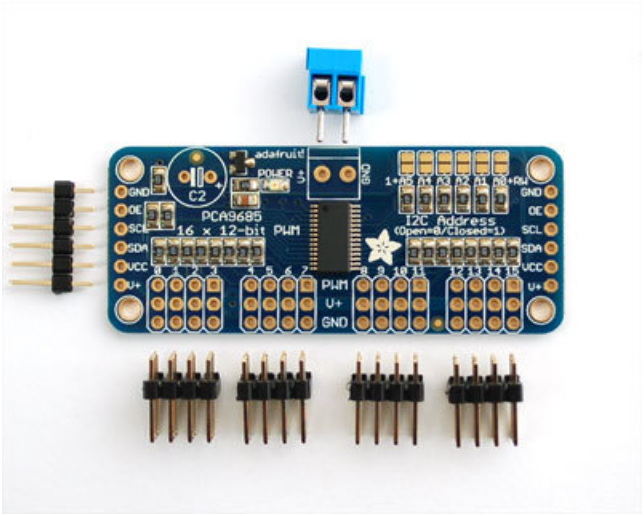
adafruit learning system

# Adafruit 16-Channel Servo Driver with Arduino

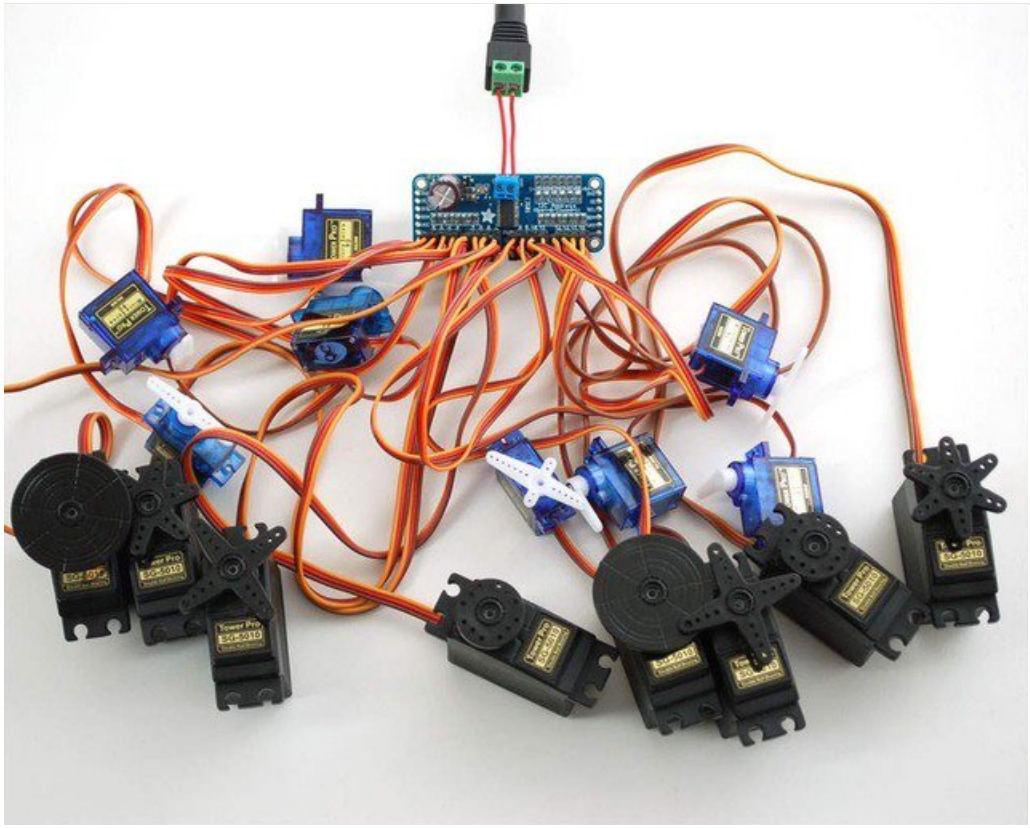
Created by Bill Earl




Last updated on 2018-01-16 12:17:12 AM UTC

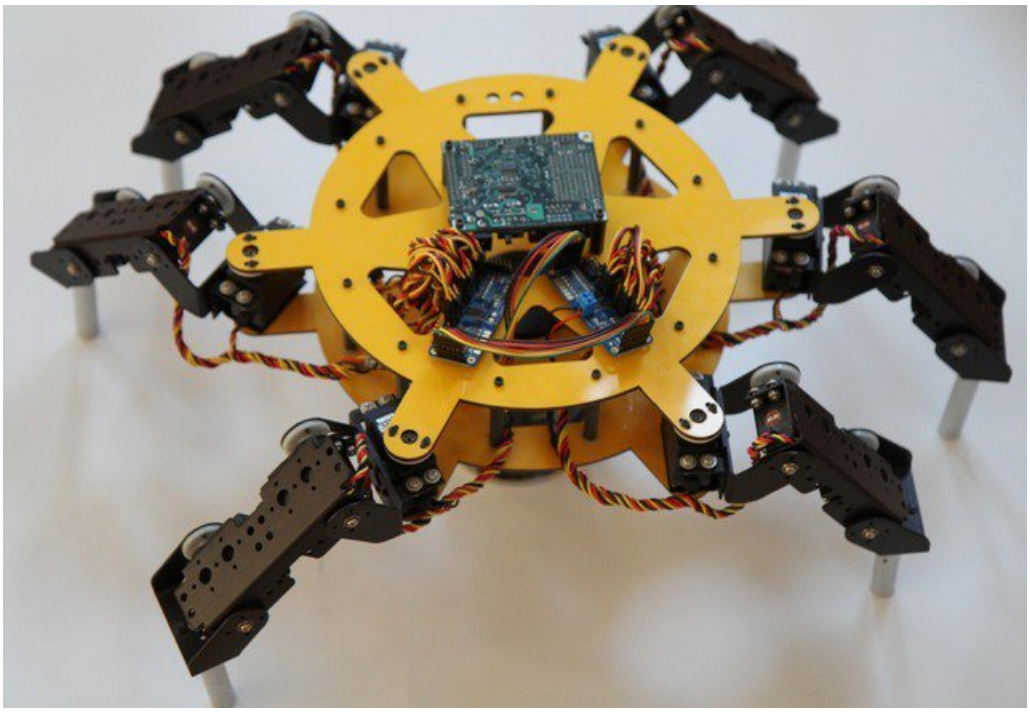
## Guide Contents

# Overview

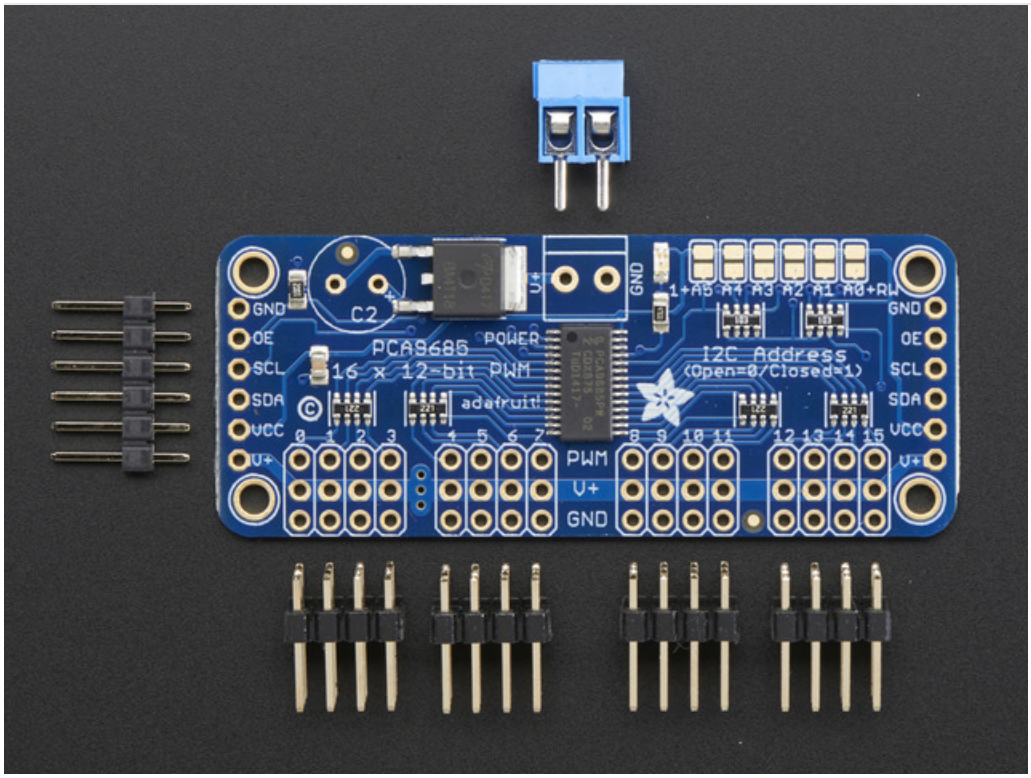


Driving servo motors with the Arduino Servo library is pretty easy, but each one consumes a precious pin - not to mention some Arduino processing power.  The Adafruit 16-Channel 12-bit PWM/Servo Driver will drive up to 16 servos over I2C with only 2 pins.  The on-board PWM controller will drive all 16 channels simultaneously with no additional Arduino processing overhead.  What's more, you can chain up to 62 of them to control up to 992 servos - all with the same 2 pins!

The Adafruit PWM/Servo Driver is the perfect solution for any project that requires a lot of servos.



## Pinouts




There are two sets of control input pins on either side. **Both sides of the pins are identical!** Use whichever side you like, you can also easily chain by connecting up two side-by-side

## Power Pins

- **GND** - This is the power and signal ground pin, must be connected
- **VCC** - This is the **logic** power pin, connect this to the logic level you want to use for the PCA9685 output, should be 3 - 5V max! It's also used for the 10K pullups on SCL/SDA so unless you have your own pullups, have it match the microcontroller's logic level too!
- **V+** - This is an *optional* power pin that will supply distributed power to the servos. If you are not using for servos you can leave disconnected. It is not used at all by the chip. You can also inject power from the 2-pin terminal block at the top of the board. You should provide 5-6VDC if you are using servos. If you have to, you can go higher to 12VDC, but if you mess up and connect VCC to V+ you could damage your board!

## Control Pins

- **SCL** - I2C clock pin, connect to your microcontrollers I2C clock line. Can use 3V or 5V logic, and has a weak pullup to **VCC**
- **SDA** - I2C data pin, connect to your microcontrollers I2C data line. Can use 3V or 5V logic, and has a weak pullup to **VCC**
- **OE** - Output enable. Can be used to quickly disable all outputs. When this pin is *low* all pins are *enabled*. When the pin is *high* the outputs are *disabled*. Pulled low by default so it's an optional pin!

## Output Ports

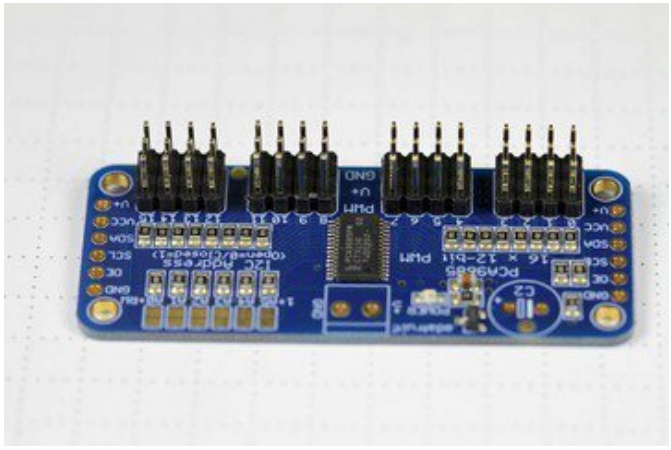
There are 16 output ports. Each port has 3 pins: V+, GND and the PWM output. Each PWM runs completely independently *but* they must all have the same PWM frequency. That is, for LEDs you probably want 1.0 KHz but servos

need 60 Hz - so you cannot use half for LEDs @ 1.0 KHz and half @ 60 Hz.

They're set up for servos but you can use them for LEDs! Max current per pin is 25mA.

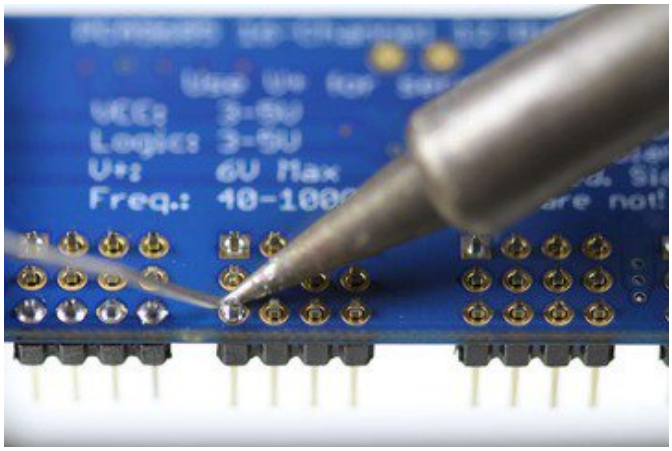
There are 220 ohm resistors in series with all PWM Pins and the output logic is the same as **VCC** so keep that in mind if using LEDs.

## Assembly



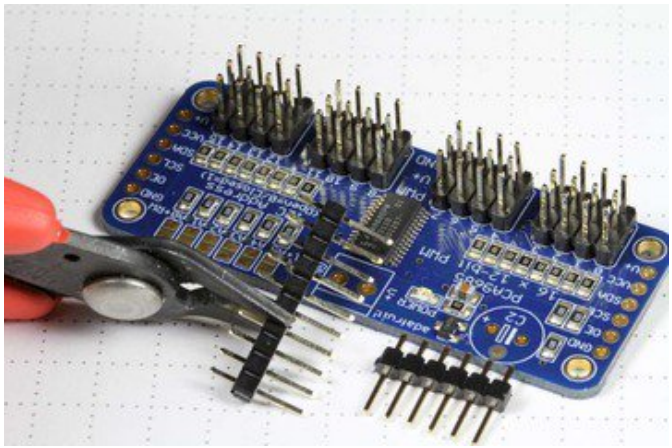
### Install the Servo Headers

Install 4 3x4 pin male headers into the marked positions along the edge of the board.



### Solder all pins

There are a lot of them!

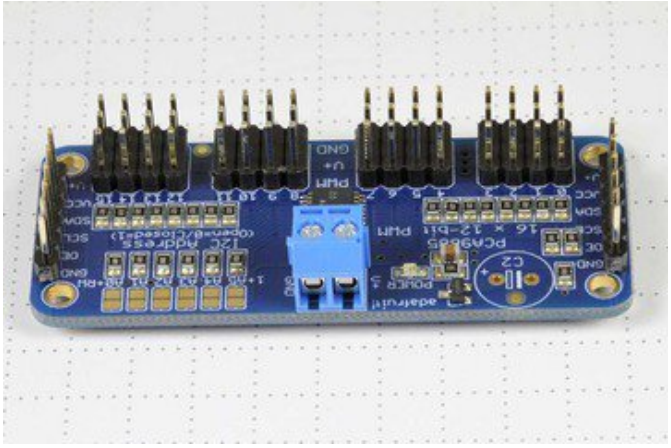


### Add Headers for Control

A strip of male header is included. Where you want to install headers and on what side depends a little on use:

- **For breadboard (http://adafru.it/239) use**, install headers on the bottom of the board.
- **For use with jumper wires (http://adafru.it/758),** install the headers on top of the board.
- **For use with our 6-pin cable (http://adafru.it/206),** install the headers on top of the board.

If you are chaining multiple driver boards, you will want headers on both ends.



## Install Power Terminals

If you are chaining multiple driver boards, you only need a power terminal on the first one.

# Hooking it Up

## Connecting to the Arduino

The PWM/Servo Driver uses I2C so it take only 4 wires to connect to your Arduino:

**"Classic" Arduino wiring:**

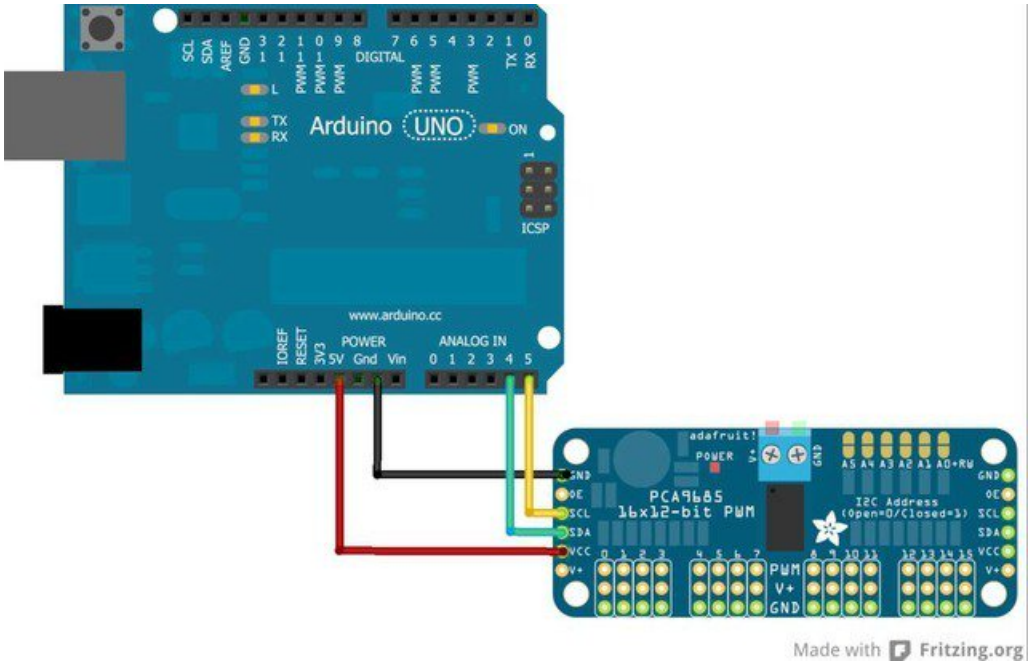
- +5v -> VCC (this is power for the BREAKOUT only, NOT the servo power!)
- GND -> GND
- Analog 4 -> SDA
- Analog 5 -> SCL

**Older Mega wiring:**

- +5v -> VCC (this is power for the BREAKOUT only, NOT the servo power!)
- GND -> GND
- Digital 20 -> SDA
- Digital 21 -> SCL

**R3 and later Arduino wiring (Uno, Mega & Leonardo):**
(These boards have dedicated SDA & SCL pins on the header nearest the USB connector)

- +5v -> VCC (this is power for the BREAKOUT only, NOT the servo power!)
- GND -> GND
- SDA -> SDA
- SCL -> SCL




The VCC pin is just power for the chip itself. If you want to connect servos or LEDs that use the V+ pins, you MUST connect the V+ pin as well. The V+ pin can be as high as 6V even if VCC is 3.3V (the chip is 5V safe). We suggest connecting power through the blue terminal block since it is polarity protected.

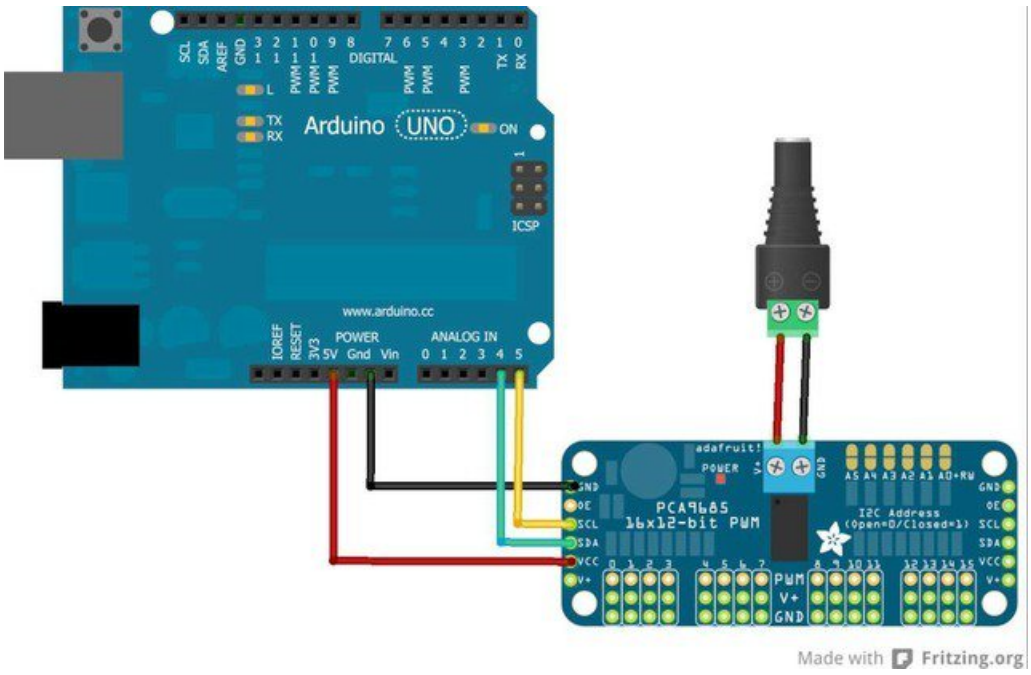
## Power for the Servos

Most servos are designed to run on about 5 or 6v. Keep in mind that a lot of servos moving at the same time (particularly large powerful ones) will need a lot of current. Even micro servos will draw several hundred mA when moving. Some High-torque servos will draw more than 1A each under load.

Good power choices are:

- 5v 2A switching power supply
- 5v 10A switching power supply
- 4xAA Battery Holder - 6v with Alkaline cells. 4.8v with NiMH rechargeable cells.
- 4.8 or 6v Rechargeable RC battery packs from a hobby store.

> It is not a good idea to use the Arduino 5v pin to power your servos. Electrical noise and 'brownouts' from excess current draw can cause your Arduino to act erratically, reset and/or overheat.

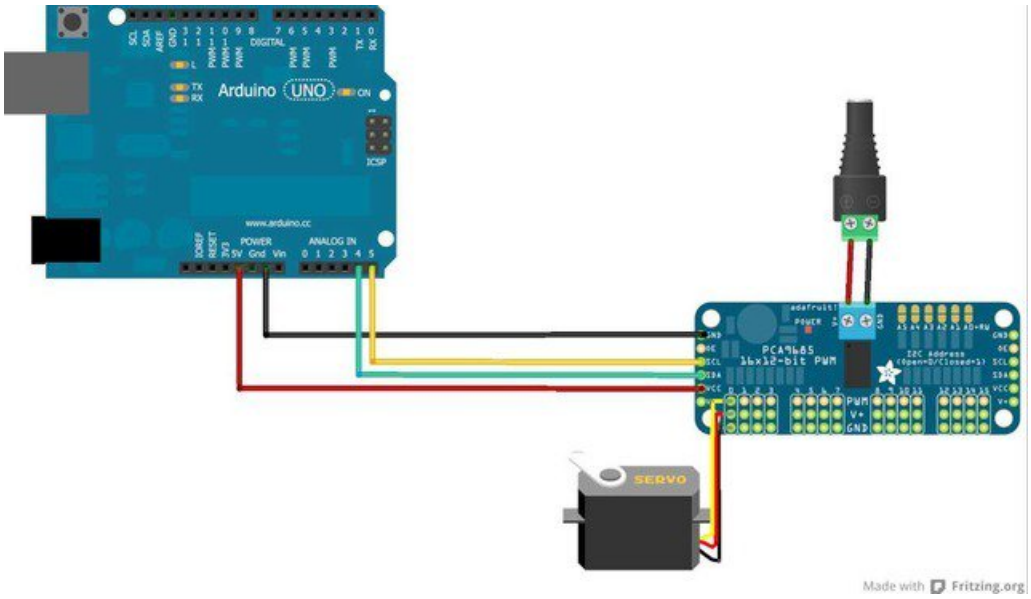



## Adding a Capacitor to the thru-hole capacitor slot

We have a spot on the PCB for soldering in an electrolytic capacitor. Based on your usage, you may or may not need a capacitor. If you are driving a lot of servos from a power supply that dips a lot when the servos move, **n** * **100uF** where **n** is the number of servos is a good place to start - eg **470uF** or more for 5 servos. Since its so dependent on servo current draw, the torque on each motor, and what power supply, there is no "one magic capacitor value" we can suggest which is why we don't include a capacitor in the kit.

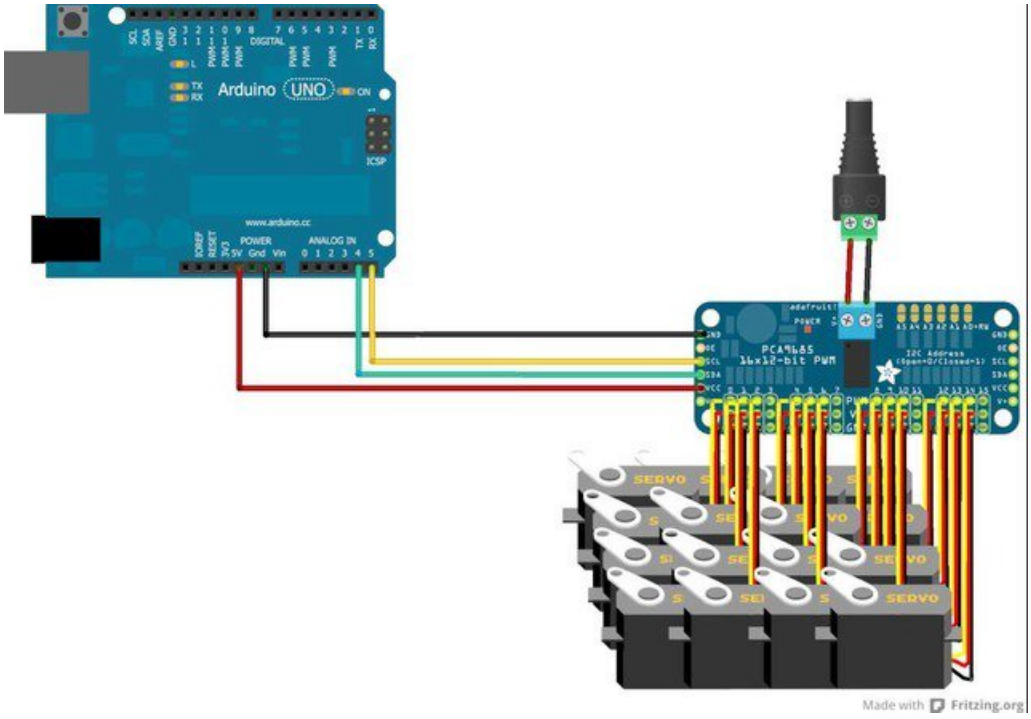
## Connecting a Servo

Most servos come with a standard 3-pin female connector that will plug directly into the headers on the Servo Driver. Be sure to align the plug with the ground wire (usually black or brown) with the bottom row and the signal wire (usually yellow or white) on the top.



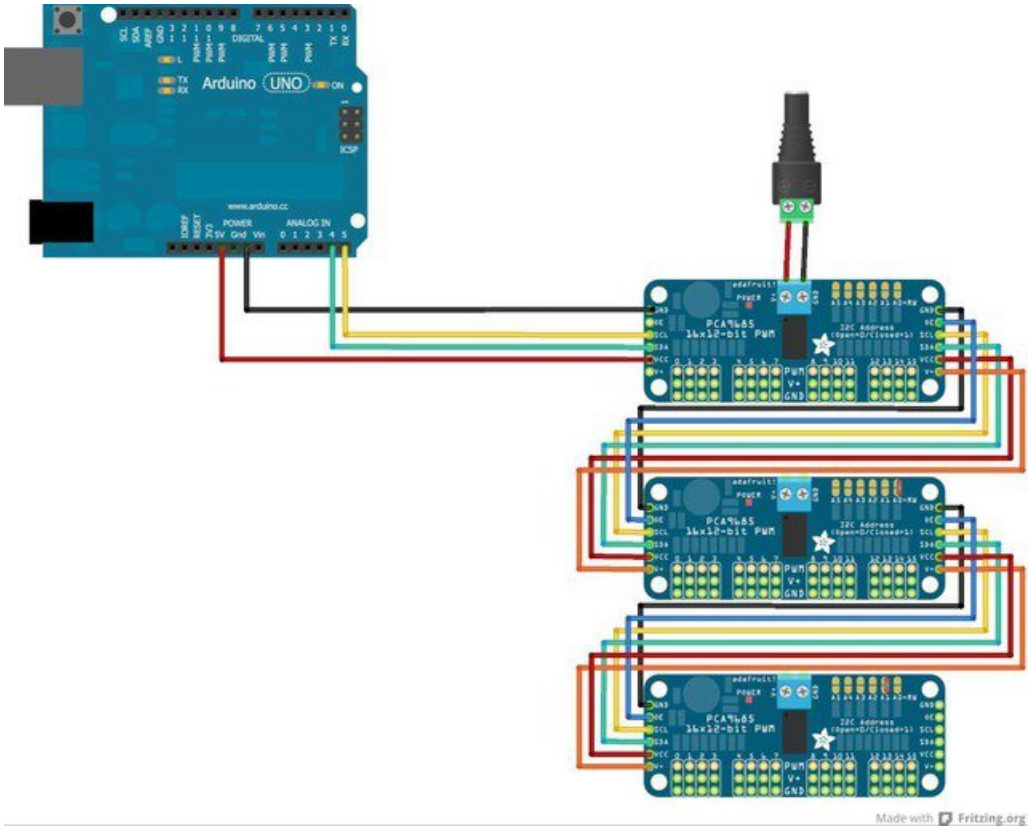

## Adding More Servos

Up to 16 servos can be attached to one board.  If you need to control more than 16 servos, additional boards can be chained as described on the next page.



## Chaining Drivers

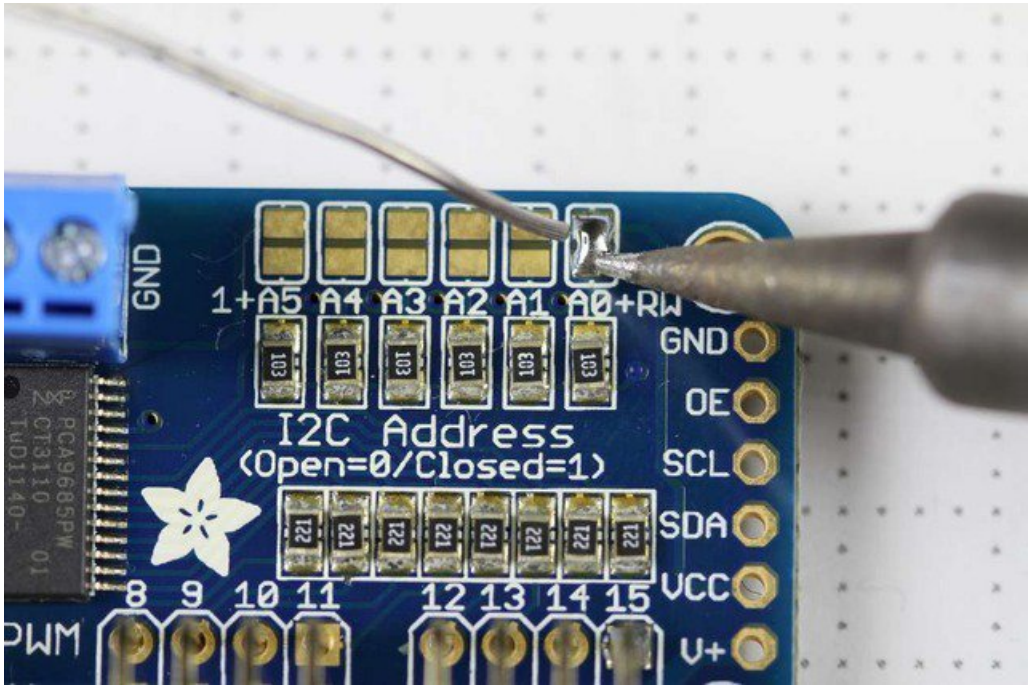
Multiple Drivers (up to 62) can be chained to control still more servos.  With headers at both ends of the board, the wiring is as simple as connecting a 6-pin parallel cable from one board to the next.




## Addressing the Boards

Each board in the chain must be assigned a unique address.  This is done with the address jumpers on the upper right edge of the board.  The I2C base address for each board is 0x40.  The binary address that you program with the address jumpers is added to the base I2C address.

To program the address offset, use a drop of solder to bridge the corresponding address jumper for each binary '1' in the address.




Board 0: Address = 0x40 Offset = binary 00000 (no jumpers required)
Board 1: Address = 0x41 Offset = binary 00001 (bridge A0 as in the photo above)
Board 2: Address = 0x42 Offset = binary 00010 (bridge A1)
Board 3: Address = 0x43 Offset = binary 00011 (bridge A0 & A1)
Board 4: Address = 0x44 Offset = binary 00100 (bridge A2)

etc.

In your sketch, you'll need to declare a separate pobject for each board. Call begin on each object, and control each servo through the object it's attached to. For example:

```
#include <Wire.h>
#include <Adafruit_PWMServoDriver.h>

Adafruit_PWMServoDriver pwm1 = Adafruit_PWMServoDriver(0x40);
Adafruit_PWMServoDriver pwm2 = Adafruit_PWMServoDriver(0x41);

void setup() {
  Serial.begin(9600);
  Serial.println("16 channel PWM test!");

  pwm1.begin();
  pwm1.setPWMFreq(1600);  // This is the maximum PWM frequency

  pwm2.begin();
  pwm2.setPWMFreq(1600);  // This is the maximum PWM frequency
}
```

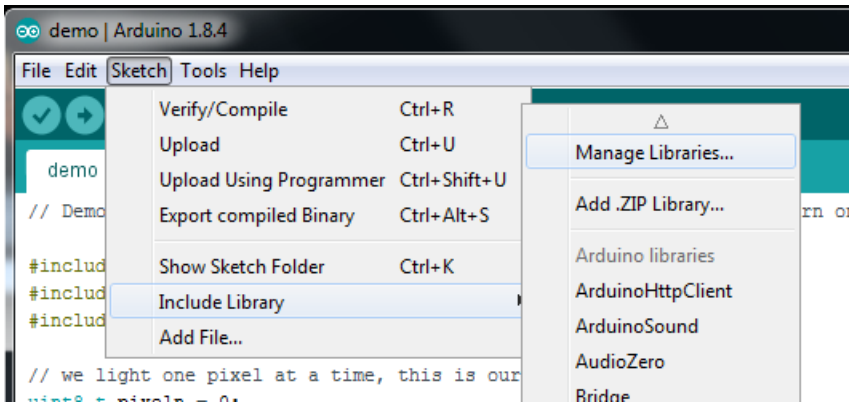
## Using the Adafruit Library

Since the PWM Servo Driver is controlled over I2C, its super easy to use with any microcontroller or microcomputer. In this demo we'll show using it with the Arduino IDE but the C++ code can be ported easily

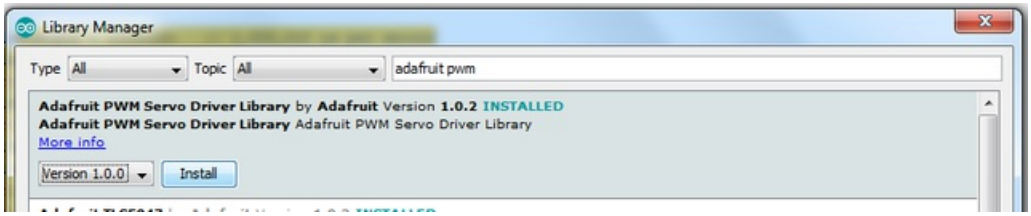
## Install Adafruit PCA9685 library

To begin reading sensor data, you will need to install the Adafruit_PWMServo library (code on our github repository). It is available from the Arduino library manager so we recommend using that.

From the IDE open up the library manager...




And type in **adafruit pwm** to locate the library. Click **Install**

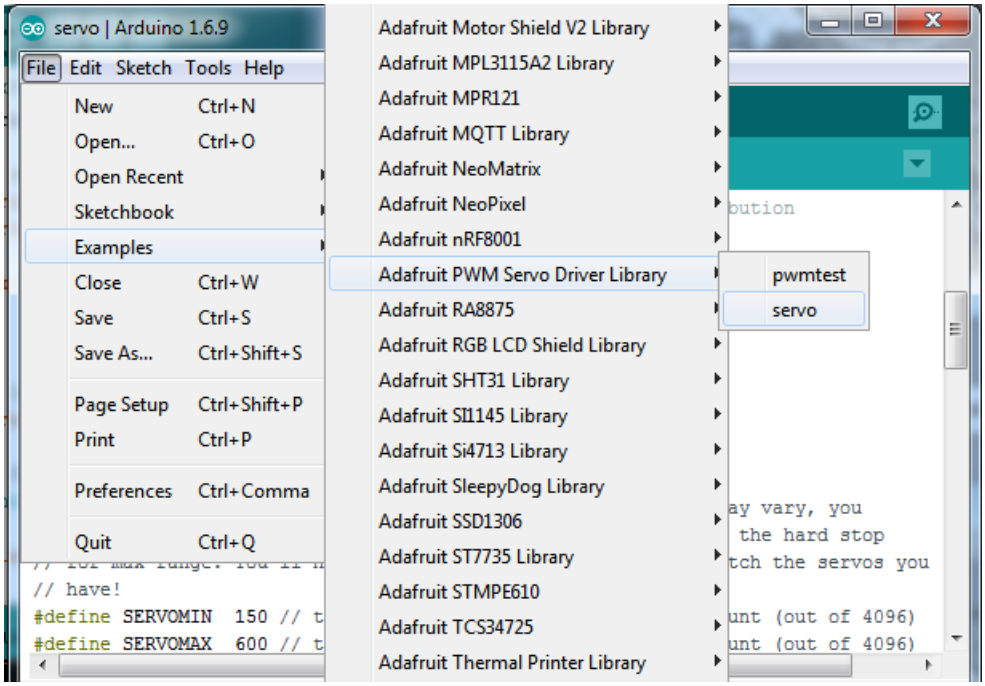



We also have a great tutorial on Arduino library installation at:
http://learn.adafruit.com/adafruit-all-about-arduino-libraries-install-use

## Test with the Example Code:

First make sure all copies of the Arduino IDE are closed.

Next open the Arduino IDE and select **File->Examples->Adafruit_PWMServoDriver->Servo**. This will open the example file in an IDE window.




If using a Breakout:

Connect the driver board and servo as shown on the previous page. Don't forget to provide power to both **Vin** (3-5V logic level) and **V+** (5V servo power). **Check the green LED is lit!**

If using a Shield:

Plug the shield into your Arduino. Don't forget you will also have to provide 5V to the V+ terminal block. **Both red and green LEDs must be lit**.

If using a FeatherWing:

Plug the FeatherWing into your Feather. Don't forget you will also have to provide 5V to the V+ terminal block. **Check the green LED is lit!**

## Connect a Servo

A single servo should be plugged into the **PWM #0** port, the first port. You should see the servo sweep back and forth over approximately 180 degrees.

### Calibrating your Servos

Servo pulse timing varies between different brands and models.  Since it is an analog control circuit, there is often some variation between samples of the same brand and model.  For precise position control, you will want to calibrate the minumum and maximum pulse-widths in your code to match known positions of the servo.

**Find the Minimum:**
Using the example code, edit SERVOMIN until the low-point of the sweep reaches the minimum range of travel.  It is best to approach this gradually and stop before the physical limit of travel is reached.

**Find the Maximum:**

Again using the example code, edit SERVOMAX until the high-point of the sweep reaches the maximum range of travel.  Again, is best to approach this gradually and stop before the physical limit of travel is reached.

> Use caution when adjusting SERVOMIN and SERVOMAX. Hitting the physical limits of travel can strip the gears and permanently damage your servo.

## Converting from Degrees to Pulse Length

The Arduino "map()" function is an easy way to convert between degrees of rotation and your calibrated SERVOMIN and SERVOMAX pulse lengths.  Assuming a typical servo with 180 degrees of rotation; once you have calibrated SERVOMIN to the 0-degree position and SERVOMAX to the 180 degree position, you can convert any angle between 0 and 180 degrees to the corresponding pulse length with the following line of code:

```
pulselength = map(degrees, 0, 180, SERVOMIN, SERVOMAX);
```

# Library Reference

## setPWMFreq(freq)

### Description

This function can be used to adjust the PWM frequency, which determines how many full 'pulses' per second are generated by the IC. Stated differently, the frequency determines how 'long' each pulse is in duration from start to finish, taking into account both the high and low segments of the pulse.

Frequency is important in PWM, since setting the frequency too high with a very small duty cycle can cause problems, since the 'rise time' of the signal (the time it takes to go from 0V to VCC) may be longer than the time the signal is active, and the PWM output will appear smoothed out and may not even reach VCC, potentially causing a number of problems.

### Arguments

- **freq**: A number representing the frequency in Hz, between 40 and 1000

### Example

The following code will set the PWM frequency to the maximum value of 1000Hz:

```
pwm.setPWMFreq(1000)
```

## setPWM(channel, on, off)

### Description

This function sets the start (on) and end (off) of the high segment of the PWM pulse on a specific channel.  You specify the 'tick' value between 0..4095 when the signal will turn on, and when it will turn off.  Channel indicates which of the 16 PWM outputs should be updated with the new values.

### Arguments

- **channel**: The channel that should be updated with the new values (0..15)
- **on**: The tick (between 0..4095) when the signal should transition from low to high
- **off**:the tick (between 0..4095) when the signal should transition from high to low

### Example

The following example will cause channel 15 to start low, go high around 25% into the pulse (tick 1024 out of 4096), transition back to low 75% into the pulse (tick 3072), and remain low for the last 25% of the pulse:

```
pwm.setPWM(15, 1024, 3072)
```

# Using as GPIO

There's also some special settings for turning the pins fully on or fully off

You can set the pin to be fully on with

```
pwm.setPWM(pin, 4096, 0);
```

You can set the pin to be fully off with

```
pwm.setPWM(pin, 0, 4096);
```

## Arduino Library Docs

Arduino Library Docs (https://adafru.it/Au7)

# CircuitPython

This guide is for version 3.0.0 of the PCA9685 library. Make sure to use a bundle from 20180110 or later.

## Adafruit CircuitPython Module Install

To use the PCA9685 with your Adafruit CircuitPython board you'll need to install the Adafruit_CircuitPython_PCA9685 module on your board. **Remember this module is for Adafruit CircuitPython firmware and not MicroPython.org firmware!**

First make sure you are running the latest version of Adafruit CircuitPython for your board. Next you'll need to install the necessary libraries to use the hardware--read below and carefully follow the referenced steps to find and install these libraries from Adafruit's CircuitPython library bundle with a version 20180110 or newer.

### Bundle Install

For express boards that have extra flash storage, like the Feather/Metro M0 express and Circuit Playground express, you can easily install the necessary libraries with Adafruit's CircuitPython bundle. This is an all-in-one package that includes the necessary libraries to use the PCA9685 with CircuitPython. To install the bundle follow the steps in your board's guide, like these steps for the Feather M0 express board.

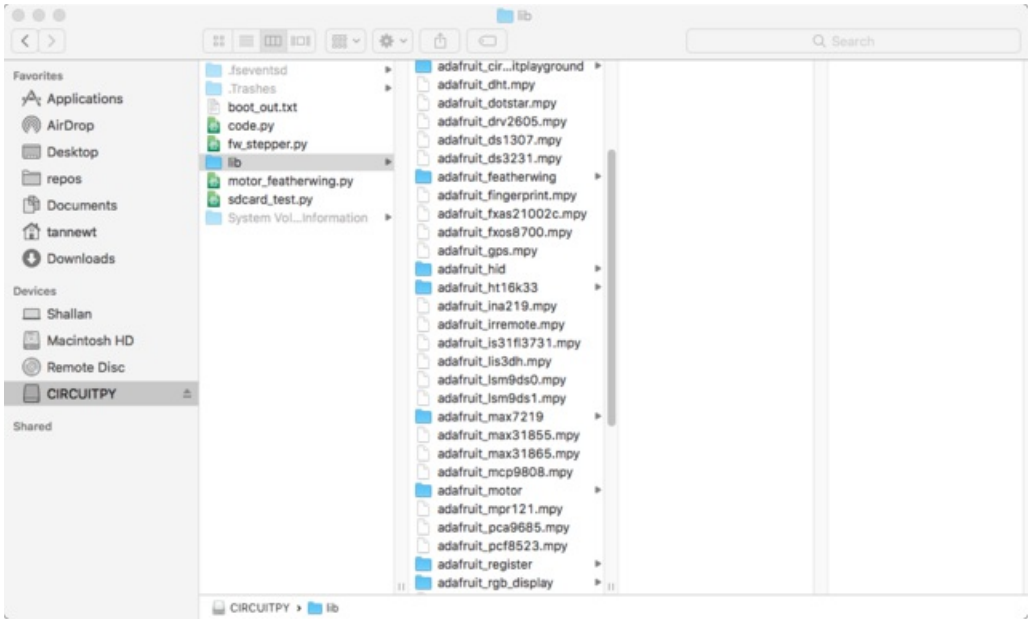
Remember for non-express boards like the Trinket M0, Gemma M0, and Feather/Metro M0 basic you'll need to manually install the necessary libraries from the bundle:

- **adafruit_pca9685**
- **adafruit_bus_device**
- **adafruit_register**
- **adafruit_motor**

If your board supports USB mass storage, like the M0-based boards, 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.**

If your board doesn't support USB mass storage, like the ESP8266, then use a tool like ampy to copy the file to the board. You can use the latest version of ampy and its new directory copy command to easily move module directories to the board.

Before continuing make sure your board's lib folder or root filesystem has the **adafruit_pca9685**, **adafruit_bus_device**, **adafruit_register** and **adafruit_motor** 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 so you are at the CircuitPython >>> prompt.

### I2C Initialization

First you'll need to initialize the I2C bus for your board. First import the necessary modules:

```
import board
import busio
```

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 = busio.I2C(board.SCL, board.SDA)
```

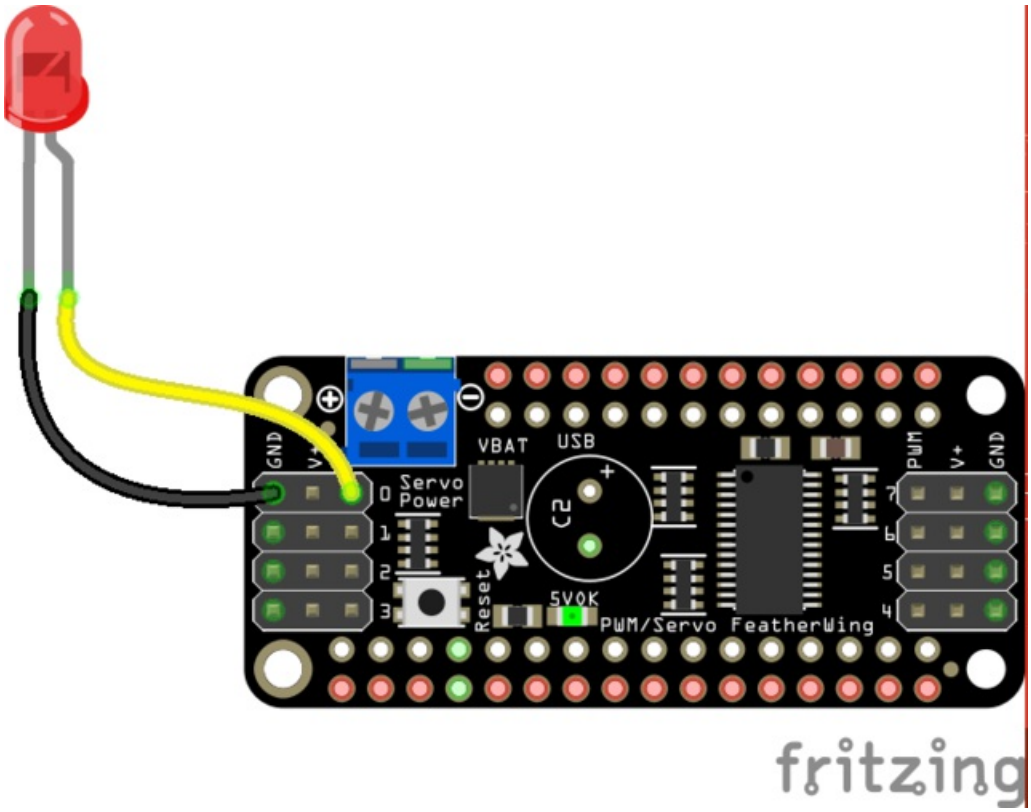
After initializing the I2C interface you need to import the PCA9685 module to use it in your own code:

```
import adafruit_pca9685
```

## Dimming 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:




Fritzing Source

https://adafru.it/zew

- **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 = adafruit_pca9685.PCA9685(i2c)
```

The PCA9685 class provides control of the PWM frequency and each channel's duty cycle. Check out the PCA9685 class documentation 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 setting the **frequency** attribute:

```
pca.frequency = 60
```

The PCA9685 supports 16 separate channels that share a frequency but can have independent duty cycles. That way you could dim 16 LEDs separately!

The PCA9685 object has a **channels** attribute which has an object for each channel that can control the duty cycle. To get the individual channel use the [] to index into **channels**.

```
led_channel = pca.channels[0]
```

Now control the LED brightness by controlling the duty cycle of the channel connected to the LED. The duty cycle value should be a 16-bit value, i.e. 0 to 0xffff, which represents what percent of the time the signal is on vs. off. A value of 0xffff 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 0xffff:

```
led_channel.duty_cycle = 0xffff
```

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:

```
led_channel.duty_cycle = 0
```

Try an in-between value like 1000:

```
led_channel.duty_cycle = 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 setting **duty_cycle** in a loop:

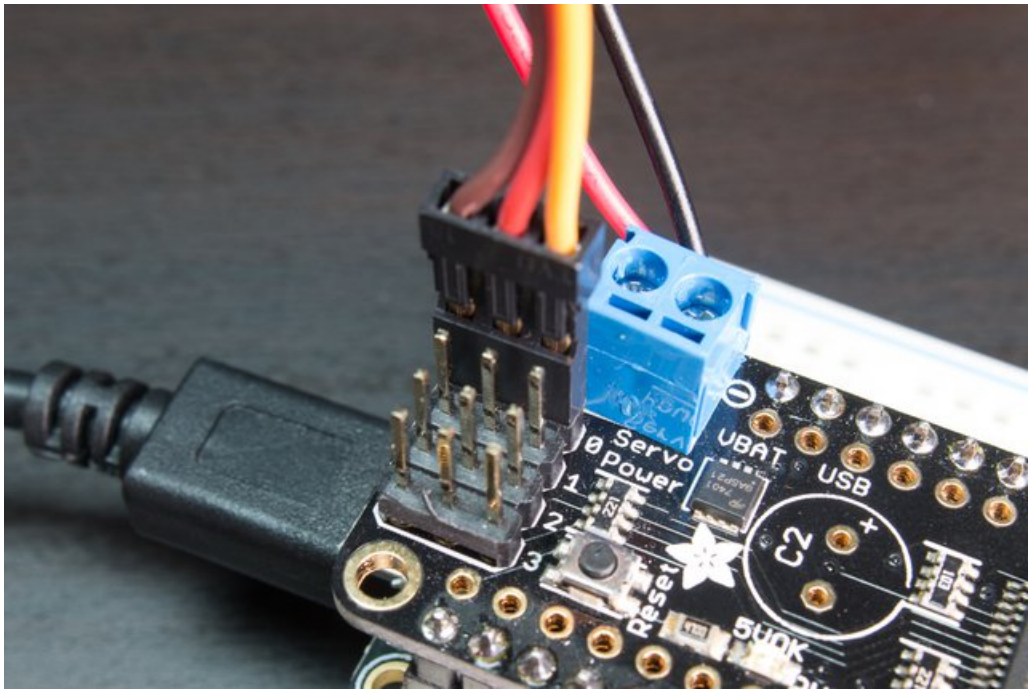
```
# Increase brightness:
for i in range(0xffff):
    led_channel.duty_cycle = i

# Decrease brightness:
for i in range(0xffff, 0, -1):
    led_channel.duty_cycle = i
```

These for loops take a while because 16-bits is a lot of numbers. **CTRL-C** to stop the loop from running and return to the REPL.

## Control Servos

Servo motors use a PWM signal to control the position of their arm or horn. Using the PCA9685 and the Motor library you can easily plug in servos and control them with Python. If you aren't familiar with servos be sure to first read this intro to servos page and this in-depth servo guide page.

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 as above with the led, save a variable for its channel:

```
servo_channel = pca.channels[0]
```

Servos typically operate at a frequency of 50 hz so update pca accordingly.

```
pca.frequency = 50
```

Now that the PCA9685 is set up for servos lets make a Servo object so that we can adjust the servo based on **angle** instead of **duty_cycle**.

By default the Servo class will use actuation range, minimum pulse-width, and maximum pulse-width values that should work for most servos. However check the Servo class documentation for more details on extra parameters to customize the signal generated for your servos.

```
import adafruit_motor.servo
servo = adafruit_motor.servo.Servo(servo_channel)
```

With Servo, you specify a position as an angle. The angle will always be between 0 and the actuation range given when Servo was created. The default is 180 degrees but your servo might have a smaller sweep--change the total angle by specifying the **actuation_angle** parameter in the Servo class initializer above.

Now set the angle to 180, one extreme of the range:

```
servo.angle = 180
```

Or to sweep back to the minimum 0 degree position:

```
servo.angle = 0
```

Often the range an individual servo recognizes varies a bit from other servos. If the servo didn't sweep the full expected range, then try adjusting min_pulse and max_pulse. Lower min_pulse until the servo stops moving or moves irregularly when angle is changed to 0. Raise max_pulse util the servo stops moving or moves irregularly when angle is change to the actuation range.

```
servo = adafruit_motor.servo.Servo(servo_channel, min_pulse=800, max_pulse=2200)
```

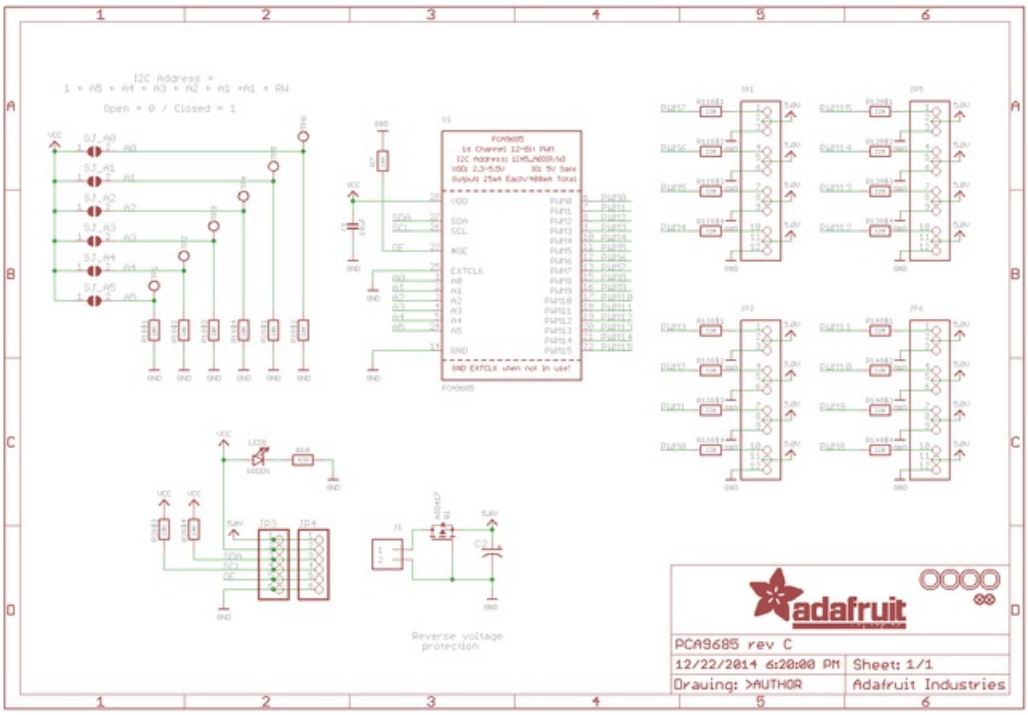
That's all there is to controlling servos with the PCA9685 and CircuitPython!  Using the **angle** attribute you can sweep and move servos in any way.  This is perfect for building robots, actuating switches, or other fun mechanical projects!

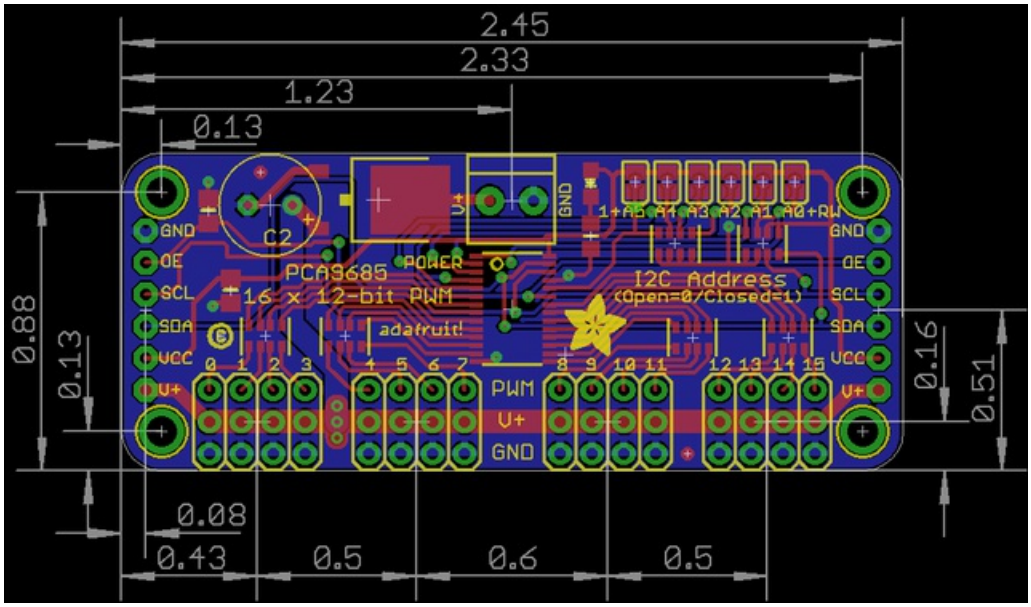
## Downloads

## Files

- PCA9685 datasheet
- Arduino driver library
- EagleCAD PCB files on GitHub
- Fritzing object in the Adafruit Fritzing library

## Schematic & Fabrication Print




Holes are 2.5mm diameter

## FAQ

**Can this board be used for LEDs or just servos?**

It can be used for LEDs as well as any other PWM-able device!

**I am having strange problems when combining this shield with the Adafruit LED Matrix/7Seg Backpacks**

The PCA9865 chip has an "All Call" address of 0x70. This is in addition to the configured address. Set the backpacks to address 0x71 or anything other than the default 0x70 to make the issue go away.

**With LEDs, how come I cant get the LEDs to turn completely off?**

If you want to turn the LEDs totally off use **setPWM(pin, 4096, 0);** not **setPWM(pin, 4095, 0);**