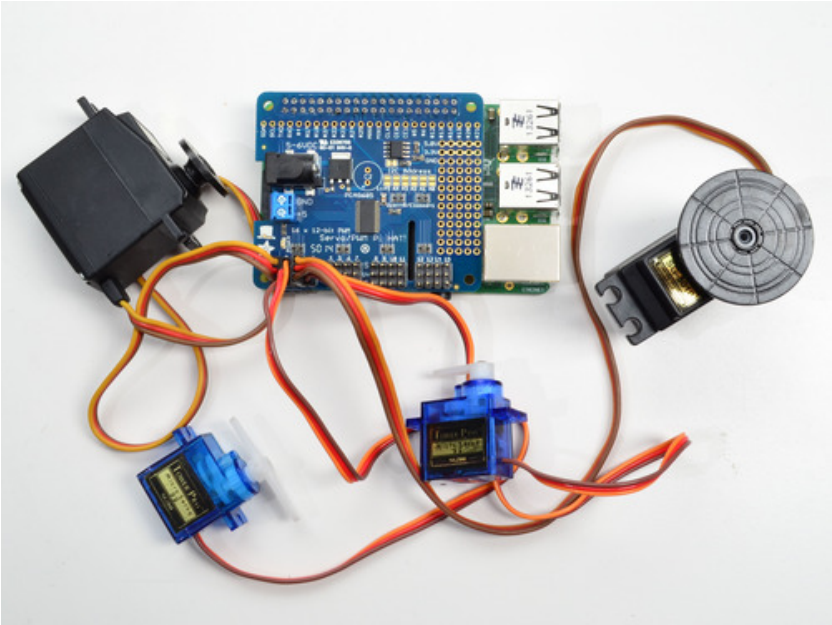
adafruit learning system

Adafruit 16-Channel PWM/Servo HAT for Raspberry Pi

Created by lady ada

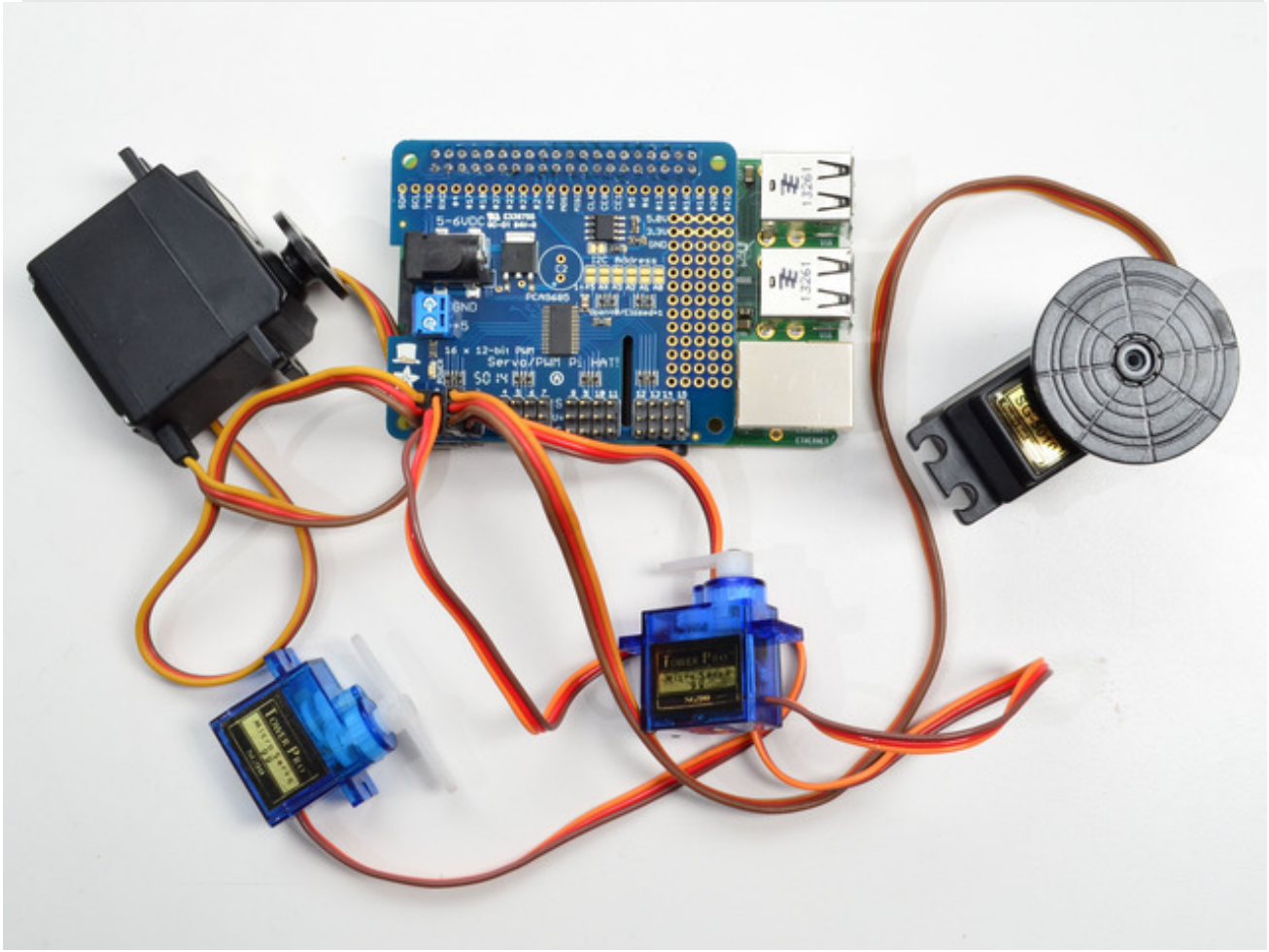


Last updated on 2015-02-03 05:30:19 PM EST

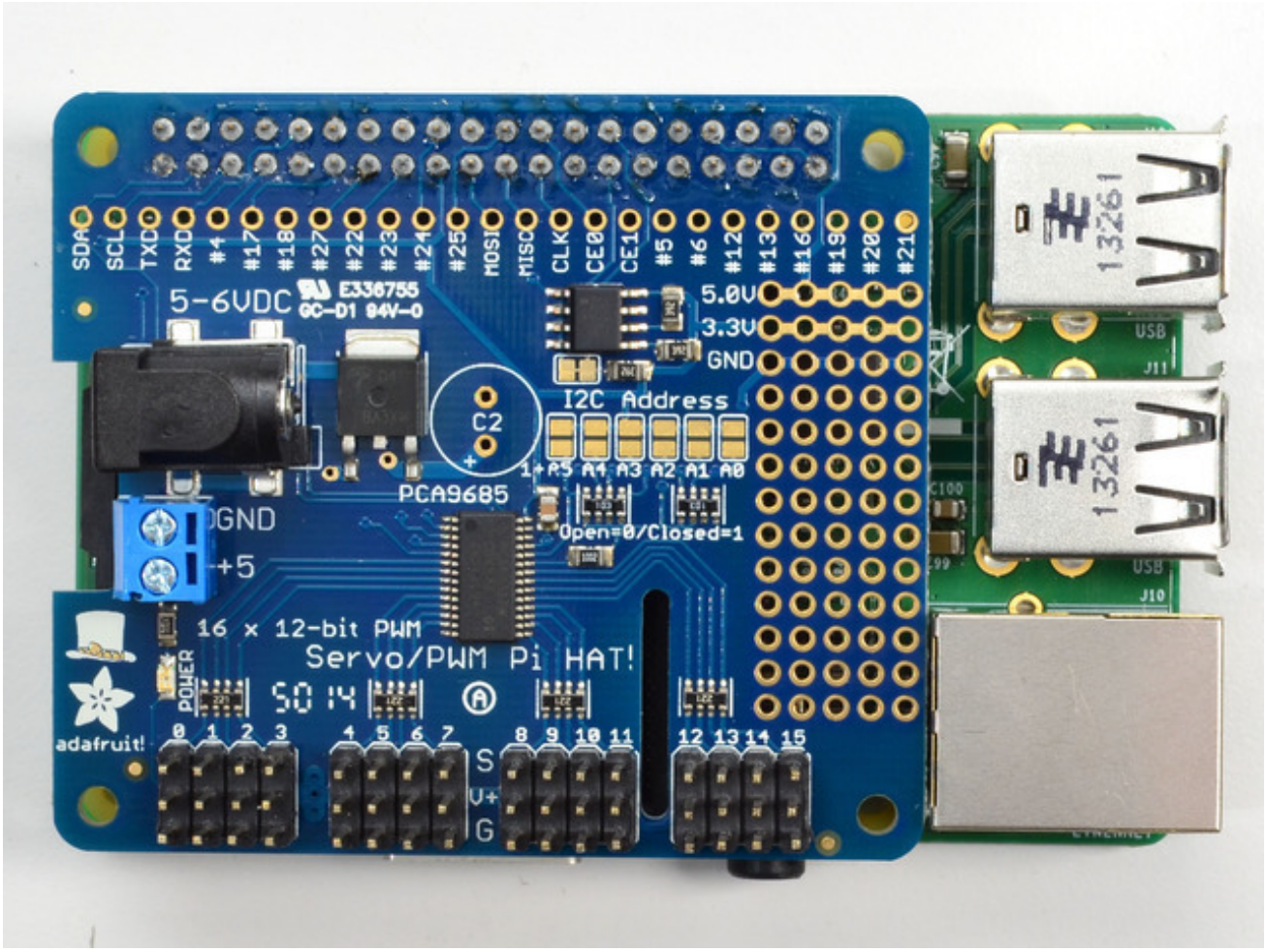
Guide Contents

Guide Contents 2
Overview 4
Powering Servos 7
Powering Servos / PWM 7
OR 8
Current Draw Requirements 9
Adding a Capacitor to the thru-hole capacitor slot 10
Connecting Servos 11
Connecting a Servo 11
Adding More Servos 12
Attach & Test the HAT 14
Step 1 - Plug in HAT 14
Step 2. Configure your Pi to use I2C devices 14
Using the Python Library 16
Downloading the Code from Github 16
Testing the Library 16
Library Reference 18
Initialize Object 18
setPWMFreq(self, freq) 18
Description 18
Arguments 18
Example 18
setPWM(self, channel, on, off) 18
Description 18
Arguments 19
Example 19
Stacking HATs 20
Extra Parts 20
Addressing the HATs 23

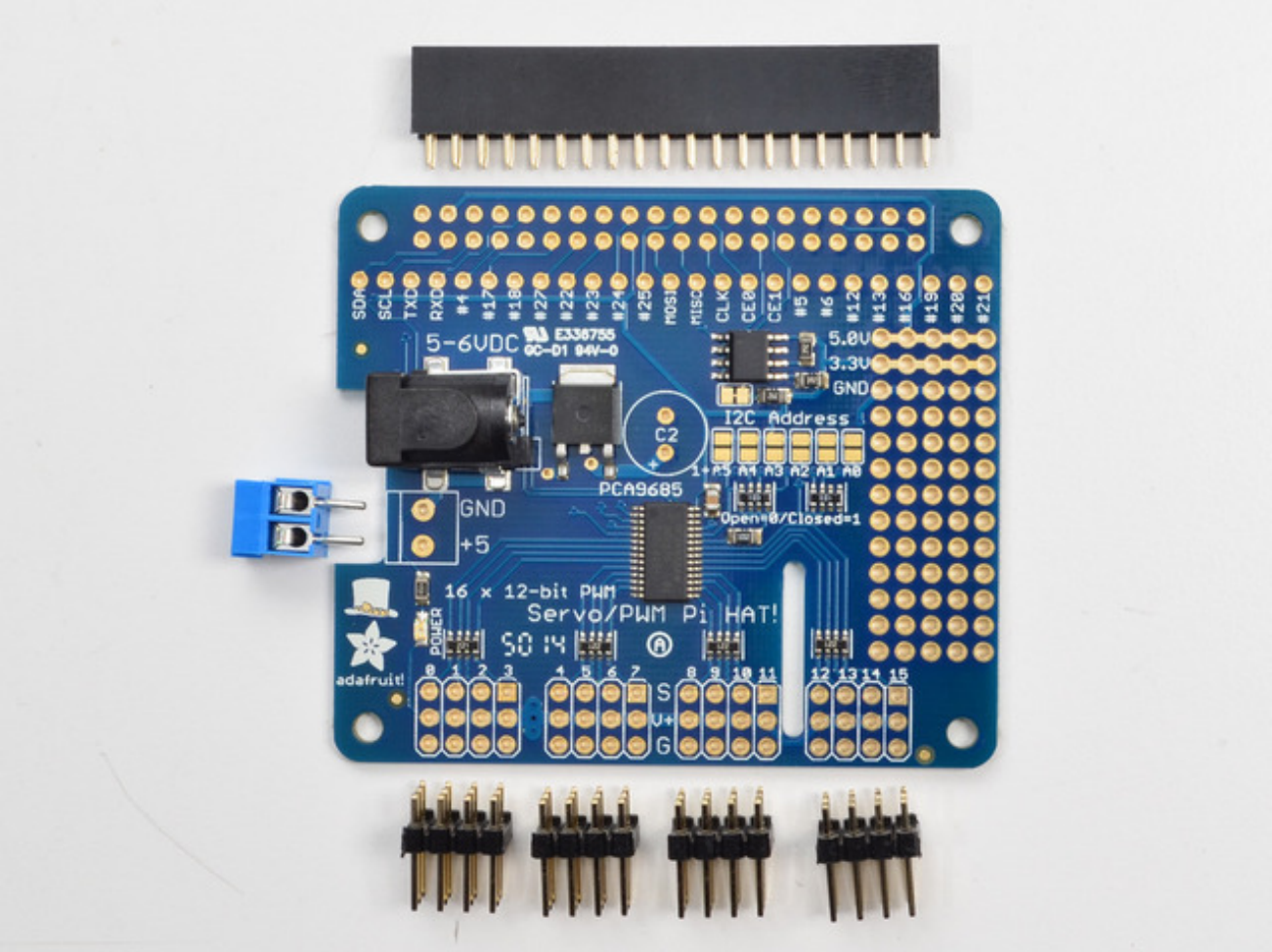
Overview



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 this handy HAT! It adds the capability to control 16 Servos with perfect timing. It can also do PWM up to 1.6 KHz with 12 bit precision, all completely free-running.

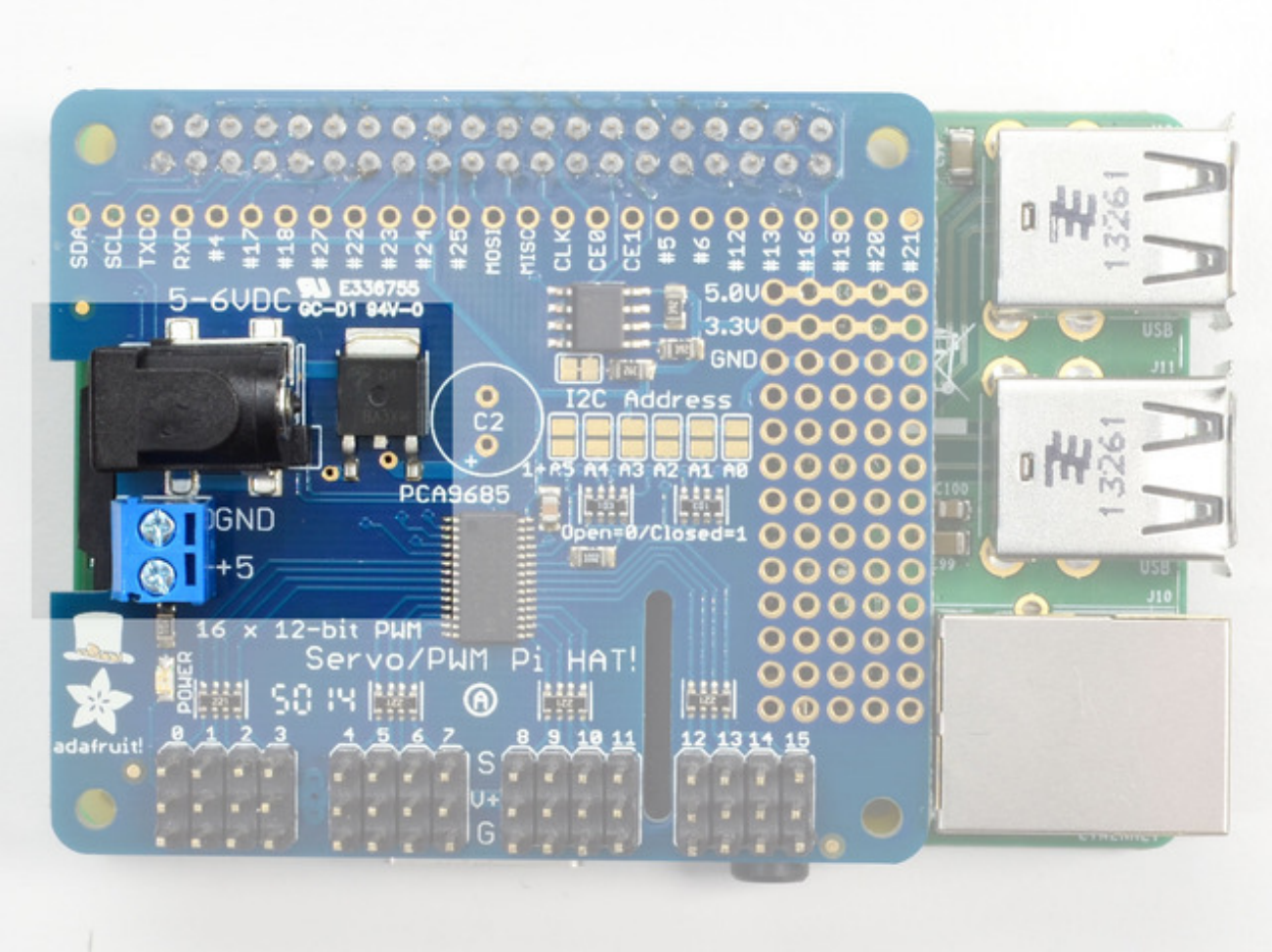


The Adafruit 16-Channel 12-bit PWM/Servo HAT will drive up to 16 servos or PWM outputs over I2C with only 2 pins. The on-board PWM controller will drive all 16 channels simultaneously with no additional Raspberry Pi processing overhead. What's more, you can stack up to 62 of them to control up to 992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

Powering Servos

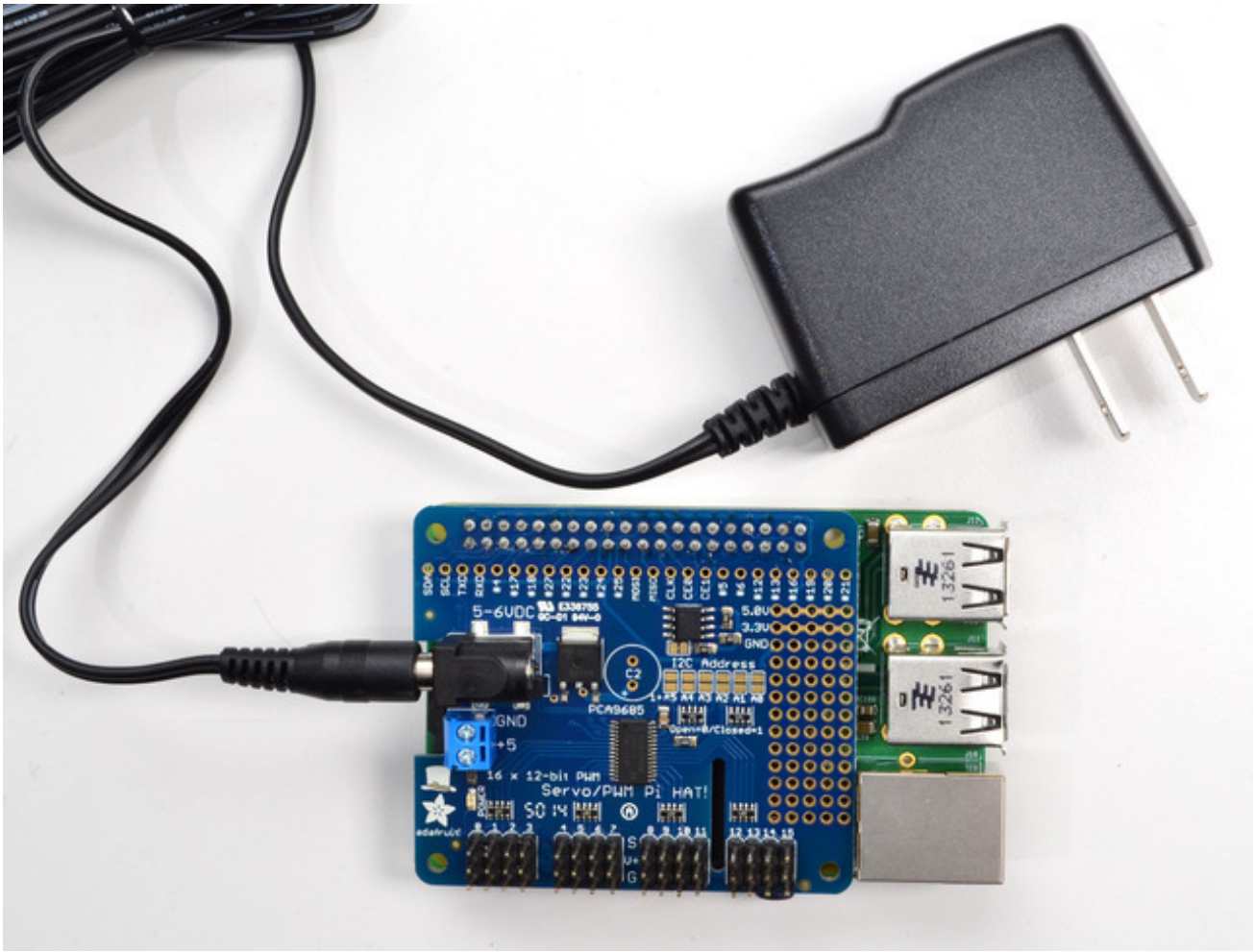


Powering Servos / PWM

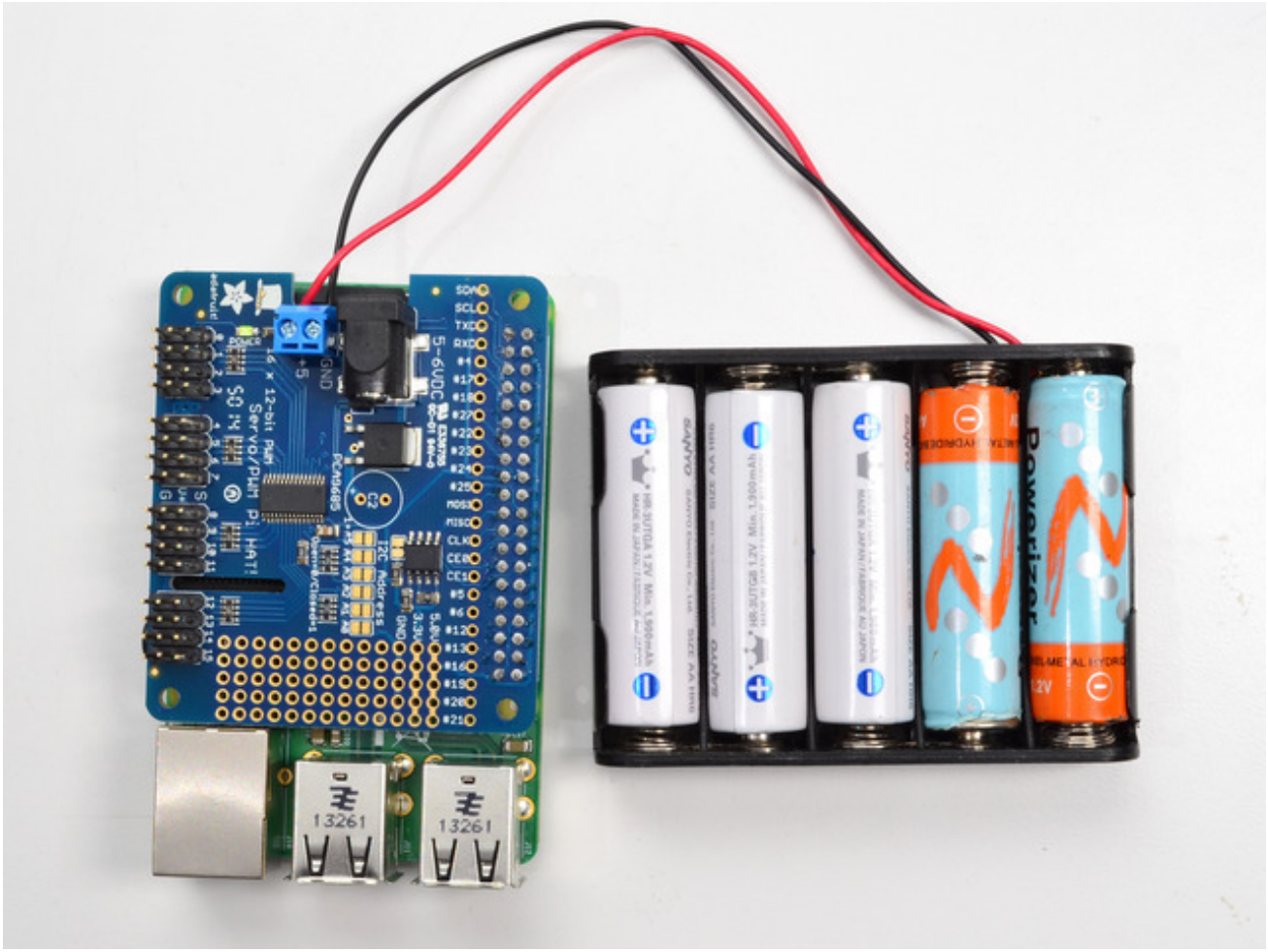
This HAT has **two** power supplies. One is VCC - that is the 3.3V power from the Raspberry Pi, it is used to power the PWM chip and determines the I2C logic level and the PWM signal logic level. This supply will always be on if the Pi is plugged in and working, check the PWR LED on the Pi (it's the red LED)

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 blue 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!


5-6VDC
GND
+5
16 x 12-bit PWM
Servo/PWM Pi HAT!
I2C Address
PCA9685
5.0V
3.3V
GND

OR



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:

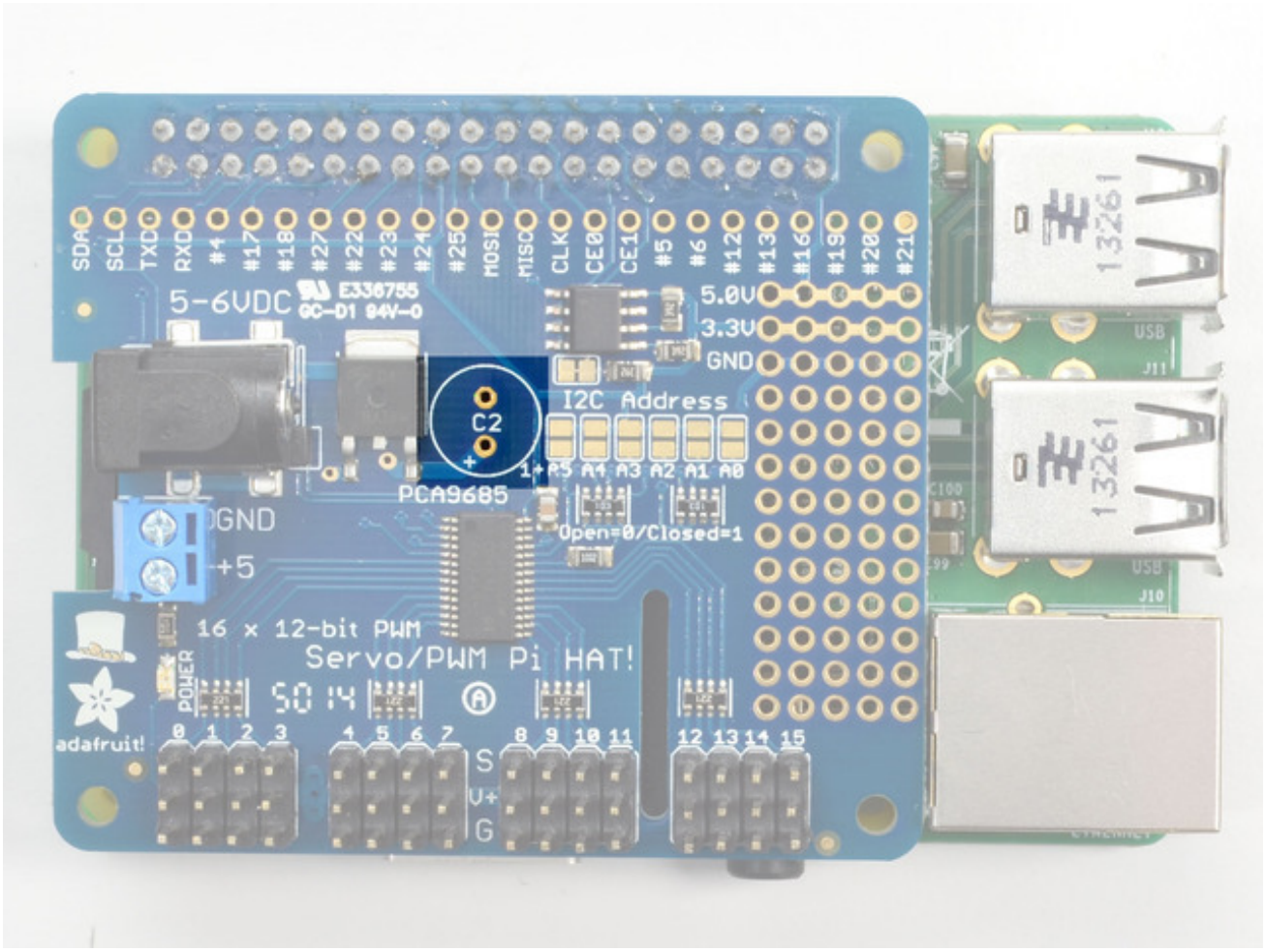
- 5v 2A switching power supply (http://adafru.it/276) (up to perhaps 4 servos)
- 5v 4A switching power supplies (http://adafru.it/e50) (up to perhaps 8 servos)
- 5v 10A switching power supply (http://adafru.it/658) (up to perhaps 16 servos)
- 4xAA Battery Holder (http://adafru.it/830) - 6v with Alkaline 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 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

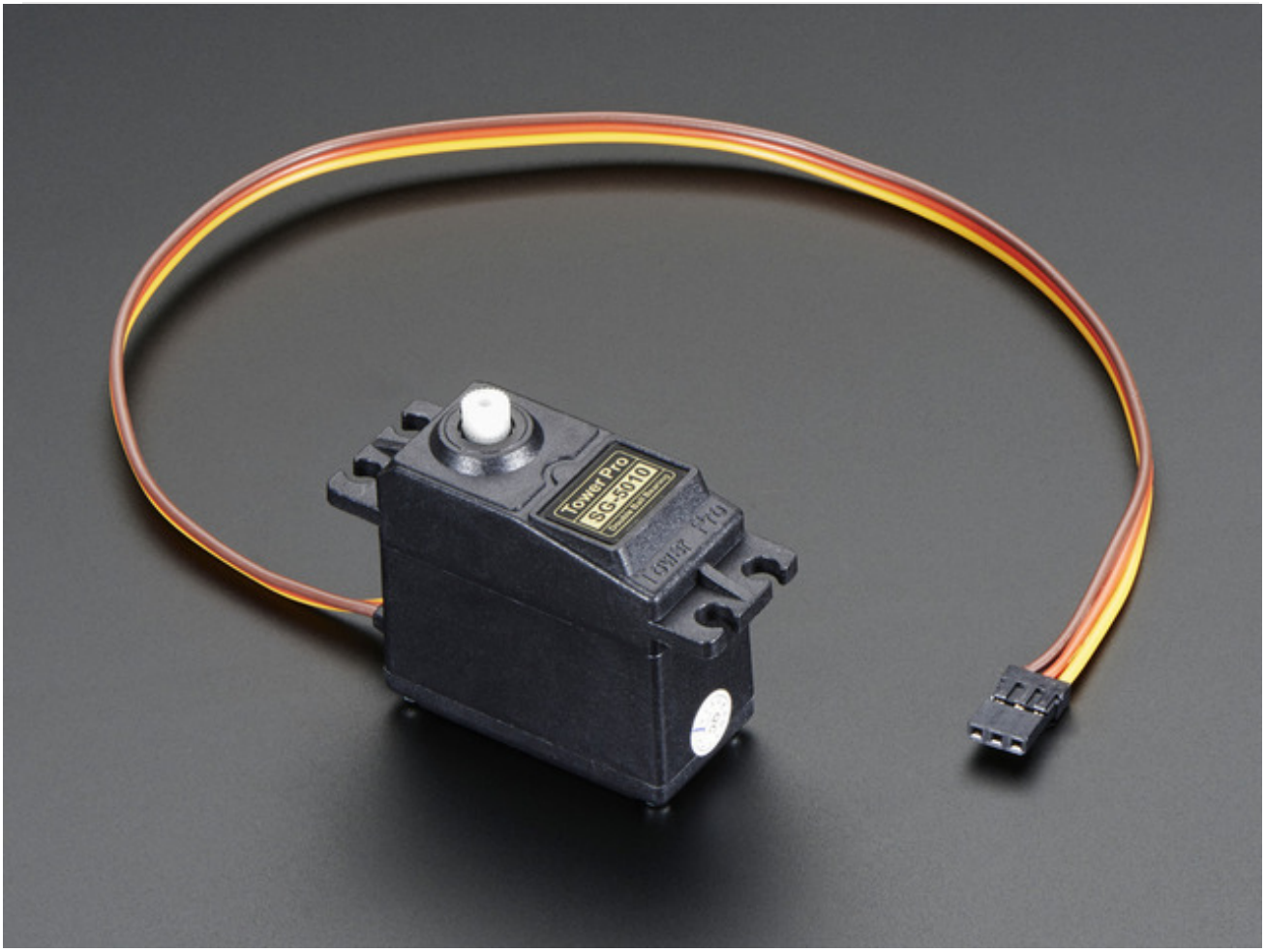
power supply and the Servo power supply completely seperate!

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.

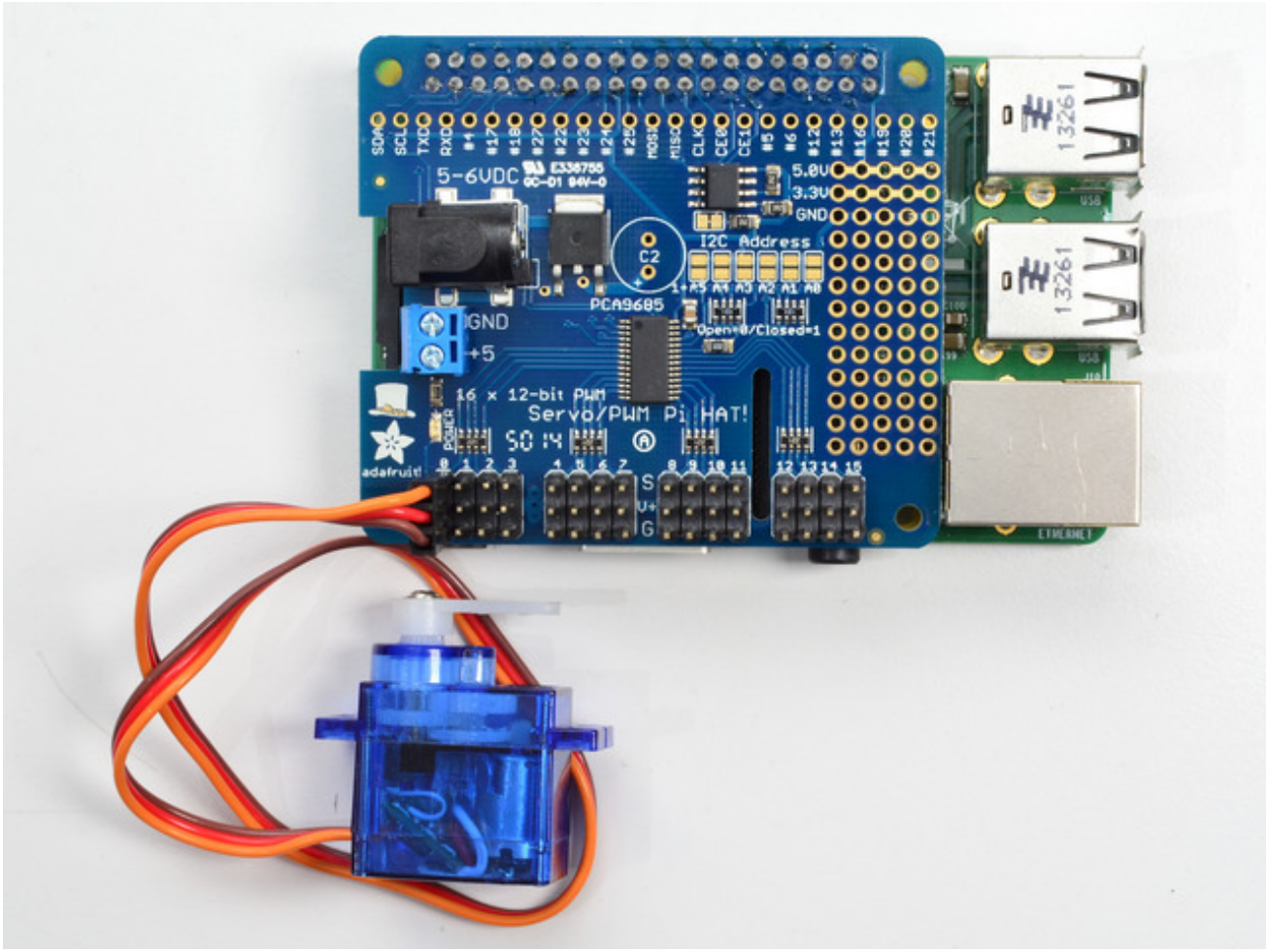


Connecting Servos



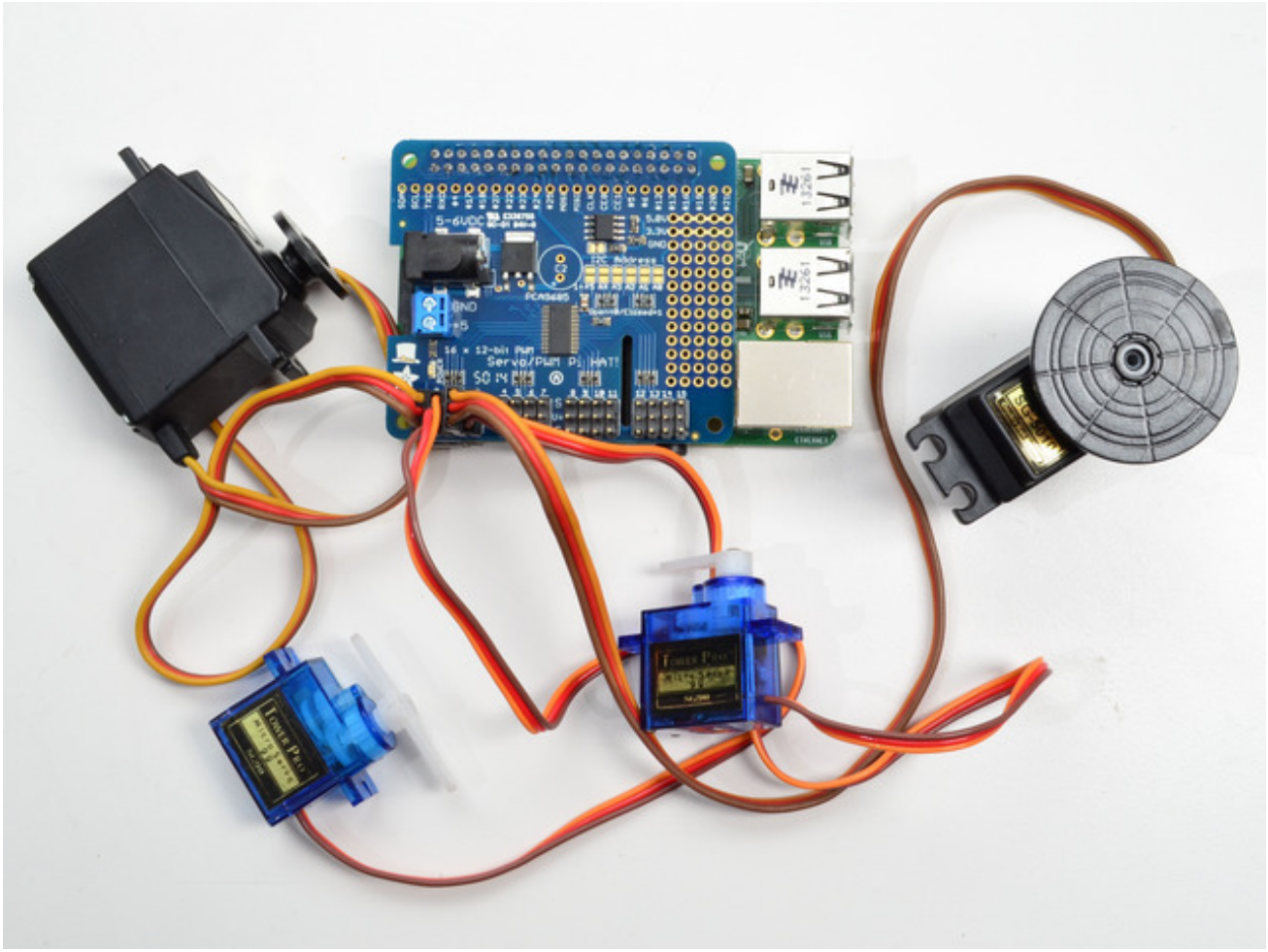
Connecting a Servo

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.



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.

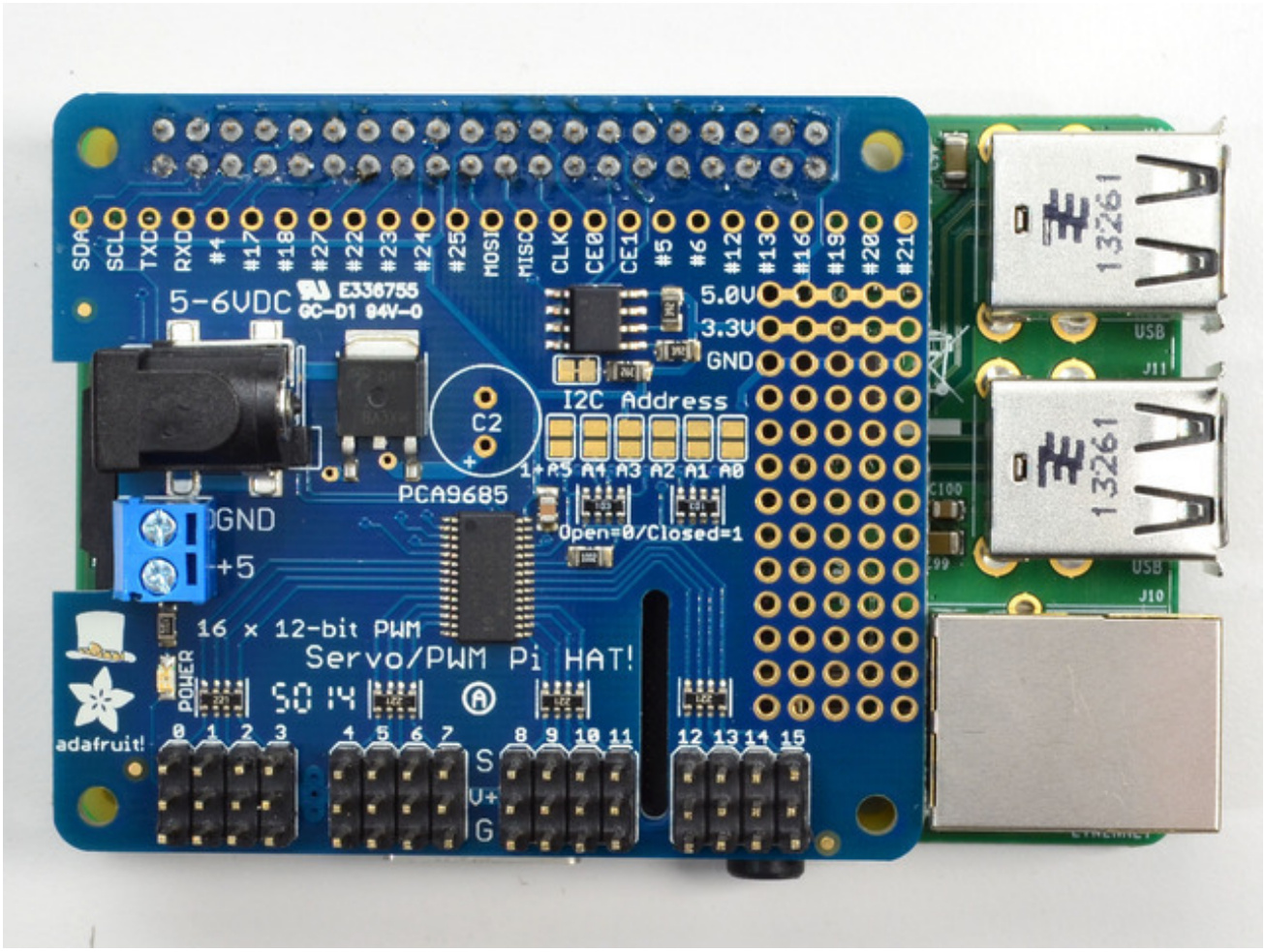

5-6VDC
16 x 12-bit PWM
Servo/PWM Pi HAT

Attach & Test the HAT

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 (http://adafru.it/aTI)

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
```

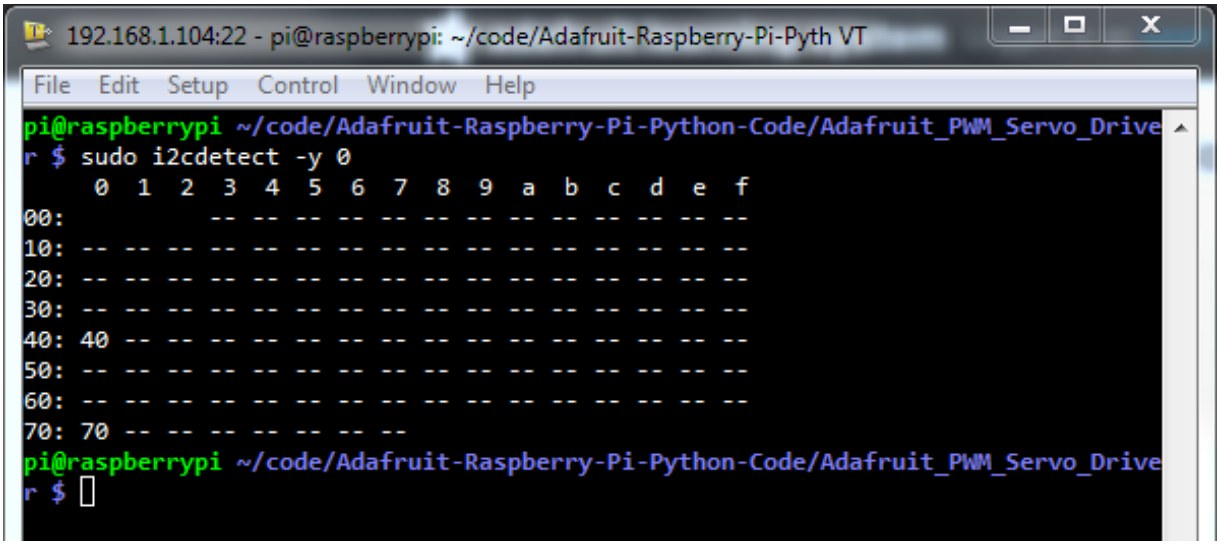
i2c-tools isn't strictly required, but it's a useful package since you can use it to scan for any I2C or SMBus devices connected to your board. If you know something is connected, but you don't know it's 7-bit I2C address, this library has a great little tool to help you find it. **python-smbus** is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

Don't forget you must add kernel support for I2C by following this tutorial! (http://adafru.it/dZY)

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:



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.

Using the Python Library

The Python code for Adafruit's PWM/Servo breakout on the Pi is available on Github at https://github.com/adafruit/Adafruit-Raspberry-Pi-Python-Code (http://adafru.it/aOg)

This code should be a good starting point to understanding how you can access SMBus/I2C devices with your Pi, and getting things moving with your PWM/Servo breakout.

Before you start, you'll need to have the python smbus library installed, run **apt-get install python-smbus**

Downloading the Code from Github

The easiest way to get the code onto your Pi is to hook up an Ethernet cable, and clone it directly using 'git', which is installed by default on most distros. Simply run the following commands from an appropriate location (ex. "/home/pi"):

```
git clone https://github.com/adafruit/Adafruit-Raspberry-Pi-Python-Code.git
cd Adafruit-Raspberry-Pi-Python-Code
cd Adafruit_PWM_Servo_Driver
```

if you get an error when running the **git** command, try running the installer,

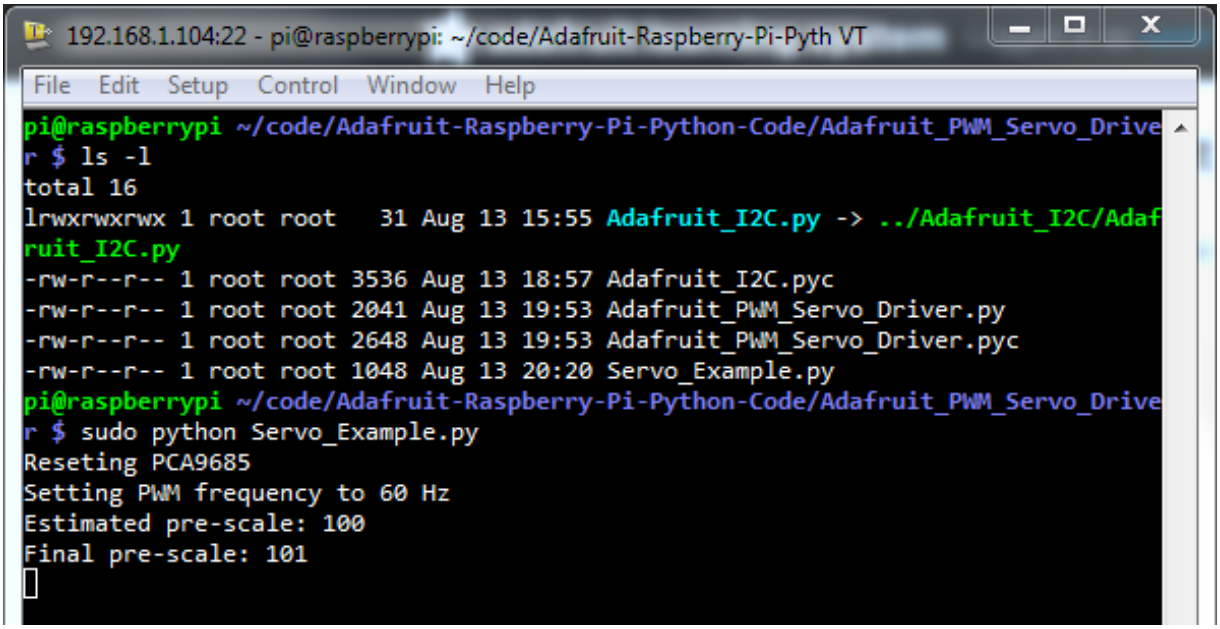
sudo apt-get install git

to install git

Testing the Library

Once the code has be downloaded to an appropriate folder, and you have your PWM/Servo HAT and motor properly connected, you can test it out with the following command (the driver includes a simple demo program):

```
sudo python Servo_Example.py
```



To stop the example, simple press CTRL+C.

Depending on if you are using a standard or continuous rotation servo, you should get results similar to the following (a continuous rotation servo is being used in this particular example):

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

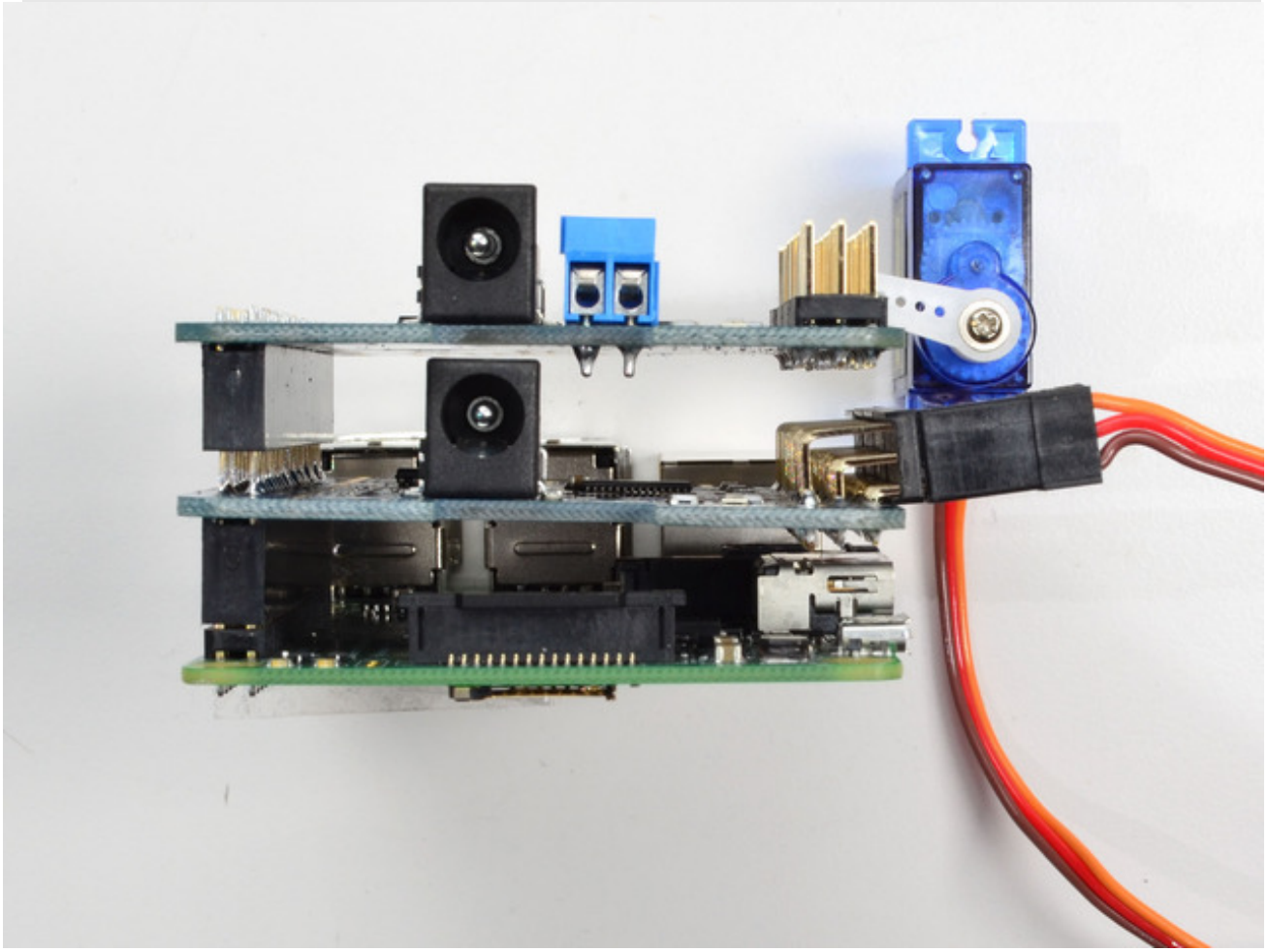
Example

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/**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 millisecond / 4096 = ~0.25 microseconds. If you want a pulse that is 10 microseconds long, divide the time by time-per-tick (10us / 0.25 us = 40) then turn on at tick 0 and turn off at tick 40.

Stacking HATs

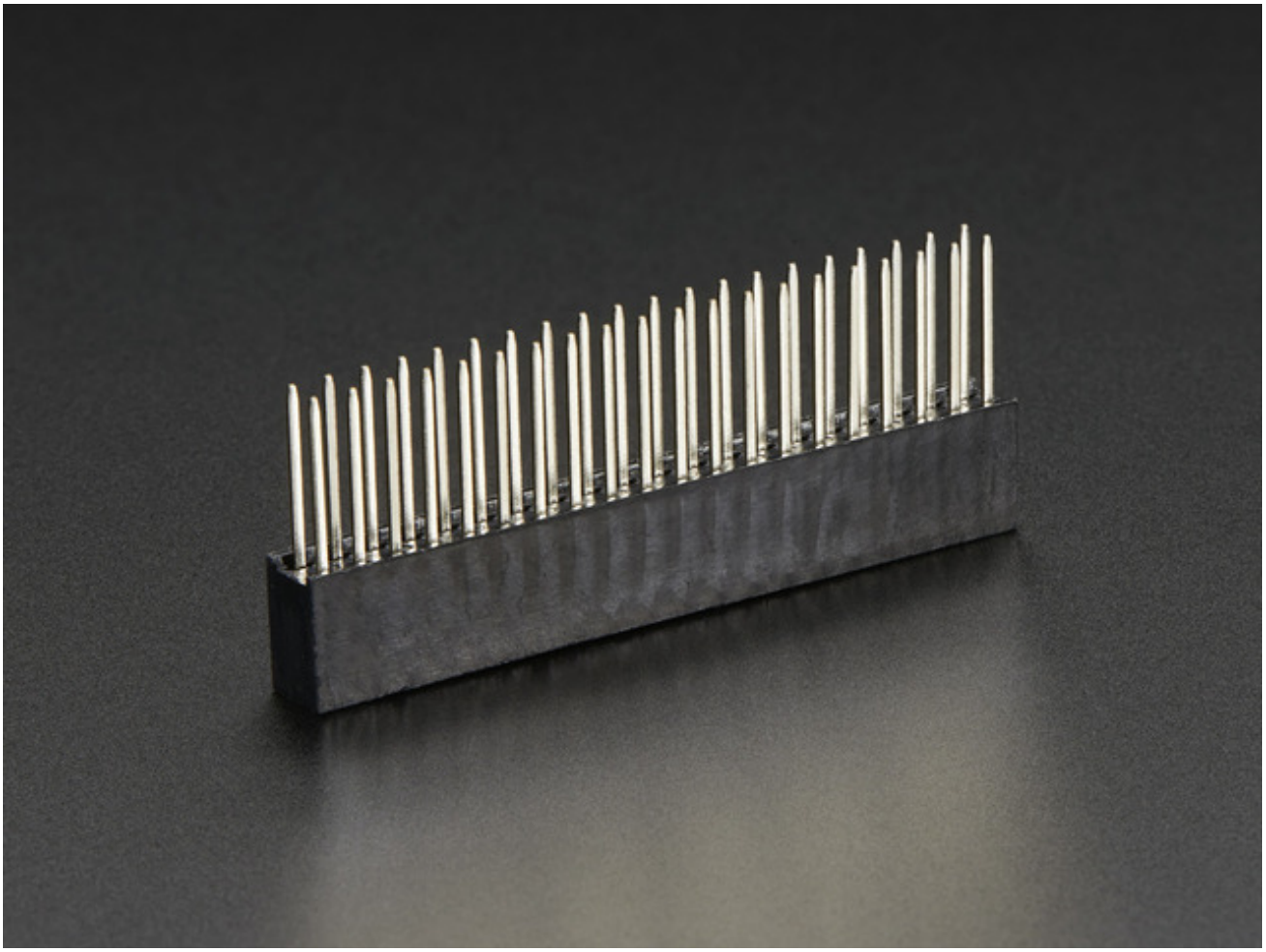


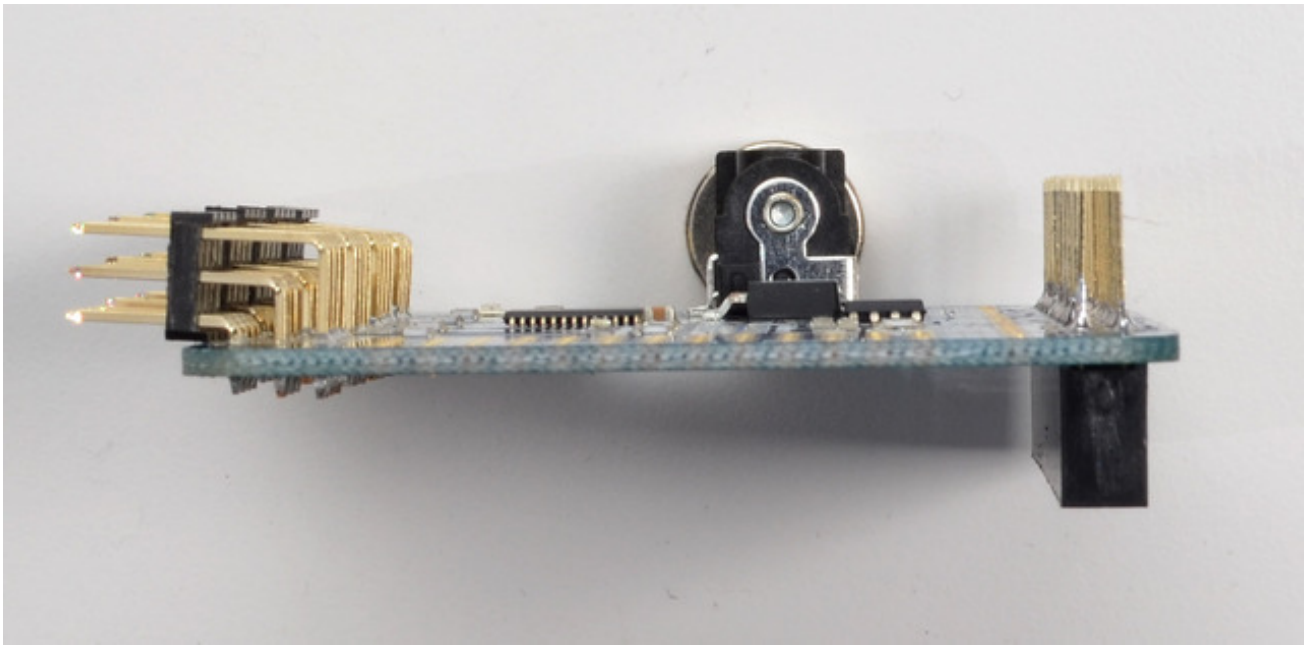
Even though HATs are not intended to be stacked, you can stack up to 62 HATs and not have an address collision, for up to 992 PWM outputs! You'll still need to provide power and write code for all those outputs but they can all share the same SDA/SCL pins no problem.

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

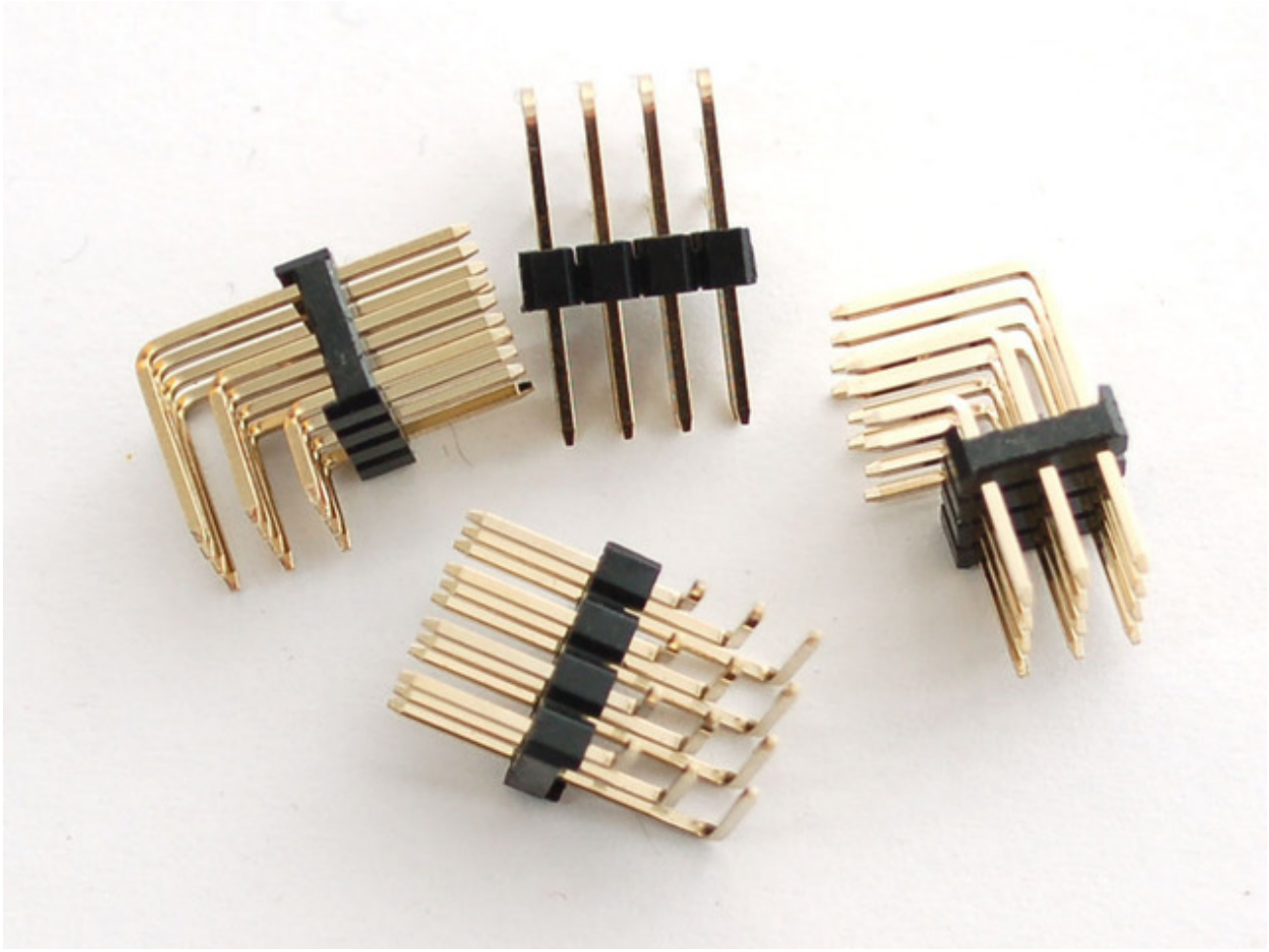
Extra Parts

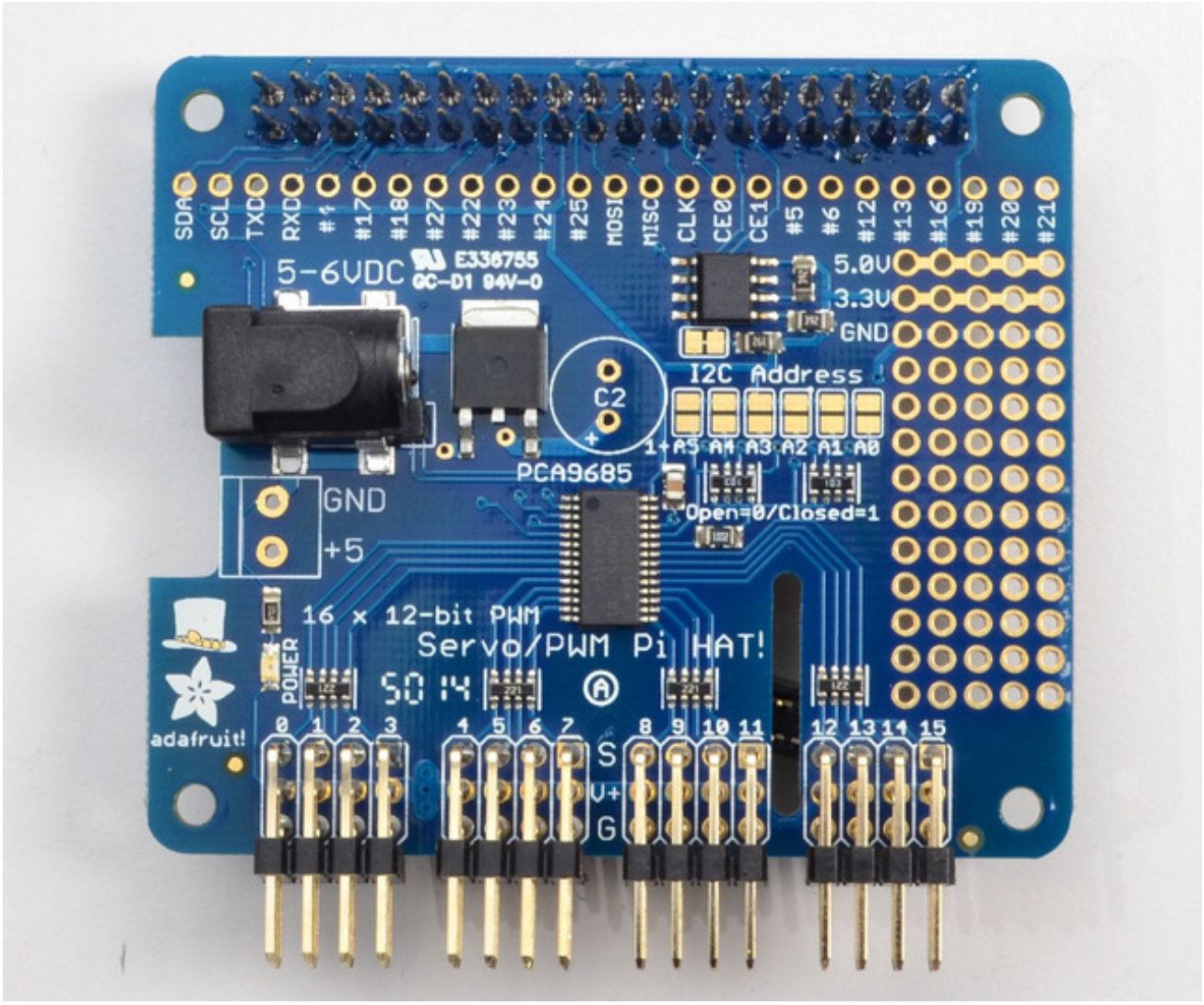
If you want to stack HATs on top of this one, make sure you pick up a HAT-stacking header (http://adafru.it/ejT) and solder them instead of the plain 2x20 header that comes in the kit





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**



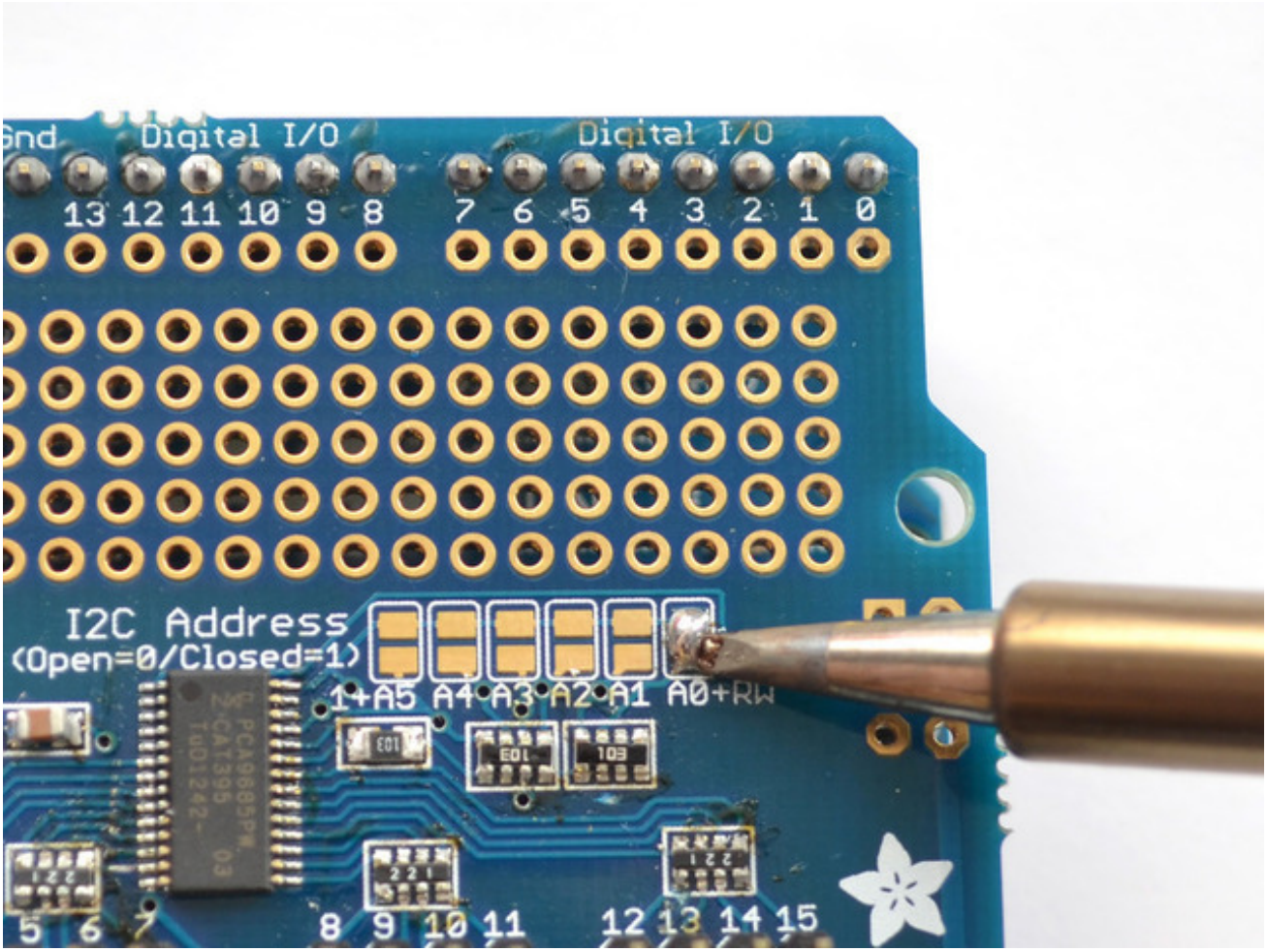


Addressing the HATs

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);**