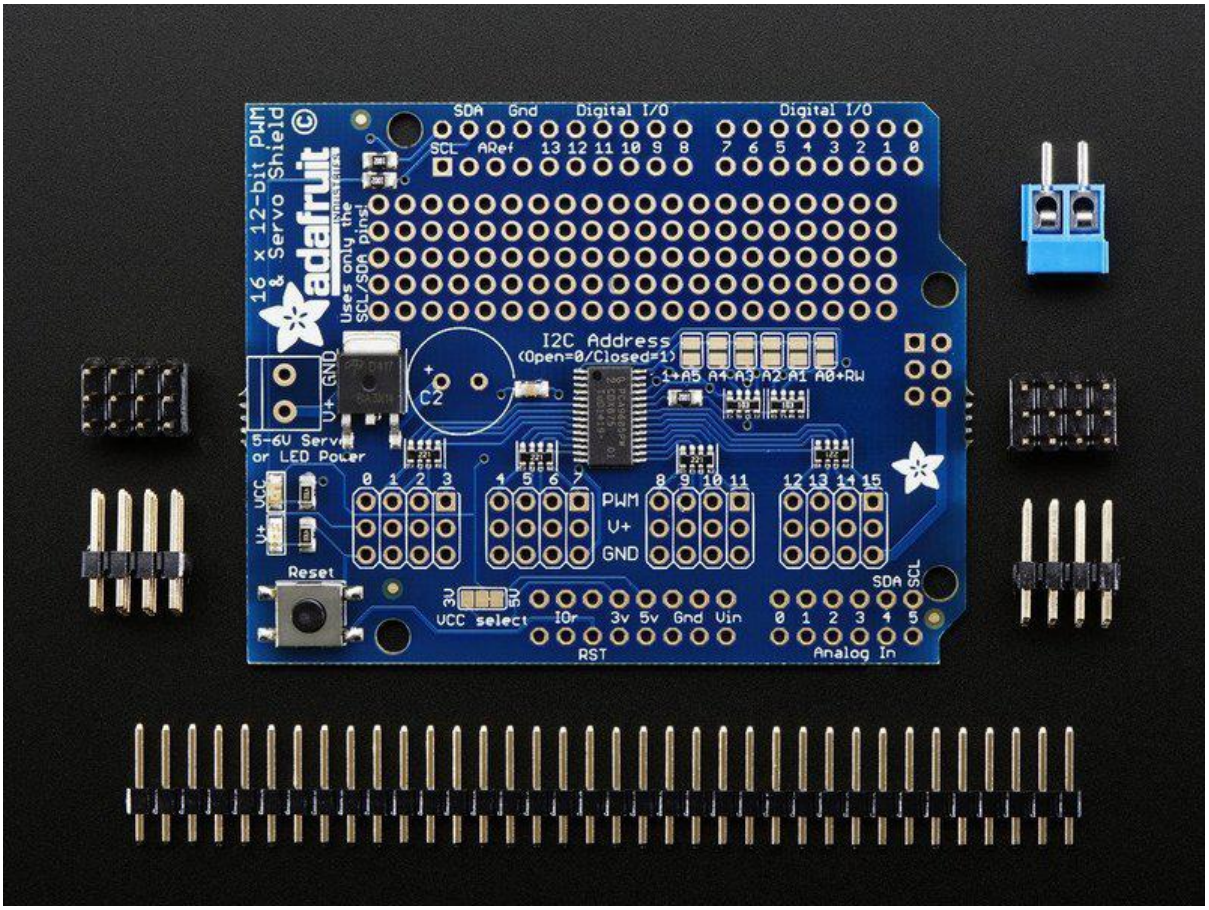
Adafruit 16-channel PWM/Servo Shield

Created by lady ada



https://learn.adafruit.com/adafruit-16-channel-pwm-slash-servo-shield

Last updated on 2023-08-29 02:12:13 PM EDT

Table of Contents

Overview 3

Assembly 4

Shield Connections 10

- Pins Used
- Connecting other I2C devices
- Powering Servos / PWM
- Adding a Capacitor to the thru-hole capacitor slot
- Connecting a Servo
- Adding More Servos

Stacking Shields 14

- Addressing the Shields

Using the Adafruit Library 16

- Install Adafruit PCA9685 library
- Test with the Example Code:
- Connect a Servo
- Calibrating your Servos
- Converting from Degrees to Pulse Length

Library Reference 19

- setPWMFreq(freq)
- Description
- setPWM(channel, on, off)
- Using as GPIO

CircuitPython Usage 21

- CircuitPython Microcontroller Wiring
- CircuitPython Installation of ServoKit and Necessary Libraries
- CircuitPython Usage
- Dimming LEDs
- Controlling Servos
- Standard Servos
- Continuous Rotation Servos
- Full Example Code

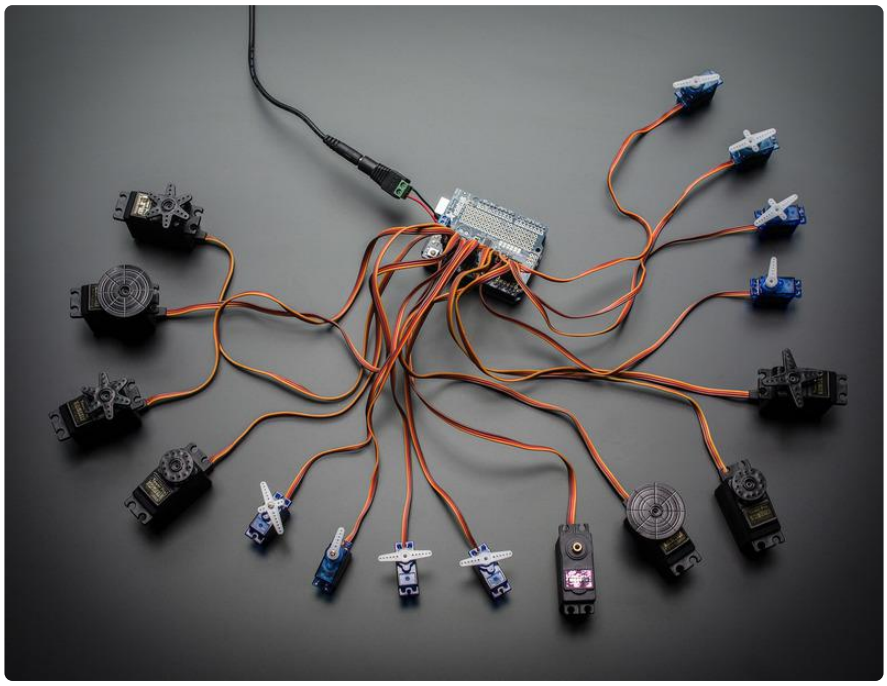
Python Docs 27

FAQ 27

Downloads & Links 28

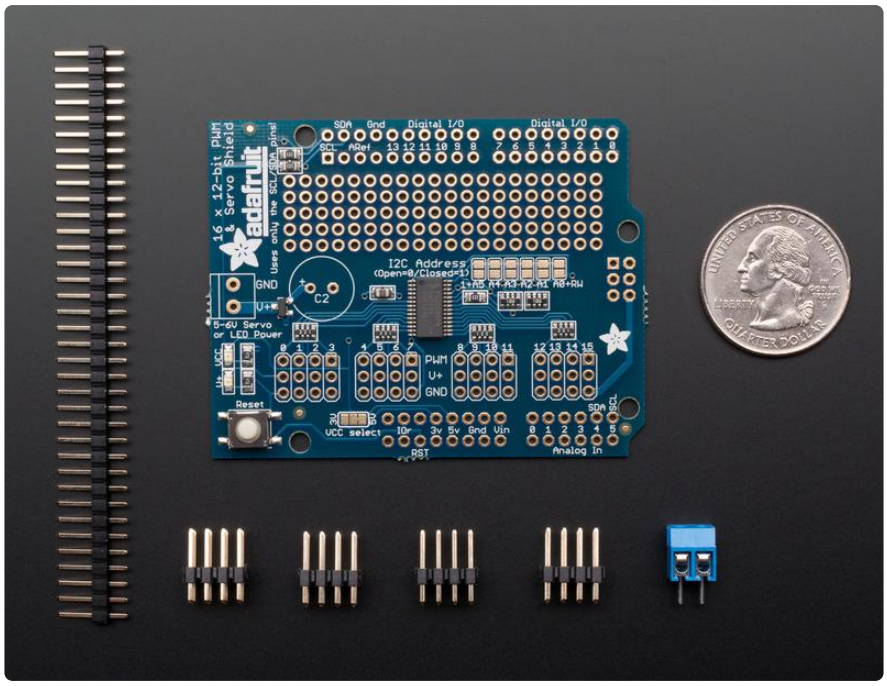
- Files
- Schematic
- Fabrication Print

Overview



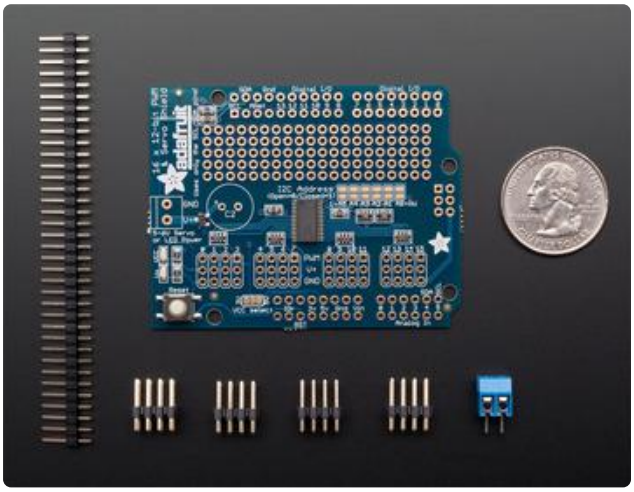
Driving servo motors with the Arduino Servo library is pretty easy, but each one consumes a precious pin - not to mention some Arduino processing power. The Adafruit 16-Channel 12-bit PWM/Servo Driver Shield will drive up to 16 servos over I2C with only 2 pins. The on-board PWM controller will drive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



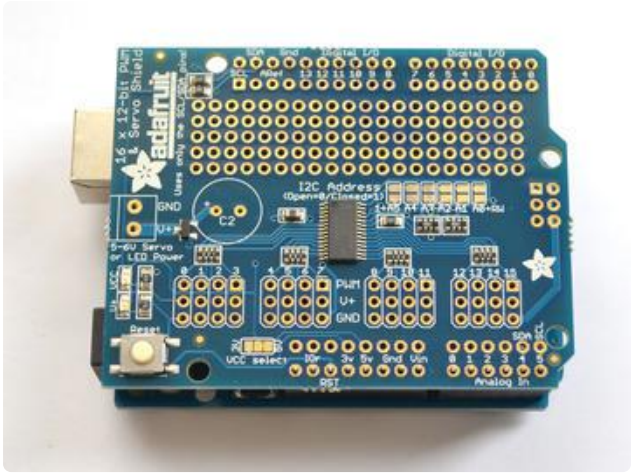
Assembly

This section assumes you do not want to stack the shield with stacking headers. If you want to stack another shield on top, skip to the Stacking Shields section!

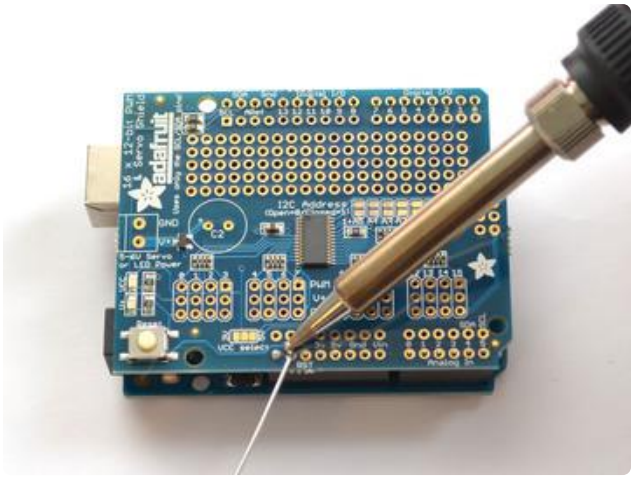


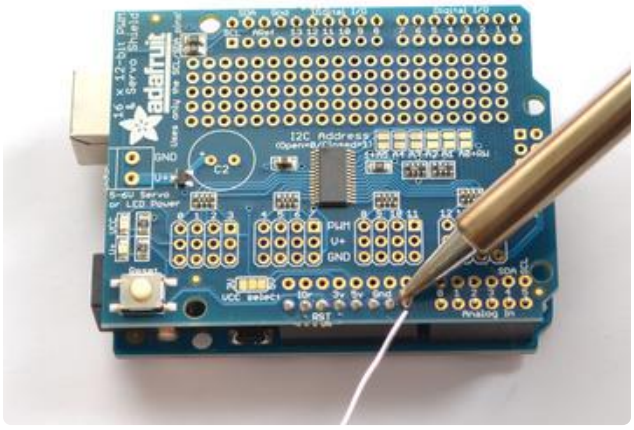
Check you have everything you need: assembled shield PCB, 0.1" male header, 4 of 3x4 male header, and a 2 pin terminal block

Break apart the 0.1" header into 6, 8 and/or 10-pin long pieces and slip the long ends into the headers of your Arduino

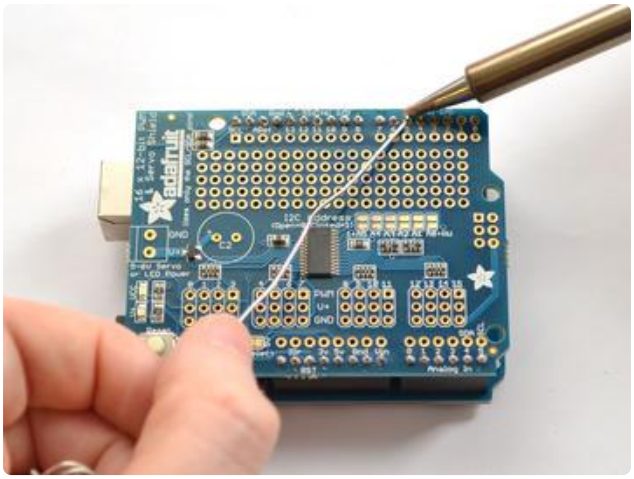


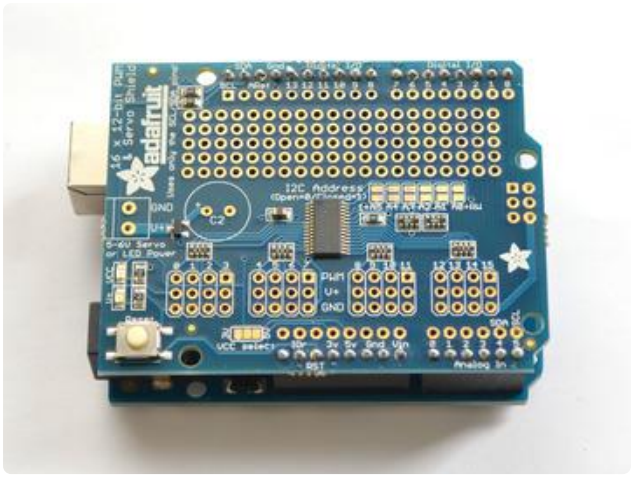
Place the shield on top of the header pins, they should fit into each of the holes along the edge

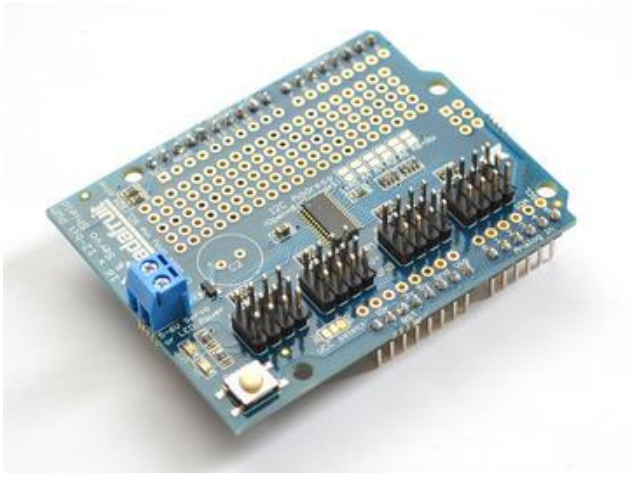




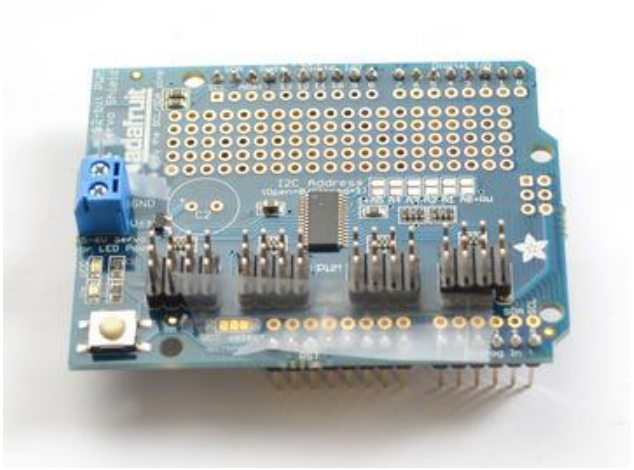
Solder each of the pins to secure the shield to the headers



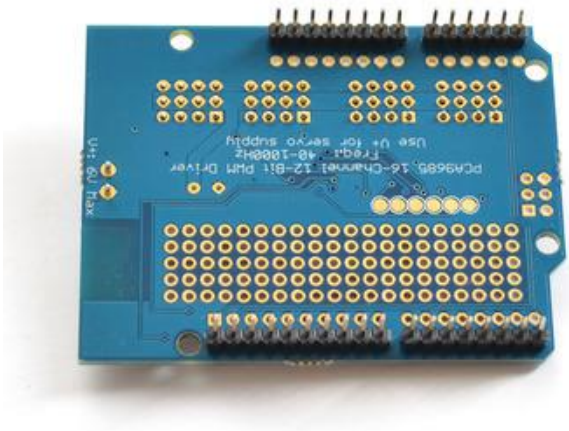


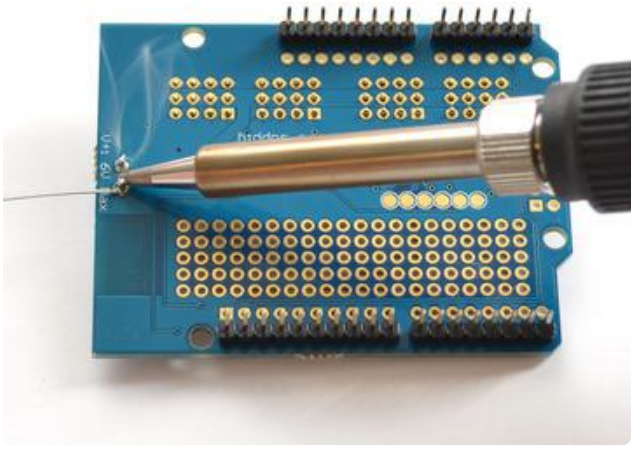


Place the 2 pin terminal block so it faces out. Also place the 3x4 headers so the short pins are plugged into the shield and the long pins are sticking up

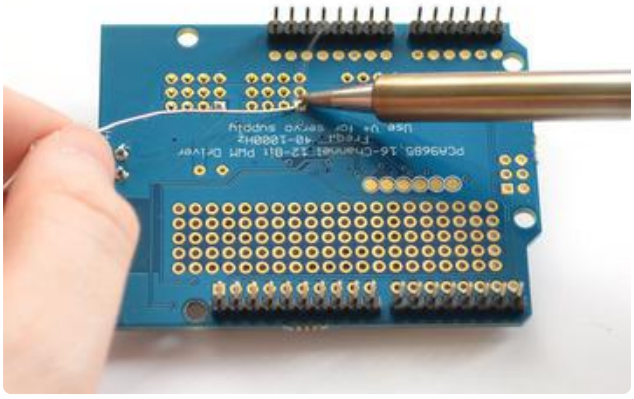


To keep them in place, a few pieces of tape will hold them for when you flip the board over.

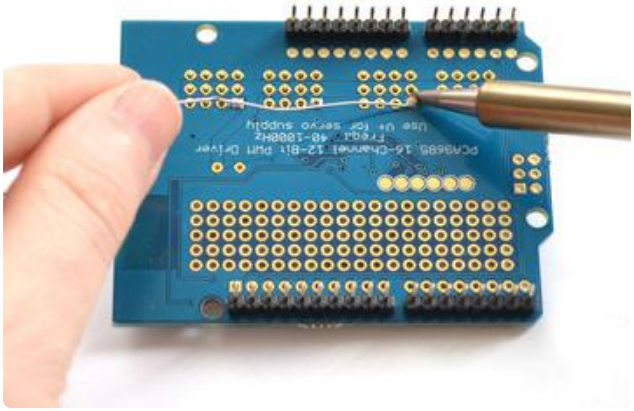


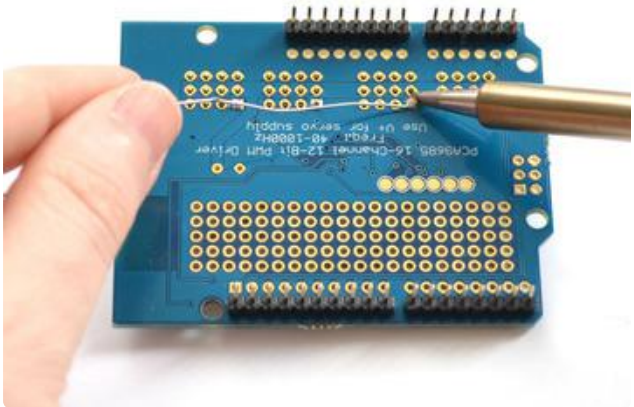


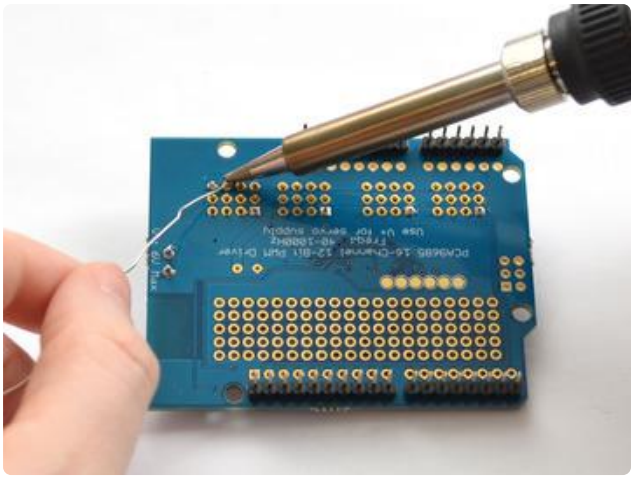
Flip the board over and solder the terminal block



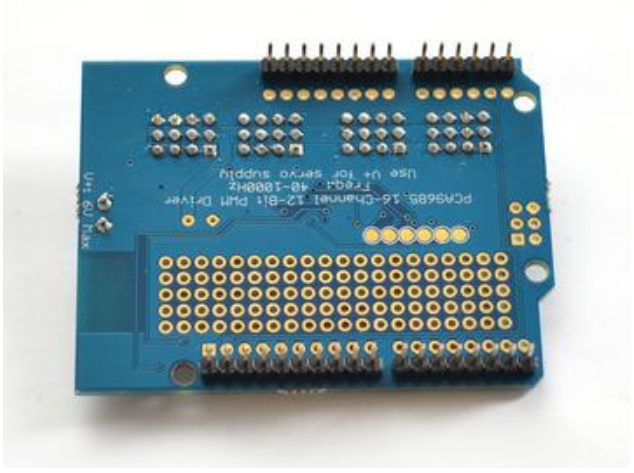
Then solder one pin of each 3x4 header to tack them in place.

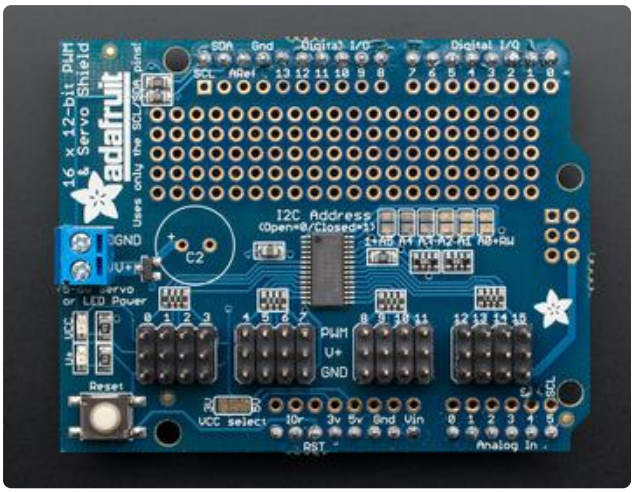






Finally...solder every pin of the 3x4 headers





You're done! Go onto the next sections to learn how to use your servo/pwm shield

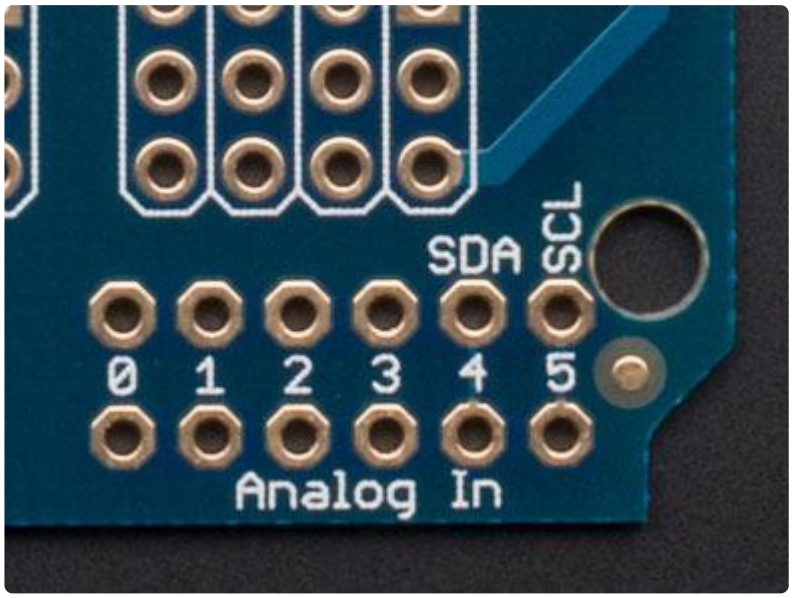
Shield Connections

Pins Used

The shield plugs in directly into any shield-compatible Arduino such as Duemilanove, Diecimila, UNO, Leonardo, Mega R3+, ADK R3+. The only pins required to run are the Ground, 5V and SDA + SCL I2C control pins.

For backwards compatibility with old Ardunos, SCL is connected to A5 and SDA is connected to A4. UNOs already have this connection on board. If you are using a Leonardo or Mega and want to use the A4/A5 pins, cut the traces on the top of the board between A4 and A5 and the two pins next to them labeled SCL/SDA.

If you are using a Mega or ADK R2 or earlier, you will have to solder a wire from SCL t o D21 and SDA to D20



Connecting other I2C devices

Since I2C is a 'shared bus' you can still connect other I2C devices to the SDA/SCL pins as long as they do not have a conflicting address. The default address for the shield is address 0x40

Powering Servos / PWM

This shield has two power supplies. One is VCC - that is the 5V power from the Arduino, it is used to power the PWM chip and determines the I2C logic level and the PWM signal logic level. When this power supply is working you will see a red LED. Th e red LED must be lit for the Arduino & shield to work! Plug in the Arduino to USB or a wall adapter to provide it.

To power servos you will need to also connect the V+ power supply - this is the power supply for the servos. (If you are lighting up single LEDs you may not need this power supply.) This power supply should be 5 or 6VDC. You can connect this power through the blue terminal block. There is reverse-polarity protection in case you hook up

power backwards.

Nearly all 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.

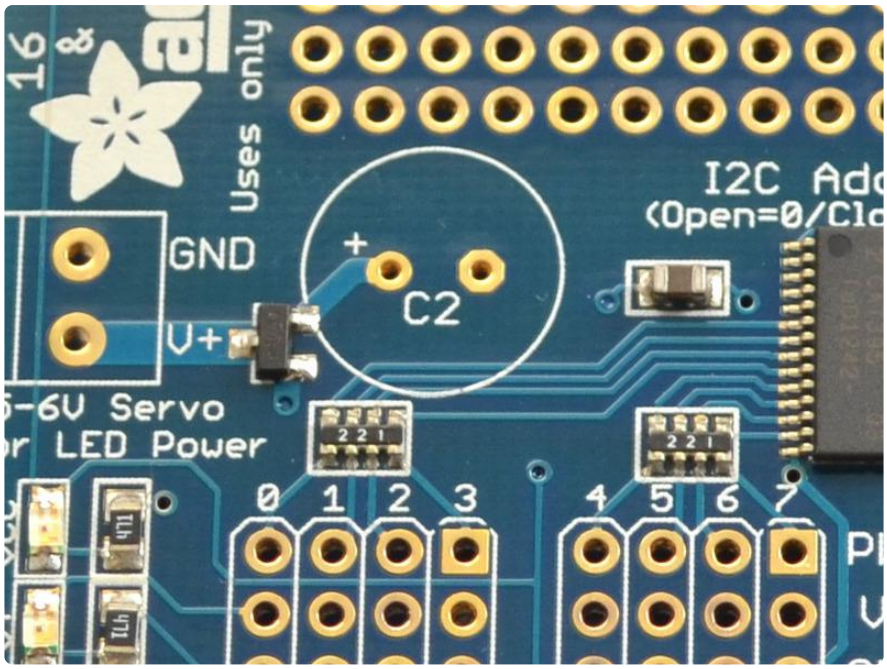
Good power choices are:

- 5v 2A switching power supply (http://adafru.it/276) (up to perhaps 4 servos)
- 5v 10A switching power supply (http://adafru.it/658) (up to perhaps 16 servos)
- 4xAA Battery Holder (http://adafru.it/830) - 6v with Alkaline cells. 4.8v with NiMH rechargeable cells, portable!
- 4.8 or 6v Rechargeable RC battery packs from a hobby store.

SERVOS CAN USE A LOT OF POWER! 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.

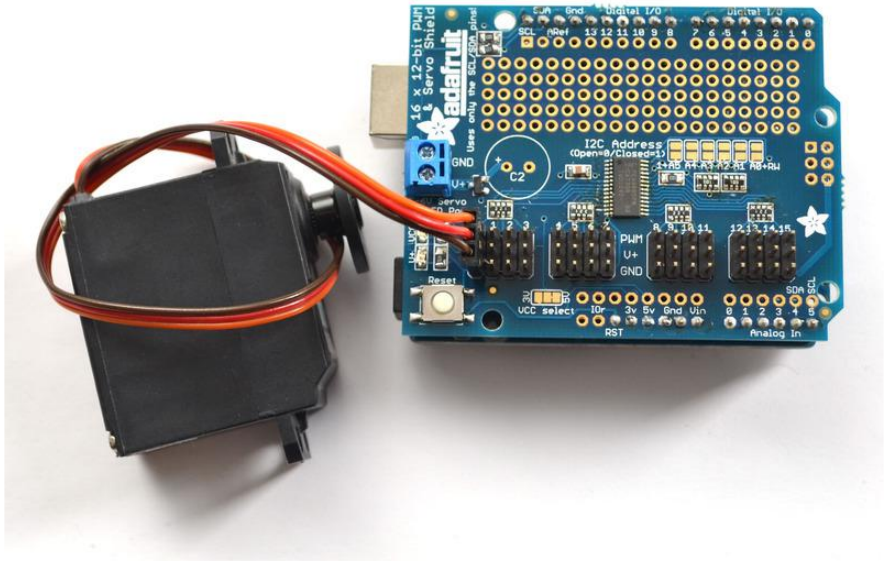
Adding a Capacitor to the thru-hole capacitor slot

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.



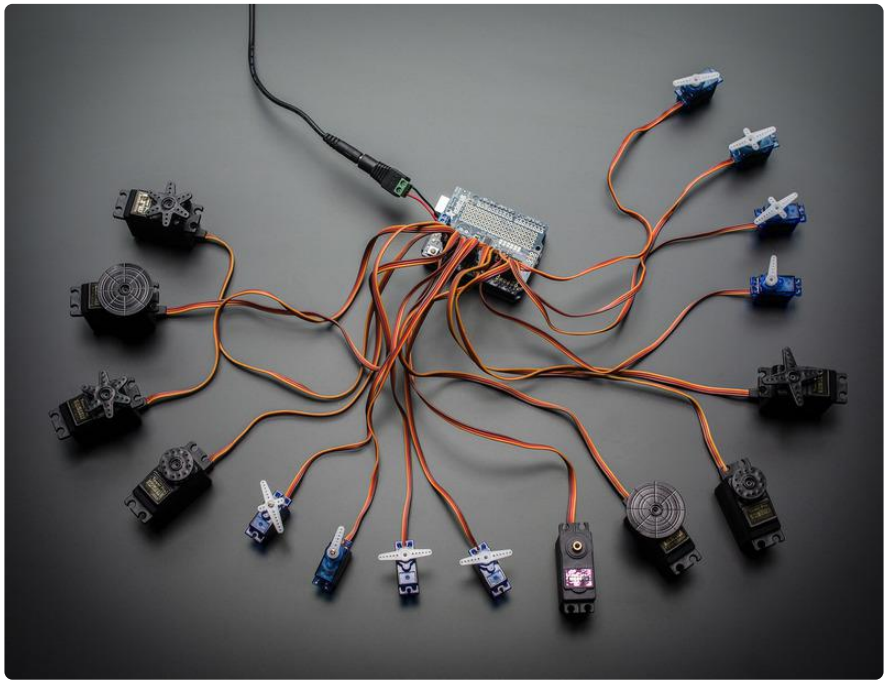
Connecting a Servo

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.



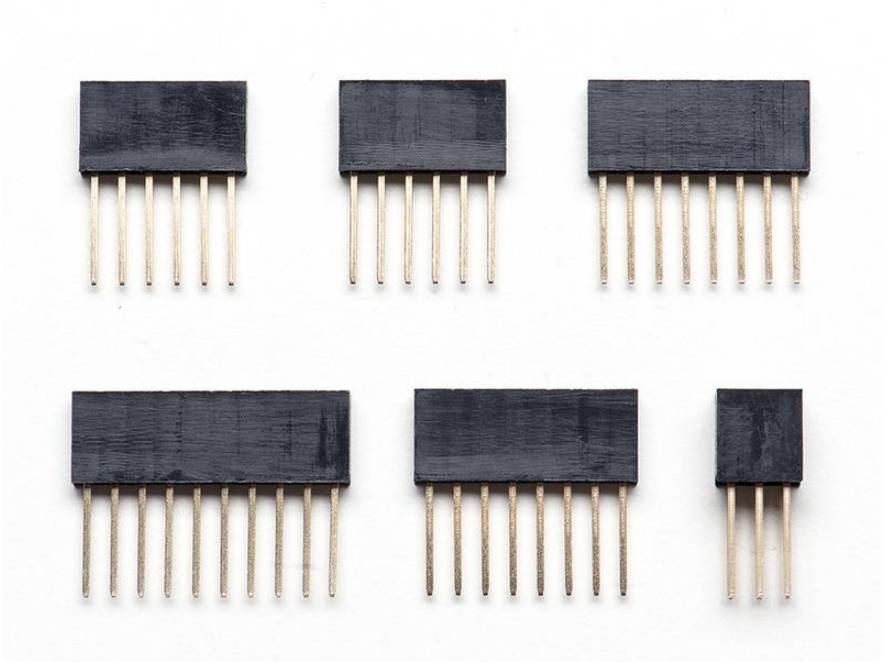
Adding More Servos

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



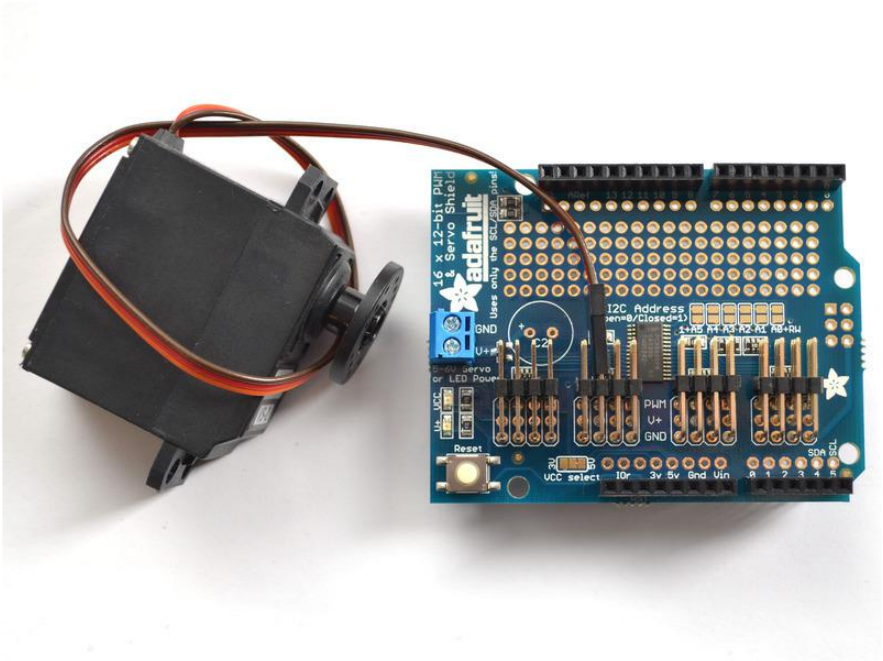
Stacking Shields

If you want to plug shields on top of this one, make sure you pick up a set of shield-stacking headers
http://www.adafruit.com/product/85 ()
and solder them instead of the male header



Since this shield only uses the I2C pins and the I2C bus is sharable, you can stack multiple shields on top of each other. You will need to have installed stacking headers & right angle 3x4 connections for it to physically connect. Multiple shields (up to 62!) can be stacked to control still more servos.

You may have difficulty with the 'left-most' header since we updated the transistor to be beefier. Use the 3 ports to the right or solder wires out to get mechanical access

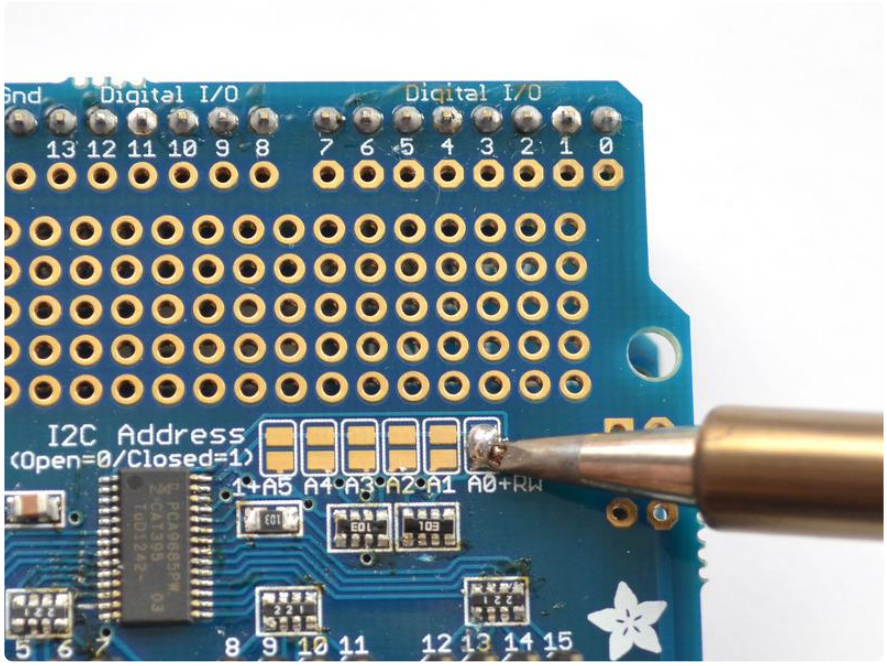


Addressing the Shields

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.

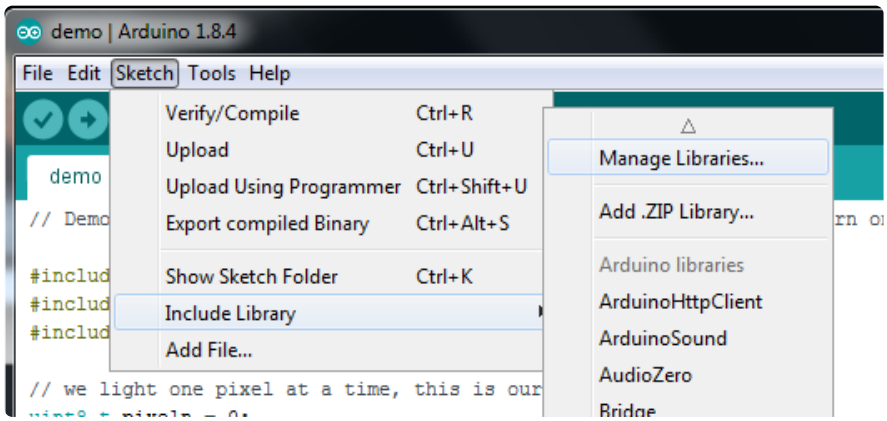
Using the Adafruit Library

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

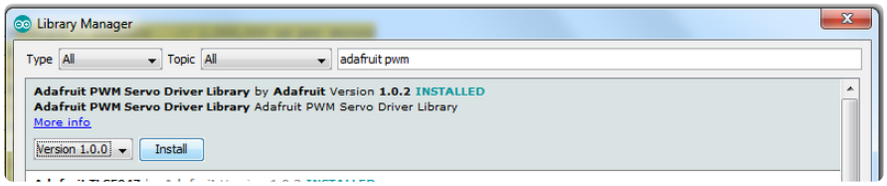
Install Adafruit PCA9685 library

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

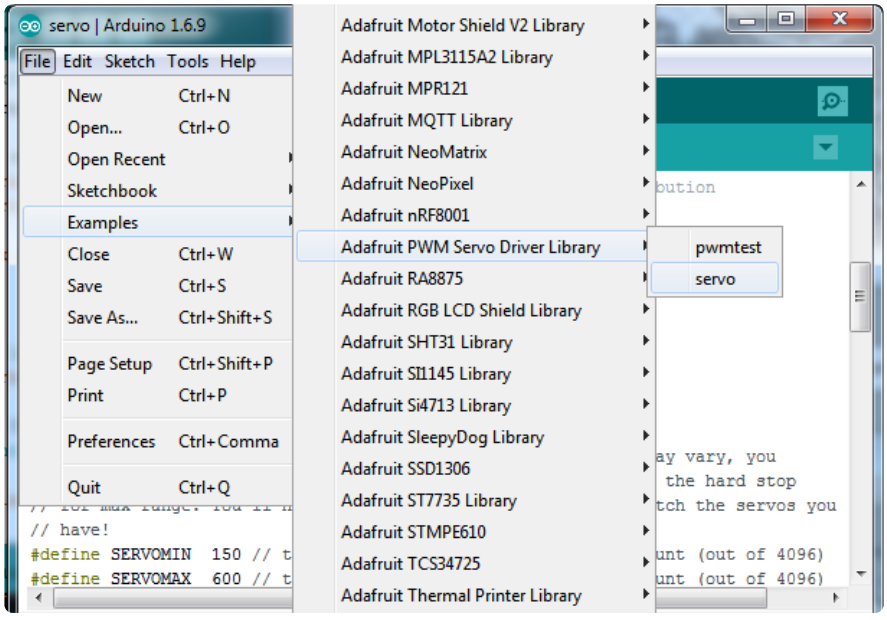


We also have a great tutorial on Arduino library installation at:
http://learn.adafruit.com/adafruit-all-about-arduino-libraries-install-use ()

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 1600

Example

The following code will set the PWM frequency to 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);`

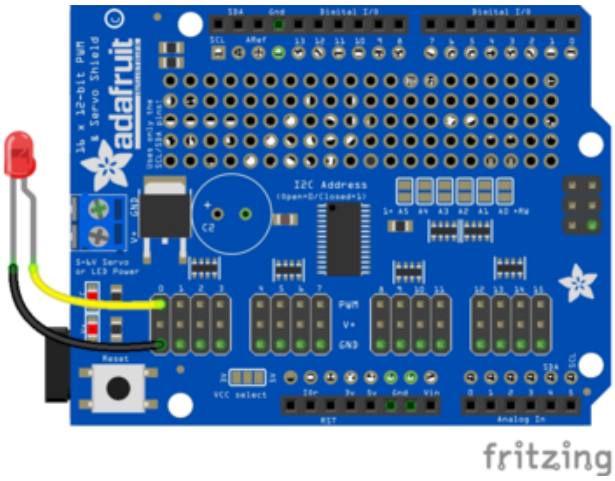
CircuitPython Usage

It's easy to control PWM or servos with the Adafruit 16-channel PWM/Servo Shield. There are multiple CircuitPython libraries available to work with the different features of this board including Adafruit CircuitPython PCA9685 (), and Adafruit CircuitPython ServoKit (). These libraries make it easy to write Python code to control PWM and servo motors.

CircuitPython Microcontroller Wiring

First assemble the shield exactly as shown in the previous pages. There's no wiring needed to connect the shield to the Metro. The example below shows the shield attached to a Metro.

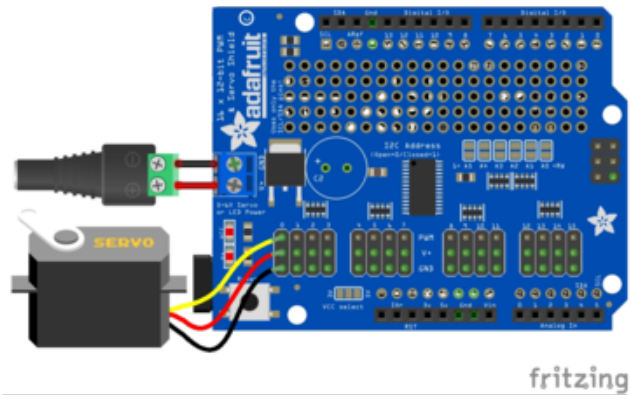
To dim an LED, wire it to the board as follows. Note: you don't need to use a resistor to limit current through the LED as the shield will limit the current to around 10mA.



Connect LED cathode / shorter leg to shield channel GND / ground.
Connect LED anode / longer leg to shield channel PWM.
External power is not necessary to PWM an LED.

To control a servo, wire it to the board as follows, including a barrel jack to the power terminal to attach an appropriate external power source to the shield. The shield will not power servos without an external power source!

Check your servo datasheet to be certain.



Typically:

Connect servo brown wire to the shield channel GND
Connect servo red wire to the shield channel V+
Connect servo yellow wire to the shield channel PWM
Connect the positive side of the power terminal to the positive side of the barrel jack.
Connect the negative side of the power terminal to the negative side of the barrel jack.
You must have an external power source to run servos!

CircuitPython Installation of ServoKit and Necessary Libraries

You'll need to install a few libraries on your Metro board.

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--carefully follow the steps to find and install these libraries from Adafruit's CircuitPython library bundle (). Our CircuitPython starter guide has a great page on how to install the library bundle ().

If you choose, you can manually install the libraries individually on your board:

- adafruit_pca9685
- adafruit_bus_device
- adafruit_register
- adafruit_motor

- adafruit_servokit

Before continuing make sure your board's lib folder or root filesystem has the adafruit_pca9685.mpy, adafruit_register, adafruit_motor, adafruit_bus_device and adafruit_servokit files and folders copied over.

Next connect to the board's serial REPL () so you are at the CircuitPython >>> prompt.

CircuitPython Usage

To demonstrate the usage, we'll use Python code to control PWM to dim an LED and to control servo motors from the board's Python REPL.

Dimming LEDs

This shield uses the PCA9685. Each channel of the shield 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 you'll need to import the necessary modules, initialize the I2C bus for your board, and create an instance of the class.

```
import board
import busio
import adafruit_pca9685
i2c = busio.I2C(board.SCL, board.SDA)
shield = 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:

```
shield.frequency = 60
```

The shield 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 = shield.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` (65535), 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.

That's all there is to dimming LEDs using CircuitPython and the PWM/Servo Shield!

Controlling Servos

We've written a handy CircuitPython library for the various PWM/Servo kits called Ada fruit CircuitPython ServoKit () that handles all the complicated setup for you. All you need to do is import the appropriate class from the library, and then all the features of that class are available for use. We're going to show you how to import the `ServoKit` class and use it to control servo motors with the Adafruit PWM/Servo Shield.

First you'll need to import and initialize the `ServoKit` class. You must specify the number of channels available on your board. The shield has 16 channels, so when you create the class object, you will specify `16`.

```
from adafruit_servokit import ServoKit
kit = ServoKit(channels=16)
```

Now you're ready to control both standard and continuous rotation servos.

Standard Servos

To control a standard servo, you need to specify the channel the servo is connected to. You can then control movement by setting `angle` to the number of degrees.

For example to move the servo connected to channel `0` to `180` degrees:

```
kit.servo[0].angle = 180
```

To return the servo to `0` degrees:

```
kit.servo[0].angle = 0
```

With a standard servo, you specify the position as an angle. The angle will always be between 0 and the actuation range. The default is 180 degrees but your servo may have a smaller sweep. You can change the total angle by setting `actuation_range`.

For example, to set the actuation range to 160 degrees:

```
kit.servo[0].actuation_range = 160
```

Often the range an individual servo recognises varies a bit from other servos. If the servo didn't sweep the full expected range, then try adjusting the minimum and maximum pulse widths using `set_pulse_width_range(min_pulse, max_pulse)`.

To set the pulse width range to a minimum of 1000 and a maximum of 2000:

```
kit.servo[0].set_pulse_width_range(1000, 2000)
```

That's all there is to controlling standard servos with the PWM/Servo Shield, CircuitPython and `ServoKit`!

Continuous Rotation Servos

To control a continuous rotation servo, you must specify the channel the servo is on. Then you can control movement using `throttle`.

For example, to start the continuous rotation servo connected to channel `1` to full throttle forwards:

```
kit.continuous_servo[1].throttle = 1
```

To start the continuous rotation servo connected to channel `1` to full reverse throttle:

```
kit.continuous_servo[1].throttle = -1
```

To set half throttle, use a decimal:

```
kit.continuous_servo[1].throttle = 0.5
```

And, to stop continuous rotation servo movement set `throttle` to `0`:

```
kit.continuous_servo[1].throttle = 0
```

That's all there is to controlling continuous rotation servos with the PWM/Servo Shield, CircuitPython and `ServoKit`!

Full Example Code

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

"""Simple test for a standard servo on channel 0 and a continuous rotation servo on
```

```
channel 1."""
import time
from adafruit_servokit import ServoKit

# Set channels to the number of servo channels on your kit.
# 8 for FeatherWing, 16 for Shield/HAT/Bonnet.
kit = ServoKit(channels=8)

kit.servo[0].angle = 180
kit.continuous_servo[1].throttle = 1
time.sleep(1)
kit.continuous_servo[1].throttle = -1
time.sleep(1)
kit.servo[0].angle = 0
kit.continuous_servo[1].throttle = 0
```

Python Docs

Python Docs ()

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?

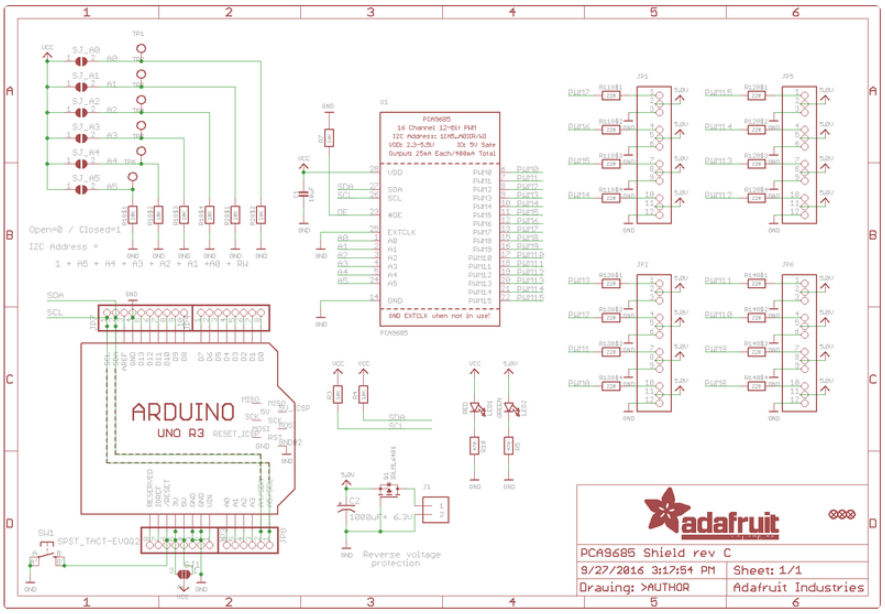
If you want to turn the LEDs totally off use (in Arduino) setPWM(pin, 0, 4096); not setPWM(pin, 0, 4095);

Downloads & Links

Files

- PCA9685 datasheet ()
- Adafruit PWM/Servo Driver Library in GitHub ()
- Adafruit Fritzing Library ()
- EagleCAD PCB files on GitHub ()

Schematic



Fabrication Print