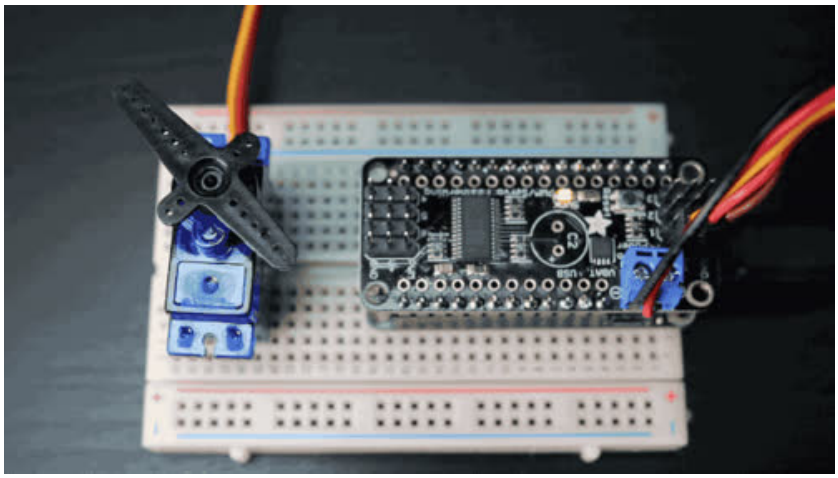
MicroPython Hardware: PCA9685 PWM & Servo Driver

Created by Tony DiCola

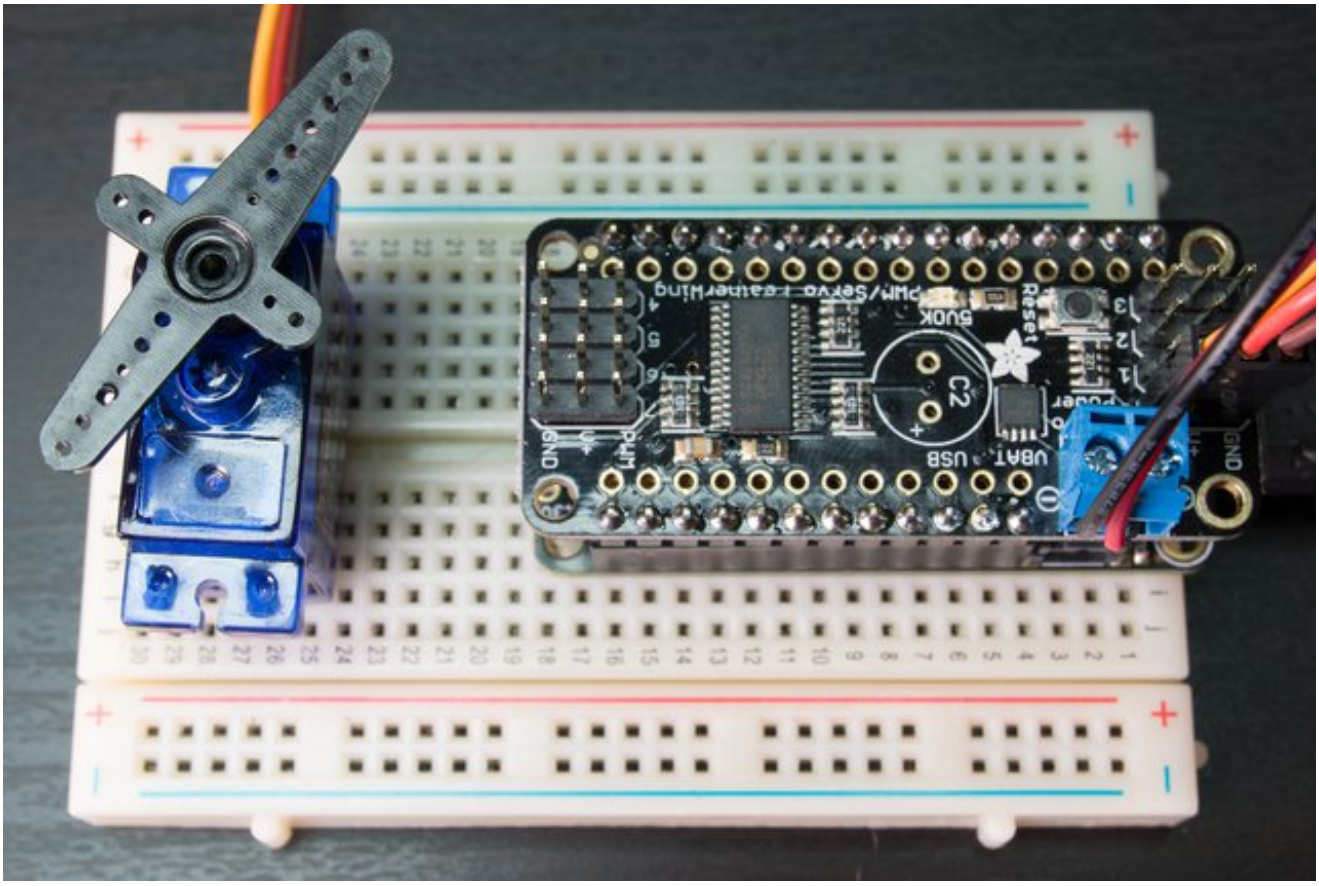


Last updated on 2017-01-27 09:54:59 AM UTC

Guide Contents

Overview

Note this guide has been updated to show how to use the PCA9685 with both MicroPython.org and Adafruit CircuitPython firmware. There are small differences between the two so be careful to read and follow the instructions that call out these differences.



Servo motors are one way to make projects come to life with exciting movements like steering robots, flipping switches, and more. To control a servo you need to generate a special PWM, or pulse-width modulation, signal. Most development boards can generate a few basic PWM signals, but what if you need to control a lot of servos or don't have a board with PWM support? The PCA9685 PWM driver board comes to the rescue! This driver board can generate up to 16-channels (or 8-channels in FeatherWing form) of PWM signals--perfect for driving many servos!

This guide explores how to use the PCA9685 PWM & Servo driver with MicroPython. You'll learn how to connect the PCA9685 to a MicroPython board and use it to control LED brightness and move servos.

To follow this guide you'll want to be familiar with MicroPython by reading these guides:

- MicroPython Basics: What is MicroPython? (http://adafru.it/pXa)
- MicroPython Basics: How to Load MicroPython on a Board (http://adafru.it/pNB)
- MicroPython Basics: Load Files & Run Code (http://adafru.it/s1f)
- MicroPython Hardware: Analog I/O (http://adafru.it/s2d) (see the PWM page (http://adafru.it/scR) specifically)

See all the MicroPython guides in the learning system (http://adafru.it/qzD) for more information.

In addition check out these guides on servo motors:

- Adafruit Motor Selection Guide: Servos (http://adafru.it/scS)
- Arduino Lesson 14: Servos (http://adafru.it/scT) (focus on the general servo information, less on the Arduino details)

Hardware

Parts

You'll need the following parts to follow this guide:

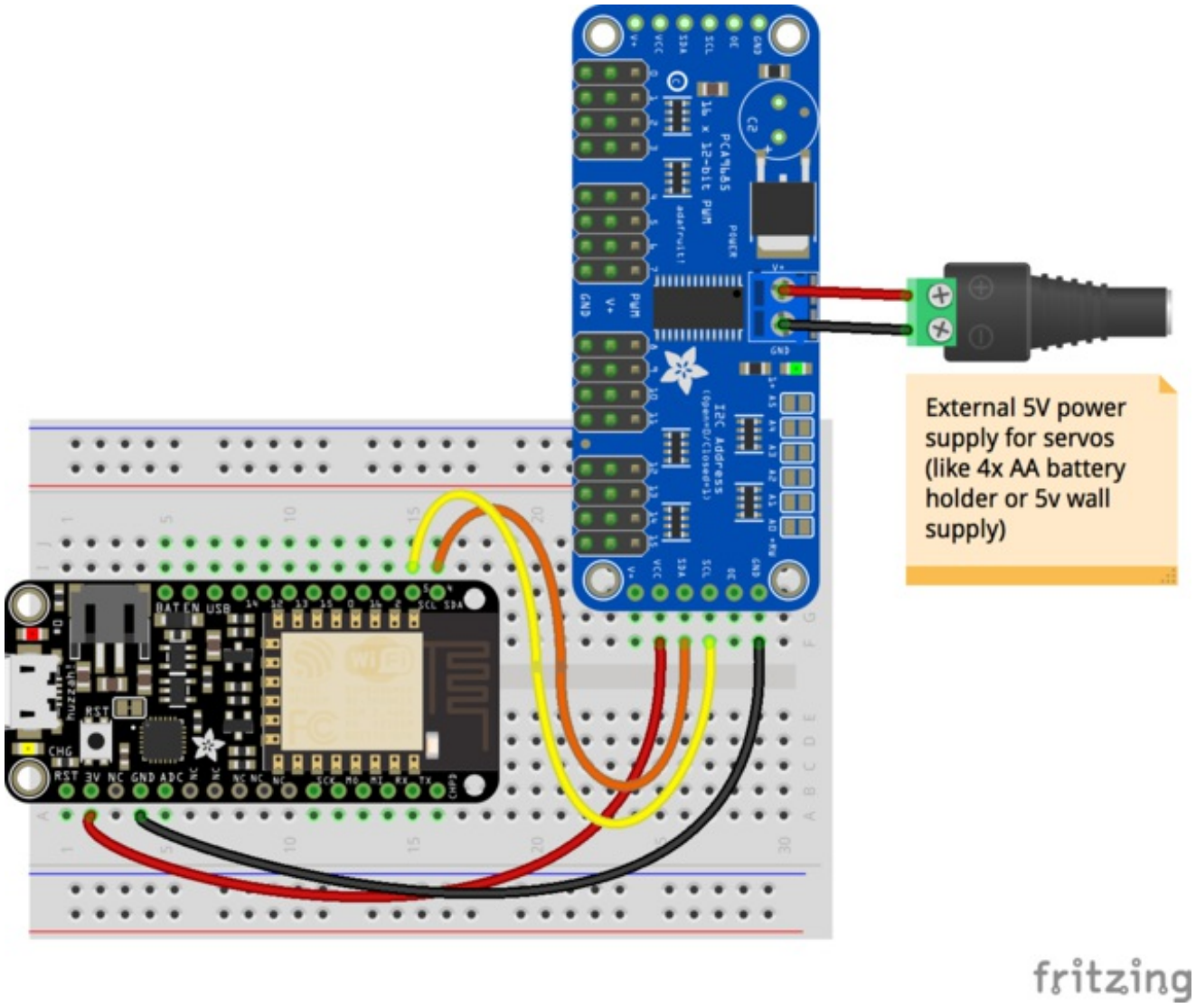
- **MicroPython board.** This guide focuses on the ESP8266 (http://adafru.it/n6A) and Feather M0/SAMD21-based boards (http://adafru.it/2772), but any MicroPython board that supports I2C should work.
    - If your board doesn't come with MicroPython running on it already then check out this how to load MicroPython on a board guide (http://adafru.it/saq).
    - If you're using a Feather board and FeatherWing you probably want a Feather female header set (http://adafru.it/2886) or Feather stacking female header set (http://adafru.it/2830).
- **PCA9685 PWM & Servo Driver Board.** If you're using a Feather the 8-channel PWM & Servo FeatherWing (http://adafru.it/2928) is the perfect option. If you need more channels or aren't using a Feather grab the 16-channel PCA9685 breakout board (http://adafru.it/815) or even 16-channel PCA9685 Arduino shield (http://adafru.it/1411).
- **5V Power Supply.** To power servos you almost always need an external 5V power supply. Servos can pull a lot of power and could damage your board if powered directly from it! A simple power option is a 4x AA battery pack (http://adafru.it/830) or a 5V power supply (http://adafru.it/276) and barrel jack screw terminal connector (http://adafru.it/368).
- **Servos or LEDs.** You'll want something to drive with the PCA9685, like servos (http://adafru.it/scU) you can move or LEDs (http://adafru.it/dLA) you can dim. Remember the PCA9685 only drives **servos**, not stepper or other motors.
- **Breadboard (http://adafru.it/64)** and **jumper wires (http://adafru.it/153)**. If you aren't using a Feather and FeatherWing you'll need a breadboard and jumper wires to connect the components.
- **Soldering tools (http://adafru.it/136)**. You'll need to solder headers to the boards Check out the guide to excellent soldering (http://adafru.it/dxy) if you're new to soldering.

Make sure to follow the board and PCA9685 driver board product guides to assemble and verify they work before continuing.

Wiring

If you're using a Feather and the PCA9685 FeatherWing just slide the wing into the Feather's headers and you're done! The wing will use an I2C connection to talk to the Feather board.

If you're using a PCA9685 breakout you'll need to connect its I2C and power pins as follows:



- **Board SCL** / **I2C clock** to **PCA9685 SCL**.
- **Board SDA** / **I2C data** to **PCA9685 SDA**.
- **Board 3.3V power** to **PCA9685 VCC**.
- **Board GND** / **ground** to **PCA9685 GND** / **ground**.

Once your board is wired to the PCA9685 continue on to learn how to use a MicroPython

module to control servos and LEDs!

Software

This page explains how to use this library with both MicroPython.org and Adafruit CircuitPython firmware. Pay careful attention to the differences as each firmware requires slightly different install and setup.

MicroPython Module Install

To use the PCA9685 with your MicroPython board you'll need to install the micropython-adafruit-pca9685 MicroPython module (http://adafru.it/scq) on your board. **Remember this module is for MicroPython.org firmware and not Adafruit CircuitPython!** Skip down to the CircuitPython Module Install section if you're using CircuitPython.

First make sure you are running the latest version of MicroPython for your board. If you're using the **ESP8266 MicroPython** port you **must** be running version **1.8.5 or higher (http://adafru.it/sas)** as earlier versions do not support using .mpy modules as shown in this guide.

Next download the latest **pca9685.mpy, servo.mpy, motor.mpy, and stepper.mpy** file from the releases page (http://adafru.it/scr) of the micropython-adafruit-pca9685 GitHub repository (http://adafru.it/scq). You'll need to copy **all** of the files to your MicroPython board's file system and can use a tool like ampy to copy the files to the board (http://adafru.it/r2B).

Adafruit CircuitPython Module Install

To use the PCA9685 with your Adafruit CircuitPython (http://adafru.it/tCy) board you'll need to install the Adafruit_CircuitPython_PCA9685 (http://adafru.it/tZF) module on your board. **Remember this module is for Adafruit CircuitPython firmware and not MicroPython.org firmware!** Jump back up to the MicroPython Module Install section if you're using MicroPython firmware.

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

Next download the latest **adafruit_pca9685.zip** file from the releases page of the Adafruit_CircuitPython_PCA9685 GitHub repository (http://adafru.it/u0a). You'll need to unzip this file and copy the entire **adafruit_pca9685** directory to the board's root filesystem.

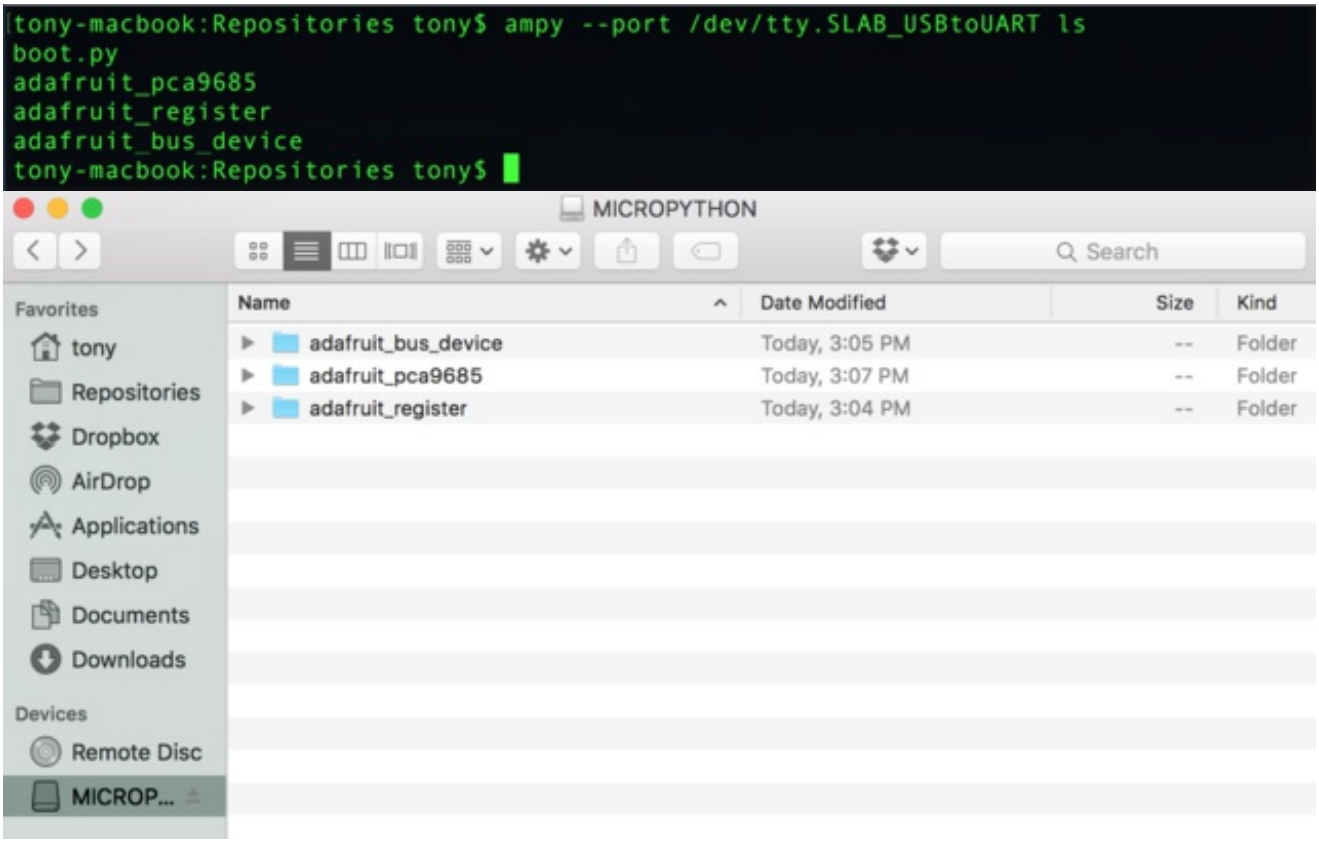
If your board supports USB mass storage, like the SAMD21 CircuitPython port, then simply drag the files to the board's file system. **Note on boards without external SPI flash, like a Feather M0 or Trinket/Gemma M0, you might run into issues on Mac OSX with hidden files taking up too much space when drag and drop copying, see this page for a workaround (http://adafru.it/u1d).**

If your board doesn't support USB mass storage, like the ESP8266, thenuse a tool like ampy to copy the file to the board (http://adafru.it/s1f). You can use the latest version of ampy and its new directory copy command (http://adafru.it/q2A) to easily move module directories to the board.

In addition you'll need both the Adafruit CircuitPython Bus Device (http://adafru.it/u0b) and Adafruit CircuitPython Register (http://adafru.it/u0c) modules installed on your board. Just like installing the module as mentioned above, download the latest release .zip file and copy the folder inside it to the board's root filesystem for each:

- Adafruit CircuitPython Bus Device Module releases (http://adafru.it/u0d)
- Adafruit CircuitPython Register Module releases (http://adafru.it/u0e)

Before continuing make sure your board's root filesystem has the**adafruit_pca9685**, **adafruit_bus_device**, and **adafruit_register** folders/modules copied over.



Usage

The following section will show how to control the PCA9685 from the board's Python prompt / REPL. You'll learn how to interactively control servos and dim LEDs by typing in the code below.

First connect to the board's serial REPL (http://adafru.it/pMf)so you are at the MicroPython **>>>** prompt.

I2C Initialization

First you'll need to initialize the I2C bus for your board. Note the I2C initialization differs slightly for MicroPython vs. Adafruit CircuitPython so pay careful attention to the right section below for your firmware.

MicroPython I2C Initialization

On MicroPython.org firmware which uses the machine API you can initialize I2C likethe MicroPython I2C guide mentions (http://adafru.it/sau). For example on a board like the ESP8266 you can run (assuming you're using the default SDA gpio #4 and SCL gpio #5 pins like on a Feather & PCA9685 FeatherWing):

```
import machine
i2c = machine.I2C(machine.Pin(5), machine.Pin(4))
```

Adafruit CircuitPython I2C Initialization

On CircuitPython the I2C initialization is slightly different as it uses the board module to more easily address board pins, and nativeio or bitbangio modules for I2C access.

First import the necessary modules, if you're using a board that has software/bitbang I2C support (like the ESP8266) run:

```
from board import *
import bitbangio as io
```

Or if your board has hardware/native I2C support (like SAMD21-based boards) run:

```
from board import *
import nativeio as io
```

Now for either board run this command to create the I2C instance using the default SCL and SDA pins (which will be marked on the boards pins if using a Feather or similar Adafruit board):

```
i2c = io.I2C(SCL, SDA)
```

You can also use a context manager for I2C as mentioned in the SAMD21 MicroPython guide (http://adafru.it/s1a). If you aren't using a context manager be sure to call **deinit** when finished with the I2C bus.

Import PCA9685 Module

After initializing the I2C interface you need to import the PCA9685 module to use it in your own code. This differs slightly between the MicroPython.org library and Adafruit CircuitPython library so pay careful attention to the command to run.

For MicroPython.org firmware and the micropython-adafruit-pca9685 module run this command:

```
import pca9685
```

For Adafruit CircuitPython firmware and the Adafruit_CircuitPython_PCA9685 module run this command:

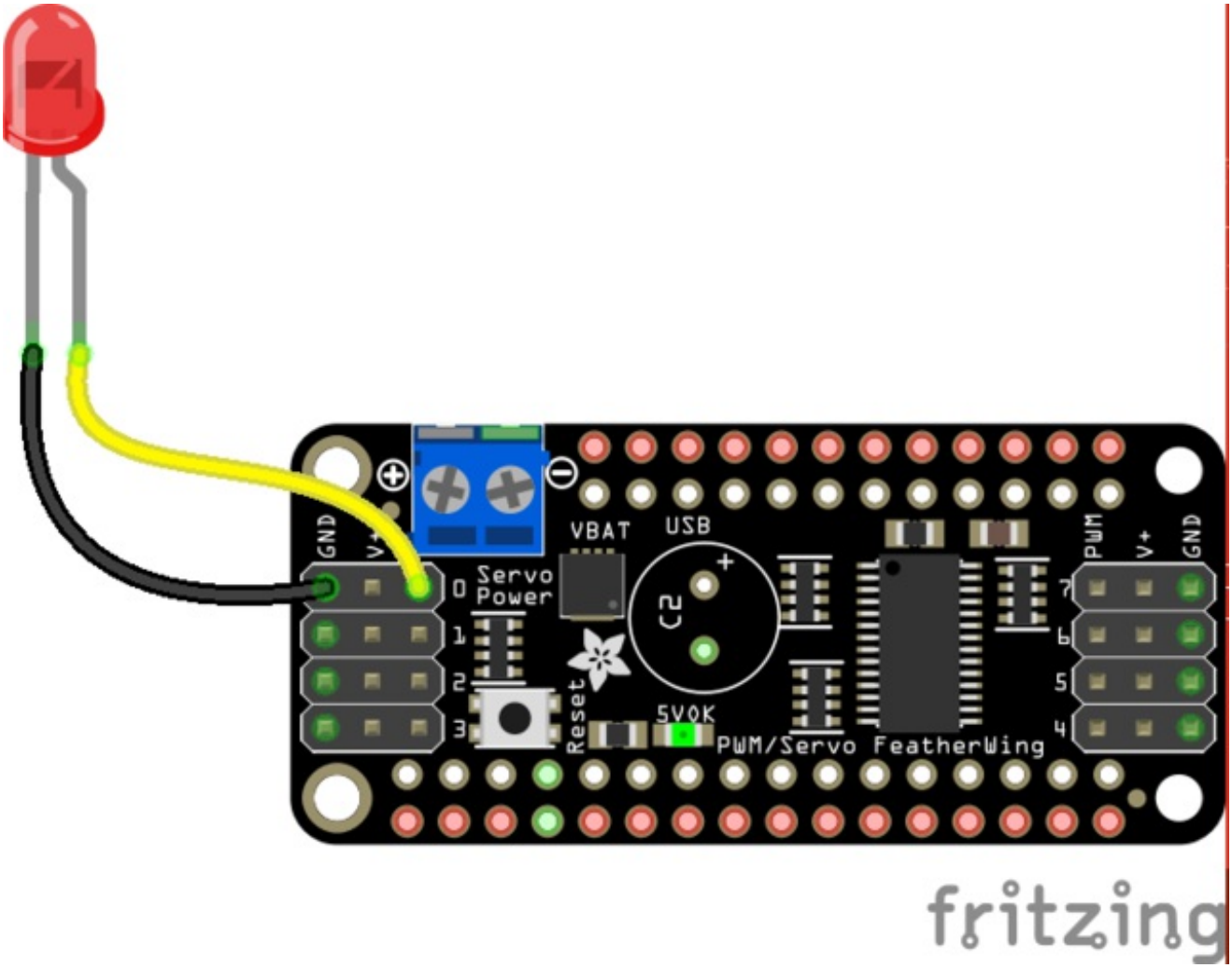
```
from adafruit_pca9685 import pca9685
```

Once the library is imported the usage of it is exactly the same between MicroPython.org and Adafruit CircuitPython firmware.

Dim LED's

Each channel of the PCA9685 can be used to control the brightness of an LED. The PCA9685 generates a high-speed PWM signal which turns the LED on and off very quickly. If the LED is turned on longer than turned off it will appear brighter to your eyes.

First wire a LED to the board as follows. Note you don't need to use a resistor to limit current through the LED as the PCA9685 will limit the current to around 10mA:



- **LED cathode** / **shorter leg** to **PCA9685 channel GND** / **ground**.
- **LED anode** / **longer leg** to **PCA9685 channel PWM**.

Now in the Python REPL you can create an instance of the basic PCA9685 class which provides low-level PWM control of the board's channels:

```
pca = pca9685.PCA9685(i2c)
```

The PCA9685 class provides control of the PWM frequency and each channel's duty cycle. Check out the PCA9685 class documentation (http://adafru.it/scV) for more details.

For dimming LEDs you typically don't need to use a fast PWM signal frequency and can set the board's PWM frequency to 60hz by calling the **freq** function:

```
pca.freq(60)
```

Now control the LED brightness by controlling the duty cycle of the channel connected to the LED. The duty cycle value should be a 12-bit value, i.e. 0 to 4095, which represents what percent of the time the signal is on vs. off. A value of 4095 is 100% brightness, 0 is

0% brightness, and in-between values go from 0% to 100% brightness.

For example set the LED completely on with a duty cycle of 4095:

```
pca.duty(0, 4095)
```

The first parameter to the **duty** function call is the channel number, in this case channel 0 if you followed the diagram above. The second parameter is the duty cycle value mentioned above.

After running the command above you should see the LED light up at full brightness!

Now turn the LED off with a duty cycle of 0:

```
pca.duty(0, 0)
```

Try an in-between value like 1000:

```
pca.duty(0, 1000)
```

You should see the LED dimly lit. Try experimenting with other duty cycle values to see how the LED changes brightness!

For example make the LED glow on and off by calling**duty** in a loop:

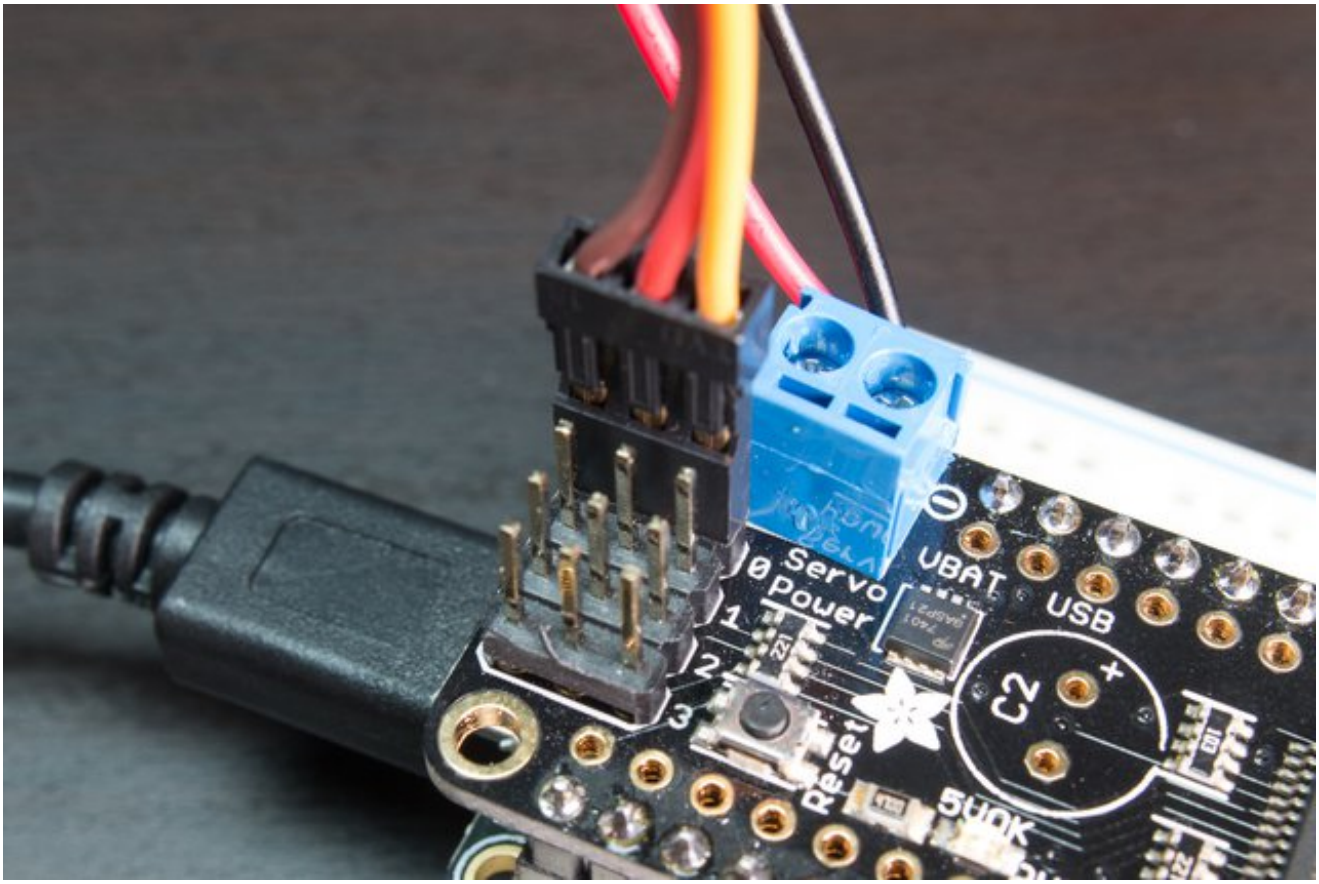
```
import time

while True:
    # Ramp up in brightness:
    for i in range(4096):
        pca.duty(0, i)
        time.sleep(0.001)
    # Ramp back down in brightness:
    for i in range(4094, 0, -1):
        pca.duty(0, i)
        time.sleep(0.001)
```

Press **Ctrl-C** to stop the loop from running and continue typing code at the REPL.

Control Servos

Servo motors use a PWM signal to control the position of their arm or horn. With the PCA9685 you can easily plug in servos and control them with MicroPython. If you aren't familiar with servos be sure to first read this intro to servos page (http://adafru.it/scW) and this in-depth servo guide page (http://adafru.it/scS).

First connect the servo to a channel on the PCA9685. Be sure to plug the servo connector in the correct way! Check your servo's datasheet to be sure, but typically the brown wire is connected to ground, the red wire is connected to 5V power, and the yellow pin is connected to PWM:



Be sure you've turned on or plugged in the external 5V power supply to the PCA9685 board too!

Now in the Python REPL you can create an instance of the Servos class which provides simple servo control (note the small difference in imports between MicroPython.org and Adafruit CircuitPython firmware):

```
# For MicroPython.org firmware use this import:
import servo
# Or for Adafruit CircuitPython firmware use this import instead:
from adafruit_pca9685 import servo

servos = servo.Servos(i2c)
```

By default the Servos class will use a frequency and minimum & maximum pulse-width values that should work for most servos. However check the Servos class documentation (http://adafru.it/scX) for more details on extra parameters to customize the signal generated for your servos.

There are a few ways to control the position of the servo using the **position** function. One way is to specify the pulse length in microseconds. Most servos will go to their center position at a pulse length of 1500 microseconds, a 90 degree extreme at 2000 microseconds, and the opposite 90 degree extreme at 1000 microseconds.

Try setting the servo to its center position with a pulse length of 1500 microseconds:

```
servos.position(0, us=1500)
```

The first parameter to the **position** function is the channel number connected to the servo (be sure to set the right channel based on how you connected your servo).

The next parameter is a keyword **us** which specifies the channel pulse length in microseconds.

Try other extremes like 2000 and 1000 microseconds to see how the servo moves:

```
servos.position(0, us=2000)
servos.position(0, us=1000)
```

Your servos might even be able to go to higher and lower values than 2000 and 1000. Check the datasheet if you can to see the miminum and maximum pulse lengths.

You can also specify a position as an angle. This is a little trickier to use since you'll need to know the total angle that your servo can sweep between. The default is 180 degrees but your servo might have a smaller sweep--change the total angle by specifying the **degrees** parameter in the Servos class initializer above.

Next call **position** and specify a **degrees** keyword, for example to set the maximum 180 degree position:

```
servos.position(0, degrees=180)
```

Or to sweep back to the minimum 0 degree position:

```
servos.position(0, degrees=0)
```

That's all there is to controlling servos with the PCA9685 and MicroPython! Using the position function you can sweep and move servos in any way. This is perfect for building robots, actuating switches, or other fun mechanical projects!