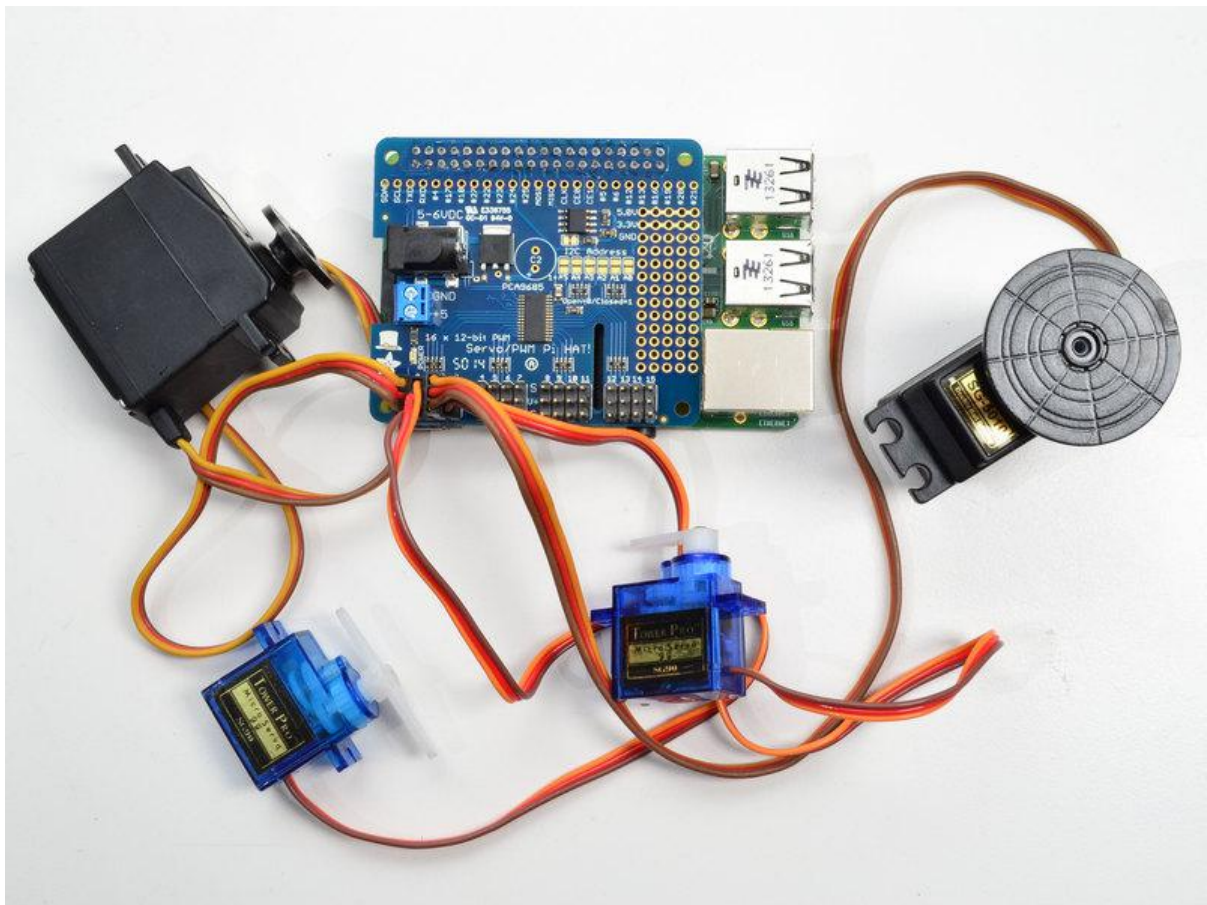




# Adafruit 16-Channel PWM/Servo HAT & Bonnet for Raspberry Pi

Created by lady ada



<https://learn.adafruit.com/adafruit-16-channel-pwm-servo-hat-for-raspberry-pi>

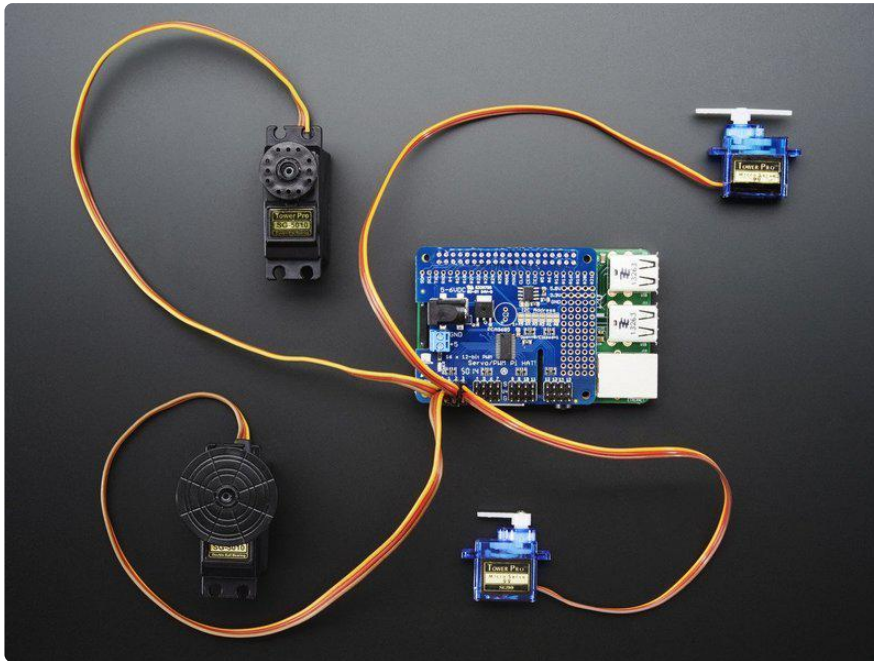
Last updated on 2023-08-29 02:43:49 PM EDT

# Table of Contents

<a href="#">Overview</a>	3
<a href="#">Powering Servos</a>	5
<ul style="list-style-type: none"><li>• <a href="#">Powering Servos / PWM</a></li><li>• <a href="#">OR</a></li><li>• <a href="#">Current Draw Requirements</a></li><li>• <a href="#">Adding a Capacitor to the thru-hole capacitor slot</a></li></ul>	
<a href="#">Connecting Servos</a>	8
<ul style="list-style-type: none"><li>• <a href="#">Connecting a Servo</a></li><li>• <a href="#">Adding More Servos</a></li></ul>	
<a href="#">Attach &amp; Test HAT/Bonnet</a>	10
<ul style="list-style-type: none"><li>• <a href="#">Step 1 - Plug in HAT</a></li><li>• <a href="#">Step 2. Configure your Pi to use I2C devices</a></li></ul>	
<a href="#">Python Usage</a>	11
<ul style="list-style-type: none"><li>• <a href="#">Python Wiring</a></li><li>• <a href="#">Python Installation of ServoKit Library</a></li><li>• <a href="#">Python Usage</a></li><li>• <a href="#">Dimming LEDs</a></li><li>• <a href="#">Controlling Servos</a></li><li>• <a href="#">Standard Servos</a></li><li>• <a href="#">Continuous Rotation Servos</a></li><li>• <a href="#">Full Example Code</a></li></ul>	
<a href="#">Library Reference</a>	17
<ul style="list-style-type: none"><li>• <a href="#">Initialize Object</a></li><li>• <a href="#">setPWMPFreq(self, freq)</a></li><li>• <a href="#">Description</a></li><li>• <a href="#">Arguments</a></li><li>• <a href="#">Example</a></li><li>• <a href="#">setPWM(self, channel, on, off)</a></li><li>• <a href="#">Description</a></li><li>• <a href="#">Arguments</a></li><li>• <a href="#">Example</a></li></ul>	
<a href="#">Stacking HATs</a>	19
<ul style="list-style-type: none"><li>• <a href="#">Extra Parts</a></li><li>• <a href="#">Addressing the HATs</a></li></ul>	
<a href="#">FAQ</a>	22
<a href="#">Downloads</a>	23
<ul style="list-style-type: none"><li>• <a href="#">Files &amp; Downloads</a></li><li>• <a href="#">Schematics</a></li><li>• <a href="#">Fabrication Print</a></li><li>• <a href="#">Schematics and Fab Print for Pi Servo Bonnet</a></li></ul>	

---

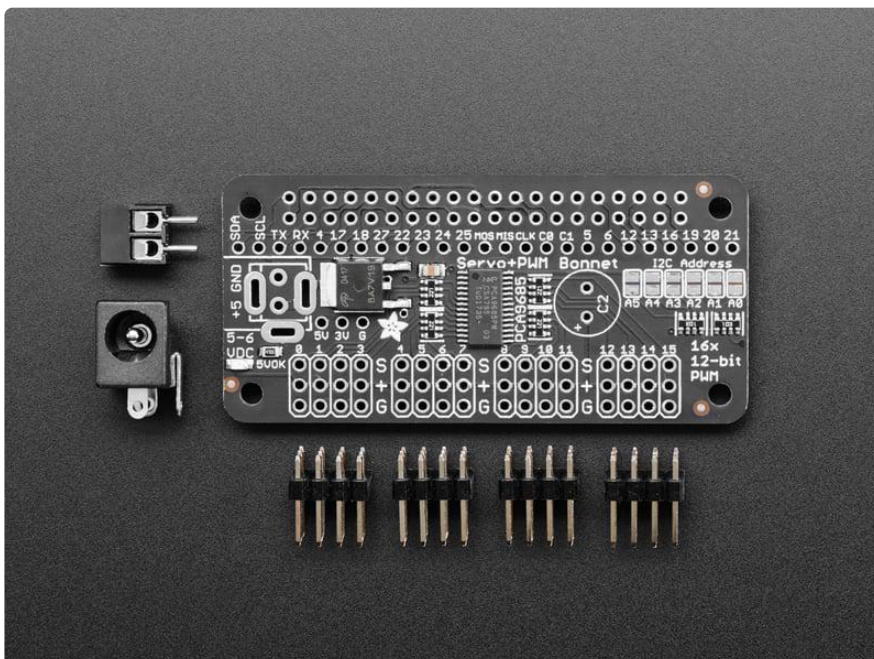
# Overview



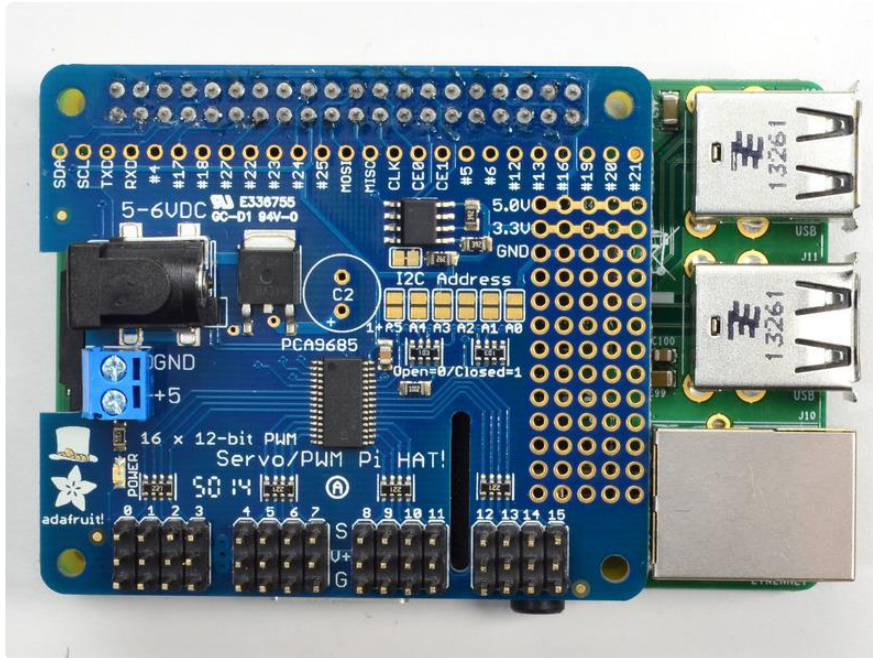
The Raspberry Pi is a wonderful little computer, but one thing it isn't very good at is controlling DC Servo Motors - these motors need very specific and repetitive timing pulses to set the position. Instead of asking the Pi Linux kernel to send these signals, pop on our handy HAT or Bonnet.

These boards add the capability to control 16 Servos with perfect timing. They can also do PWM up to 1.6 KHz with 12 bit precision, all completely free-running.

Works with any servo that can be powered by 5V and take 3.3V logic level signals.

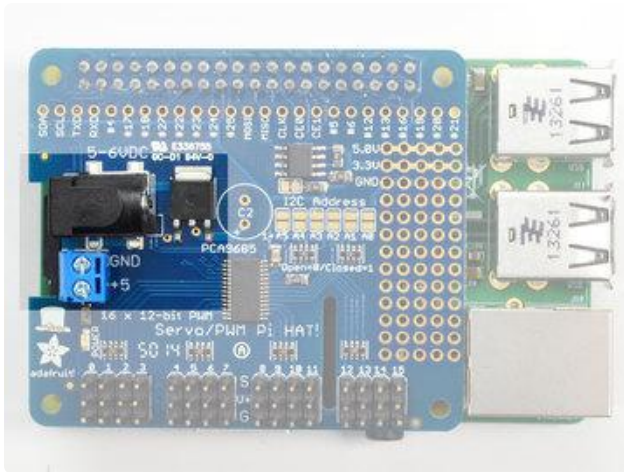


The Adafruit 16-Channel 12-bit PWM/Servo HAT or Bonnet will drive up to 16 servos or PWM outputs over I2C with only 2 pins. The on-board PWM controller will drive all 16 channels simultaneously with no additional Raspberry Pi processing overhead. What's more, you can stack up to 62 of them to control up to 992 servos - all with the same 2 pins!



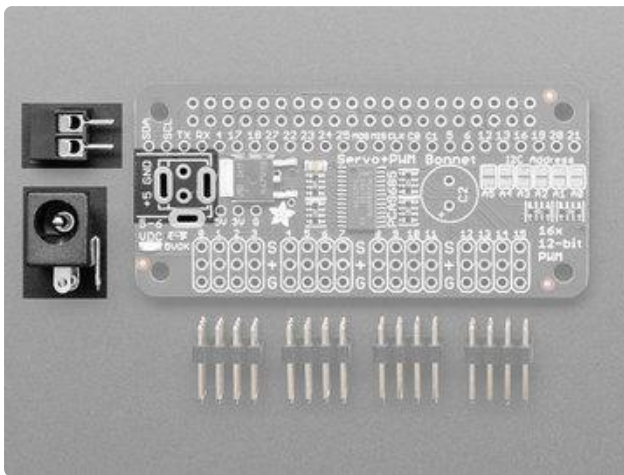
Best of all, we even have a Python library you can use, so you'll be up and running instantly, to make your robotic creation com to life. The Adafruit PWM/Servo Driver is the perfect solution for any project that requires a lot of servos!

# Powering Servos



The power input section of the HAT and Bonnet are both on the left hand side.

The HAT has both 2.1mm DC jack and a terminal block



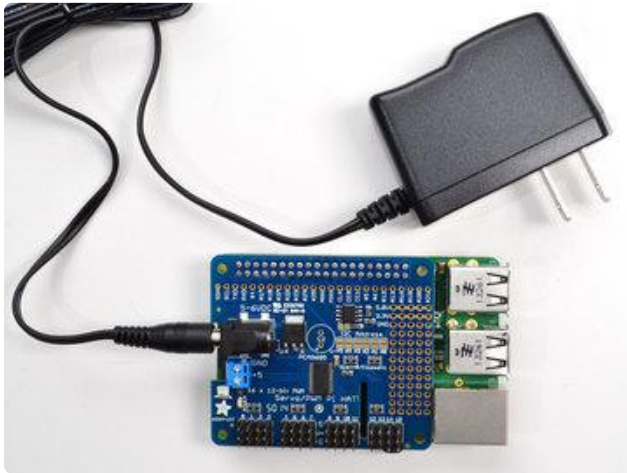
The Bonnet has a spot for either DC jack or terminal block

## Powering Servos / PWM

The drivers have two power supplies. One is VCC - that is the 3.3V power from the Raspberry Pi, it is used to power the PWM chip and determines the I2C logic level and the PWM signal logic level. This supply will always be on if the Pi is plugged in and working, check the PWR LED on the Pi (it's the red LED on the Pi 2, 3. Pi Zero does not have a PWR LED, look for a blinking activity LED)

To power servos you will need to also connect the 5-6V V+ power supply - this is the power supply for the servos. (If you are lighting up single 20mA standard draw LEDs you may not need this power supply, but I'm assuming you want to use servos here.) This power supply should be 5 or 6VDC, most servos work well at 5V and if you give them 6V will be a little stronger.

You can connect this power through the terminal block or the 2.1mm DC jack. There is reverse-polarity protection in case you hook up power backwards, however you should use either the DC jack or the terminal block, not BOTH!



Use either a 5V wall adapter, 2 Amp+ is recommended

OR



Or, for portable use, a 4 or 5 x AA battery pack can be connected to the terminal block.

## Current Draw Requirements

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:

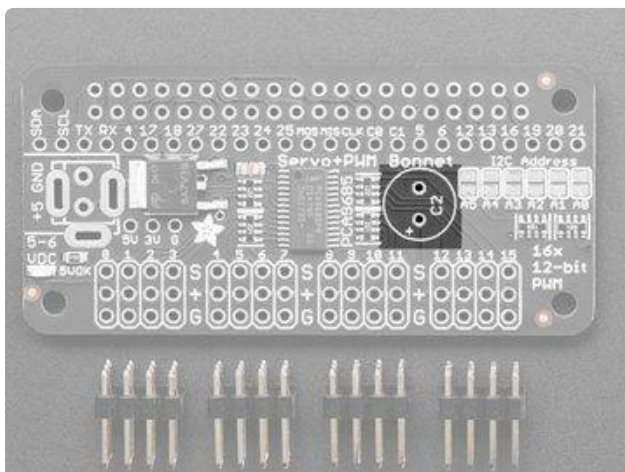
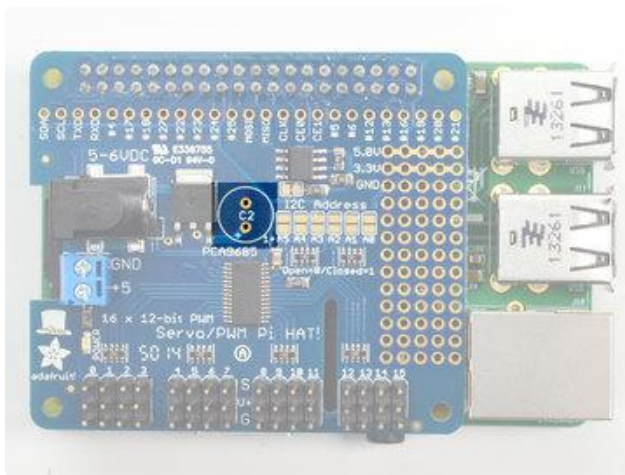
- [5v 2A switching power supply \(http://adafru.it/276\)](http://adafru.it/276) (up to perhaps 4 servos)
- [5v 4A switching power supplies \(\)](#) (up to perhaps 8 servos)
- [5v 10A switching power supply \(http://adafru.it/658\)](http://adafru.it/658) (up to perhaps 16 servos)

- [4xAA Battery Holder \(http://adafru.it/830\)](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.

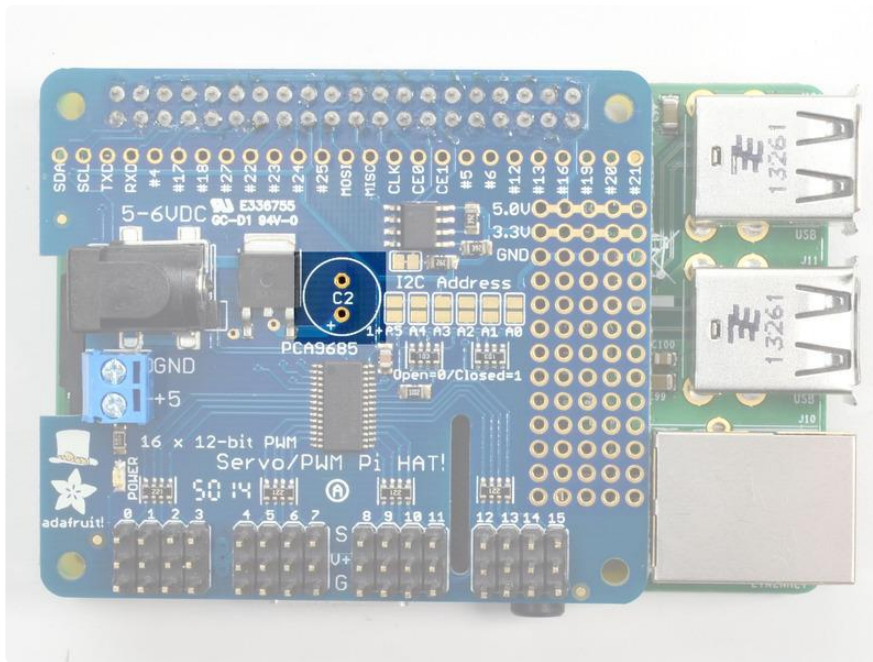
**SERVOS CAN USE A LOT OF POWER!** It is not a good idea to use the Raspberry Pi's 5v pin to power your servos! Electrical noise and 'brownouts' from excess current draw could cause your Pi to act erratically, reset and/or overheat. Seriously, keep the Pi power supply and the Servo power supply completely separate!

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

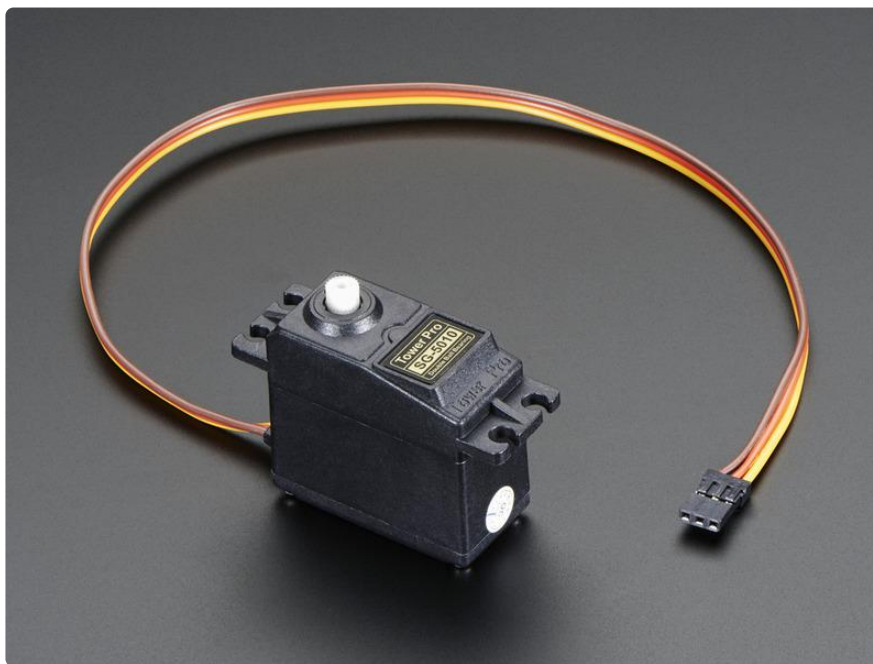
We have a spot on the PCB for soldering in an electrolytic capacitor. Based on your usage, you may or may not need a capacitor. If you are driving a lot of servos from a power supply that dips a lot when the servos move,  $n * 100\mu\text{F}$  where  $n$  is the number of servos is a good place to start - eg 470 $\mu\text{F}$  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.



There are slots on both the bonnet and HAT for an optional capacitor. You may not need the capacitor, it's only if you find that you servo power supply is drooping enough to affect functionality



## Connecting Servos

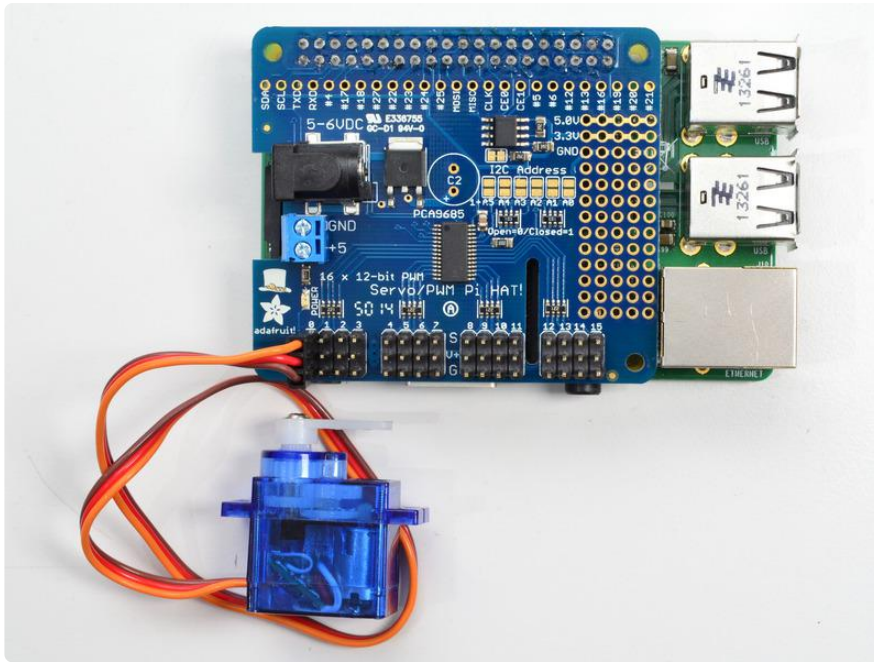


### Connecting a Servo

Most servos come with a standard 3-pin female connector that will plug directly into the headers on the Servo HAT headers. 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.

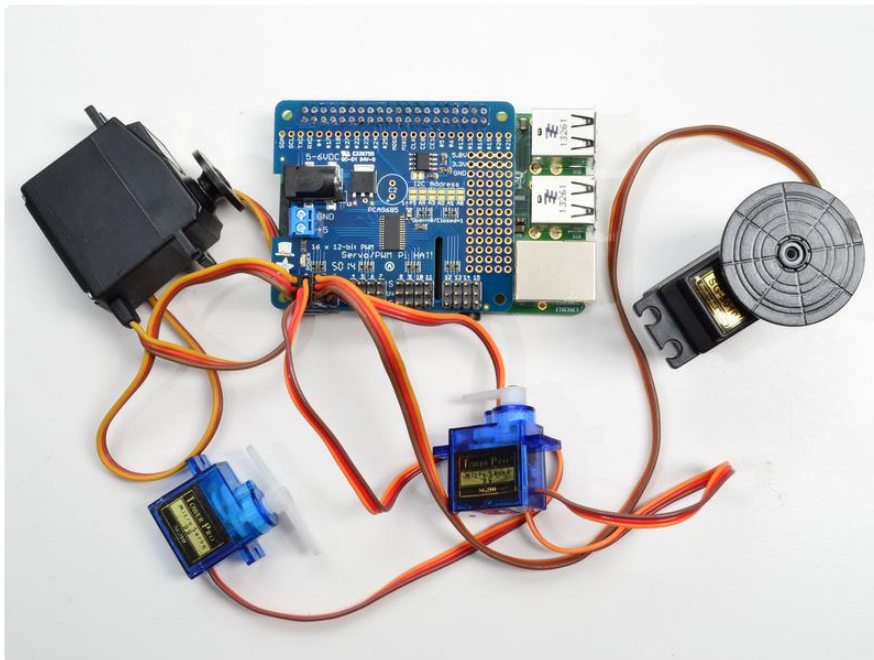
Works with any servo that can be powered by 5V and take 3.3V logic level signals.





## 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.



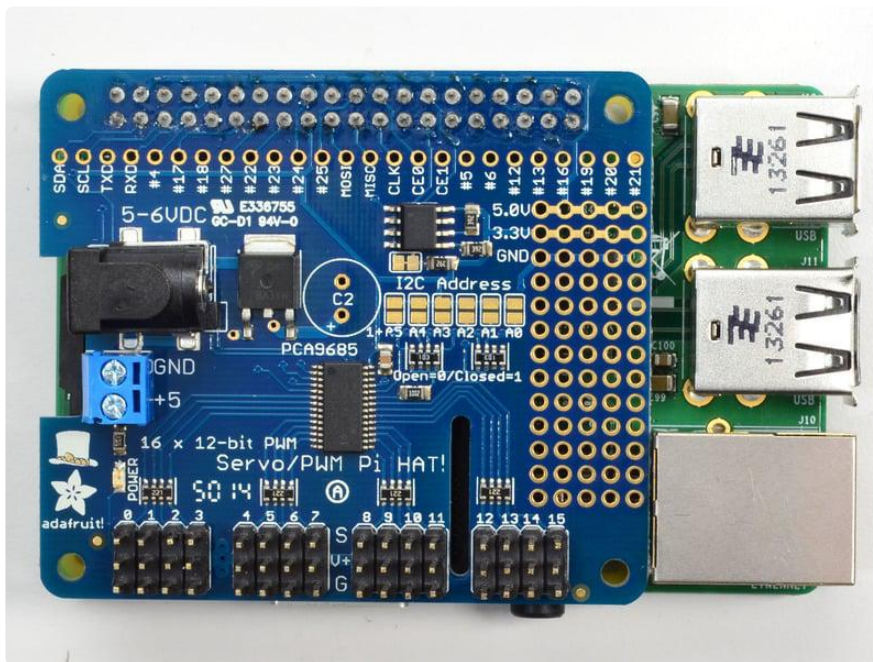
---

# Attach & Test HAT/Bonnet

## Step 1 - Plug in HAT

Now you have soldered the HAT up and you know how to power the servos, we can install the HAT

Begin by having the Pi shutdown and not powered, plug the HAT on top to match the 2x20 headers, and power up the Pi



## Step 2. Configure your Pi to use I2C devices

To learn more about how to setup I2C with either Raspbian or Occidentalis, please take a minor diversion to this Adafruit Tutorial: <http://learn.adafruit.com/adafruits-raspberry-pi-lesson-4-gpio-setup/configuring-i2c> ()

When you are ready to continue, enter the following commands to add SMBus support (which includes I2C) to Python:

```
sudo apt-get install python-smbus
sudo apt-get install i2c-tools
```

i2c-tools isn't strictly required, but it's a useful package since you can use it to scan for any I2C or SMBus devices connected to your board. If you know something is connected, but you don't know its 7-bit I2C address, this library has a great little tool to help you find it. python-smbus is required, it adds the I2C support for python!

[Don't forget you must add kernel support for I2C by following this tutorial! \(\)](#)

You can then detect if the HAT is found on the #1 I2C port with:

```
sudo i2cdetect -y 1
```

This will search /dev/i2c-1 for all address, and if an Adafruit PWM/Servo HAT is properly connected and it's set to its default address -- meaning none of the 6 address solder jumpers at the top of the board have been soldered shut -- it should show up at 0x40 (binary 1000000) as follows:



```
192.168.1.104:22 - pi@raspberrypi: ~/code/Adafruit-Raspberry-Pi-Pyth VT
File Edit Setup Control Window Help
pi@raspberrypi ~/code/Adafruit-Raspberry-Pi-Python-Code/Adafruit_PWM_Servo_Drive
r $ sudo i2cdetect -y 0
 0 1 2 3 4 5 6 7 8 9 a b c d e f
00: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: 40 -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: 70 -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
pi@raspberrypi ~/code/Adafruit-Raspberry-Pi-Python-Code/Adafruit_PWM_Servo_Drive
r $
```

Once both of these packages have been installed, and i2cdetect finds the 0x40 I2C address, you have everything you need to get started accessing I2C and SMBus devices in Python.

---

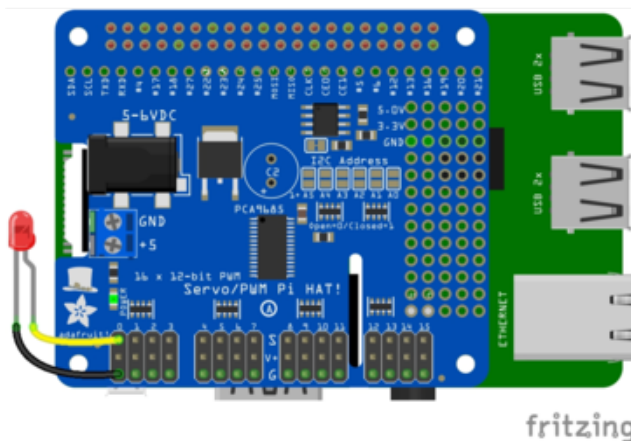
## Python Usage

It's easy to control PWM or servos with the Adafruit 16-channel PWM/Servo HAT and Bonnet. There are multiple CircuitPython libraries available to work with the different features of these boards including [Adafruit CircuitPython PCA9685 \(\)](#), and [Adafruit CircuitPython ServoKit \(\)](#). These libraries make it easy to write Python code to control PWM and servo motors.

# Python Wiring

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

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 HAT and Bonnet will limit the current to around 10mA.



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

To control a servo, wire it to the board as shown in the previous pages, including a barrel jack to the power terminal to attach an appropriate external power source to the HAT or Bonnet. The HAT and Bonnet will not power servos without an external power source!

## Python Installation of ServoKit Library

You'll need to install the Adafruit\_Blinka library that provides the CircuitPython support in Python. This may also require enabling I2C on your platform and verifying you are running Python 3. [Since each platform is a little different, and Linux changes often, please visit the CircuitPython on Linux guide to get your computer ready \(!\)](#)

Once that's done, from your command line run the following command:

- `sudo pip3 install adafruit-circuitpython-servokit`

If your default Python is version 3 you may need to run 'pip' instead. Just make sure you aren't trying to use CircuitPython on Python 2.x, it isn't supported!

# Python Usage

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

## Dimming LEDs

This HAT and Bonnet use the PCA9685. Each channel of the HAT and Bonnet 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)
hat = 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:

```
hat.frequency = 60
```

The HAT and Bonnet support 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 = hat.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 HAT and Bonnet!

## Controlling Servos

We've written a handy CircuitPython library for the various PWM/Servo kits called [Adafruit 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 HAT or Bonnet.

First you'll need to import and initialize the `ServoKit` class. You must specify the number of channels available on your board. The HAT and Bonnet have 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 HAT or Bonnet, Python 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 HAT or Bonnet, Python 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
```



---

# Library Reference

The driver consists of the following functions, which you can use to drive the underlying hardware when writing your own application in Python:

## Initialize Object

You can create a new object for each HAT with

```
pwm = PWM(0x40)
```

In this case, `pwm` (lowercase) is the object created, and `PWM(0x40)` is the creation call. By default, all HATs are address `0x40`, but by changing the address jumpers, you can create objects that use other addresses such as `0x60`, `0x42`, etc.

## setPWMFreq(self, 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(self, 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

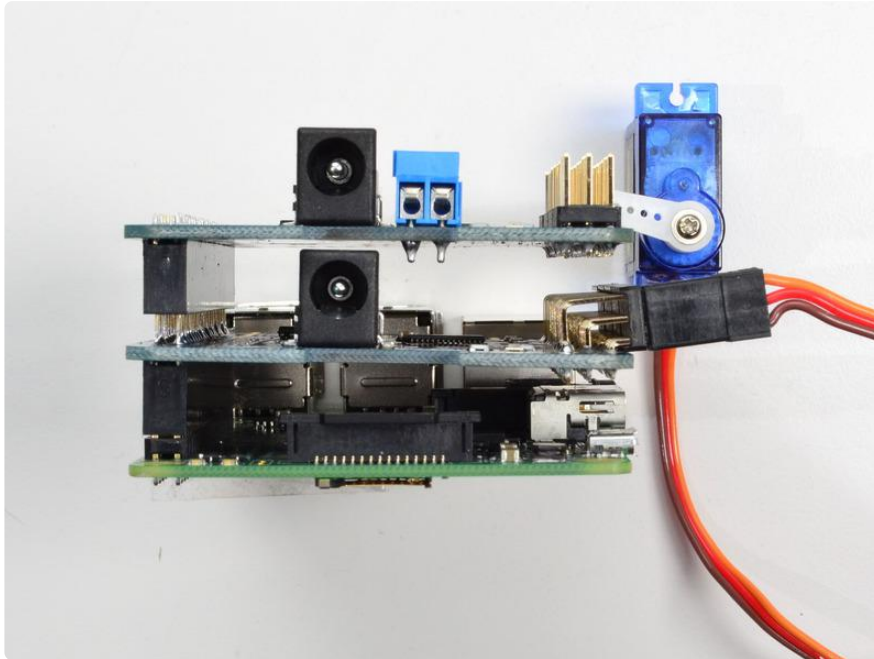
The following example will cause channel 15 to start low, go high around 25% into the pulse (tick 1024 out of 4096), transition back to low 75% into the pulse (tick 3072), and remain low for the last 25% of the pulse:

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

If you need to calculate pulse-width in microseconds, you can do that by first figuring out how long each cycle is. That would be  $1/\text{freq}$  where freq is the PWM frequency you set above. For 1000 Hz, that would be 1 millisecond. Then divide by 4096 to get the time per tick, eg  $1 \text{ millisecond} / 4096 = \sim 0.25 \text{ microseconds}$ . If you want a pulse that is 10 microseconds long, divide the time by time-per-tick ( $10\text{us} / 0.25 \text{ us} = 40$ ) then turn on at tick 0 and turn off at tick 40.

---

# Stacking HATs

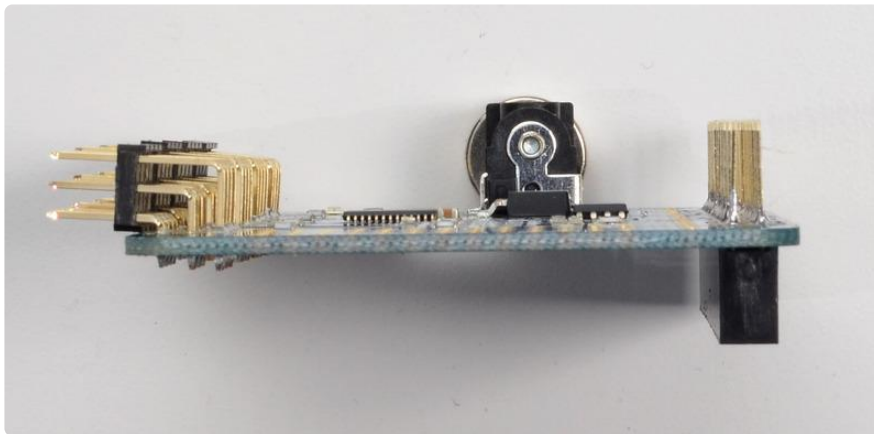
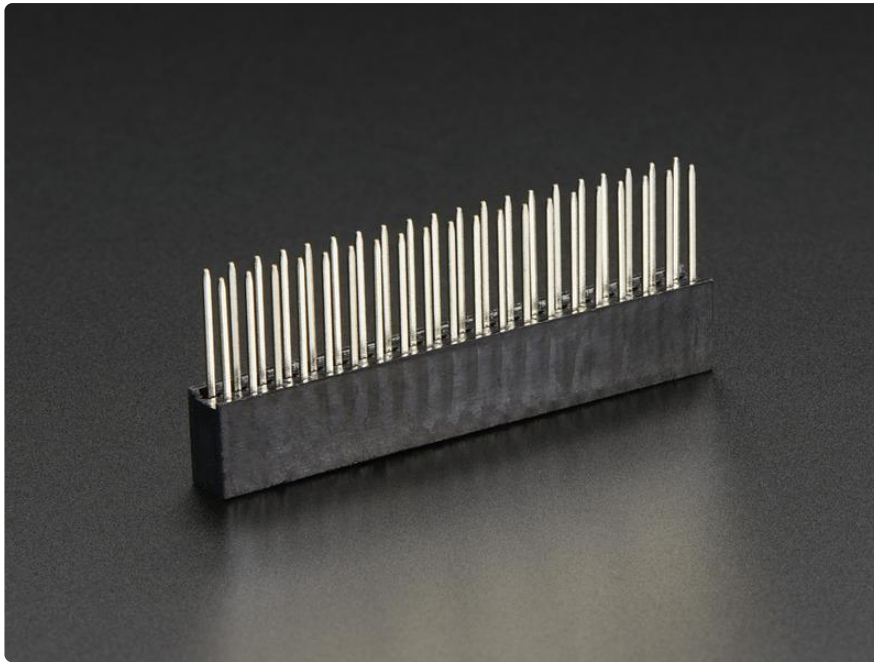


Even though HATs are not intended to be stacked, you can stack up to 62 HATs and not have an address collision, for up to 992 PWM outputs! You'll still need to provide power and write code for all those outputs but they can all share the same SDA/SCL pins no problem.

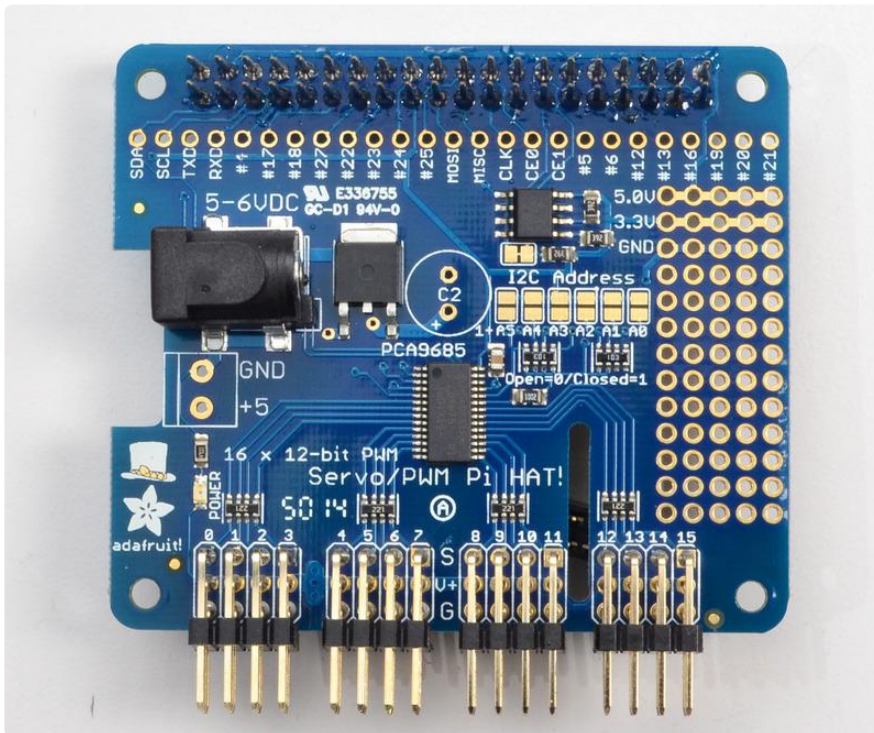
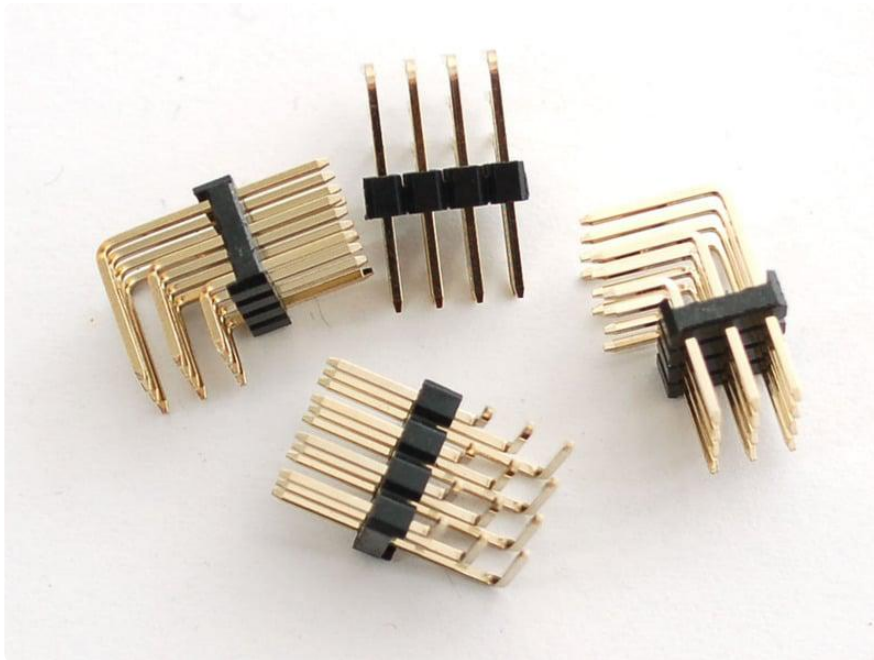
You will need to have installed stacking headers & right angle 3x4 connections for it to physically connect.

## Extra Parts

If you want to stack HATs on top of this one, [make sure you pick up a HAT-stacking header \(\)](#) and solder them instead of the plain 2x20 header that comes in the kit



You'll also need a set of right-angle 3x4 headers, since you will have to have the servo connections stick out instead of up

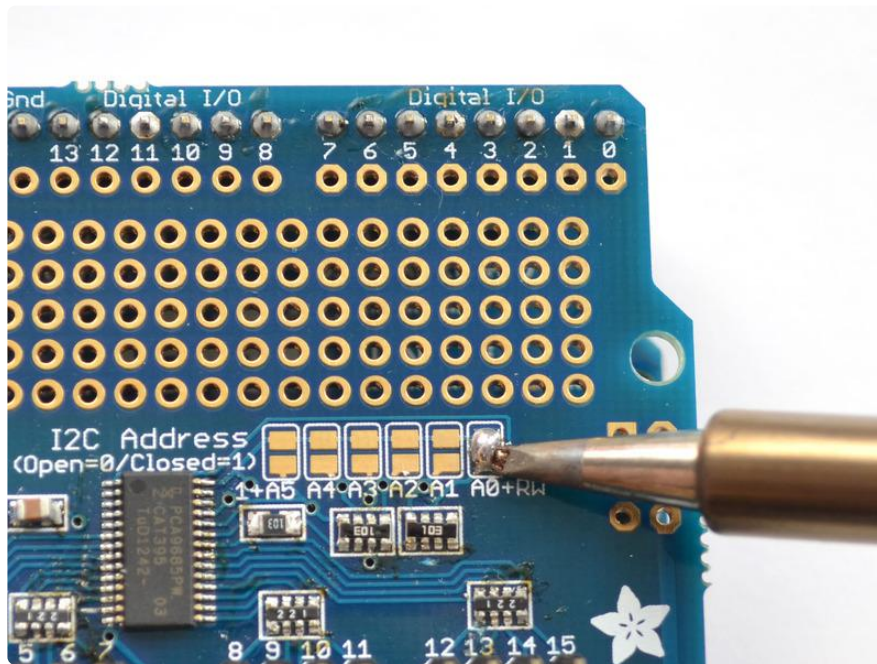


## Addressing the HATs

Each HAT in the stack must be assigned a unique address. This is done with the address jumpers on the middle right 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.

This photo is from the Arduino Shield version of this driver but its the same setup



- 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.

---

## FAQ

---

### Can this HAT be used for LEDs or just servos?

It can be used for LEDs as well as any other PWM-able device! Use the Signal and Ground pins if you dont mind the LEDs powered by 3.3V and 220ohm series resistor. Or V+ and your own resistor & LED, if you want up to 5V power for the LEDs

---

### 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.

---

If I'm using it with LEDs I cant quite get the PWM to be totally off?

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

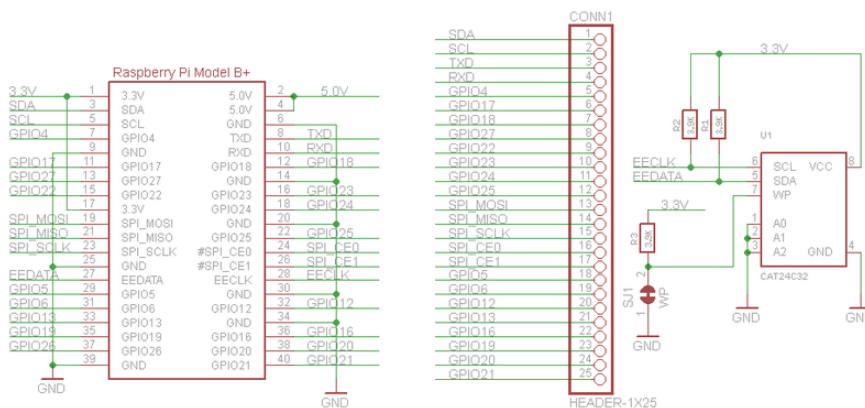
## Downloads

## Files & Downloads

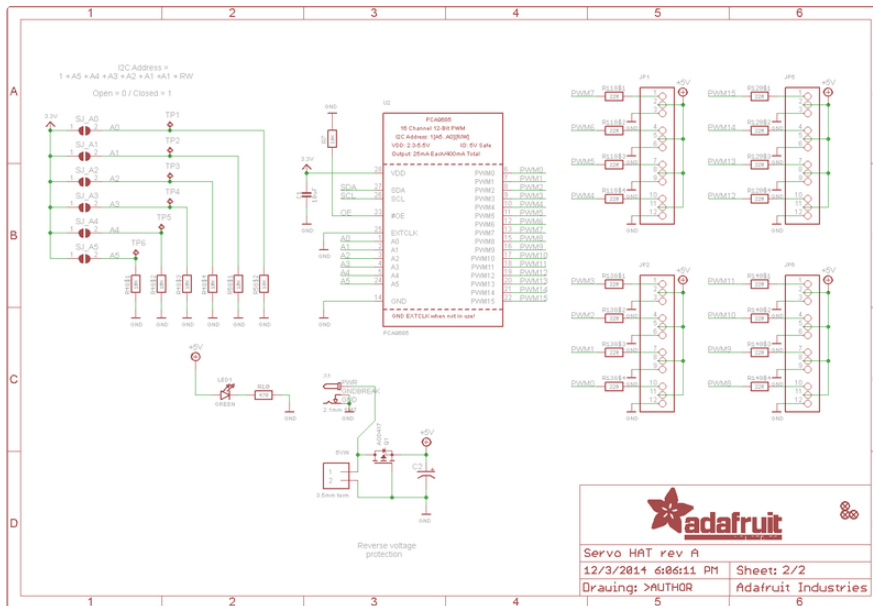
- Datasheet for servo/PWM control chip [PCA9685 \(\)](#)
- [Full Official Specifications for Pi HAT dimensions \(\)](#)
- [EagleCAD PCB files on GitHub \(\)](#)
- [Fritzing object in Adafruit Fritzing Library \(\)](#)

## Schematics

Pi HAT and GPIO Breakout:

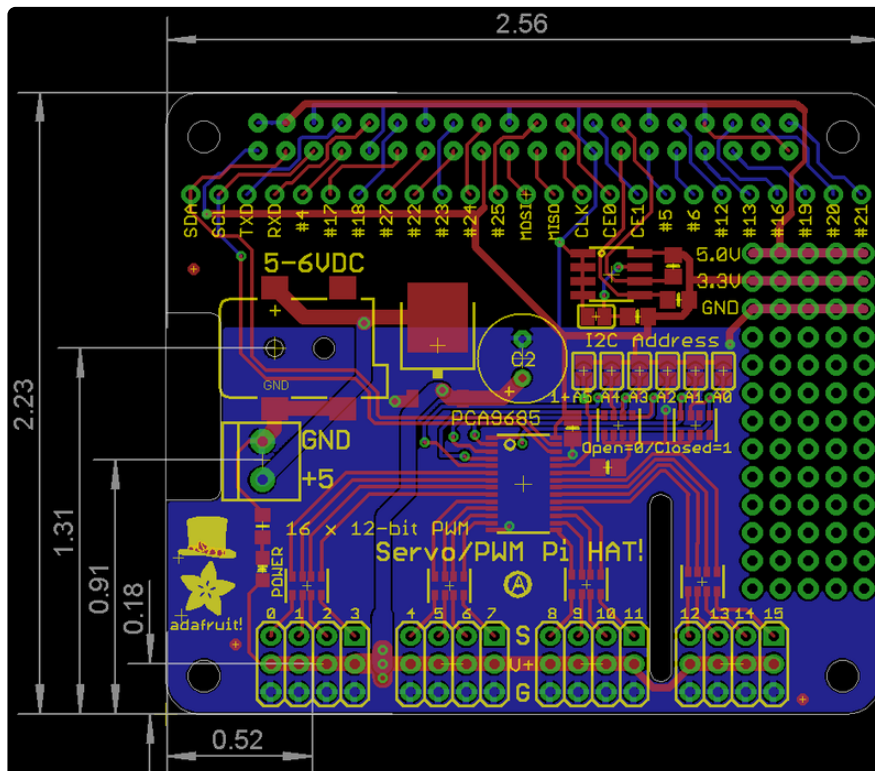


Motor Control Section:



# Fabrication Print

Dimensions in Inches. [For more dimensional details, see the official Pi HAT mechanical specification \(\)](#).





# Schematics and Fab Print for Pi Servo Bonnet

