAC7578**, and select the **Adafruit DAC7578** library:




If asked about dependencies, click "Install all".



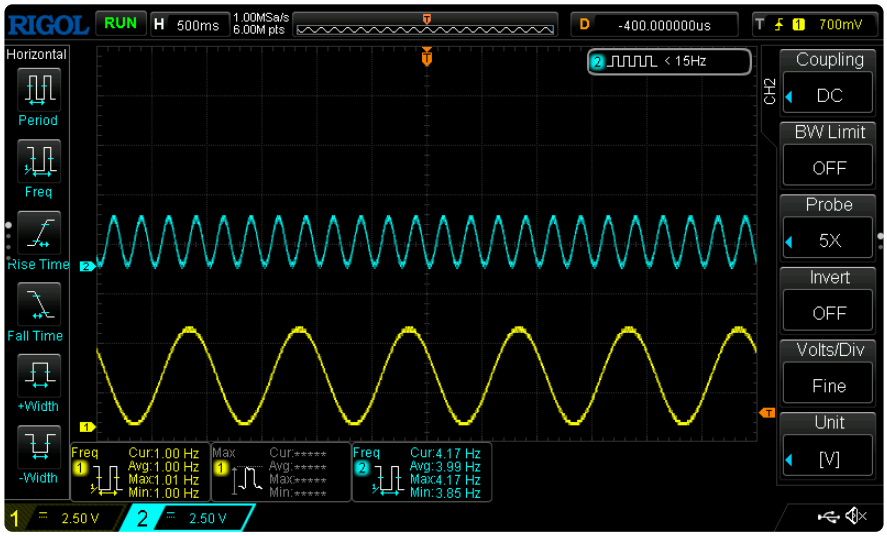

If the "Dependencies" window does not come up, then you already have the dependencies installed.

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

## Example Code

```
#include <Adafruit_DACX578.h>
#include <math.h>

Adafruit_DACX578 dac(12); // Assuming 12-bit resolution

const float frequencies[8] = {1, 2, 4, 8, 16, 32, 40, 50}; // Hz
const uint16_t amplitude =
    2048;                     // Half of full scale for 12-bit DAC (0 to 4095)
const uint16_t offset = 2048; // DC offset to keep sine wave positive
const uint32_t sampleRate = 370; // Measured actual rate
const uint32_t period = 2700;    // Measured cycle time in microseconds

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

```
  Serial.println("Adafruit DACX578 Sine Wave Test");

  if (!dac.begin()) {
    Serial.println("Failed to find DAC7578 chip");
    while (1)
      delay(10);
  }

  Serial.println("DAC7578 initialized");

  // Set I2C frequency to 800 kHz for faster communication
  Wire.setClock(800000);
}

void loop() {
  static uint32_t lastTime = micros();
  static float phase[8] = {0};
  uint32_t currentTime = micros();

  if (currentTime - lastTime >= period) {
    lastTime = currentTime; // Use actual time

    for (uint8_t channel = 0; channel < 8; channel++) {
      float sineValue = sin(phase[channel]) * amplitude + offset;
      dac.writeAndUpdateChannelValue(channel, (uint16_t)sineValue);

      // Update phase using actual sample rate
      phase[channel] += 2 * M_PI * frequencies[channel] / sampleRate;
      if (phase[channel] >= 2 * M_PI) {
        phase[channel] -= 2 * M_PI;
      }
    }
  }
}
```




Upload the sketch to your board and open up the Serial Monitor (**Tools -> Serial Monitor**) at 115200 baud. You'll see the DACx578 recognized over I2C. Then, sinewaves will be output to each of the 8 channels on the DAC. You can connect these outputs to your oscilloscope to measure the frequency. In the image above, channels 0 (1 Hz) and 2 (4 Hz) on the DAC are connected to the two input channels on the oscilloscope.

# Arduino Docs

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

# Downloads

## Files

- DACx578 Datasheet (https://adafru.it/1aeS)
- EagleCAD PCB Files on GitHub (https://adafru.it/1aeT)
- Fritzing object in the Adafruit Fritzing Library (https://adafru.it/1aeU)

## Schematic and Fab Print