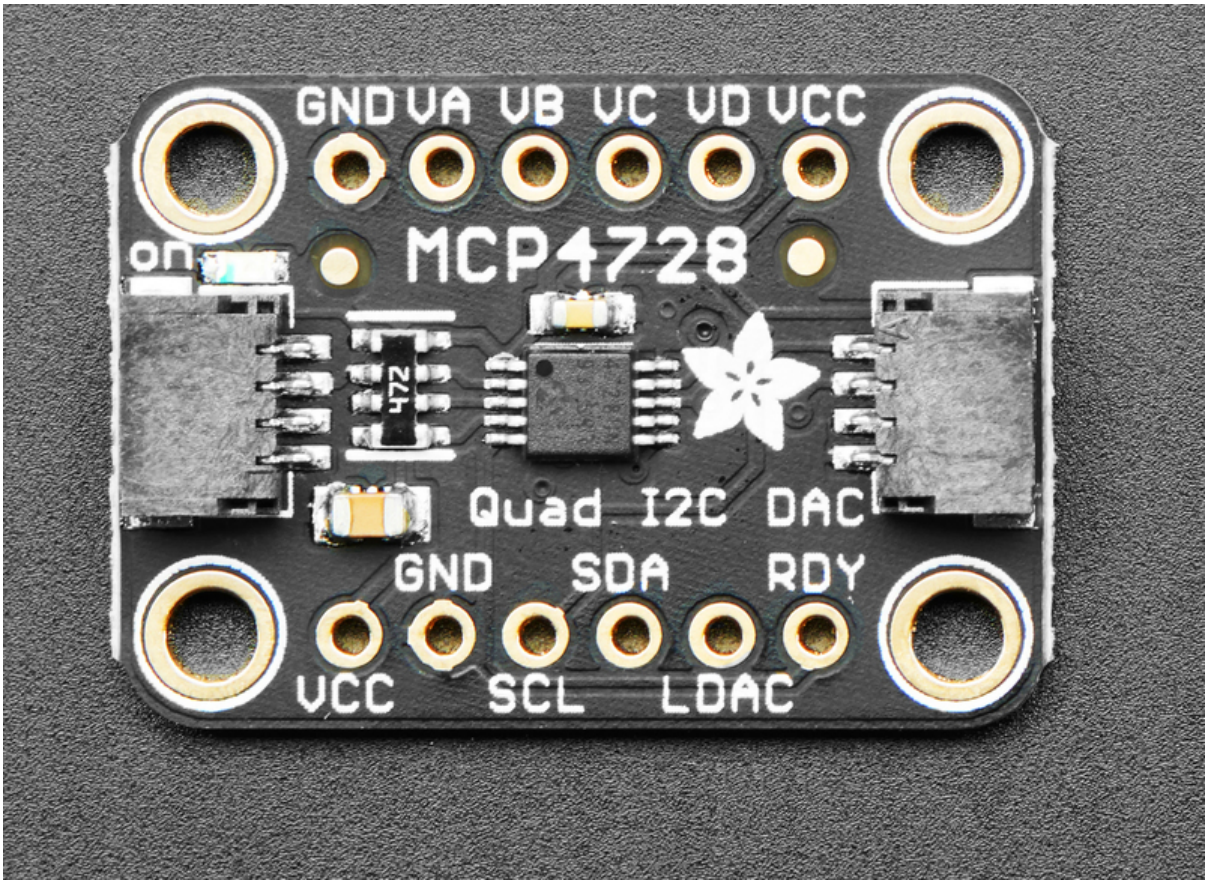
Adafruit MCP4728 I2C Quad DAC

Created by Bryan Siepert

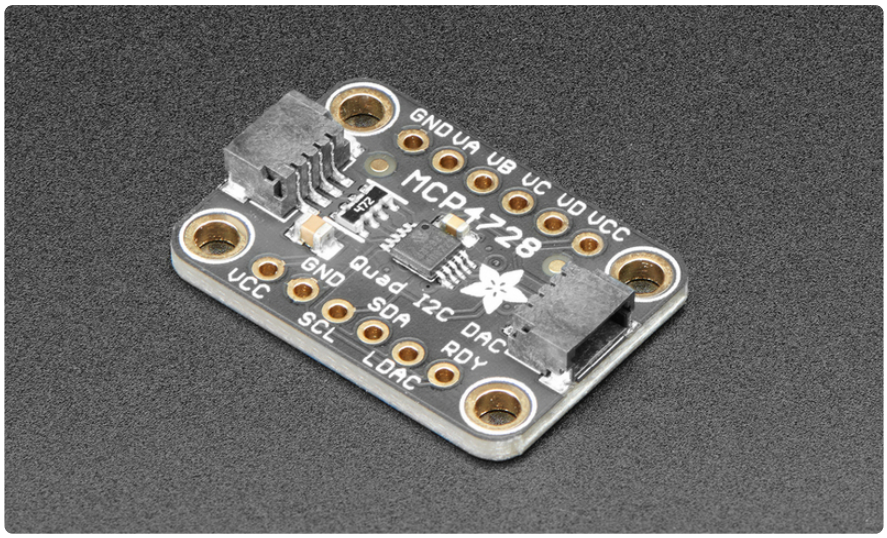


https://learn.adafruit.com/adafruit-mcp4728-i2c-quad-dac

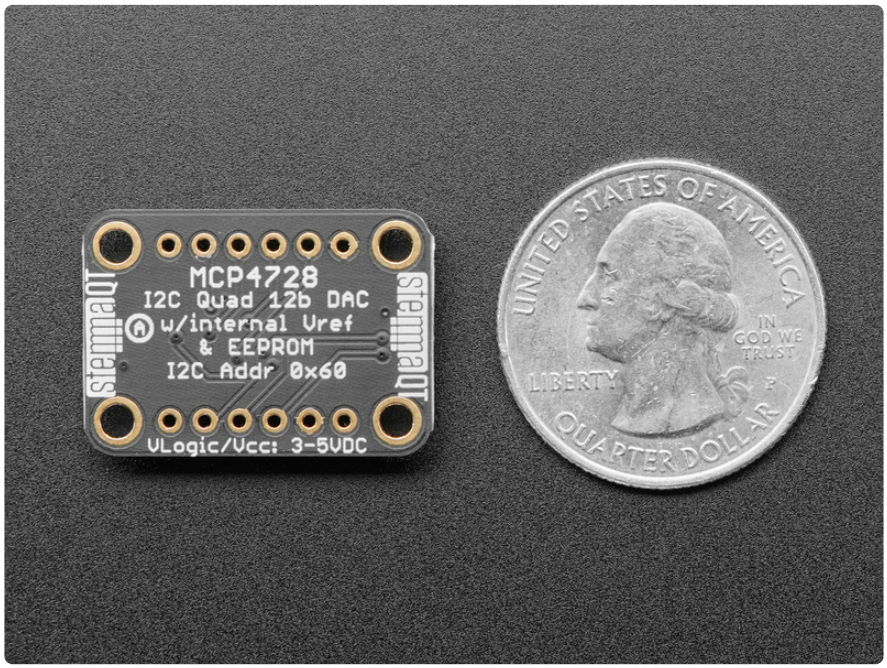
Last updated on 2024-06-03 03:00:02 PM EDT

Table of Contents

Overview



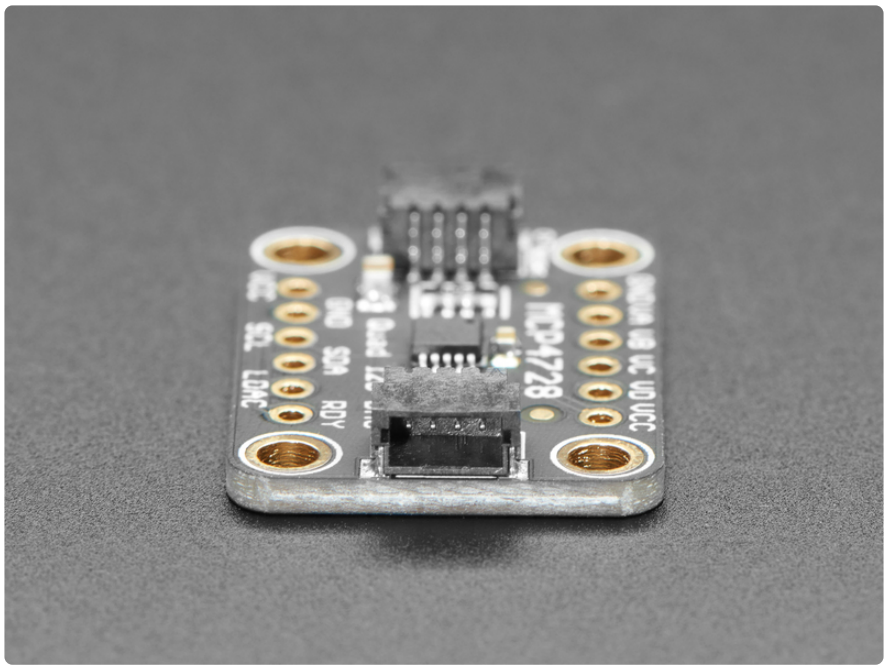
If you've ever said to yourself "Gee, I wish these four 12-bit DACs came in a single package with the ability to save their settings to an EEPROM", well I have good news. The MCP4728 is the answer to your wishes! Within its little package, the MCP4728 has four 12-bit DACs for whatever voltage setting needs you may have. In addition, it has the ability to store the settings for the DACs to an internal EEPROM. Once saved to the internal non-volatile memory, the settings will be loaded by default when the DAC powers up.



To take things even further, the MCP4728 lets you chose between two sources for your reference voltage: the input voltage that you use to power it on the VCC pin, or an **internal 2.048V reference voltage**. **If you use the internal reference** voltage (**Vref** in DAC speak) you can choose between **1X and 2X gain** for the output, allowing your

voltages to range from 0V to 2.048 or 0V- 4.096V as your application requires.

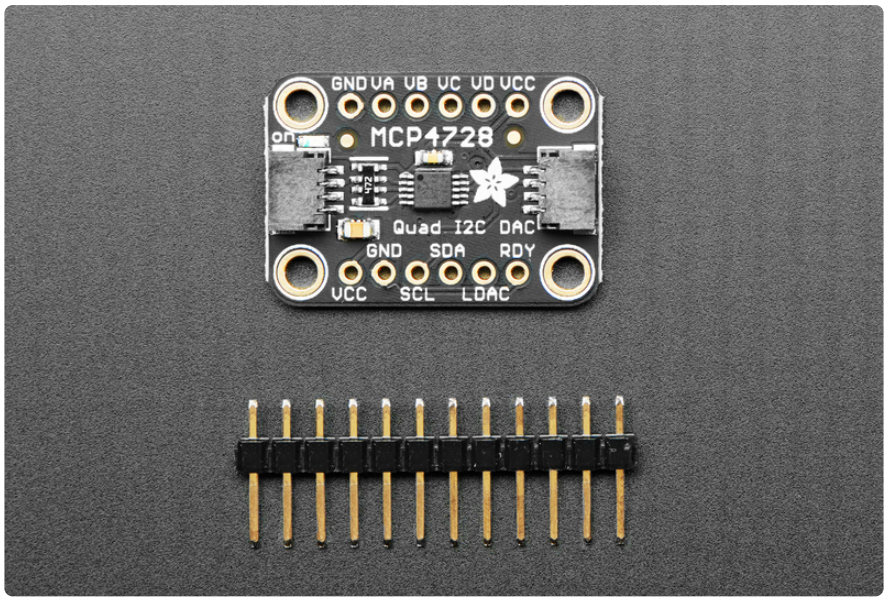
By default you'll use the input voltage as your Vref, allowing you to scale the voltages from 0V-3.3V or 5V depending on your input voltage.



The breakout for the MCP4728 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 MCP4728 to your project and allows you to easily share an I2C bus with other STEMMA QT, Qwiic, Grove or other compatible sensors.

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



As of Aug 1, 2022 - we may make this board with the **MCP4728** (default I2C address **0x60**) or **MCP4728A4** (default address **0x64**). To check your board's I2C address, use either the Arduino (https://adafru.it/10oD) or CircuitPython (https://adafru.it/10oE) I2C scanner programs.

Power Pins

- **Vcc** - this is the power pin. The MCP4728 will handle both 3.3V logic level boards like Feathers or 5V logic level boards like the Arduino Uno
- **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 connectors to dev boards with **STEMMA QT** connectors or to other things with various associated accessories (https://adafru.it/Ft6)

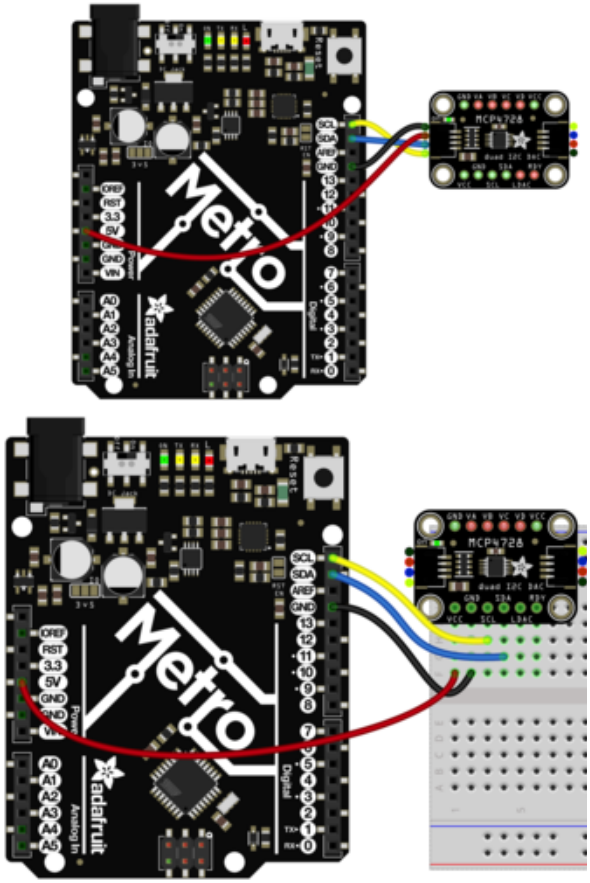
Other Pins

- **LDAC** - Output latching pin. Can be used to latch the output when updating channels.
- **RDY** - Ready status pin, It is driven low by the DAC when the EEPROM is being written to, signifying that he DAC will not process commands. Goes High when the write is done
- **VA, VB, VC, VD** - DAC output pins for each channel

Arduino

Wiring

Wiring the MCP4728 to communicate with your microcontroller is straightforward thanks to the I2C interface. For these examples, we can use the Metro or Arduino to update the DAC. The instructions below show a Metro (http://adafru.it/2488), but the same applies to an Arduino

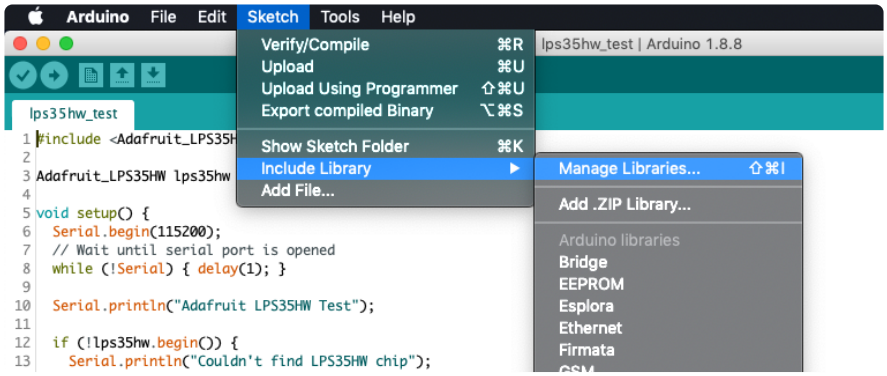


Arduino 5V to **MCP4728 VCC (red wire)** if you are running a 5V board Arduino (Uno, etc.). If your board is 3V, connect to that instead.
Arduino GND to **MCP4728 GND (black wire)**
Arduino SCL to **MCP4728 SCL (yellow wire)**
Arduino SDA to **MCP4728 SDA (blue wire)**
Multimeter Positive Lead to **MCP4728 VA, VB, VC, and VD** in sequence
Multimeter Negative Lead to **GND**

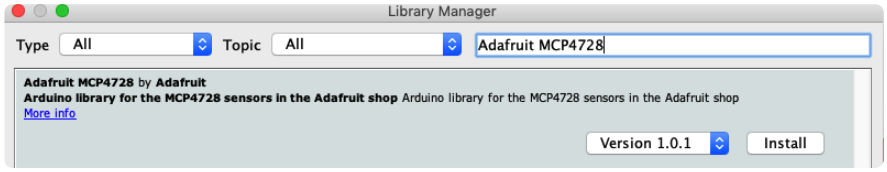
Library Installation

Once wired up, to start using the MCP4728, you'll need to install the Adafruit_MCP4728 library (https://adafru.it/laJ). 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 MCP4728**, and select the **Adafruit MCP4728** library and click **Install**:



Basic Reading Example

Open up **File -> Examples -> Adafruit MCP4728 -> basic_demo** and upload to your Arduino wired up to the sensor.

One you've uploaded the sketch to your board, measure the voltages on each of the four channel outputs and verify that they're within the ranges given for the logic level of your microcontroller.

The values will differ slightly for you but they should be close to the following for a **5V logic level** device such as a Metro 328 or Arduino Uno:

- **5V** on **VA**
- **2.5V** on **VB**
- **1.25V** on **VC**

- **0V** on **VD**

For a **3.3V logic level** device such as a Metro M4 or Feather M0 the voltages should be close to:

- **3.3V** on **VA**
- **1.65V** on **VB**
- **0.825V** on **VC**
- **0V** on **VD**

Basic Example Code

```
// Basic demo for configuring the MCP4728 4-Channel 12-bit I2C DAC
#include <Adafruit_MCP4728.h>
#include <Wire.h>

Adafruit_MCP4728 mcp;

void setup(void) {
  Serial.begin(115200);
  while (!Serial)
    delay(10); // will pause Zero, Leonardo, etc until serial console opens

  Serial.println("Adafruit MCP4728 test!");

  // Try to initialize!
  if (!mcp.begin()) {
    Serial.println("Failed to find MCP4728 chip");
    while (1) {
      delay(10);
    }
  }

  mcp.setChannelValue(MCP4728_CHANNEL_A, 4095);
  mcp.setChannelValue(MCP4728_CHANNEL_B, 2048);
  mcp.setChannelValue(MCP4728_CHANNEL_C, 1024);
  mcp.setChannelValue(MCP4728_CHANNEL_D, 0);
}

void loop() { delay(1000); }
```

If your board has a **MCP4728A4** with an I2C address of **0x64**, then you'll need to edit this line of code:

```
if (!mcp.begin()) {
```

To this:

```
if (!mcp.begin(0x64)) {
```

Vrefs and You

Reference Voltages (**Vref** for short) are an important topic to understand when working with DACs. The **Vref determines the top of the voltage range that the DAC will output**. This is because the DAC starts with the Vref as the output voltage and if the DAC settings specify a lower voltage, the internal circuitry will reduce the Vref voltage down to the specified amount.

If your Vref is 5V, you will be able to have the DAC output voltages from 0V up to 5V. Similarly if you are using a 3.3V source as your Vref, you will be able to scale the DAC's output voltages from 0V to 3.3V. **If you need your DAC to output a voltage, your Vref will need to be the same or a higher voltage.**

The MCP4728 can choose one of two sources for its Vref. The first is the VCC pin which can take a supply voltage between 2.7V and 5.5V. This will match the output range of the DAC to the logic level of your microcontroller.

The second option is to use the 2.048V Vref in the MCP4728 itself. Normally this would mean the highest voltage you can output is 2.048V however when **using the internal Vref,** you can optionally **apply a 2X gain** to the Vref, doubling it and allowing your output voltages to range from 0V to 4.096V

Vref Example

The second included demo has an example of how to set the Vref and Gain for a channel. Open up **File -> Examples -> Adafruit MCP4728 -> vref_demo** and upload to your board wired up to the sensor.

With the exception of Channel A, the examples below show how using the same value for the channels but different Vref values will change the output voltage for the channel.

Use your multimeter on the channel's voltage pin to verify the values.

Channel A

```
// Vref = MCP_VREF_VDD, value = 0, 0V
  mcp.setChannelValue(MCP4728_CHANNEL_A, 0);
```

The value for Channel A is being set to 0, so the Vref doesn't matter, the voltage on the **VA pin** will be **0V** either way.

Channel B

```
// value is vref/2, with 2.048V internal Vref and 1X gain
  // = 2.048/2 = 1.024V
  mcp.setChannelValue(MCP4728_CHANNEL_B, 2048, MCP4728_VREF_INTERNAL,
                      MCP4728_GAIN_1X);
```

For Channel B we're using the Internal 2.048 Vref with the 1X gain, so the Vref is 2.048V. The value is set to 2048 or Vref/2, so the resulting voltage on the **VB pin** is half of 2.048V, **1.024V**

Channel C

```
// value is vref/2, with 2.048V internal vref and *2X gain*
  // = 4.096/2 = 2.048V
  mcp.setChannelValue(MCP4728_CHANNEL_C, 2048, MCP4728_VREF_INTERNAL,
                      MCP4728_GAIN_2X);
```

Channel C uses the 2.048V internal Vref with a 2X gain, bringing the Vref up to 4.096V. The value is again set to Vref/ 2, so the resulting voltage on the **VC pin is 2.048V**

Channel D

```
// value is vref/2, Vref is MCP4728_VREF_VDD(default), the power supply voltage
(usually 3.3V or 5V)
  // For Vdd/Vref = 5V, voltage = 2.5V
  // For 3.3V, voltage = 1.65V
  // Values will vary depending on the actual Vref/Vdd
  mcp.setChannelValue(MCP4728_CHANNEL_D, 2048);
```

Finally, channel D is using the power supply voltage as Vref, so with the **5V logic level** of an Arduino Uno or similar Vref will be 5V. Just like the previous examples, the value for Channel D is set to Vref/2 but because the Vref in this case is 5V, the resulting voltage on **pin VD is 2.5V.**

Note that if you are using **a 3.3V logic level** board, **Vref will be 3.3V** so to voltage on **VD will be 1.65V**

Vref Example

```
// Demo for configuring the Vref of the MCP4728 4-Channel 12-bit I2C DAC
#include <Adafruit_MCP4728.h>
#include <Wire.h>

Adafruit_MCP4728 mcp;
```

```
void setup(void) {
  Serial.begin(115200);
  while (!Serial)
    delay(10); // will pause Zero, Leonardo, etc until serial console opens

  Serial.println("Adafruit MCP4728 test!");

  // Try to initialize!
  if (!mcp.begin()) {
    Serial.println("Failed to find MCP4728 chip");
    while (1) {
      delay(10);
    }
  }
  Serial.println("MCP4728 Found!");
  /*
   * @param channel The channel to update
   * @param new_value The new value to assign
   * @param new_vref Optional vref setting - Defaults to `MCP4728_VREF_VDD`
   * @param new_gain Optional gain setting - Defaults to `MCP4728_GAIN_1X`
   * @param new_pd_mode Optional power down mode setting - Defaults to
   * `MCP4728_PD_MOOE_NORMAL`
   * @param udac Optional UDAC setting - Defaults to `false`, latching (nearly).
   * Set to `true` to latch when the UDAC pin is pulled low
   *
   */

  // Vref = MCP_VREF_VDD, value = 0, 0V
  mcp.setChannelValue(MCP4728_CHANNEL_A, 0);

  // value is vref/2, with 2.048V internal Vref and 1X gain
  // = 2.048/2 = 1.024V
  mcp.setChannelValue(MCP4728_CHANNEL_B, 2048, MCP4728_VREF_INTERNAL,
                      MCP4728_GAIN_1X);

  // value is vref/2, with 2.048V internal vref and *2X gain*
  // = 4.096/2 = 2.048V
  mcp.setChannelValue(MCP4728_CHANNEL_C, 2048, MCP4728_VREF_INTERNAL,
                      MCP4728_GAIN_2X);

  // value is vref/2, Vref is MCP4728_VREF_VDD(default), the power supply
  // voltage (usually 3.3V or 5V) For Vdd/Vref = 5V, voltage = 2.5V For 3.3V,
  // voltage = 1.65V Values will vary depending on the actual Vref/Vdd
  mcp.setChannelValue(MCP4728_CHANNEL_D, 2048);

  mcp.saveToEEPROM();
}

void loop() { delay(1000); }
```

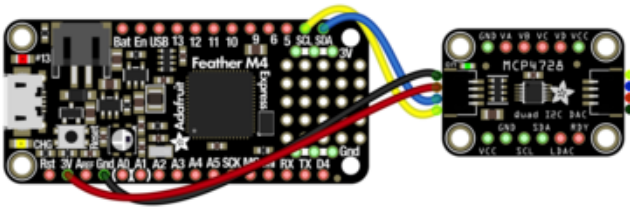
Arduino Docs

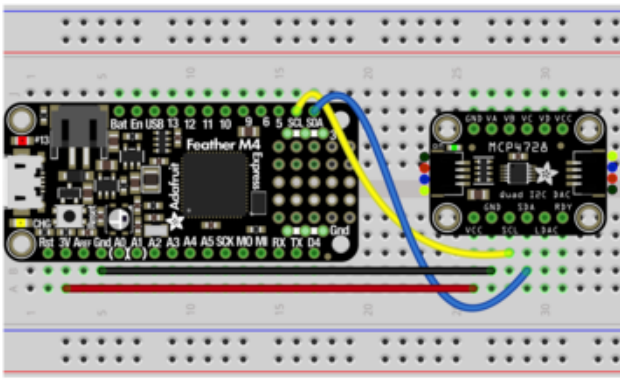
Arduino Docs (https://adafru.it/HE5)

Python & CircuitPython

CircuitPython Microcontroller Wiring

Wiring the MCP4728 to communicate with your microcontroller is straightforward thanks to the I2C interface. For these examples, we can use a multimeter to measure the voltages output on each of the DAC's channels. The instructions below reference an Adafruit Feather (http://adafru.it/3857) microcontroller, but the same applies to a Metro M0 or M4 or other CircuitPython compatible board (https://adafru.it/Em8).



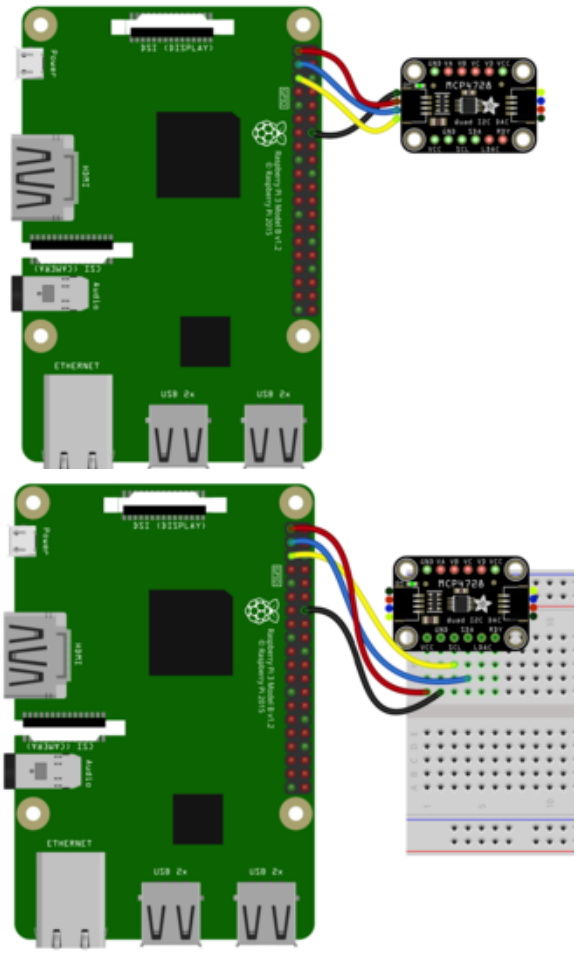


Feather 3.3V to **MCP4728 VCC (red wire)**
Feather GND to **MCP4728 GND (black wire)**
Feather SCL to **MCP4728 SCL (yellow wire)**
Feather SDA to **MCP4728 SDA (blue wire)**
Multimeter Positive Lead to **MCP4728 VA, VB, VC, and VD in sequence**
Multimeter Negative Lead to **GND**

Python Computer Wiring

Since there are dozens of Linux computers/boards you can use, we will show wiring for Raspberry Pi (http://adafru.it/3055). 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:



RPi 3.3V to **MCP4728 VCC (red wire)**
RPi GND to **MCP4728 GND (black wire)**
RPi SCL to **MCP4728 SCL (yellow wire)**
RPi SDA to **MCP4728 SDA (blue wire)**
Multimeter Positive Lead to **MCP4728 VA, VB, VC, and VD in sequence**
Multimeter Negative Lead to **GND**

CircuitPython Installation of MCP4728 Library

You'll need to install the Adafruit CircuitPython MCP4728 (https://adafru.it/laK) library on your CircuitPython board.

First make sure you are running the latest version of Adafruit CircuitPython (https://adafru.it/Amd) for your board.

Next 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/uap). Our CircuitPython starter guide has a great page on how to install the library bundle (https://adafru.it/ABU).

For non-express boards like the Trinket M0 or Gemma M0, you'll need to manually install the necessary libraries from the bundle:

- **adafruit_mcp4728.mpy**
- **adafruit_bus_device**

Before continuing make sure your board's lib folder or root filesystem has the **adafruit_mcp4728.mpy** file and **adafruit_bus_device** folder copied over.

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

Python Installation of MCP4728 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-mcp4728`

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 values for each of the channels. Type the following code in the CircuitPython REPL to import the necessary modules and initialize the I2C connection with the DAC:

```
import board
import busio
import adafruit_mcp4728

i2c = busio.I2C(board.SCL, board.SDA)
mcp4728 =  adafruit_mcp4728.MCP4728(i2c)
```

```
Adafruit CircuitPython 5.0.0-beta.1 on 2019-12-10; Adafruit Metro M4 Express with samd51j19
>>> import board
>>> import busio
>>> import adafruit_mcp4728
>>> i2c = busio.I2C(board.SCL, board.SDA)
>>> mcp4728 =  adafruit_mcp4728.MCP4728(i2c)
```

If your board has a **MCP4728A4** with an I2C address of **0x64**, then you'll need to edit this line of code:

```
mcp4728 = adafruit_mcp4728.MCP4728(i2c)
```

To this:

```
mcp4728 = adafruit_mcp4728.MCP4728(i2c, 0x64)
```

Next, we'll set the values for each channel and then test the voltages with a multimeter

```
mcp4728.channel_a.value = 65535
mcp4728.channel_b.value = int(65535/2)
mcp4728.channel_c.value = int(65535/4)
mcp4728.channel_d.value = 0
```

```
>>> mcp4728.channel_a.value = 65535
>>> mcp4728.channel_b.value = int(65535/2)
>>> mcp4728.channel_c.value = int(65535/4)
>>> mcp4728.channel_d.value = 0
```

You can now use a multimeter to check the voltages on each of the output pins, **VA, VB, VC, and VD.** The values will differ slightly for you but they should be close to:

- **3.3V** on **VA**
- **1.65V** on **VB**
- **0.825V** on **VC**
- **0V** on **VD**

The keen-eyed will have noticed that the MCP4728 is a 12-bit DAC but we're using 16-bit numbers to set the value. CircuitPython uses 16-bit values for voltages, regardless of the underlying hardware. Use "raw_value" to set an unscaled value

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

import board
import adafruit_mcp4728

MCP4728_DEFAULT_ADDRESS = 0x60
MCP4728A4_DEFAULT_ADDRESS = 0x64

i2c = board.I2C()  # uses board.SCL and board.SDA
# i2c = board.STEMMA_I2C()  # For using the built-in STEMMA QT connector on a
microcontroller
#  use for MCP4728 variant
mcp4728 = adafruit_mcp4728.MCP4728(i2c, adafruit_mcp4728.MCP4728_DEFAULT_ADDRESS)
#  use for MCP4728A4 variant
#  mcp4728 = adafruit_mcp4728.MCP4728(i2c,
adafruit_mcp4728.MCP4728A4_DEFAULT_ADDRESS)

mcp4728.channel_a.value = 65535  # Voltage = VDD
mcp4728.channel_b.value = int(65535 / 2)  # VDD/2
mcp4728.channel_c.value = int(65535 / 4)  # VDD/4
mcp4728.channel_d.value = 0  # 0V
```

Vrefs and You

Reference Voltages (**Vref** for short) are an important topic to understand when working with DACs. The **Vref determines the top of the voltage range that the DAC will output**. This is because the DAC starts with the Vref as the output voltage and if the DAC settings specify a lower voltage, the internal circuitry will reduce the Vref voltage down to the specified amount.

If your Vref is 5V, you will be able to have the DAC output voltages from 0V up to 5V. Similarly if you are using a 3.3V source as your Vref, you will be able to scale the DAC's output voltages from 0V to 3.3V. **If you need your DAC to output a voltage, your Vref will need to be the same or a higher voltage.**

The MCP4728 can choose one of two sources for its Vref,. The first is the VCC pin which can take a supply voltage between 2.7V and 5.5V. This will match the output range of the DAC to the logic level of your microcontroller.

The second option is to use the 2.048V Vref in the MCP4728 itself. Normally this would mean the highest voltage you can output is 2.048V however when **using the internal Vref,** you can optionally **apply a 2X gain** to the Vref, doubling it and allowing your output voltages to range from 0V to 4.096V

Vref Example

For brevity, we will assume a logic level of 3.3V. If the logic level for your device is something else, the resulting voltages will be different. Subtle difference in the VDD voltage will also affect the measured values

The included Vref example code shows how the Vref and gain settings work. Typing the following into the CircuitPython REPL to follow along and setup the required library imports, and instantiate the mcp4728 library object.

Additionally we'll define a variable that represents the maximum value for a 12-bit DAC like this one. When the `raw_value` of a channel of the mcp4728 is set to this, it will output the maximum possible voltage for that channel, as **defined by the current Vref**.

```
from time import sleep
import board
import busio
import adafruit_mcp4728
```

```
i2c = busio.I2C(board.SCL, board.SDA)
mcp4728 =  adafruit_mcp4728.MCP4728(i2c)

FULL_VREF_RAW_VALUE =  4095
```

```
>>> from time import sleep
>>> import board
>>> import busio
>>> import adafruit_mcp4728
>>> i2c = busio.I2C(board.SCL, board.SDA)
>>> mcp4728 =  adafruit_mcp4728.MCP4728(i2c)
>>> FULL_VREF_RAW_VALUE =  4095
```

Next we will set the raw value for channel a to half of this full value so that the output will be half of the Vref. We're wrapping the result of dividing the max value in an `int` call to make sure the resulting value is an integer type that the `raw_value` property expects.

Once this is run, measure the voltage on the VA pin with your multimeter. Since the Vref is set to VDD, which is the logic level of the microcontroller, the Vref will be 3.3V. With setting the value to half of the maximum, the resulting voltage should be **3.3V/2 or approximately 1.65V**

```
mcp4728.channel_a.raw_value = int(FULL_VREF_RAW_VALUE/2)
mcp4728.channel_a.vref = adafruit_mcp4728.Vref.VDD
```

```
>>> mcp4728.channel_a.raw_value = int(FULL_VREF_RAW_VALUE/2)
>>> mcp4728.channel_a.vref = adafruit_mcp4728.Vref.VDD
```

```
mcp4728.channel_b.raw_value = int(FULL_VREF_RAW_VALUE/2) # VDD/2
mcp4728.channel_b.vref = adafruit_mcp4728.Vref.INTERNAL
mcp4728.channel_b.gain = 1
```

Next we'll set the Vref for channel B to use the DAC's internal **Vref of 2.048V.** Since we are using the internal Vref, we can set the gain on the internal Vref to 1X, so the final Vref voltage remains 2.048V.

We set the value of the channel to the same value as the previous example, however when you measure the voltage on channel B's **output pin VB**, you will see that because we're using a different lower Vref, the resulting voltage is **approximately 1.024V,** again half of the 2.048V Vref.

```
>>> mcp4728.channel_b.raw_value = int(FULL_VREF_RAW_VALUE/2)
>>> mcp4728.channel_b.vref = adafruit_mcp4728.Vref.INTERNAL
>>> mcp4728.channel_b.gain = 1
```

```
mcp4728.channel_c.raw_value = int(FULL_VREF_RAW_VALUE/2) # VDD/2
mcp4728.channel_c.vref = adafruit_mcp4728.Vref.INTERNAL
mcp4728.channel_c.gain = 2
```

```
>>> mcp4728.channel_c.raw_value = int(FULL_VREF_RAW_VALUE/2)
>>> mcp4728.channel_c.vref = adafruit_mcp4728.Vref.INTERNAL
>>> mcp4728.channel_c.gain = 2
```

Finally for channel C, we again set the raw value to half of the maximum, so the resulting voltage should be half of the Vref voltage. We then set the Vref source to the **internal 2.048V Vref,** however this time we set the internal Vref's **gain to 2X**, making the final **Vref voltage 4.096V**. With the value for the channel set to half of the maximum, we get half of the Vref value, or in this case approximately **2.048V on pin VC**

Without changing output value setting compared to the first channel, we were able to change the output voltage by changing the Vref.

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

from time import sleep
import board
import adafruit_mcp4728

i2c = board.I2C()  # uses board.SCL and board.SDA
# i2c = board.STEMMA_I2C()  # For using the built-in STEMMA QT connector on a
microcontroller
mcp4728 = adafruit_mcp4728.MCP4728(i2c)

FULL_VREF_RAW_VALUE = 4095

# pylint: disable=no-member
mcp4728.channel_a.raw_value = int(FULL_VREF_RAW_VALUE / 2)  # VDD/2
mcp4728.channel_a.vref = (
    adafruit_mcp4728.Vref.VDD
)  # sets the channel to scale between 0v and VDD

mcp4728.channel_b.raw_value = int(FULL_VREF_RAW_VALUE / 2)  # VDD/2
mcp4728.channel_b.vref = adafruit_mcp4728.Vref.INTERNAL
mcp4728.channel_b.gain = 1

mcp4728.channel_c.raw_value = int(FULL_VREF_RAW_VALUE / 2)  # VDD/2
mcp4728.channel_c.vref = adafruit_mcp4728.Vref.INTERNAL
mcp4728.channel_c.gain = 2

mcp4728.save_settings()

while True:
    sleep(1)
```

Note the use of the mcp4728.save_settings() command -- this stores the current settings into the breakout board's EEPROM, so at boot-up it will always supply the amount of voltage that was saved.

Python Docs

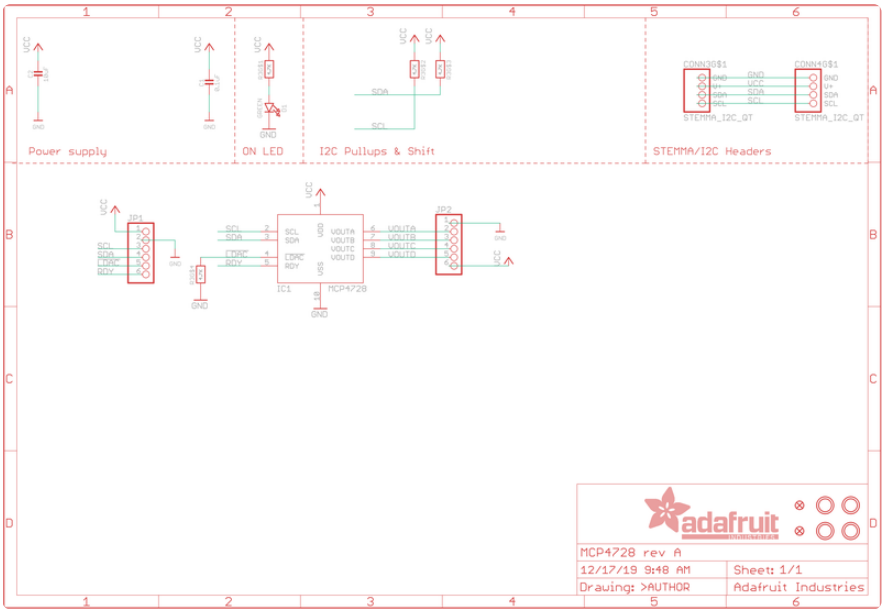
Python Docs (https://adafru.it/HE6)

Downloads

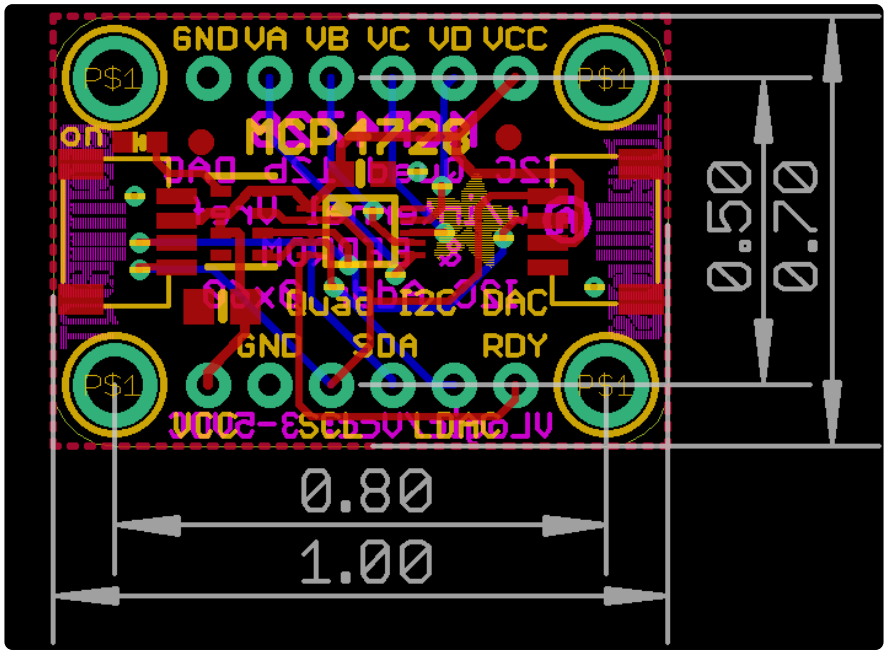
Files

- MCP4728 Datasheet (https://adafru.it/cfQ)
- EagleCAD files on GitHub (https://adafru.it/laL)
- Fritzing object in Adafruit Fritzing Library (https://adafru.it/laM)

Schematic



Fab Print


GND VA VB VC VD VCC
P$1
P$1
MCP4728
Quad I2C DAC
GND SDA RDY
P$1
P$1
0.50
0.70
0.80
1.00