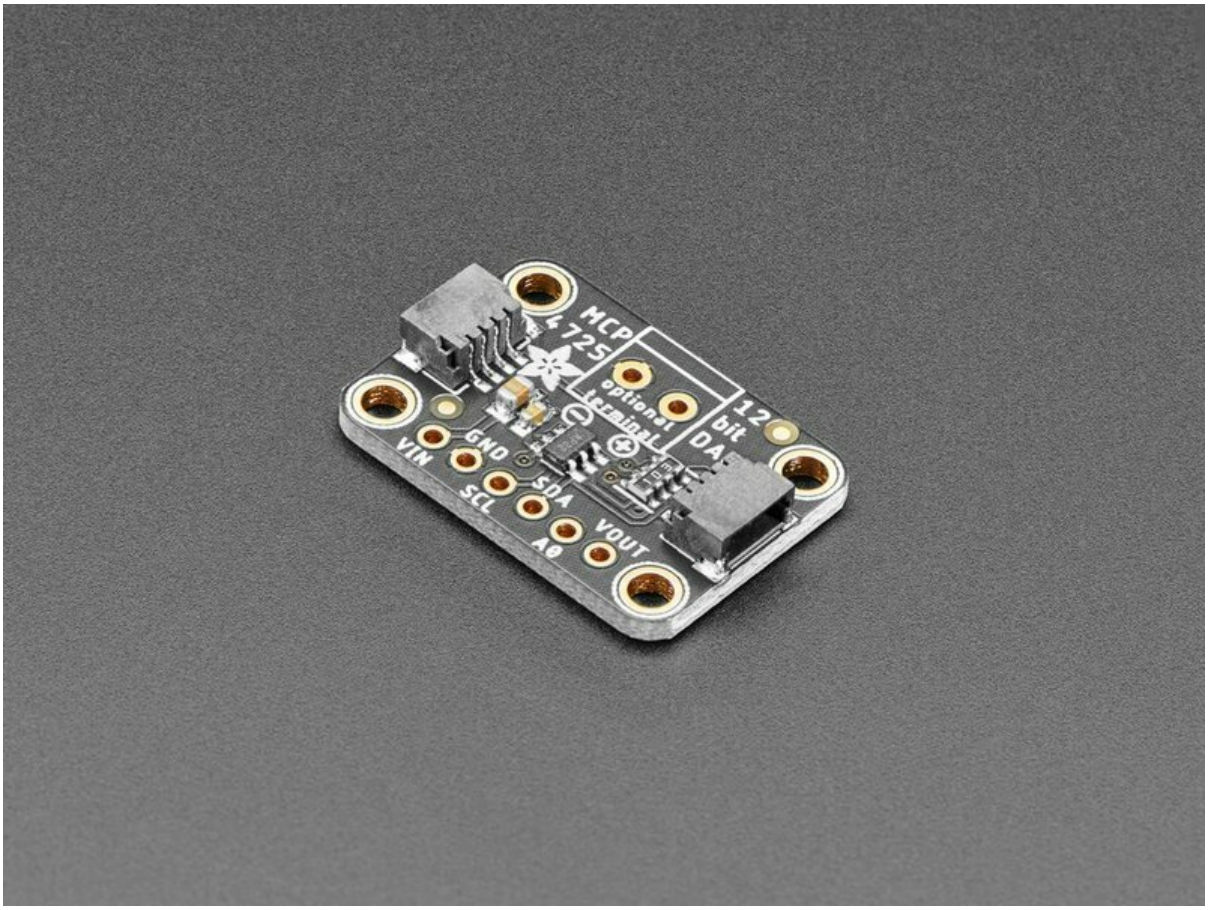
MCP4725 12-Bit DAC Tutorial

Created by lady ada



https://learn.adafruit.com/mcp4725-12-bit-dac-tutorial

Last updated on 2024-06-03 01:11:44 PM EDT

Table of Contents

Overview 3

Pinouts 6

- Power Pins
- I2C Logic Pins
- Other Pins

Arduino Wiring 7

Arduino Code 11

- Library Installation
- Triangle Wave Example
- Increasing the speed
- Using the library

Python & CircuitPython 13

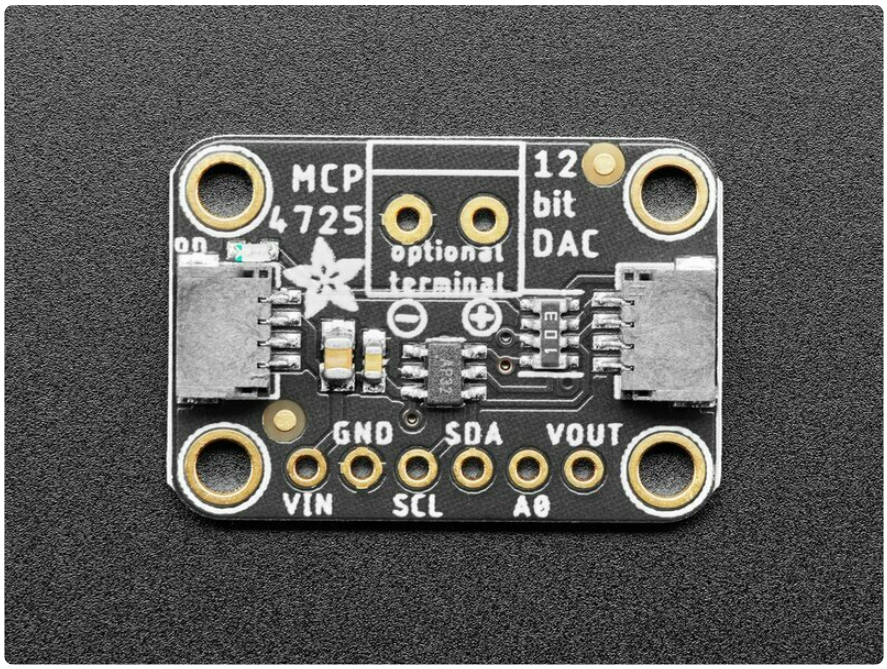
- CircuitPython Microcontroller Wiring
- Python Computer Wiring
- CircuitPython Installation of MCP4725 Library
- Python Installation of MCP4725 Library
- CircuitPython & Python Usage
- Full Example Code

Python Docs 19

Downloads 19

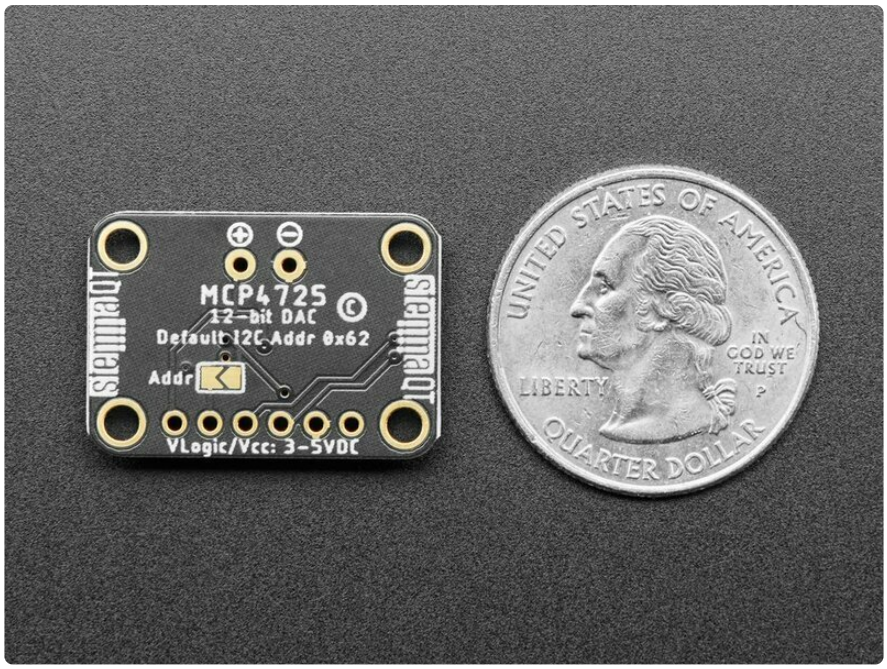
- Files
- Schematic & Fabrication Print STEMMA QT Version
- Schematic & Fabrication Print Original Version

Overview



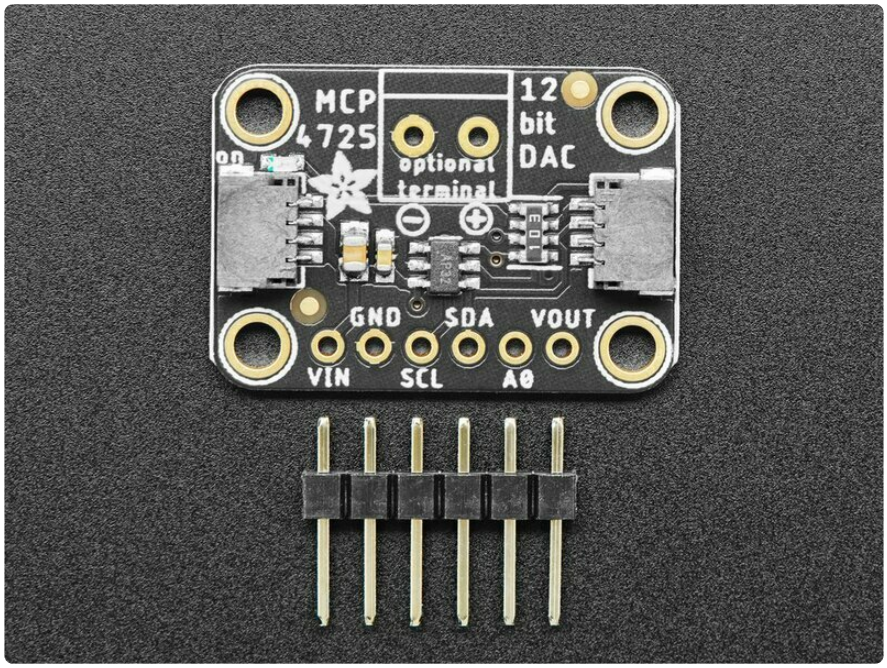
Your microcontroller probably has an ADC (analog -> digital converter) but does it have a DAC (digital -> analog converter)??? Now it can! This breakout board features the easy-to-use MCP4725 12-bit DAC. Control it via I2C and send it the value you want it to output, and the VOUT pin will have it. Great for audio / analog projects, such as when you can't use PWM but need a sine wave or adjustable bias point.

We break out the ADDR/A0 pin so you can connect two of these DACs on one I2C bus, just tie that pin of one high (or close the jumper on the back) to keep it from conflicting. Also included is a 6-pin header, for use in a breadboard. Works with both 3.3V or 5V logic.

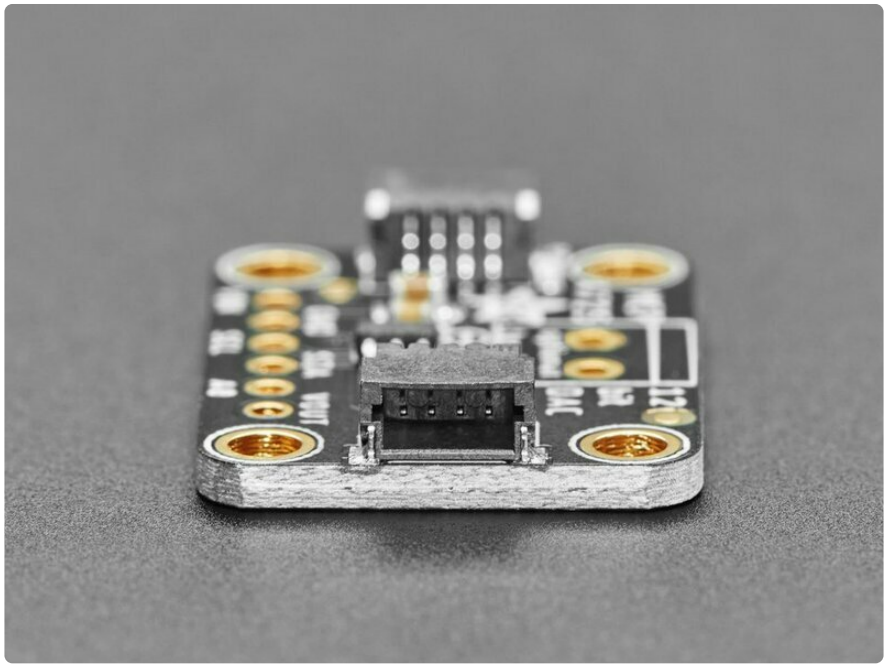


Some nice extras with this chip: for chips that have 3.4Mbps Fast Mode I2C (Arduino's don't) you can update the Vout at ~200 KHz. There's an EEPROM so if you write the output voltage, you can 'store it' so if the device is power cycled it will restore that voltage. The output voltage is rail-to-rail and proportional to the power pin so if you run it from 3.3V, the output range is 0-3.3V. If you run it from 5V the output range is 0-5V.

We have an easy-to-use Arduino and CircuitPython/Python library and tutorial with a triangle-wave and sine-wave output example (https://adafru.it/YIf) that can be used with just about any microcontroller or microcomputer with I2C host.



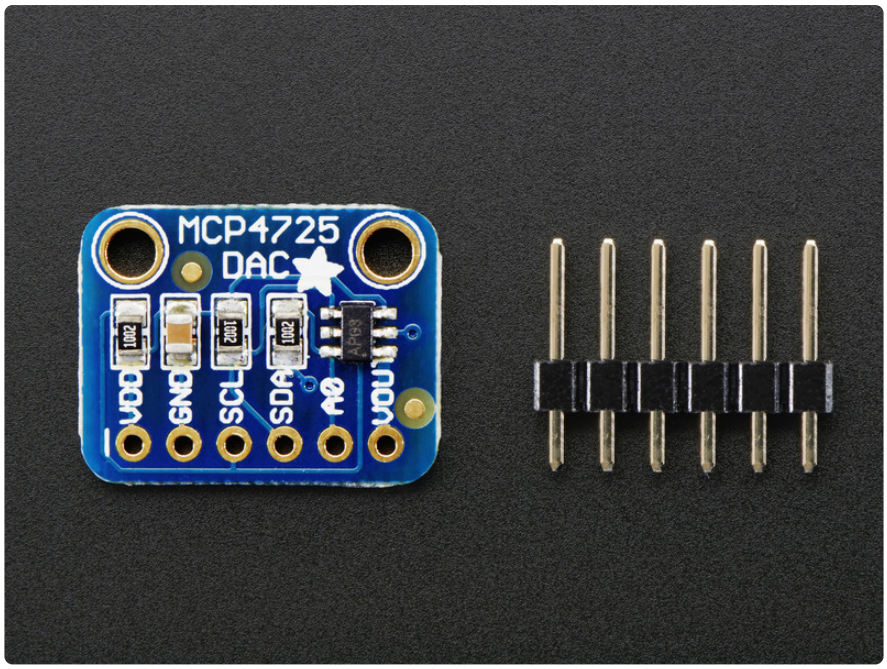
Comes with a bit of 0.1" standard header in case you want to use it with a breadboard or perfboard. Four mounting holes for easy attachment. There's an optional 3.5mm terminal block spot on the PCB - we don't include a 3.5mm terminal block but they're both common and stocked in the shop (http://adafru.it/724) - that you can solder in place if you like.



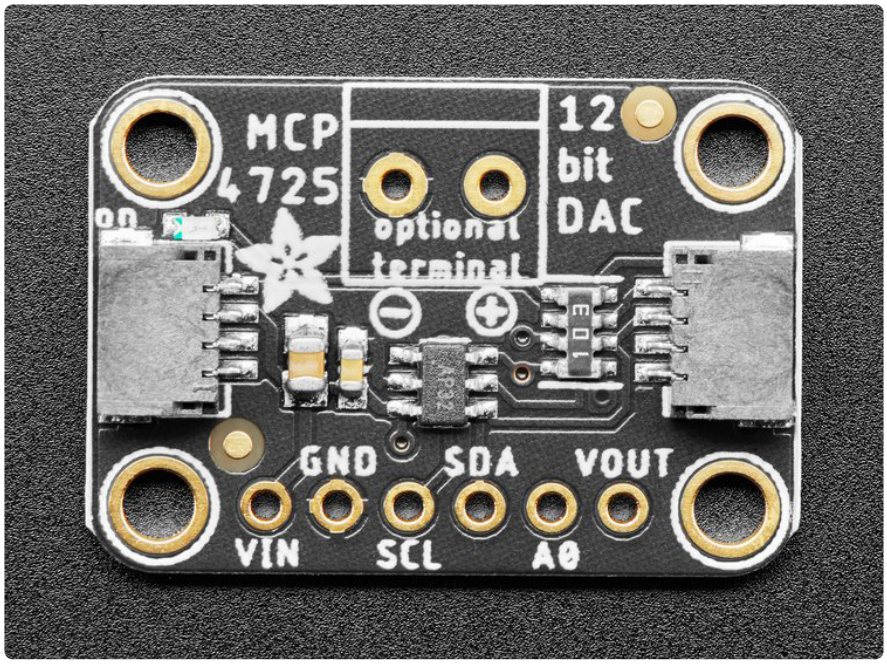
To get you going fast, we spun up a custom-made PCB in the **STEMMA QT** form factor (https://adafru.it/LBQ), making it easy to interface with. The STEMMA QT connectors (https://adafru.it/JqB) on either side are compatible with the SparkFun Qwiic (https://adafru.it/Fpw) I2C connectors. This allows you to make solderless connections between your development board and the MCP4725 or to chain it with a wide range of other sensors and accessories using a **compatible cable** (https://adafru.it/JnB).

QT Cable is not included, but we have a variety in the shop (https://adafru.it/17VE).

There are two versions of this board - the STEMMA QT version shown above, and the original header-only version shown below. Code works the same on both!



Pinouts



The default I2C address is **0x62**.

Power Pins

The sensor on the breakout requires between a 2.7V and 5.5V, and can be easily used with most microcontrollers from an Arduino to a Feather or something else.

- **VIN** - 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
- **GND** - common ground for power and logic

I2C Logic Pins

- **SCL** - I2C clock pin, connect to your microcontroller's I2C clock line. This pin is level shifted so you 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 is level shifted so you 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 development boards with **STEMMA QT** connectors, or to other things, with various associated accessories (https://adafru.it/Ft6).

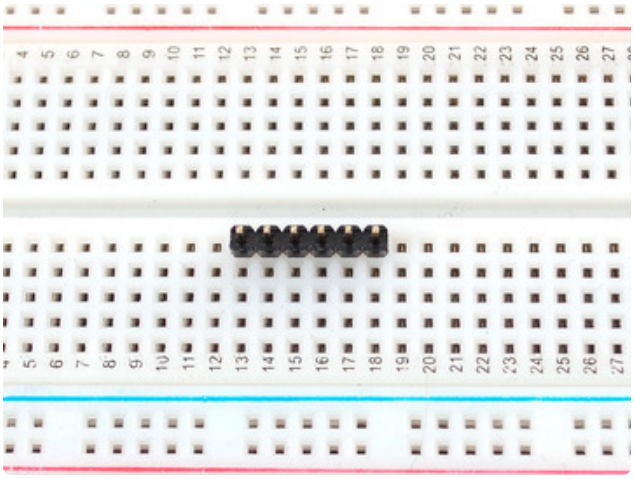
Other Pins

- **A0 pin / ADDR jumper** - Tie A0 high, or close the jumper on the back to change the I2C address to **0x63**, so you can connect two MCP4725 breakouts on the same I2C bus.
- **VOUT** - This is the output pin for the I2C signal.
- **Optional terminal block** - There is an optional 3.5mm terminal block spot on the board that breaks out **VOUT** and **GND**. You can solder a terminal block on if you like. The terminal block does not come with the board, but they are common and you can purchase one from the Adafruit shop (http://adafru.it/724).

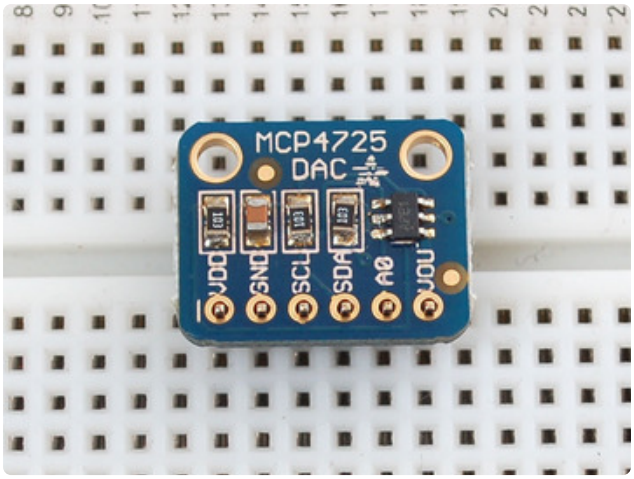
Arduino Wiring

Wiring up the MCP4725 breakout PCB is super easy. To start, we'll attach the breakout headers so we can plug it into a breadboard.

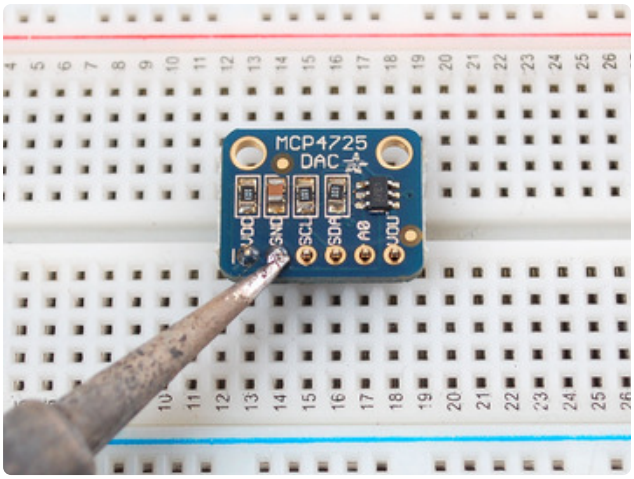
Break off a strip of 6-pins of 0.1" male header and stick the LONG pins down into a breadboard



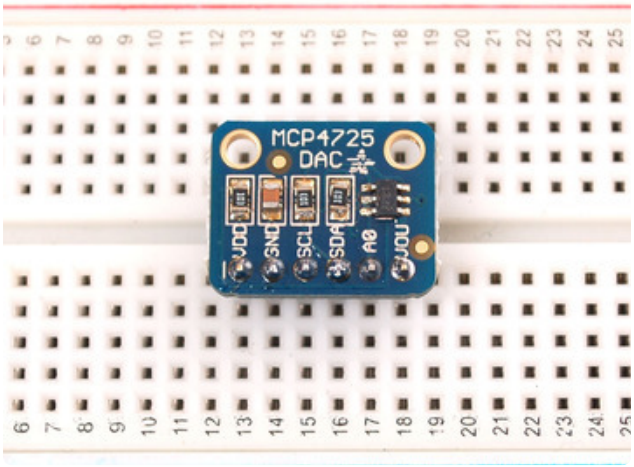
Break off a strip of 6-pins of 0.1" male header and stick the LONG pins down into a breadboard



Place the breakout board on top so the short ends of the header stick up through the pads

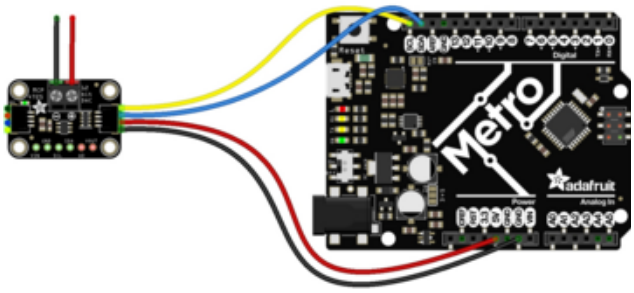


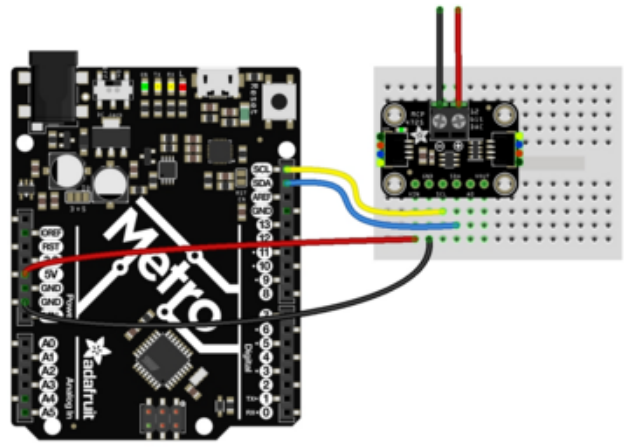
Solder each pin using a soldering iron and solder, to make solid connection on each pin.

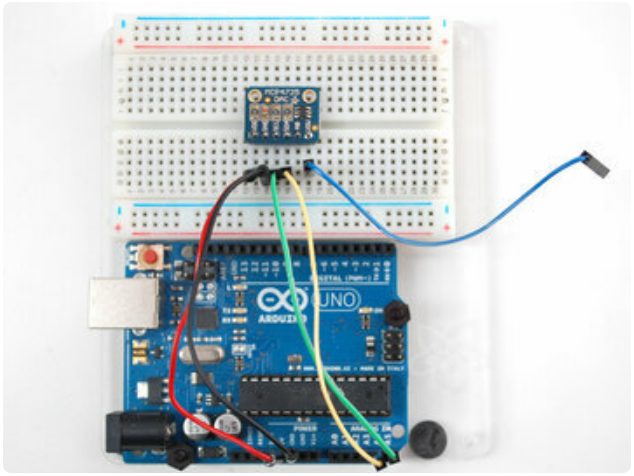


This part is not optional! You cannot 'press fit' the header on, it must be attached permanently

Now that the header is attached, we can wire it up. We'll demonstrate using an Arduino







Breakout VDD (power) to **5V** if your microcontroller is 5V logic, or to 3V if your micro is 3V logic **(red wire on STEMMA QT version)**
Breakout GND to **microcontroller GND (black wire on STEMMA QT version)**
Breakout SDA to **microcontroller I2C Data (blue wire on STEMMA QT version)** (on the Uno, this is **A4** on the Mega it is **20** and on the Leonardo digital **2**)
Breakout SCL to **microcontroller I2C Clock (yellow wire on STEMMA QT version)** (on the Uno, this is **A5** on the Mega it is **21** and on the Leonardo digital **3**)

There's two other pins remaining.

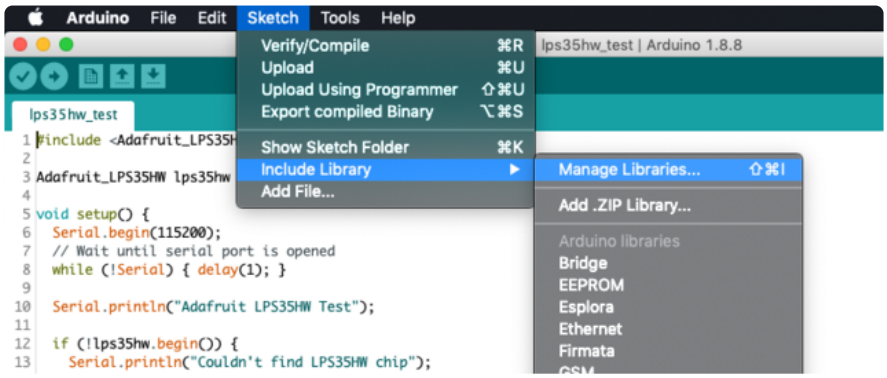
- **A0** allow you to change the I2C address. By default (nothing attached to A0) the address is hex **0x62**. If A0 is connected to **VDD** the address is **0x63**. This lets you have two DAC boards connected to the same SDA/SCL I2C bus pins.
- **VOUT** is the voltage out from the DAC! The voltage will range from 0V (when the DAC value is 0) to VDD (when the DAC 'value' is the max 12-bit number: 0xFFF)

Arduino Code

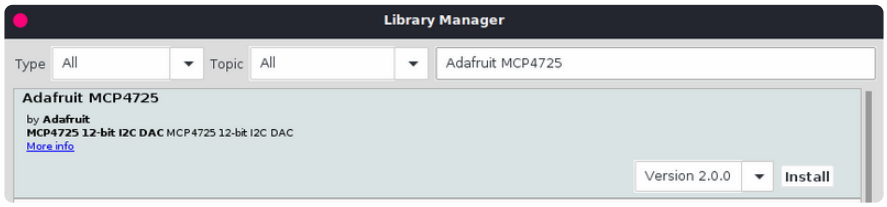
Library Installation

Once wired up, to start using the MCP4725, you'll need to install the Adafruit_MCP4725 library (https://adafru.it/aPz). The library is available through the Arduino library manager so we recommend taking that approach.

From the Arduino IDE, open up the Library Manager:



Click the **Manage Libraries ...** menu item, search for **Adafruit MCP4725**, and select the **Adafruit MCP4725** library and click **Install**:



Next up, download the Adafruit MCP4725 library. This library does all of the interfacing, so you can just "set and forget" the DAC output. It also has some examples to get you started

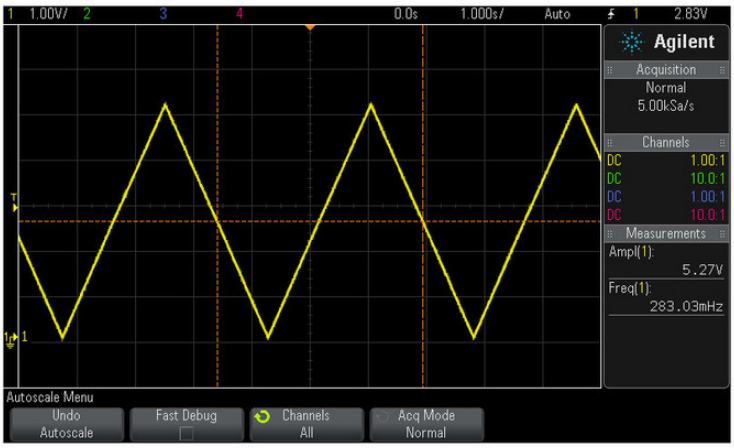
The library is available on GitHub (https://adafru.it/aPz). You can download it by clicking the button below.

Download Adafruit_MCP4725 Library

https://adafru.it/cDA

Triangle Wave Example

Open up the **File→Examples→Adafruit_MCP4725→trianglewave** sketch and upload it to the Arduino. Then connect your oscilloscope (or an LED + resistor if you don't have access to an oscilloscope)



We also have a sine wave version showing how to use a lookup table to create a more complex waveform.

Using the library

The library is very simple, so you can adapt it very quickly.

First, be sure to call **begin(addr)** where **addr** is the i2c address (default is 0x62, if A0 is connected to VCC its 0x63). Then call **setVoltage(value, storeflag)** to set the DAC output. **value** should range from 0 to 0x0FFF. **storeflag** indicates to the DAC whether it should store the value in EEPROM so that next time it starts, it'll have that same value output. You shouldn't set the flag to true unless you require it as it will take longer to do, and you could wear out the EEPROM if you write it over 20,000 times.

Increasing the speed

One thing thats a little annoying about the Arduino Wire library in this case is it is set for 100KHz transfer speed. In the MCP4725 library we update the speed to 400KHz by setting the TWBR

```
TWBR = 12; // 400 khz
```

You can speed this up a bit more, if you'd like, check the ATmega328 datasheet for how to calculate the **TWBR** register.

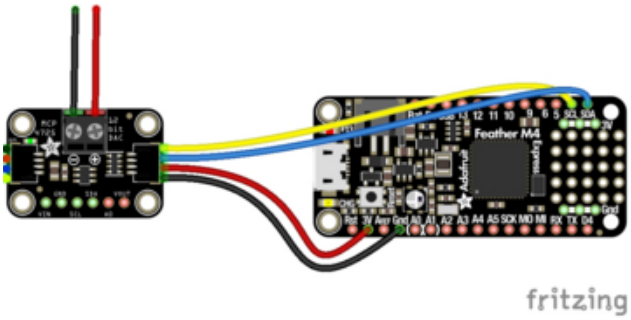
Python & CircuitPython

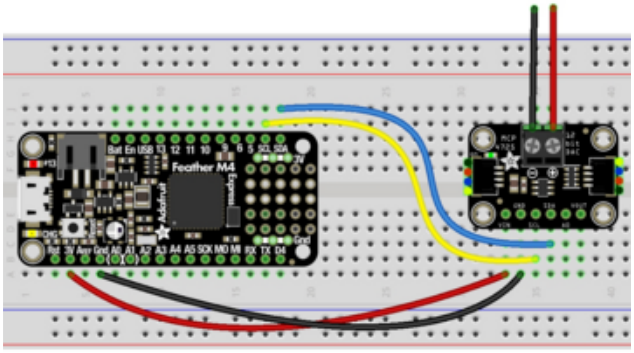
It's easy to use the MCP4725 digital to analog converter with Python and CircuitPython, and the Adafruit CircuitPython MCP4725 (https://adafru.it/C3P) module. This module allows you to easily write Python code that controls the output voltage from the DAC.

You can use this sensor 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).

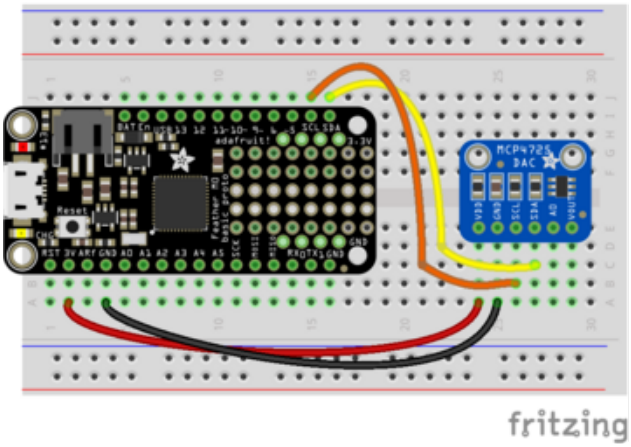
CircuitPython Microcontroller Wiring

First wire up a MCP4725 to your board exactly as shown on the previous pages for Arduino using an I2C connection. Here's an example of wiring a Feather M0 to the sensor with I2C:





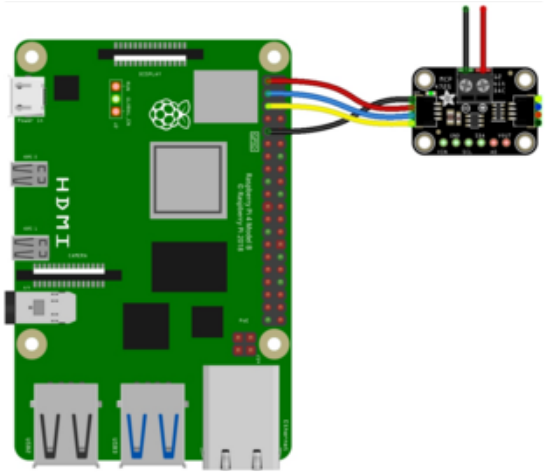
Board 3V to **sensor VIN (red wire on STEMMA QT version)**
Board GND to **sensor GND (black wire on STEMMA QT version)**
Board SCL to **sensor SCL (yellow wire on STEMMA QT version**
Board SDA to **sensor SDA (blue wire on STEMMA QT version**

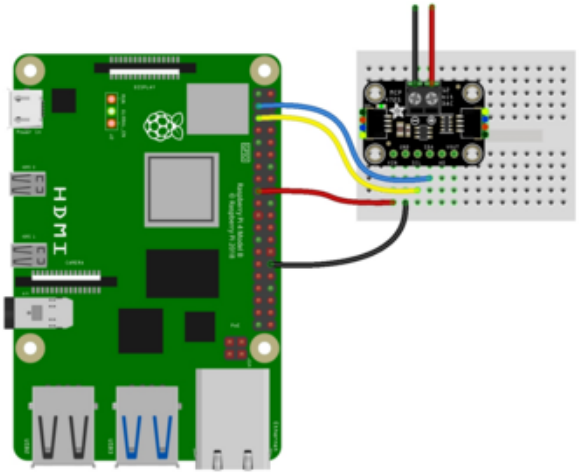


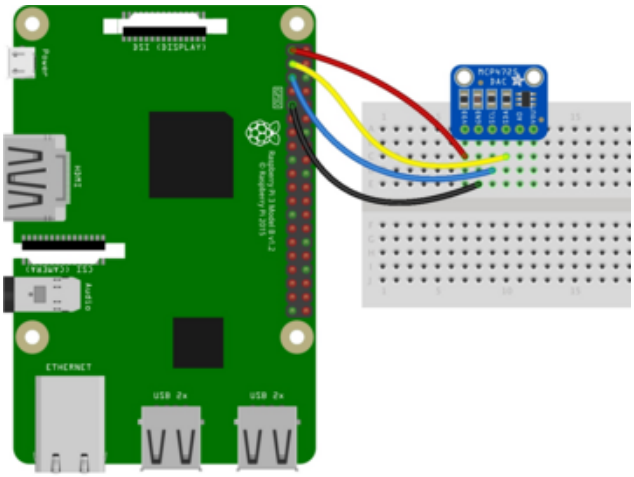
Python Computer Wiring

Since there's 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).

Here's the Raspberry Pi wired with I2C:







Pi 3V3 to **sensor VIN (red wire on STEMMA QT version)**
Pi GND to **sensor GND (black wire on STEMMA QT version)**
Pi SCL to **sensor SCL (yellow wire on STEMMA QT version)**
Pi SDA to **sensor SDA (blue wire on STEMMA QT version)**

CircuitPython Installation of MCP4725 Library

Next you'll need to install the Adafruit CircuitPython MCP4725 (https://adafru.it/C3P) library on your CircuitPython board. Make sure you are running the latest version of Adafruit CircuitPython (https://adafru.it/tBa) for your board before starting..

You'll need to install the necessary libraries to use the hardware--carefully follow the steps to find and install these libraries from Adafruit's CircuitPython library bundle (https://adafru.it/zdx). For example the Circuit Playground Express guide has a great page on how to install the library bundle (https://adafru.it/Bf2) for both express and non-express boards.

Remember for non-express boards like the Trinket M0, Gemma M0, and Feather/Metro M0 basic you'll need to manually install the necessary libraries from the bundle:

- **adafruit_mcp4725.mpy**

You can also download the **adafruit_mcp4725.mpy** from its releases page on Github (https://adafru.it/C3Q).

Before continuing make sure your board's lib folder or root filesystem has the **adafruit_mcp4725.mpy** file copied over.

Next connect to the board's serial REPL (https://adafru.it/pMf)so you are at the CircuitPython >>> prompt.

Python Installation of MCP4725 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:

- `sudo pip3 install adafruit-circuitpython-mcp4725`

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 & Python Usage

To demonstrate the usage of the DAC we'll initialize it and set the output voltage from the board's Python REPL. Run the following code to import the necessary modules and initialize the I2C connection with the sensor:

```
import board
import busio
import adafruit_mcp4725
i2c = busio.I2C(board.SCL, board.SDA)
dac = adafruit_mcp4725.MCP4725(i2c)
```

Now you can set the output voltage just like controlling a DAC with CircuitPython's built-in AnalogOut class and the **value** property. Simply set this to any 16-bit value (0-65535) and the output of the Vout pin will change to a voltage proportional to 0-3.3V. For example to set the output to 1.65V or about halfway within its range:

```
dac.value = 32767
```

Hook up a multimeter to the **Vout** pin of the board (positive/red lead to **Vout**, ground/black lead to board GND) and you should see about 1.65 volts DC output. Try setting **dac.value** to other numbers like 0 or 65535 to see how the voltage changes.

Hook up a multimeter to the **Vout** pin of the board (positive/red lead to **Vout**, ground/black lead to board GND) and you should see about 1.65 volts DC output. Try setting **dac.value** to other numbers like 0 or 65535 to see how the voltage changes.

You can use the MCP4725 instance anywhere you might use the AnalogOut class!

However you might prefer a few other simpler properties to change the output voltage:

- **normalized_value** - Set this to a floating point number between 0 and 1.0. A value of 0 is ground/0V and 1.0 is Vdd or max voltage/3.3V. Anything in-

between is a proportional voltage. This is handy for scaling the output value without having to worry about how many bits of resolution it has.
- **raw_value** - Set this to a 12-bit value 0-4095 to control the raw 12-bit output of the DAC. Unlike the **value** property this **raw_value** exposes the true 12-bit resolution of the DAC and is free from quantization errors. If you need the most precise output use the raw_output value for setting voltage.

```
dac.normalized_value = 0.5  # ~1.65V output
dac.raw_output = 2047 # Also ~1.65V output
```

That's all there is to using the MCP4725 DAC with CircuitPython!

Below is a complete example that shows changing the DAC voltage to a triangle wave that goes up and down repeatedly. Save this as **code.py** on your board and connect a multimeter to measure the Vout pin voltage to see it oscillate up and down from 0 to 3.3V and back.

Full Example Code

```
# SPDX-FileCopyrightText: 2018 Tony DiCola for Adafruit Industries
# SPDX-License-Identifier: MIT

# Simple demo of setting the DAC value up and down through its entire range
# of values.
import board
import busio

import adafruit_mcp4725


# Initialize I2C bus.
i2c = busio.I2C(board.SCL, board.SDA)

# Initialize MCP4725.
dac = adafruit_mcp4725.MCP4725(i2c)
# Optionally you can specify a different addres if you override the A0 pin.
# amp = adafruit_max9744.MAX9744(i2c, address=0x63)

# There are a three ways to set the DAC output, you can use any of these:
dac.value = 65535  # Use the value property with a 16-bit number just like
# the AnalogOut class.  Note the MCP4725 is only a 12-bit
# DAC so quantization errors will occur.  The range of
# values is 0 (minimum/ground) to 65535 (maximum/Vout).

dac.raw_value = 4095  # Use the raw_value property to directly read and write
# the 12-bit DAC value.  The range of values is
# 0 (minimum/ground) to 4095 (maximum/Vout).

dac.normalized_value = 1.0  # Use the normalized_value property to set the
# output with a floating point value in the range
# 0 to 1.0 where 0 is minimum/ground and 1.0 is
# maximum/Vout.

# Main loop will go up and down through the range of DAC values forever.
while True:
```

```
    # Go up the 12-bit raw range.
    print("Going up 0-3.3V...")
    for i in range(4095):
        dac.raw_value = i
    # Go back down the 12-bit raw range.
    print("Going down 3.3-0V...")
    for i in range(4095, -1, -1):
        dac.raw_value = i
```

Python Docs

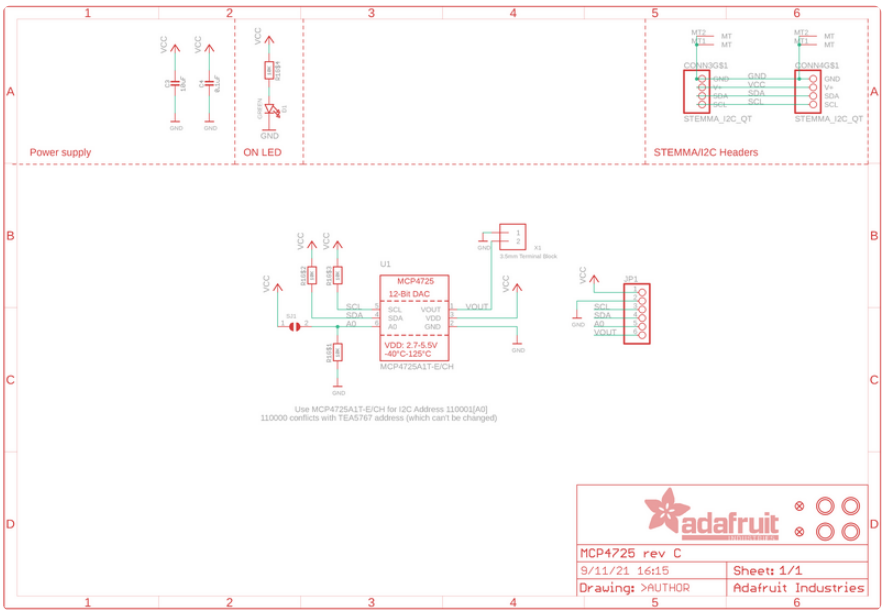
Python Docs (https://adafru.it/C3S)

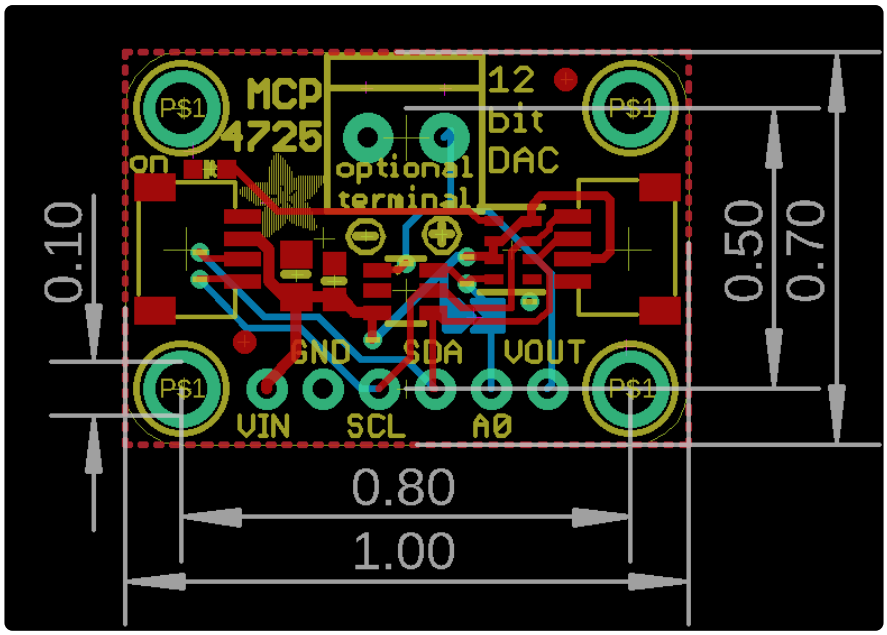
Downloads

Files

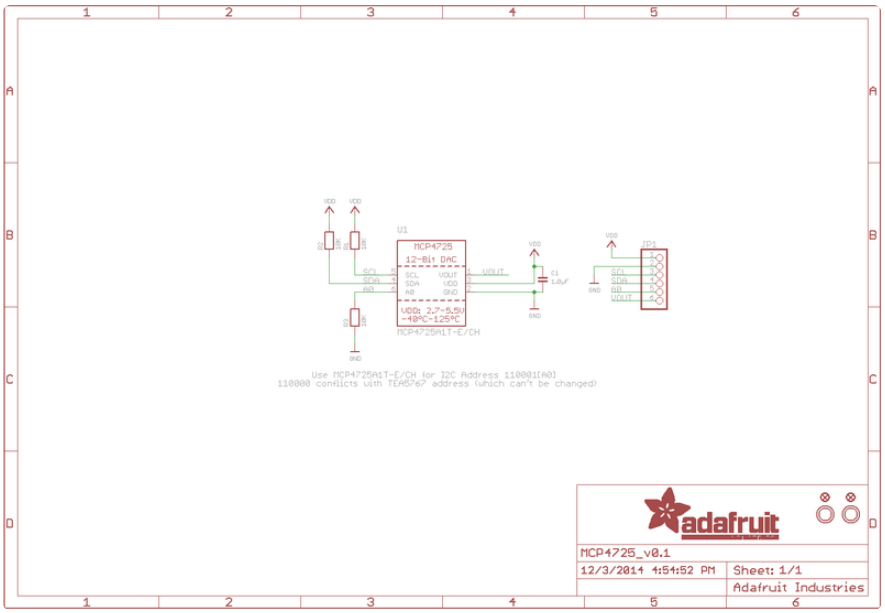
- For more details about the chip, please check out the MCP4725 datasheet (https://adafru.it/aRW)
- MCP4725 Arduino Library is on GitHub (https://adafru.it/aPz)
- Fritzing object in the Adafruit Fritzing library (https://adafru.it/aP3)
- EagleCAD PCB files on GitHub (https://adafru.it/rMC)

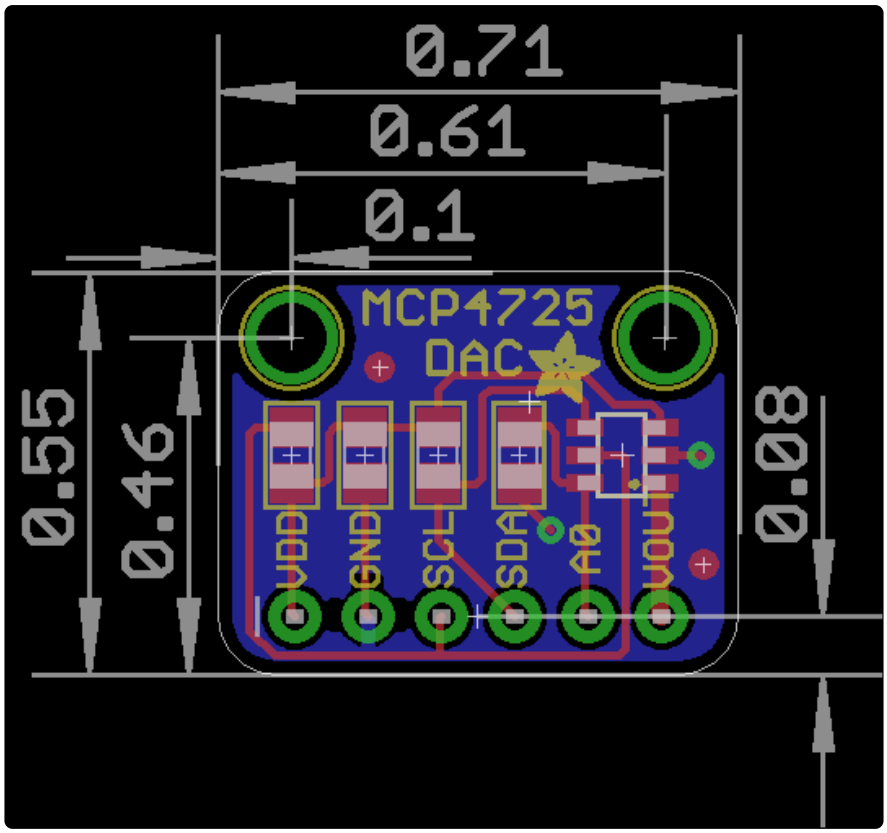
Schematic & Fabrication Print STEMMA QT Version





Schematic & Fabrication Print Original Version




0.71
0.61
0.1
0.55
0.46
0.08
MCP4725
DAC
VDD
GND
SCL
SDA
A0
VOUT