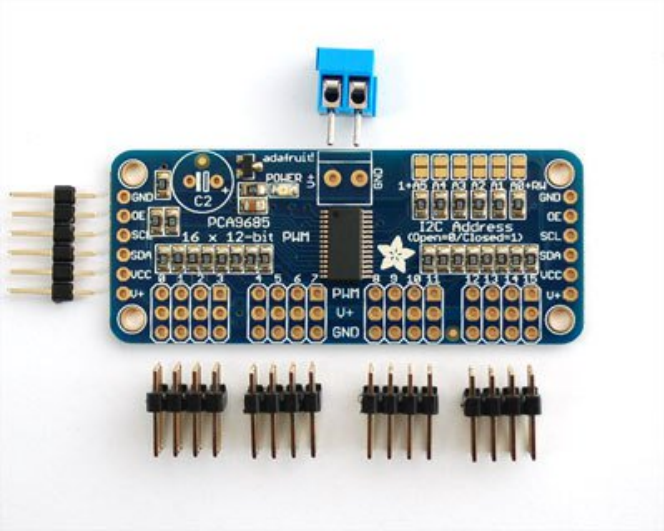
Adafruit 16-Channel Servo Driver with Arduino

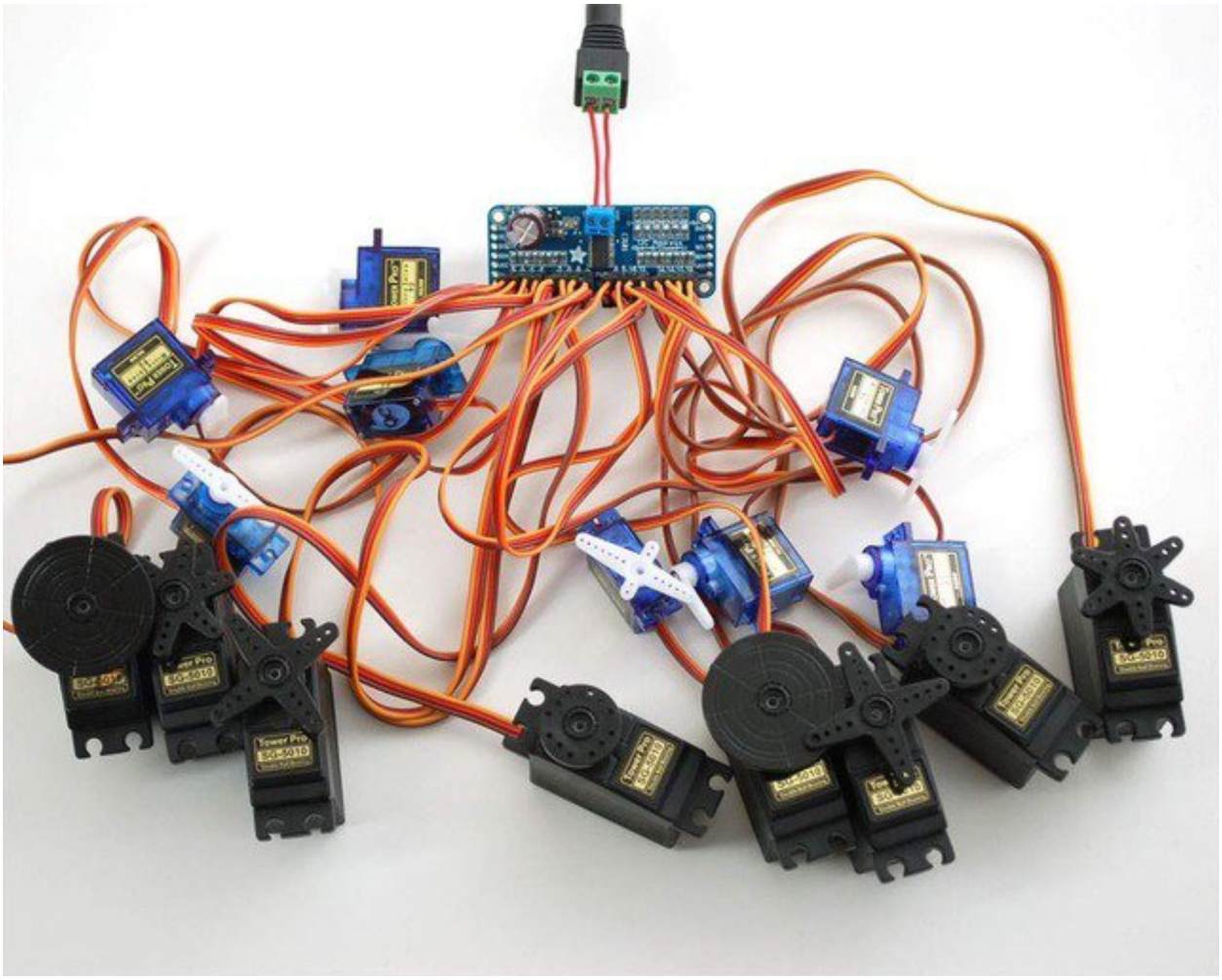
Created by Bill Earl



Last updated on 2016-10-07 04:59:20 PM UTC

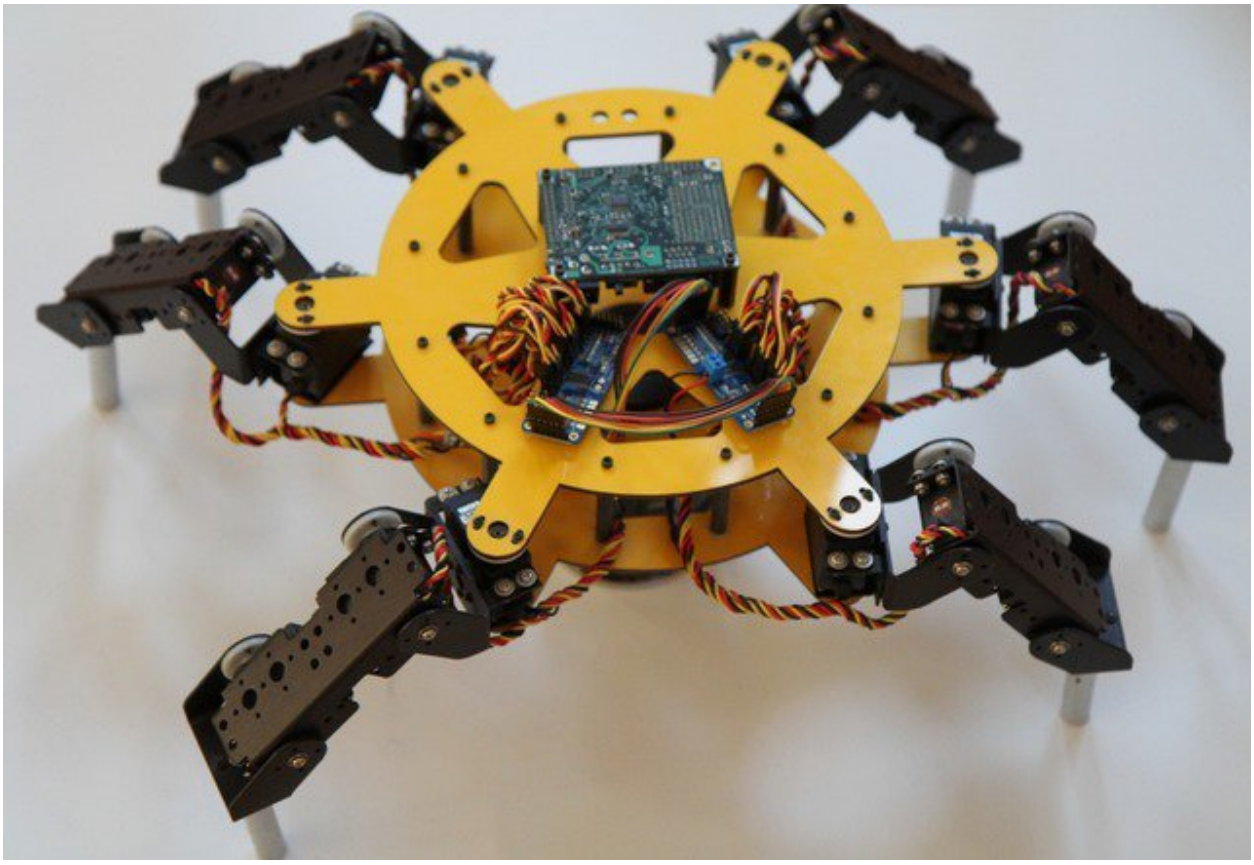
Guide Contents

Overview

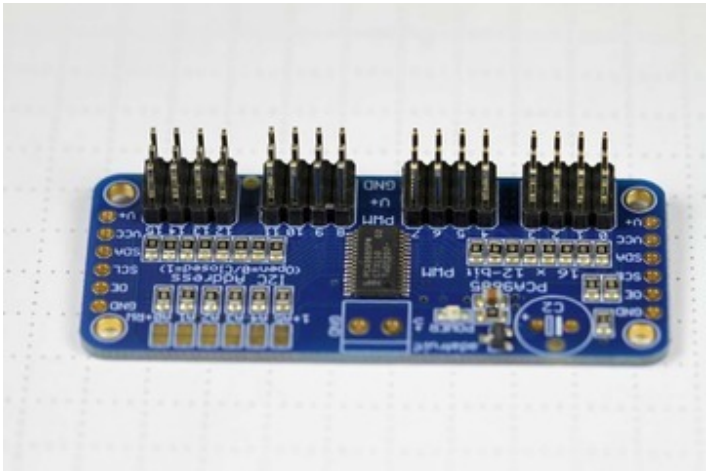


Driving servo motors with the Arduino Servo library is pretty easy, but each one consumes a precious pin - not to mention some Arduino processing power. The Adafruit 16-Channel 12-bit PWM/Servo Driver will drive up to 16 servos over I2C with only 2 pins. The on-board PWM controller will drive all 16 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.

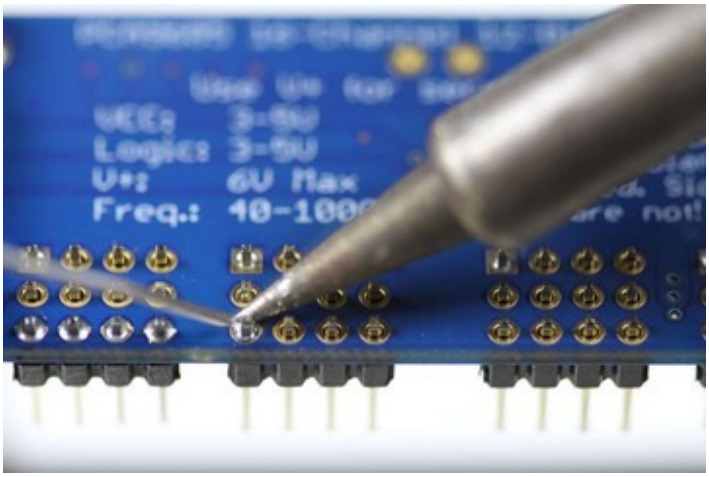


Assembly

- 

Install the Servo Headers

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

- 

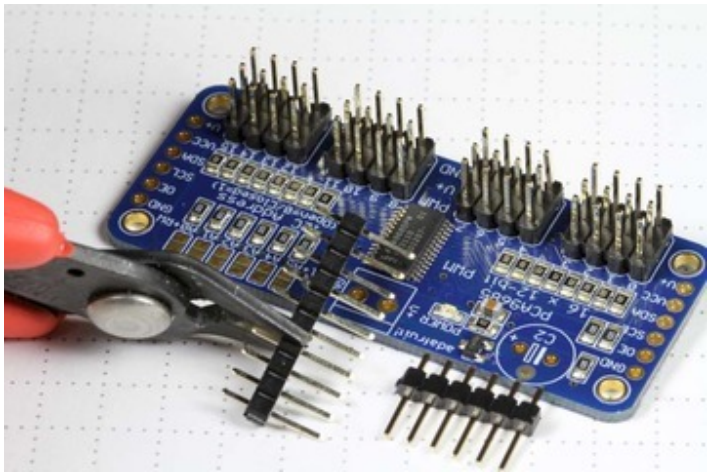
Solder all pins

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:

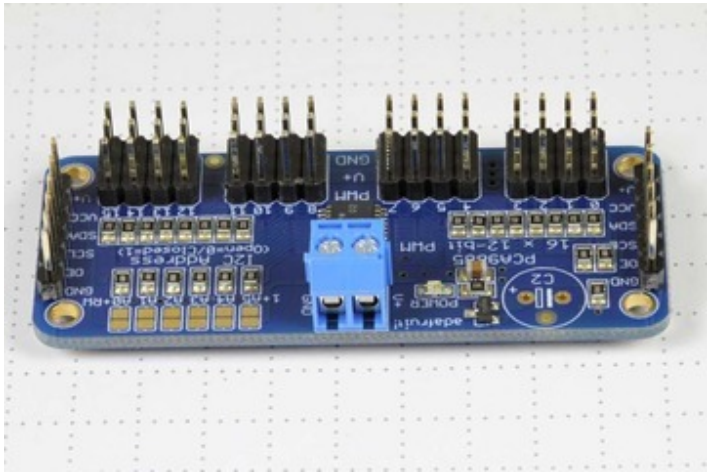
- **For [breadboard](http://adafru.it/239) use**, install headers on the

- 

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:

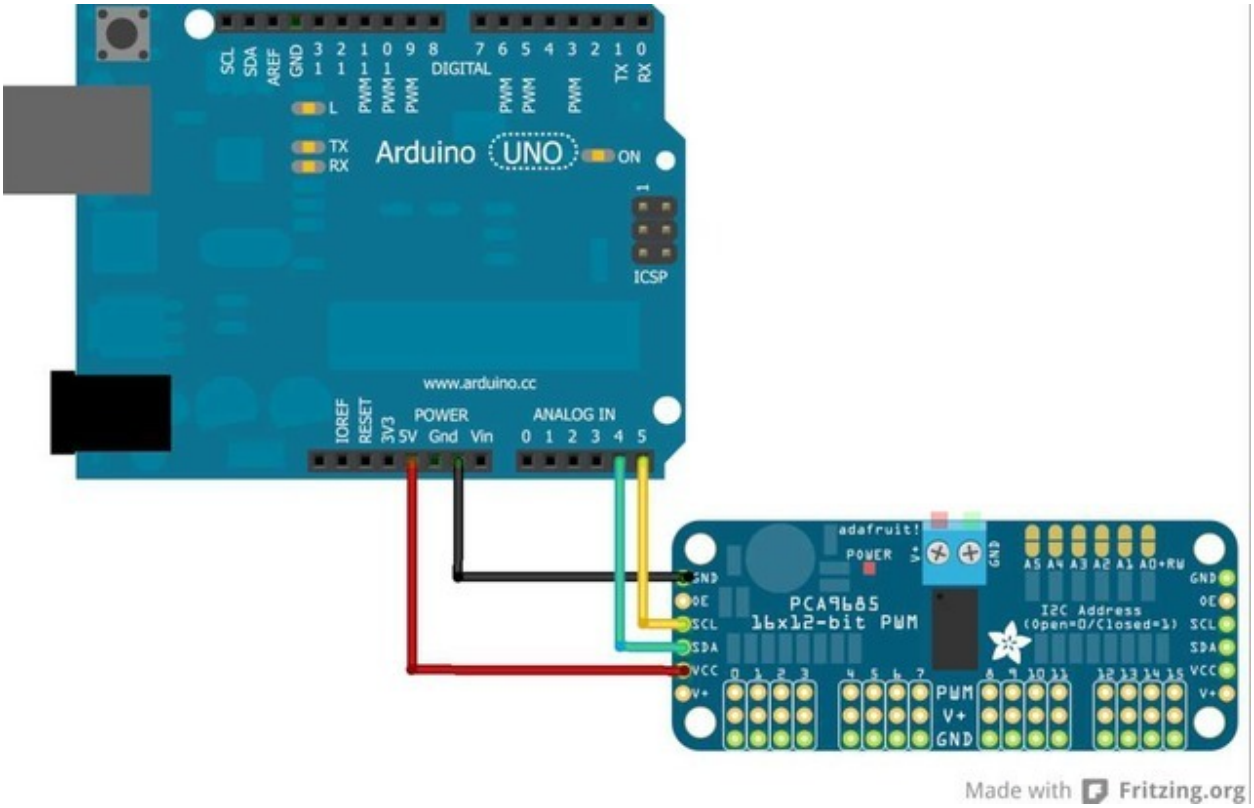
- +5v -> VCC (this is power for the BREAKOUT only, NOT the servo power!)
- GND -> GND
- Analog 4 -> SDA
- Analog 5 -> SCL

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.

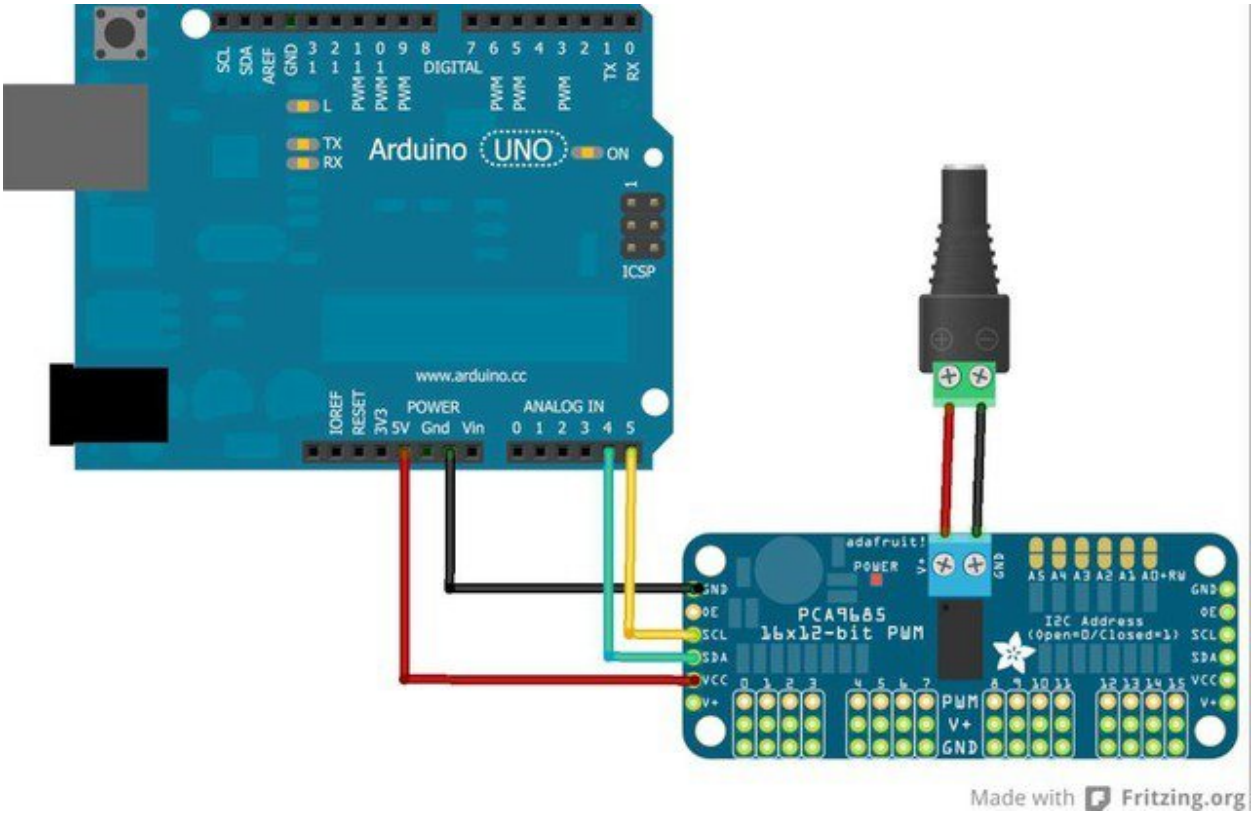
Power for the Servos

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 (http://adafru.it/276)
- 5v 10A switching power supply (http://adafru.it/658)
- 4xAA Battery Holder (http://adafru.it/830) - 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.

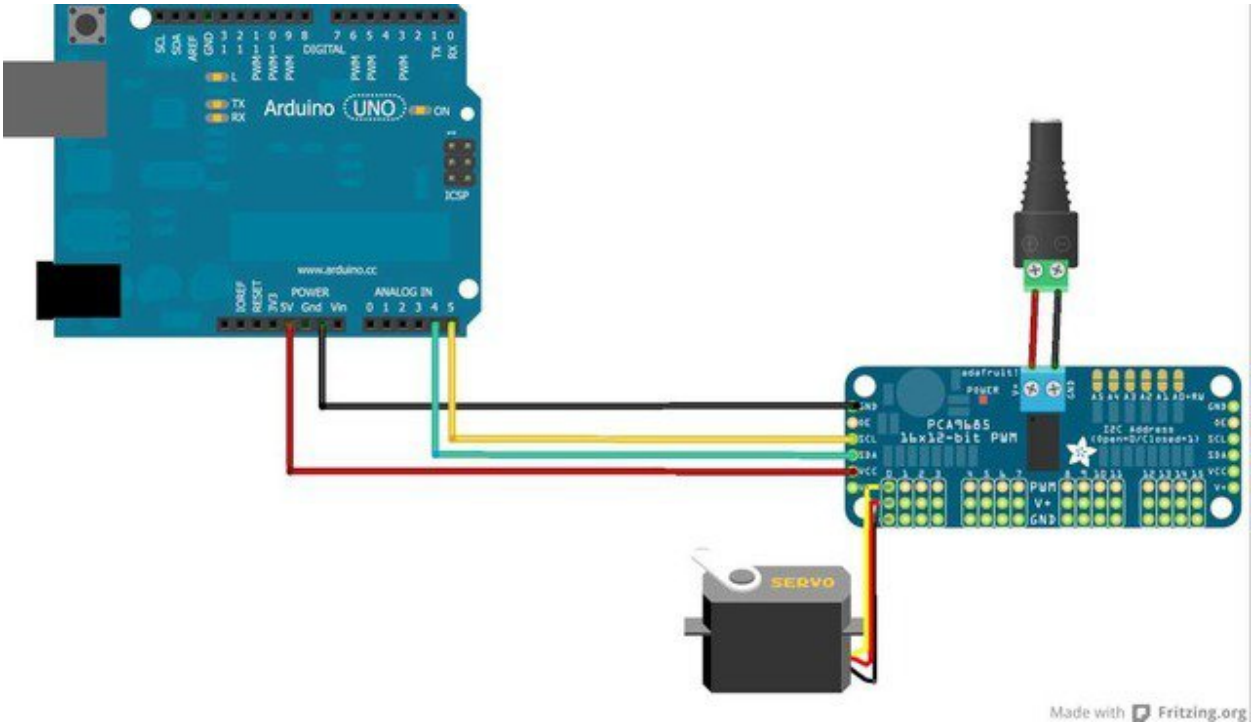


Adding a Capacitor to the thru-hole capacitor slot

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.

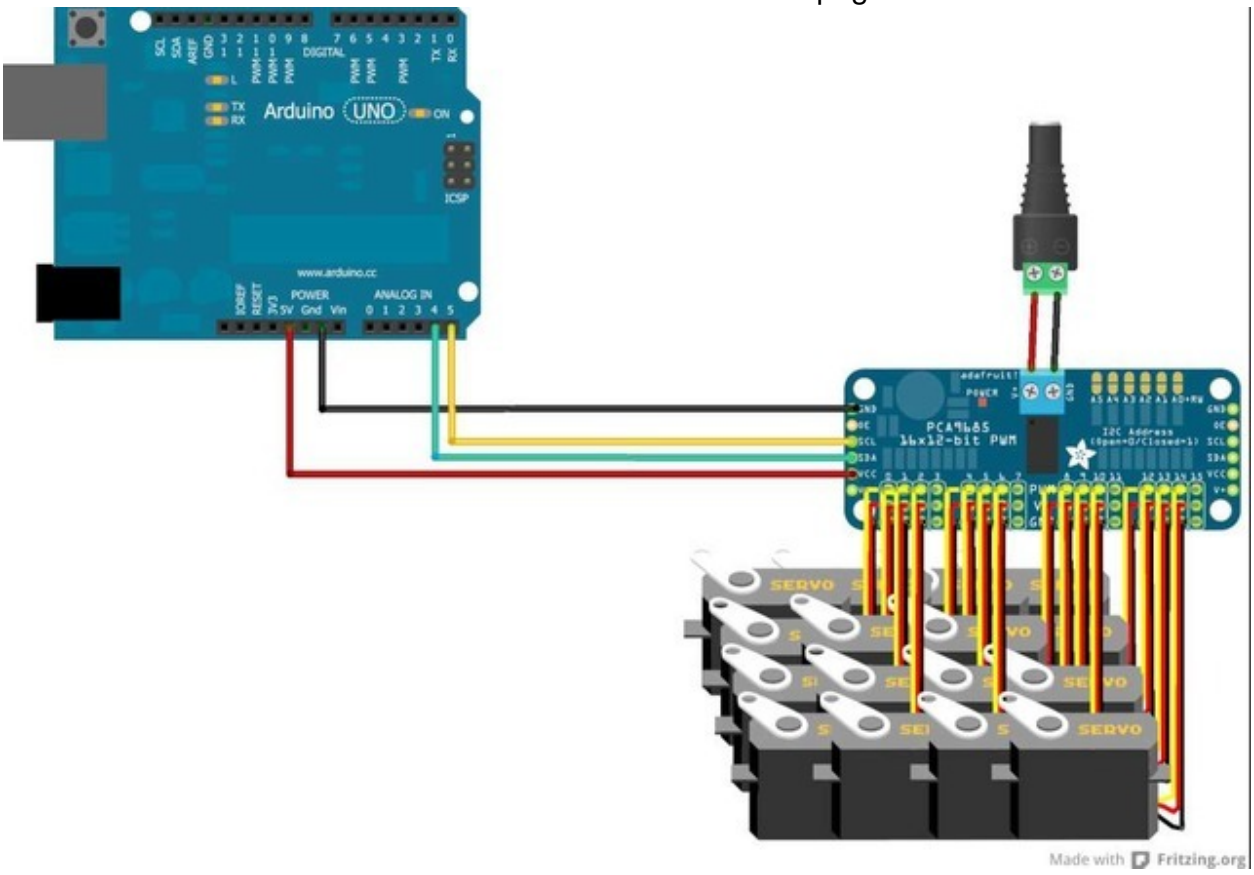
Connecting a Servo

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.



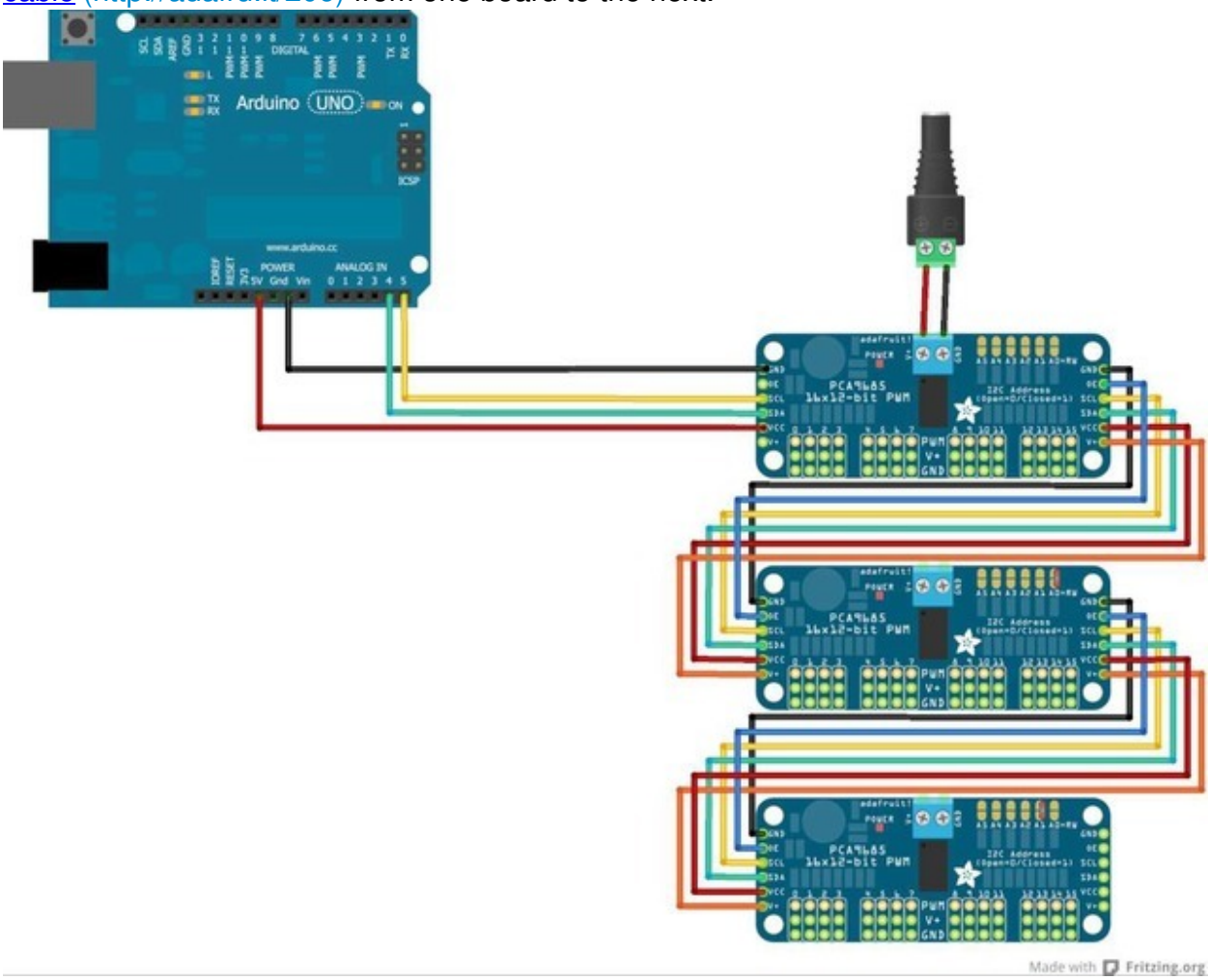
Adding More Servos

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

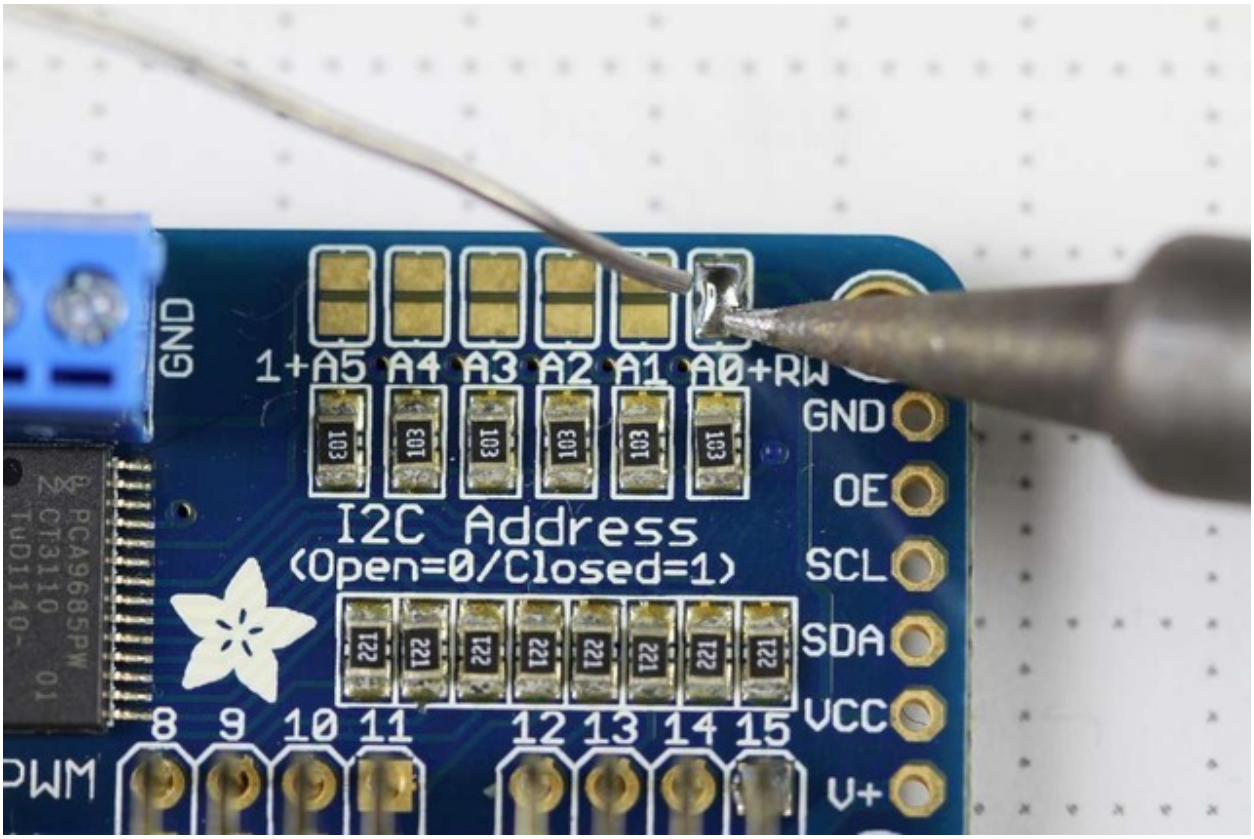
Multiple Drivers (up to 62) can be chained to control still more servos. With headers at both ends of the board, the wiring is as simple as connecting a [6-pin parallel cable (http://adafru.it/206)](http://adafru.it/206) from one board to the next.



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

Download the library from Github

Start by downloading the library from the GitHub repository (http://adafru.it/aQl) You can do that by visiting the github repo and manually downloading or, easier, just click this button to download the zip

Download Adafruit PWM/Servo Library
http://adafru.it/cDw

Rename the uncompressed folder **Adafruit_PWMServoDriver** and check that the **Adafruit_PWMServoDriver** folder contains **Adafruit_PWMServoDriver.cpp** and **Adafruit_PWMServoDriver.h**

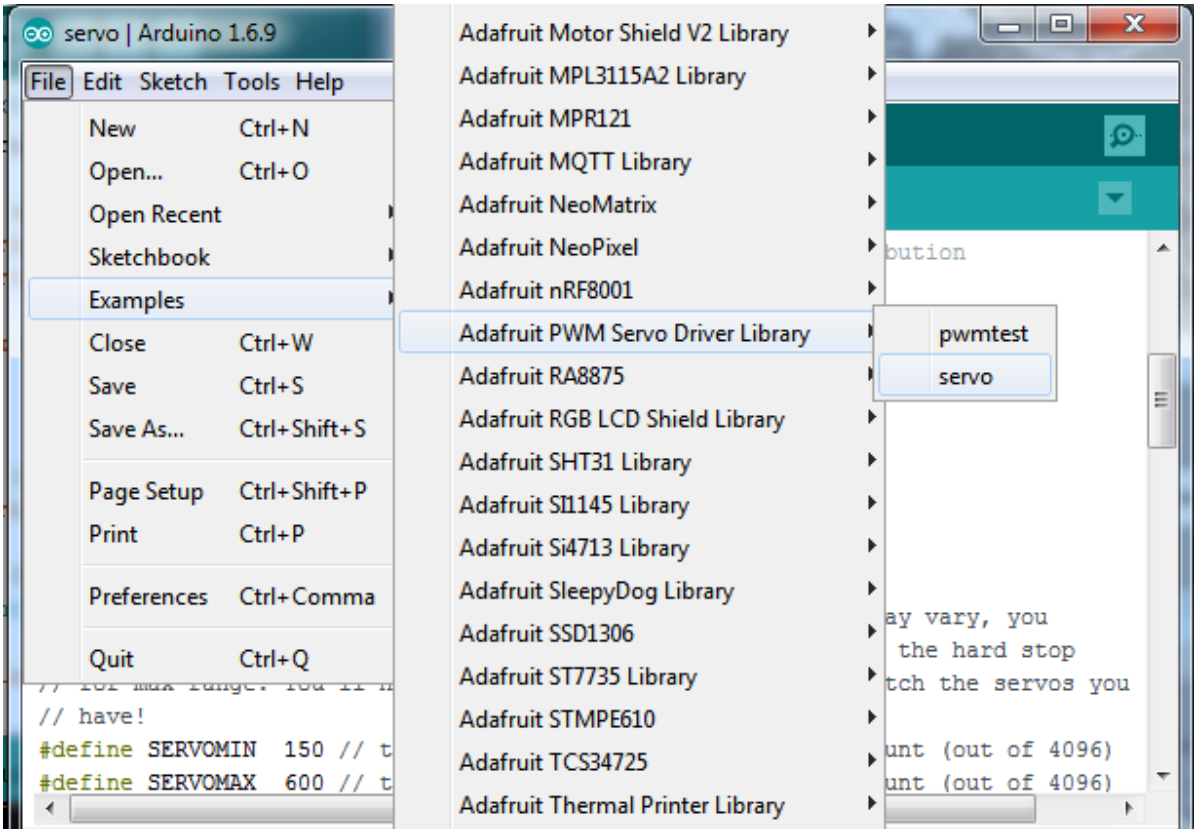
Place the **Adafruit_PWMServoDriver** library folder your **arduinosketchfolder/libraries/** folder. You may need to create the**libraries** subfolder if its your first library. Restart the IDE.

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

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 (http://adafru.it/aQm) 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 of. 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)
```

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
We are not sure why this occurs but there is an address collision even though the address are different! 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);**
I can't get this to work with my 7-segment LED display.

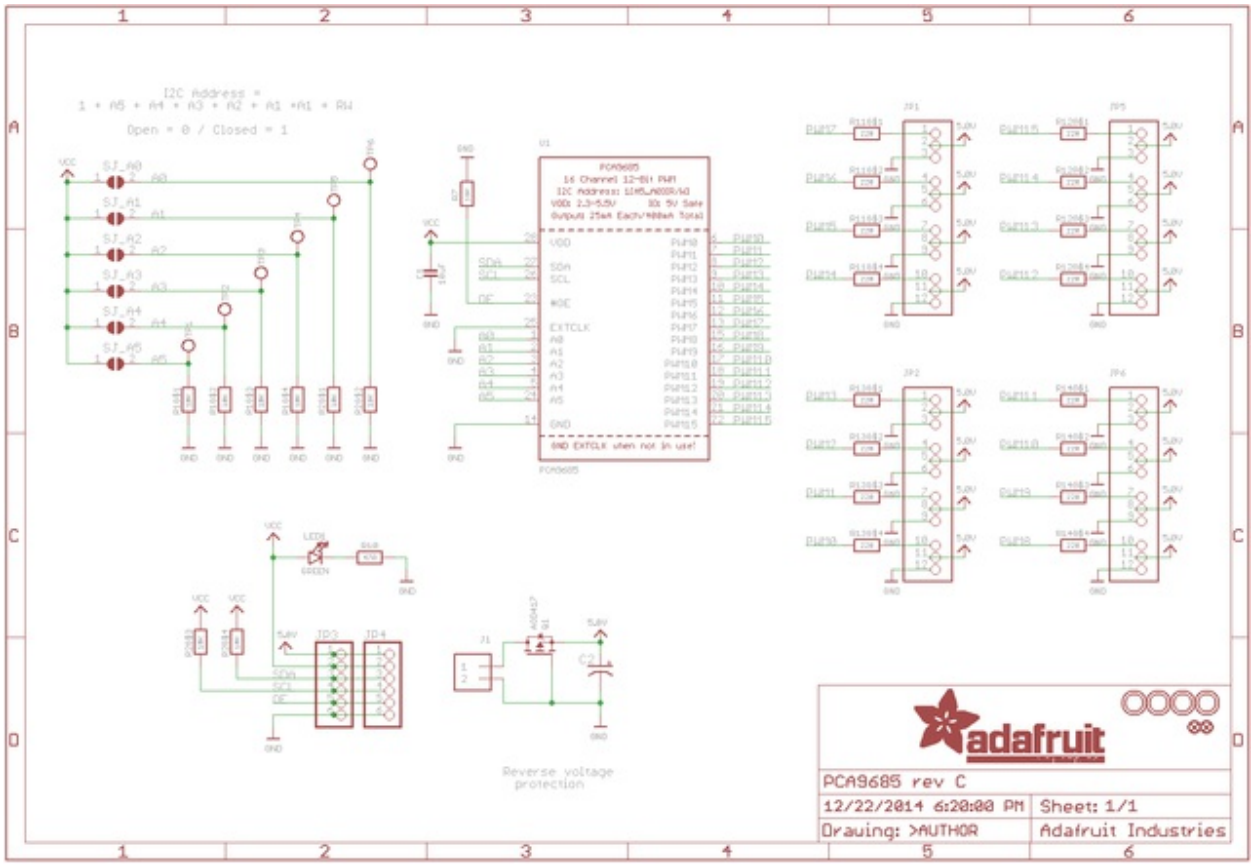
The PCA9865 chip has an "All Call" address of 0x70. This is in addition to the configured address. This will cause an address collision if used with any other chip addressed to 0x70.

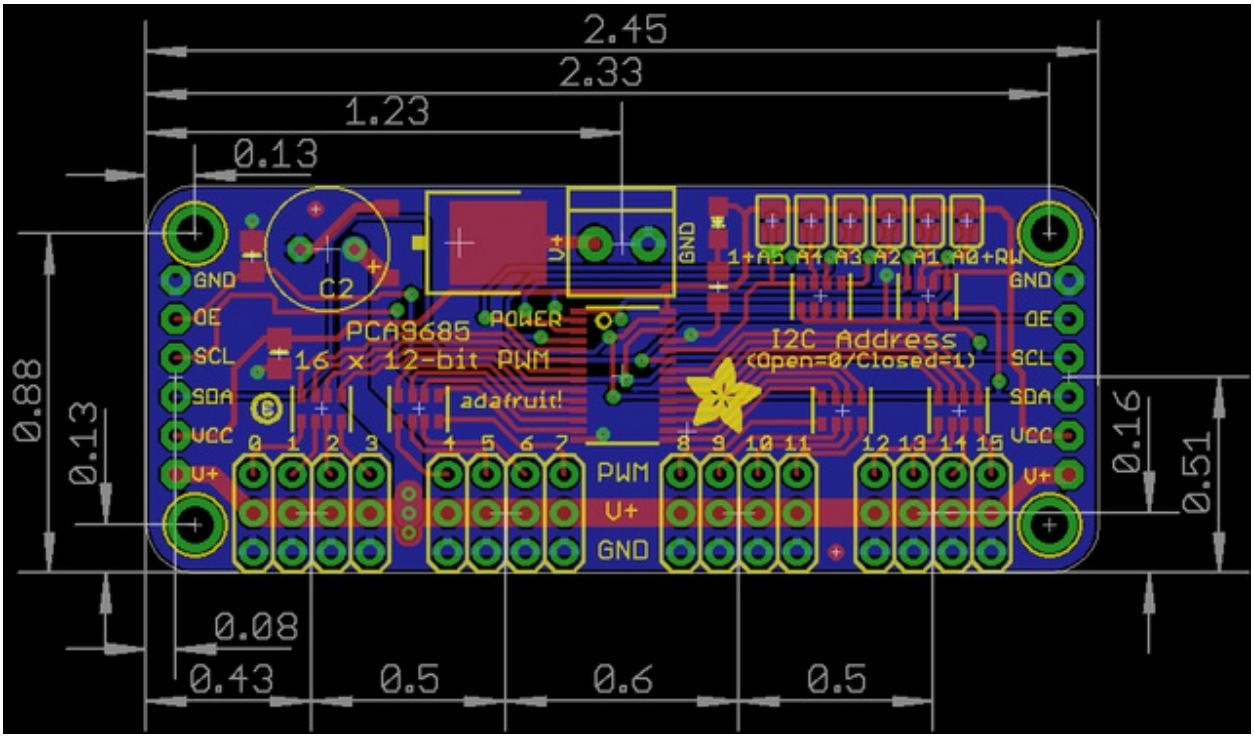
Downloads

Files

- PCA9685 datasheet (http://adafru.it/okB)
- Arduino driver library (http://adafru.it/aQl)
- EagleCAD PCB files on GitHub (http://adafru.it/rME)
- Fritzing object in the Adafruit Fritzing library (http://adafru.it/aP3)

Schematic & Fabrication Print




2.45
2.33
1.23
0.13
0.88
0.13
0.16
0.51
0.08
0.43
0.5
0.6
0.5
GND
OE
SCL
SDA
VCC
V+
C2
PCA9685
16 x 12-bit PWM
POWER
V+
GND
1+A5 A4 A3 A2 A1 A0+RW
I2C Address
(Open=0/Closed=1)
adafruit
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
PWM
V+
GND