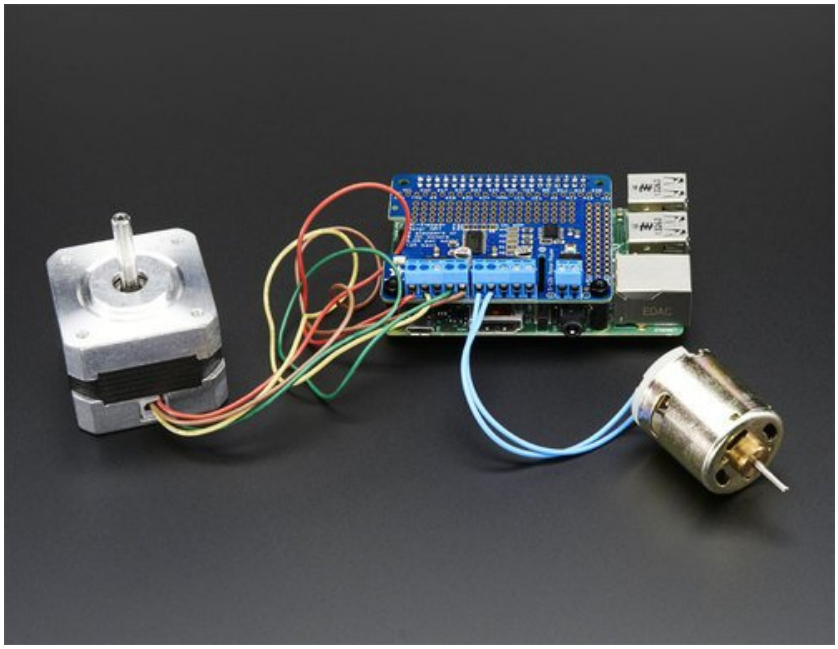
adafruit learning system

Adafruit DC and Stepper Motor HAT for Raspberry Pi

Created by lady ada



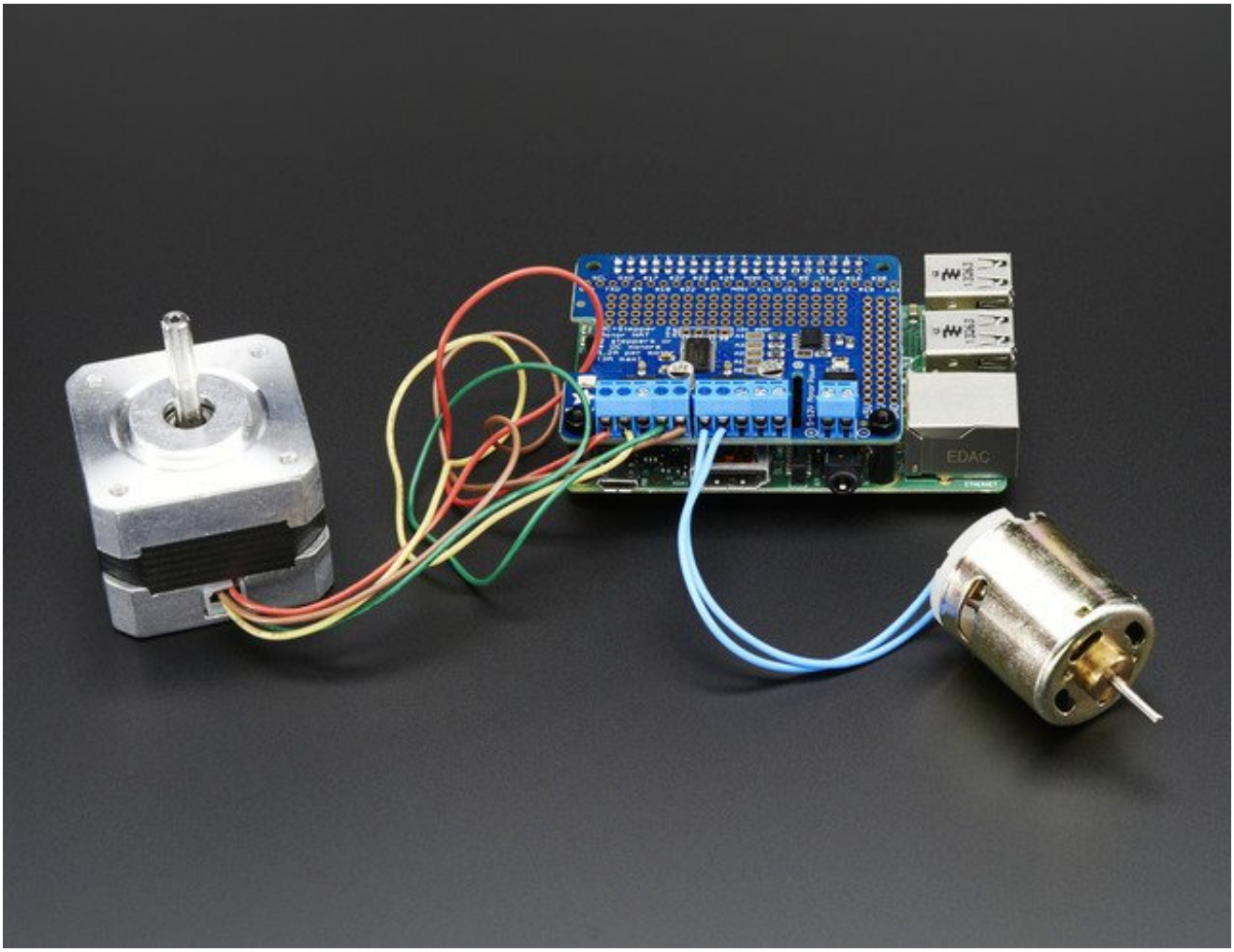
Last updated on 2016-05-27 01:27:44 PM EDT

Guide Contents

Guide Contents 2
Overview 4
Assembly 9
Solder on Headers and Terminal Block 9
And Solder! 9
Powering Motors 14
Voltage requirements: 14
Current requirements: 14
Power it up 16
Installing Software 17
Enable I2C 17
Downloading the Code from Github 17
Using DC Motors 19
Connecting DC Motors 19
DC motor control walkthru 20
Creating the DC motor object 21
Setting DC Motor Speed 21
Setting DC Motor Direction 21
Using Stepper Motors 23
Connecting Stepper Motors 23
Stepper motor control walkthru 24
Creating the Stepper motor object 25
Stepping 25
step() - blocking steps 27
Using "Non-blocking" oneStep() 28
Stacking HATs 29
Addressing the HATs 30
Stacking in Code 31
Downloads 32
Motor ideas and tutorials 32
Datasheets 32

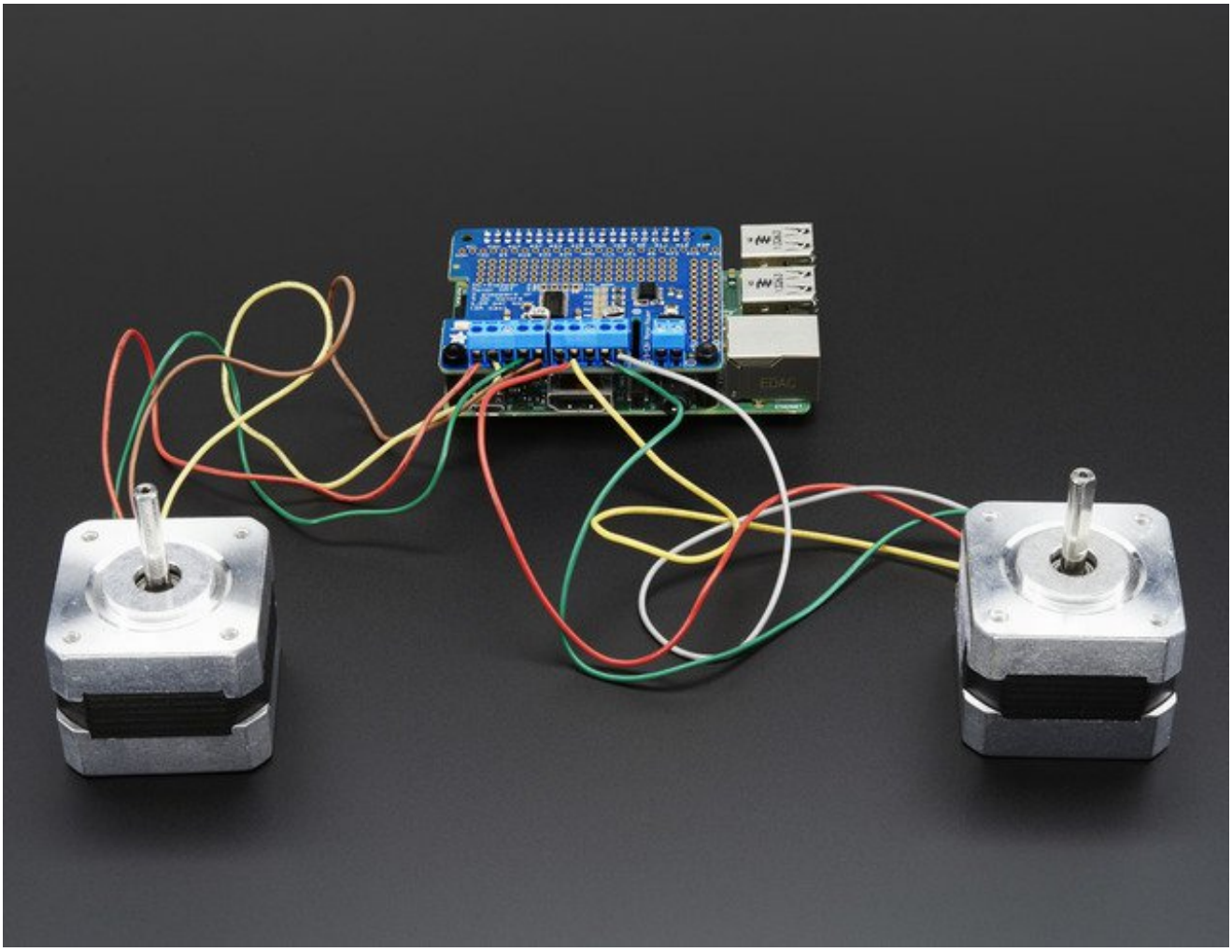
Schematic 32
Fabrication Print 32

Overview



Let your robotic dreams come true with the new DC+Stepper Motor HAT from Adafruit. This Raspberry Pi add-on is perfect for any motion project as it can drive up to 4 DC or 2 Stepper motors with full PWM speed control.

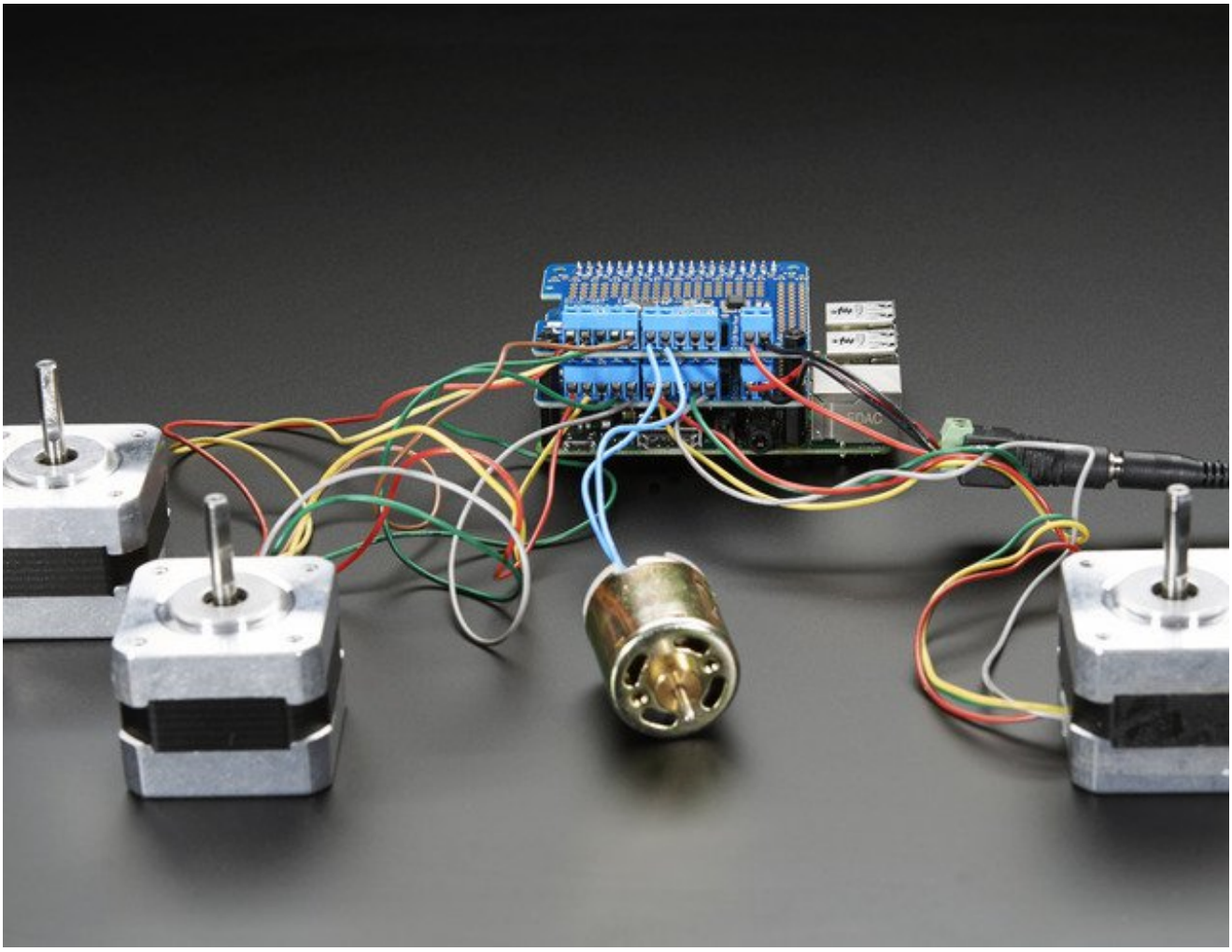
Raspberry Pi and motors are not included



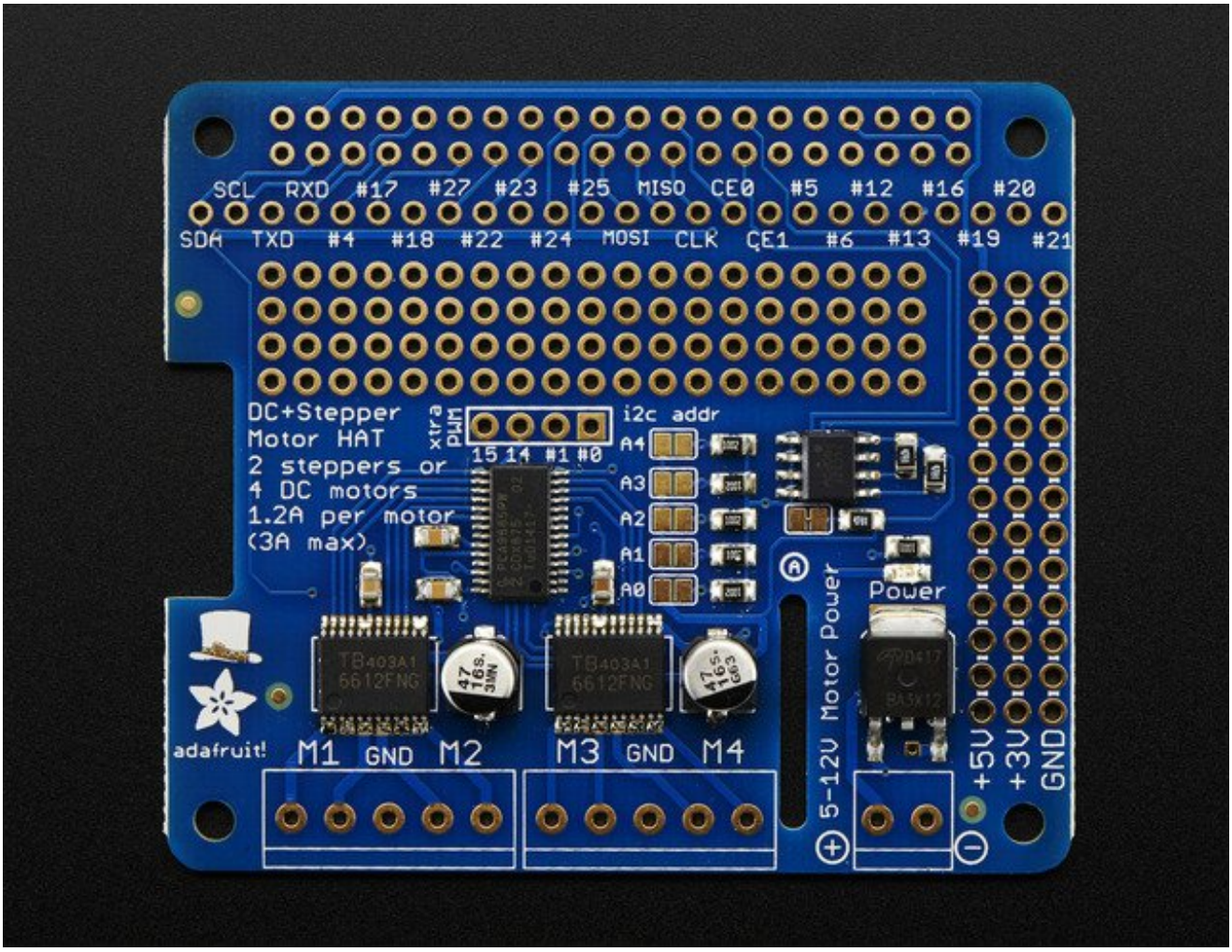
Since the Raspberry Pi does not have a lot of PWM pins, we use a **fully-dedicated PWM driver chip** onboard to both control motor direction and speed. This chip handles all the motor and speed controls over I2C. Only two GPIO pins (SDA & SCL) are required to drive the multiple motors, and since it's I2C you can also connect any other I2C devices or HATs to the same pins.

In fact, **you can even stack multiple Motor HATs**, up to 32 of them, for controlling up to 64 stepper motors or 128 DC motors - just remember to purchase and solder in a stacking header instead of the one we include.

> If connecting the Hat off-the Pi via jumpers, you will need to connect GND and 3.3v in addition to SDA and SCL.



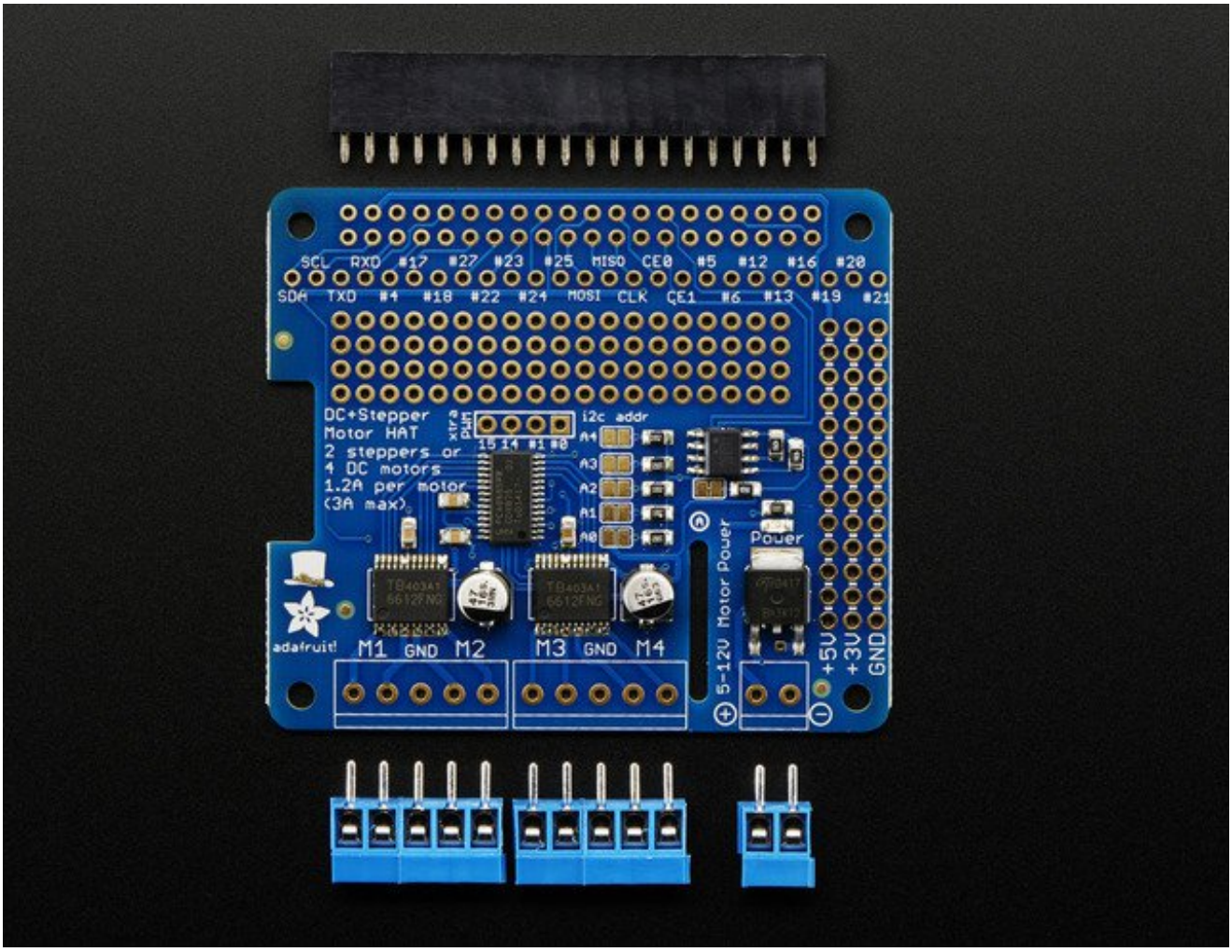
Motors are controlled TB6612 MOSFET driver: with **1.2A per channel current capability** (20ms long bursts of 3A peak)**,** a big improvement over L293D drivers and there are built-in flyback diodes as well.



We even had a little space so we added a polarity protection FET on the power pins and a bit of prototyping area. And the HAT is assembled and tested here at Adafruit so all you have to do is solder on the included 2x20 plain header and the terminal blocks.

Lets check out these specs again:

- 4 H-Bridges: TB6612 chipset provides **1.2A per bridge** (3A brief peak) with thermal shutdown protection, internal kickback protection diodes. Can run motors on 4.5VDC to 13.5VDC.
- **Up to 4 bi-directional DC** motors with individual 8-bit speed selection (so, about 0.5% resolution)
- **Up to 2 stepper motors** (unipolar or bipolar) with single coil, double coil, interleaved or micro-stepping.
- Big terminal block connectors to easily hook up wires (18-26AWG) and power
- Polarity protected 2-pin terminal block and jumper to connect external 5-12VDC power
- Works best with Raspberry Pi model B+ and A+, but can be used with a model A or B if you purchase a 2x13 extra-tall header and solder that instead of the 2x20 (http://adafru.it/eib)
- Install the easy-to-use Python library, check out the examples and you're ready to go!



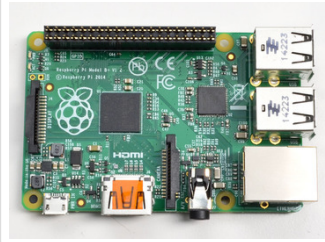
Comes with an assembled & tested HAT, terminal blocks, and 2x20 plain header. Some soldering is required to assemble the headers on. Stacking header not included, but we sell them in the shop so if you want to stack HATs, please pick one up at the same time. (http://adafru.it/2223)

Raspberry Pi and motors are not included but we have lots of motors in the shop and all our DC motors, and stepper motors work great.

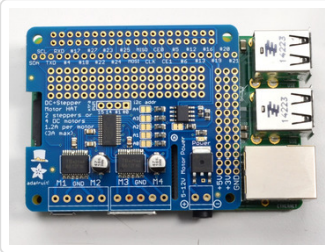
Assembly

Solder on Headers and Terminal Block

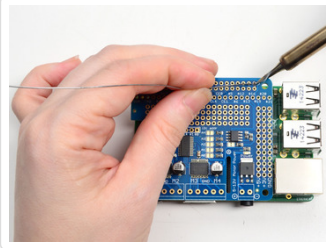
Before we can motorin' there's a little soldering to be done. This step will attach the 2x20 socket header so that we can plug this HAT into a Raspberry Pi, and the terminal blocks so you can attach external power and motors.

- 

  Start by plugging the 2x20 header into a Raspberry Pi, this will keep the header stable while you solder. Make sure the Pi is powered down!

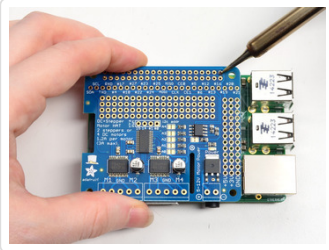
- 

  Place the HAT on top so that the short pins of the 2x20 header line up with the pads on the HAT

- 

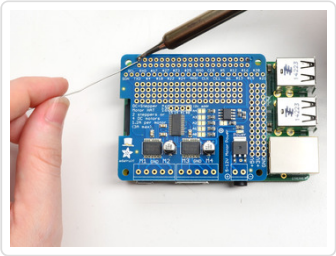
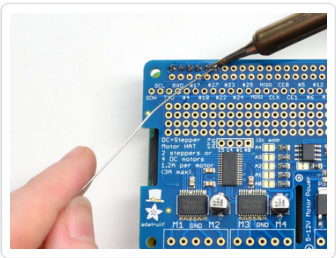
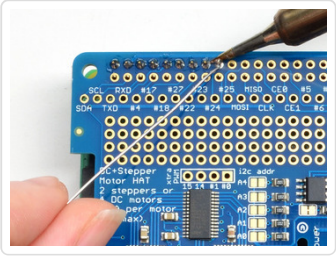
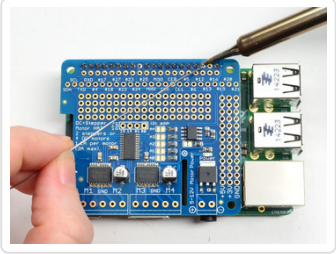
  ### And Solder!

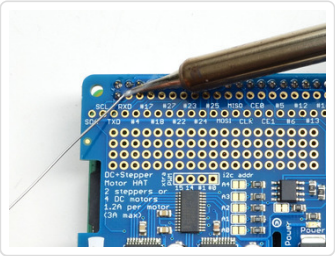
  Heat up your iron and solder in one header connection on the right.

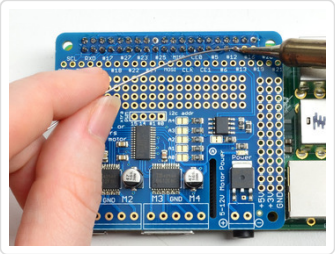
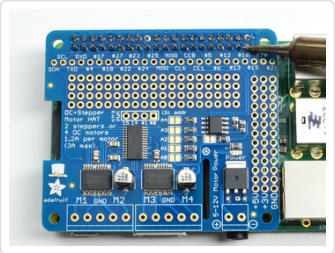
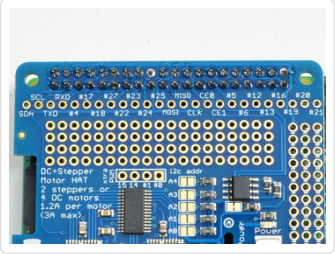
- 

  Once it is soldered, put down the solder and reheat the solder point with your iron while straightening the HAT so it isn't leaning down

(For tips on soldering, be sure to check out our Guide to Excellent Soldering (http://adafru.it/aTk).

- 
  Solder one point on the opposite side of the connector
- 
- 
  Solder each of the connections for the top row
- 

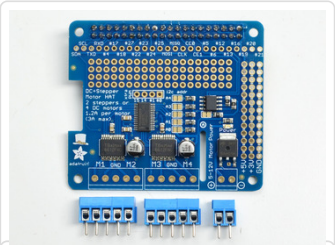
- 
-

  Flip the board around and solder all the connections for the other half of the 2x20 header
- 
- 
- 

  Check over your work so far, make sure each solder point is shiny, and isn't bridged or dull or cracked

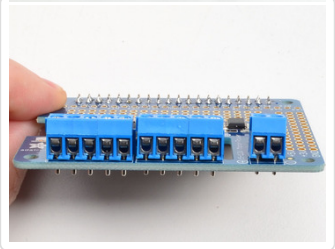
Now grab the 3.5mm-spaced terminal blocks. These will let you quickly connect up your motor and power supply using only a screw

- driver



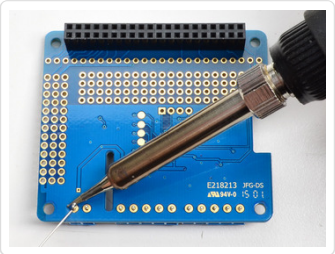
You will have 3 x 2-pin terminal blocks and 2 x 3-pin terminal blocks

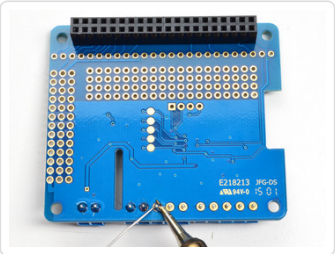
Slide each of the 3-pin terminal blocks into a 2-pin to create two 5-pin blocks



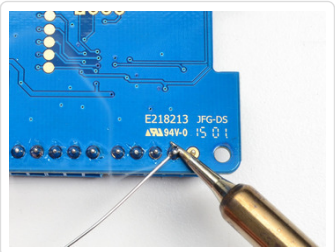
Slide the terminal blocks along the edge of the HAT, so that the 'mouth' of each block is facing out.

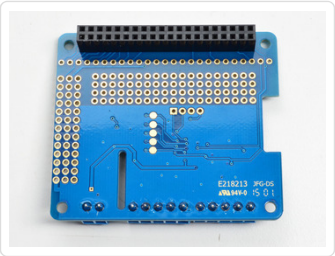
You can use scotch or other plain tape to keep the terminal blocks flat against the PCB while you solder





Flip over the board and solder in all of the terminal block pins



- 

  Check over your work so far, make sure each solder point is shiny, and isn't bridged or dull or cracked

  You're done! You can now move onto the software side

Powering Motors

Motors need a lot of energy, especially cheap motors since they're less efficient.

Voltage requirements:

The first important thing to figure out what voltage the motor is going to use. If you're lucky your motor came with some sort of specifications. Some small hobby motors are only intended to run at 1.5V, but its just as common to have 6-12V motors. The motor controllers on this HAT are designed to run from **5V to 12V**.

MOST 1.5-3V MOTORS WILL NOT WORK or will be damaged by 5V power

Current requirements:

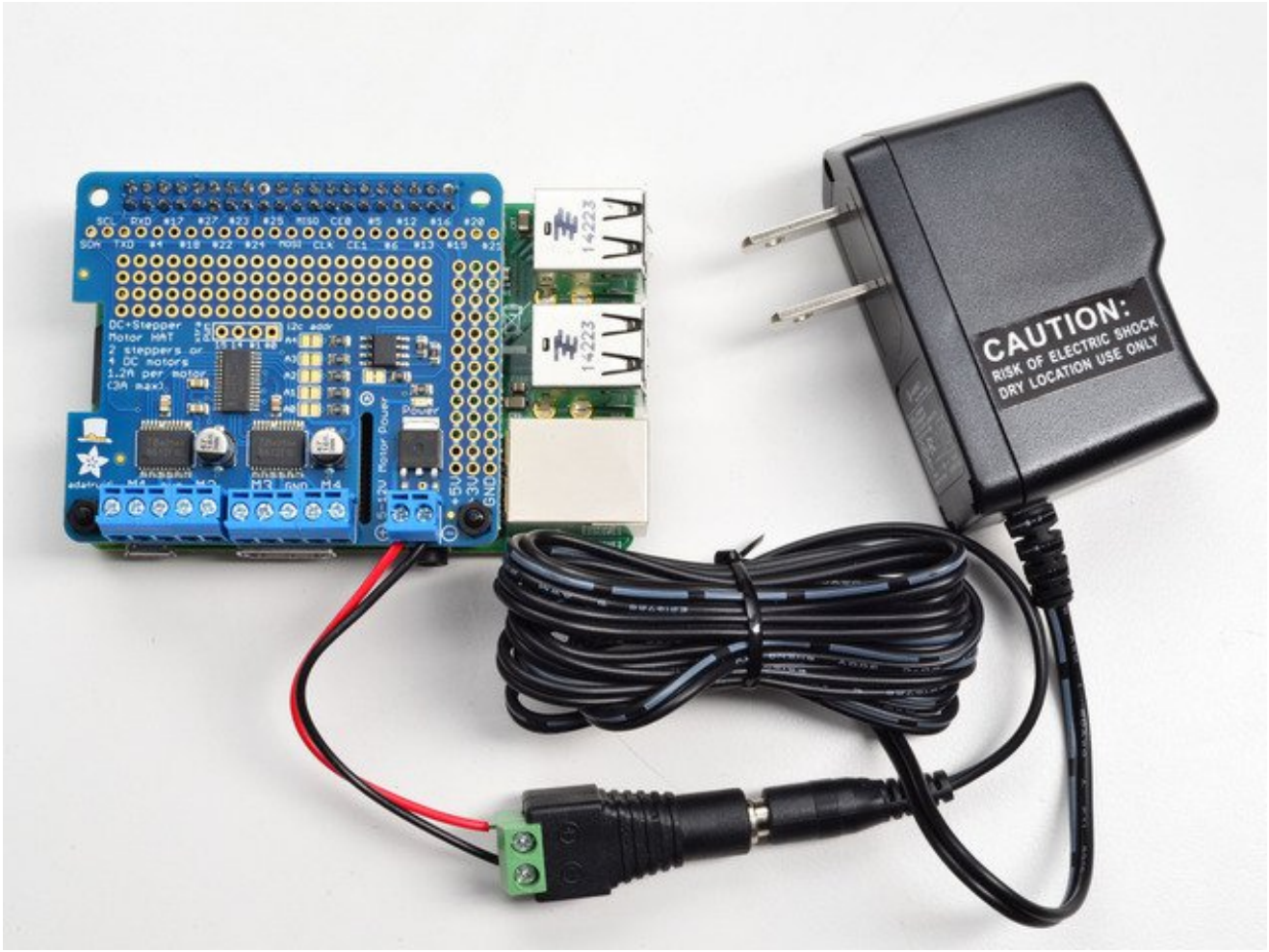
The second thing to figure out is how much current your motor will need. The motor driver chips that come with the kit are designed to provide up to 1.2 A per motor, with 3A peak current. Note that once you head towards 2A you'll probably want to put a heat-sink on the motor driver, otherwise you will get thermal failure, possibly burning out the chip.

> You can't run motors off of a 9V battery so don't waste your time/batteries!

Honestly, for portable we recommend you use a big Lead Acid or multiple-AA NiMH battery pack - use 4 to 8 batteries to vary the voltage from about 6V to 12V as your motors require

If you don't have to take your project on the go, a 9V 1A (http://adafru.it/dO6), 12V 1A (http://adafru.it/eny), or 12V 5A w (http://adafru.it/enz)ill work nicely

99% of 'weird motor problems' are due to having a voltage mismatch (too low a voltage, too high a voltage) or not having a powerful enough supply! *Even small DC motors can draw up to 3 Amps when they stall.*



Power it up

Wire up your battery pack to the Power terminal block on the right side of the HAT. It is polarity protected but still its a good idea to check your wire polarity. Once the HAT has the correct polarity, you'll see the green LED light up

Please note the HAT does not power the Raspberry Pi, and we strongly recommend having two seperate power supplies - one for the Pi and one for the motors, as motors can put a lot of noise onto a power supply and it could cause stability problems!

Installing Software

We have a Python library you can use to control DC and stepper motors, its probably the easiest way to get started, and python has support for multithreading which can be really handy when running multiple stepper motors at onces!

Enable I2C

You will have to make I2C support work on your Pi before you begin, visit our tutorial to enable I2C in the kernel! (http://adafru.it/dEO)

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

Downloading the Code from Github

The easiest way to get the code onto your Pi is to hook up an Ethernet cable or with a WiFi setup, 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"):

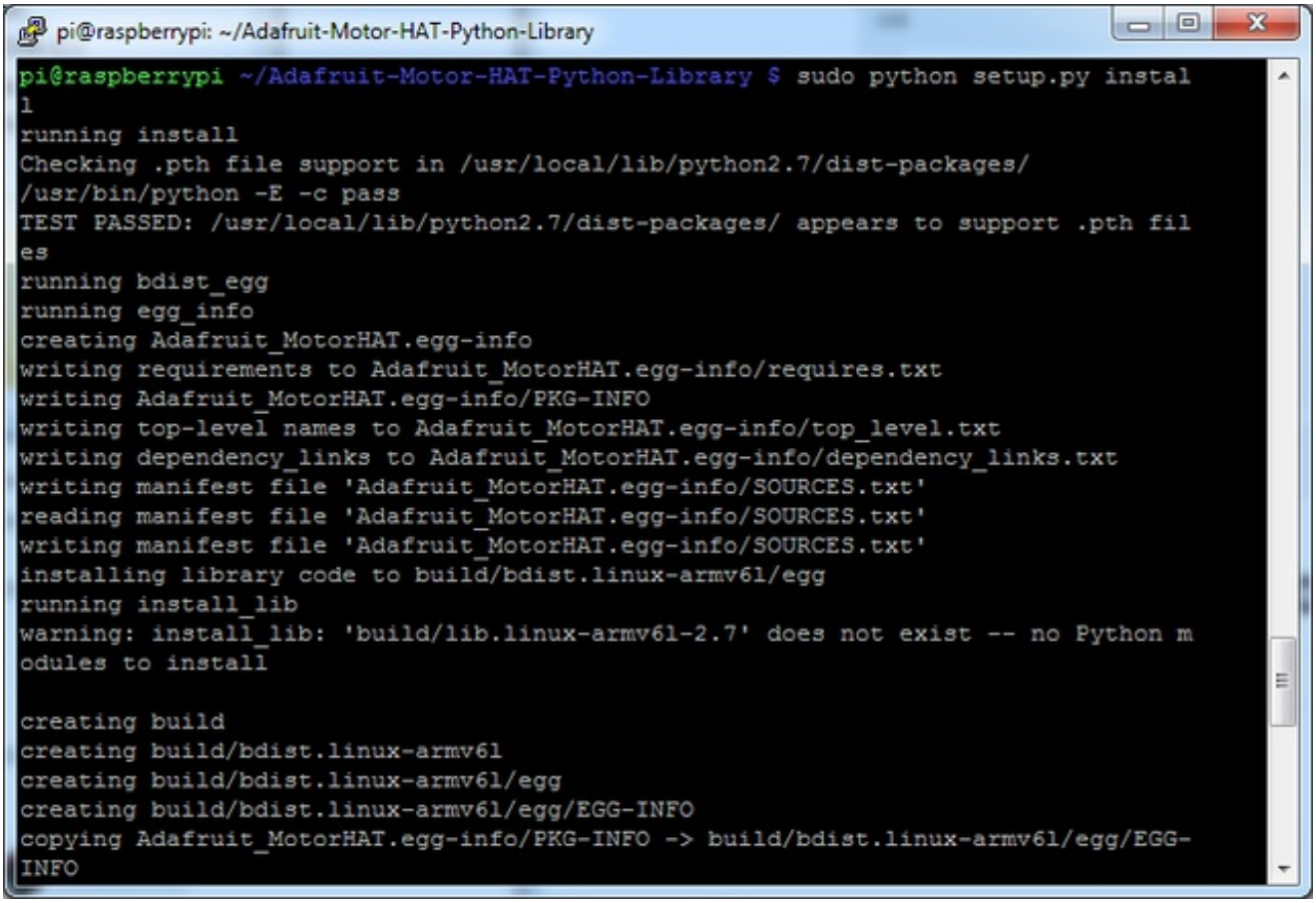
```
git clone https://github.com/adafruit/Adafruit-Motor-HAT-Python-Library.git
cd Adafruit-Motor-HAT-Python-Library
```

Install python-dev if you havent already:

```
sudo apt-get install python-dev
```

Now install it with the setup install command:

```
sudo python setup.py install
```



That's it! Now you can get started with testing. Run

cd examples

from within the Motor HAT library folder, we have a couple examples to demonstrate the different types of motors and configurations. The next few pages will explain them

Using DC Motors

DC motors are used for all sort of robotic projects.

The Motor HAT can drive up to 4 DC motors bi-directionally. That means they can be driven forwards and backwards. The speed can also be varied at 0.5% increments using the high-quality built in PWM. This means the speed is very smooth and won't vary!

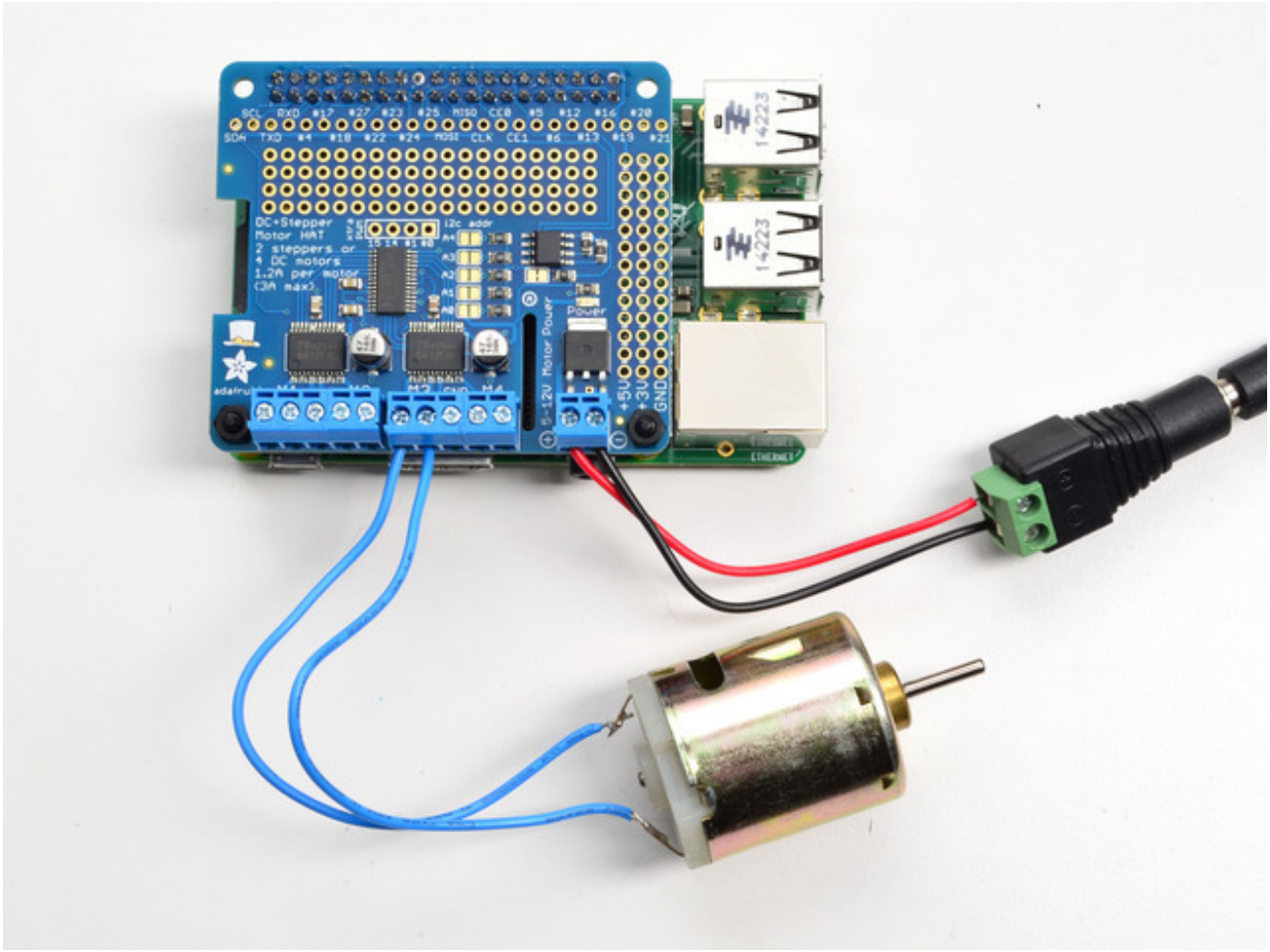
Note that the H-bridge chip is not meant for driving continuous loads over 1.2A or motors that peak over 3A, so this is for small motors. Check the datasheet for information about the motor to verify its OK!

Connecting DC Motors

To connect a motor, simply solder two wires to the terminals and then connect them to either the **M1**, **M2**, **M3**, or **M4.** If your motor is running 'backwards' from the way you like, just swap the wires in the terminal block

For this demo, please connect it to **M3**

Now go into the Adafruit-Motor-HAT-Python/examples folder and run **sudo python DCTest.py** to watch your motor spin back and forth.



DC motor control walkthru

Here's a walkthru of the code which shows you everything the MotorHAT library can do and how to do it.

Start with importing at least these libraries:

```
#!/usr/bin/python
from Adafruit_MotorHAT import Adafruit_MotorHAT, Adafruit_DCMotor

import time
import atexit
```

The MotorHAT library contains a few different classes, one is the MotorHAT class itself which is the main PWM controller. You'll always need to create an object, and set the address. By default the address is 0x60 (see the stacking HAT page on why you may want to change the address)

```
# create a default object, no changes to I2C address or frequency
mh = Adafruit_MotorHAT(addr=0x60)
```

The PWM driver is 'free running' - that means that even if the python code or Pi linux kernel crashes, the PWM driver will still continue to work. This is good because it lets the Pi focus on linuxy things while the PWM driver does its PWMy things. **But it means that the motors DO NOT STOP when the python code quits**

For that reason, we strongly recommend this 'at exit' code when using DC motors, it will do its best to shut down all the motors.

```
# recommended for auto-disabling motors on shutdown!
def turnOffMotors():
 mh.getMotor(1).run(Adafruit_MotorHAT.RELEASE)
 mh.getMotor(2).run(Adafruit_MotorHAT.RELEASE)
 mh.getMotor(3).run(Adafruit_MotorHAT.RELEASE)
 mh.getMotor(4).run(Adafruit_MotorHAT.RELEASE)

atexit.register(turnOffMotors)
```

Creating the DC motor object

OK now that you have the motor HAT object, note that each HAT can control up to 4 motors. And you can have multiple HATs!

To create the actual DC motor object, you can request it from the MotorHAT object you created above with **getMotor(*num*)** with a value between 1 and 4, for the terminal number that the motor is attached to

```
myMotor = mh.getMotor(3)
```

DC motors are simple beasts, you can basically only set the speed and direction.

Setting DC Motor Speed

To set the speed, call **setSpeed(*speed*)** where speed varies from 0 (off) to 255 (maximum!). This is the PWM duty cycle of the motor

```
# set the speed to start, from 0 (off) to 255 (max speed)
myMotor.setSpeed(150)
```

Setting DC Motor Direction

To set the direction, call **run(*direction*)** where *direction* is a constant from one of the following:

- **Adafruit_MotorHAT.FORWARD** - DC motor spins forward
- **Adafruit_MotorHAT.BACKWARD** - DC motor spins forward

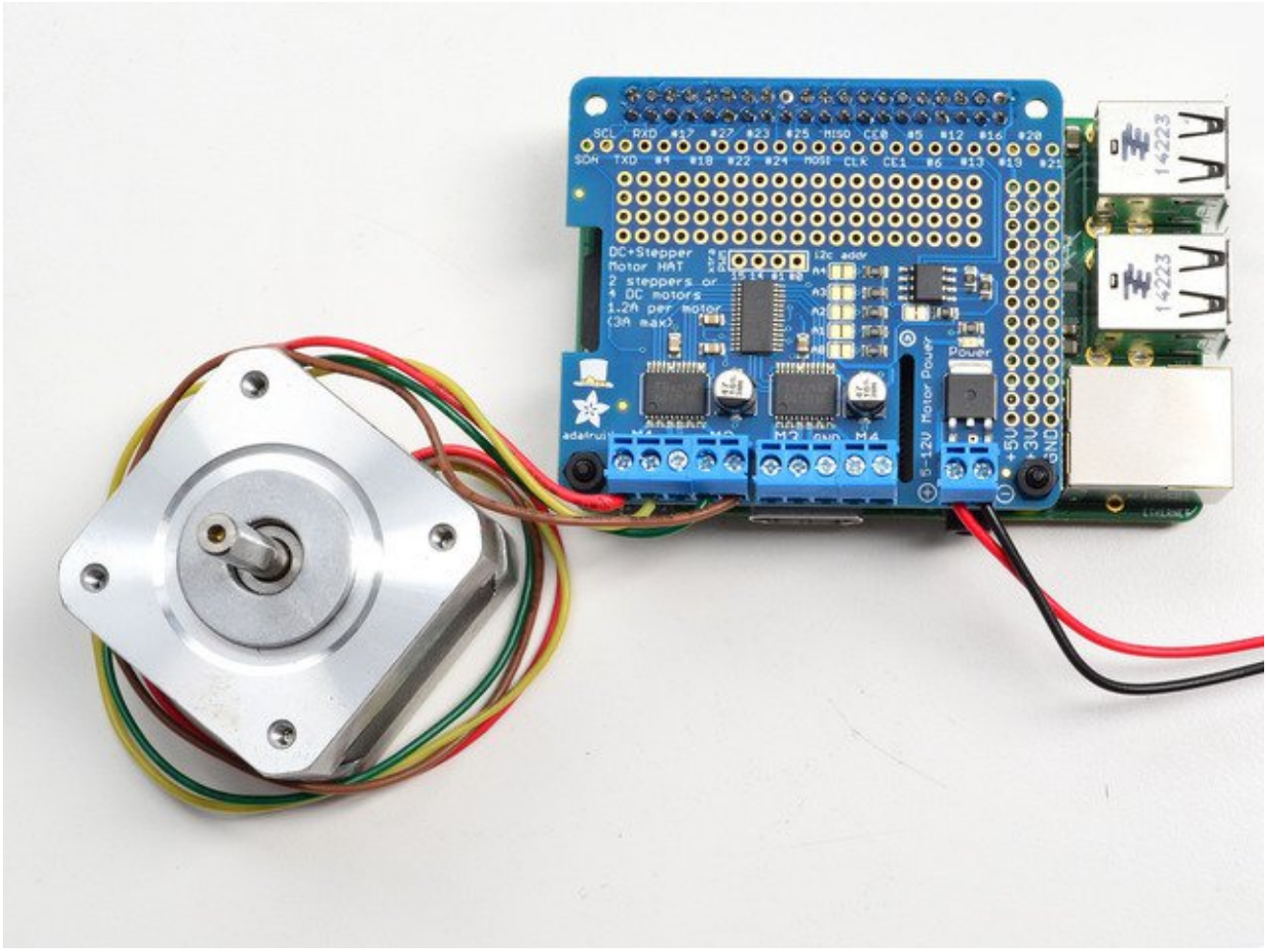
- **Adafruit_MotorHAT.RELEASE** - DC motor is 'off', not spinning but will also not hold its place.

```
while (True):
 print "Forward! "
 myMotor.run(Adafruit_MotorHAT.FORWARD)

 print "\tSpeed up..."
 for i in range(255):
  myMotor.setSpeed(i)
  time.sleep(0.01)

 print "\tSlow down..."
 for i in reversed(range(255)):
  myMotor.setSpeed(i)
  time.sleep(0.01)

 print "Backward! "
 myMotor.run(Adafruit_MotorHAT.BACKWARD)

 print "\tSpeed up..."
 for i in range(255):
  myMotor.setSpeed(i)
  time.sleep(0.01)

 print "\tSlow down..."
 for i in reversed(range(255)):
  myMotor.setSpeed(i)
  time.sleep(0.01)

 print "Release"
 myMotor.run(Adafruit_MotorHAT.RELEASE)
 time.sleep(1.0)
```

Using Stepper Motors

Stepper motors are great for (semi-)precise control, perfect for many robot and CNC projects. This HAT supports up to 2 stepper motors. The python library works identically for bi-polar and uni-polar motors

Running a stepper is a little more intricate than running a DC motor but its still very easy



Connecting Stepper Motors

For unipolar motors: to connect up the stepper, first figure out which pins connected to which coil, and which pins are the center taps. If its a 5-wire motor then there will be 1 that is the center tap for both coils. Theres plenty of tutorials online on how to reverse engineer the coils pinout. (http://adafru.it/aOO) The center taps should both be connected together to the **GND** terminal on the Motor HAT output block. then coil 1 should connect to one motor port (say **M1** or **M3**) and coil 2 should connect to the other motor port (**M2** or **M4**).

For bipolar motors: its just like unipolar motors except theres no 5th wire to connect to ground. The code is exactly the same.

For this demo, please connect it to **M1 and M2**

Now go into the Adafruit-Motor-HAT-Python/examples folder and run **sudo python StepperTest.py** to watch your stepper motor spin back and forth.

Stepper motor control walkthru

Here's a walkthru of the code which shows you everything the MotorHAT library can do and how to do it.

Start with importing at least these libraries:

```
#!/usr/bin/python
from Adafruit_MotorHAT import Adafruit_MotorHAT, Adafruit_DCMotor, Adafruit_StepperMotor

import time
import atexit
```

The MotorHAT library contains a few different classes, one is the MotorHAT class itself which is the main PWM controller. You'll always need to create an object, and set the address. By default the address is 0x60 (see the stacking HAT page on why you may want to change the address)

```
# create a default object, no changes to I2C address or frequency
mh = Adafruit_MotorHAT(addr=0x60)
```

Even though this example code does not use DC motors, it's still important to note that the PWM driver is 'free running' - that means that even if the python code or Pi linux kernel crashes, the PWM driver will still continue to work. This is good because it lets the Pi focus on linuxy things while the PWM driver does its PWMy things.

Stepper motors will not continue to move when the Python script quits, but it's still strongly recommend that you keep this 'at exit' code, it will do its best to shut down all the motors:

```
# recommended for auto-disabling motors on shutdown!
def turnOffMotors():
    mh.getMotor(1).run(Adafruit_MotorHAT.RELEASE)
    mh.getMotor(2).run(Adafruit_MotorHAT.RELEASE)
    mh.getMotor(3).run(Adafruit_MotorHAT.RELEASE)
    mh.getMotor(4).run(Adafruit_MotorHAT.RELEASE)


atexit.register(turnOffMotors)
```

Creating the Stepper motor object

OK now that you have the motor HAT object, note that each HAT can control up to 2 steppers. And you can have multiple HATs!

To create the actual Stepper motor object, you can request it from the MotorHAT object you created above with **getStepper(*steps, portnum*)** where *steps* is how many steps per rotation for the stepper motor (usually some number between 35 - 200) ith a value between 1 and 2. Port #1 is **M1** and **M2**, port #2 is **M3** and **M4**

```
myStepper = mh.getStepper(200, 1)       # 200 steps/rev, motor port #1
```

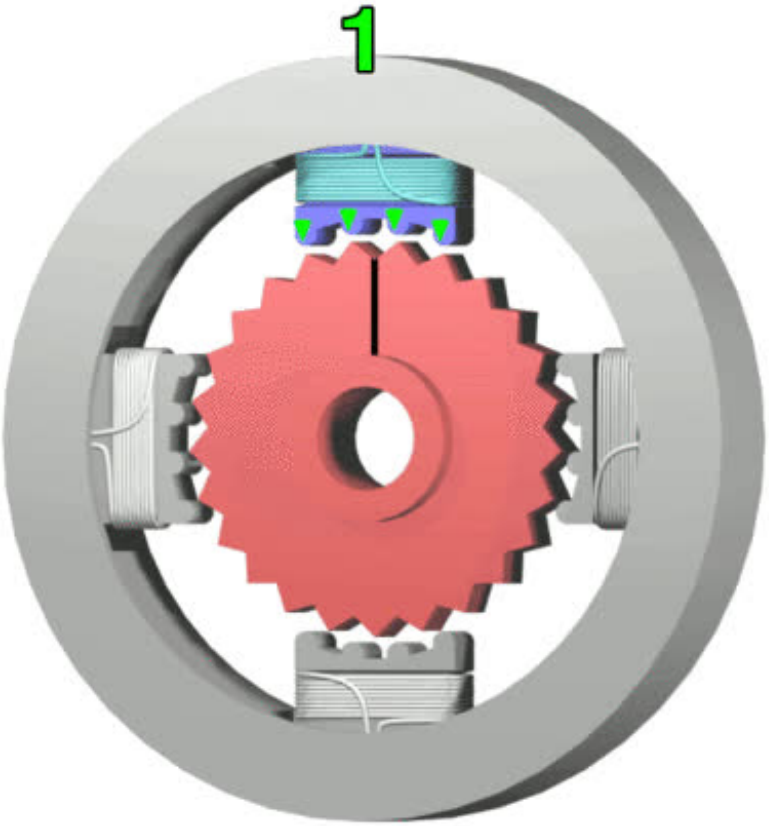
Next, if you are planning to use the 'blocking' **step()** function to take multiple steps at once you can set the speed in RPM. If you end up using oneStep() then this step isn't necessary. Also, the speed is approximate as the Raspberry Pi can't do precision delays the way an Arduino would. Anyways, we wanted to keep the Arduino and Pi versions of this library similar so we kept **setSpeed()** in:

```
myStepper.setSpeed(30)             # 30 RPM
```

Stepping

Stepper motors differ from DC motors in that the controller (in this case, Raspberry Pi) must tick each of the 4 coils in order to make the motor move. Each two 'ticks' is a step. By alternating the coils, the stepper motor will spin all the way around. If the coils are fired in the opposite order, it will spin the other way around.

If the python code or Pi crashes or stops responding, the motor will no longer move. Compare this to a DC motor which has a constant voltage across a single coil for movement.



"StepperMotor" by Wapcaplet; Teravolt. (http://adafru.it/enC)

There are four essential types of steps you can use with your Motor HAT. All four kinds will work with any unipolar or bipolar stepper motor

1. **Single Steps** - this is the simplest type of stepping, and uses the least power. It uses a single coil to 'hold' the motor in place, as seen in the animated GIF above
2. **Double Steps** - this is also fairly simple, except instead of a single coil, it has two coils on at once. For example, instead of just coil #1 on, you would have coil #1 and #2 on at once. This uses more power (approx 2x) but is stronger than single stepping (by maybe 25%)
3. **Interleaved Steps** - this is a mix of Single and Double stepping, where we use single steps interleaved with double. It has a little more strength than single stepping, and about 50% more power. What's nice about this style is that it makes your motor appear to have 2x as many steps, for a smoother transition between steps
4. **Microstepping** - this is where we use a mix of single stepping with PWM to slowly transition between steps. It's slower than single stepping but has much higher precision. We recommend 8 microstepping which multiplies the # of steps your stepper motor has by 8.

```
while (True):
  print("Single coil steps")
  myStepper.step(100, Adafruit_MotorHAT.FORWARD, Adafruit_MotorHAT.SINGLE)
  myStepper.step(100, Adafruit_MotorHAT.BACKWARD, Adafruit_MotorHAT.SINGLE)
```

```
print("Double coil steps")
myStepper.step(100, Adafruit_MotorHAT.FORWARD, Adafruit_MotorHAT.DOUBLE)
myStepper.step(100, Adafruit_MotorHAT.BACKWARD, Adafruit_MotorHAT.DOUBLE)
print("Interleaved coil steps")
myStepper.step(100, Adafruit_MotorHAT.FORWARD, Adafruit_MotorHAT.INTERLEAVE)
myStepper.step(100, Adafruit_MotorHAT.BACKWARD, Adafruit_MotorHAT.INTERLEAVE)
print("Microsteps")
myStepper.step(100, Adafruit_MotorHAT.FORWARD, Adafruit_MotorHAT.MICROSTEP)
myStepper.step(100, Adafruit_MotorHAT.BACKWARD, Adafruit_MotorHAT.MICROSTEP)
```

step() - blocking steps

As you can see above, you can step mulitple steps at a time with **step()**

step(*numberofsteps*, *direction*, *type*)

Where *numbersteps* is the number of steps to take, *direction* is either FORWARD or BACKWARD and *type* is **SINGLE, DOUBLE, INTERLEAVE** or **MICROSTEP**

Note that **INTERLEAVE** will move half the distance of **SINGLE** or **DOUBLE** because there are twice as many steps. And **MICROSTEP** will move 1/8 the distance because each microstep counts as a step!

This is the easiest way to move your stepper but is **blocking** - that means that the Python program is completely busy moving the motor. If you have two motors and you call these two procedures in a row:

```
stepper1.step(100, Adafruit_MotorHAT.FORWARD, Adafruit_MotorHAT.SINGLE)
stepper2.step(100, Adafruit_MotorHAT.BACKWARD, Adafruit_MotorHAT.SINGLE)
```

Then the first stepper will move 100 steps, stop, and *then* the second stepper will start moving.

Chances are you'd like to have your motors moving at once!

For that, you'll need to take advantage of Python's ability to multitask with threads which you can see in **DualStepperTest.py**

The key part of the DualStepperTest example code is that we define a function that will act as a 'wrapper' for the **step()** procedure:

```
    def stepper_worker(stepper, numsteps, direction, style):
#print("Steppin!")
stepper.step(numsteps, direction, style)
#print("Done")
```

We have some commented-out print statements in case you want to do some debugging.

Then, whenever you want to make the first stepper move, you can call:

```
st1 = threading.Thread(target=stepper_worker, args=(myStepper1, numsteps, direction, stepping_style))
st1.start()
```

Which will spin up a background thread to move Stepper1 and will return immediately. You can then do the same with the second stepper:

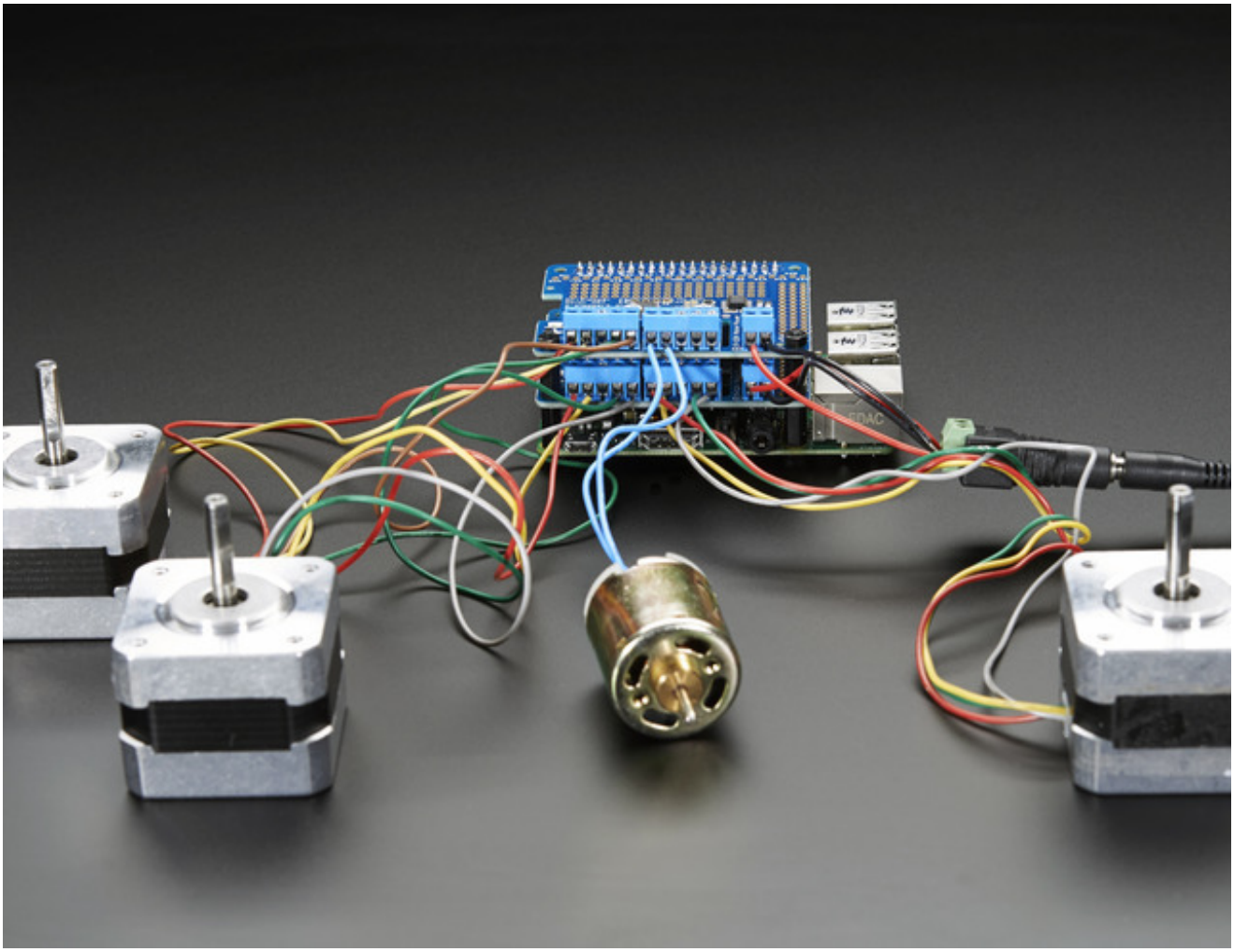
```
st2 = threading.Thread(target=stepper_worker, args=(myStepper2, numsteps, direction, stepping_style))
st2.start()
```

You can tell when the stepper is done moving because the stepper thread you created will 'die' - test it with **st2.isAlive()** or **st2.isAlive()** - if you get **True** that means the stepper is still moving.

Using "Non-blocking" oneStep()

OK lets say you want a lot of control over your steppers, you can use the one `oneStep(direction, stepstyle)` which will make a single step in the style you request, with no delay. This will let you step exactly when you like, for the most control

Stacking HATs



One of the cool things about this HAT design is that it is possible to stack them. Every HAT you stack can control another 2 steppers or 4 DC motors (or a mix of the two)

You can stack up to 32 HAT for a total of 64 steppers or 128 DC motors! Most people will probably just stack two or maybe three but hey, you never know. (PS if you drive 64 steppers from one Raspberry Pi send us a photo, OK?)

If you need to control a bunch of servos as well, you can use our 16-channel servo HAT and stack it with this HAT to add a crazy large # of servos. (http://adafru.it/enA)

Stacking HATs is very easy. Each HAT you want to stack on top of must have stacking headers installed. (http://adafru.it/2223) The top HAT does not have to have stacking headers unless you eventually want to put something on top of it.

The only thing to watch for when stacking HATs is every HAT must have a unique I2C address. The

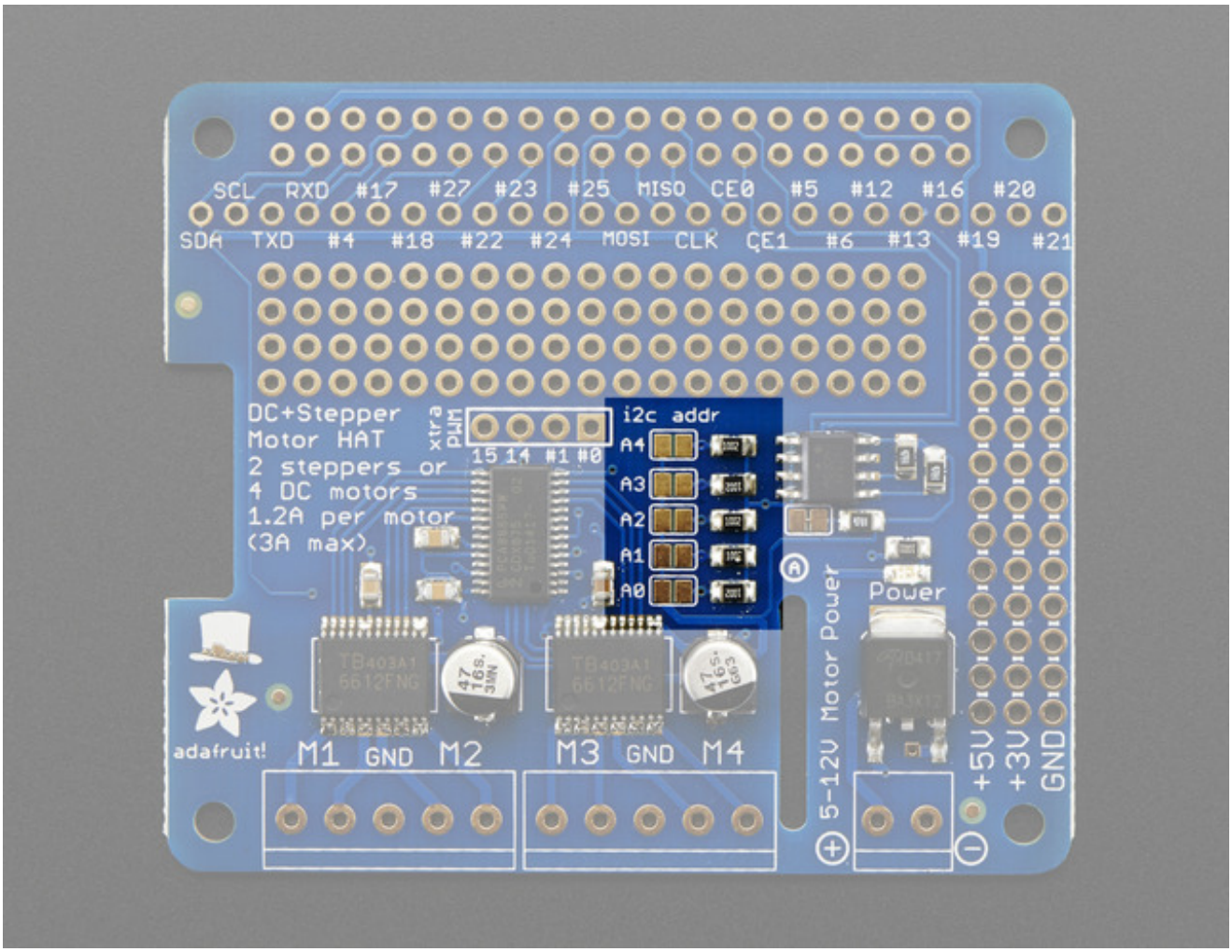
default address is **0x60**. You can adjust the address of the motor HATs to range from 0x60 to 0x80 for a total of 32 unique addresses.

Addressing the HATs

Each board in the stack must be assigned a unique address. This is done with the address jumpers on the left side of the board. The I2C base address for each board is 0x60. 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.

The bottom-most jumper is address bit #0, then the one above of that is address bit #1, etc up to address bit #5



Board 0: Address = 0x60 Offset = binary 0000 (no jumpers required)
Board 1: Address = 0x61 Offset = binary 0001 (bridge A0)
Board 2: Address = 0x62 Offset = binary 0010 (bridge A1, the one above A0)
Board 3: Address = 0x63 Offset = binary 0011 (bridge A0 & A1, two bottom jumpers)

Board 4: Address = 0x64 Offset = binary 0100 (bridge A2, middle jumper)

etc.

> Note that address 0x70 is the "all call" address for the controller chip on the HAT. All boards will respond to address 0x70 - regardless of the address jumper settings.

Stacking in Code

We have an example in our Python library called StackingTest.py

The key part is to create two MotorHAT objects:

```
# bottom hat is default address 0x60
bottomhat = Adafruit_MotorHAT(addr=0x60)
# top hat has A0 jumper closed, so it's address 0x61
tophat = Adafruit_MotorHAT(addr=0x61)
```

then use other of those for getting motors. We recommend using threading for control of the steppers to make them turn together

Downloads

Motor ideas and tutorials

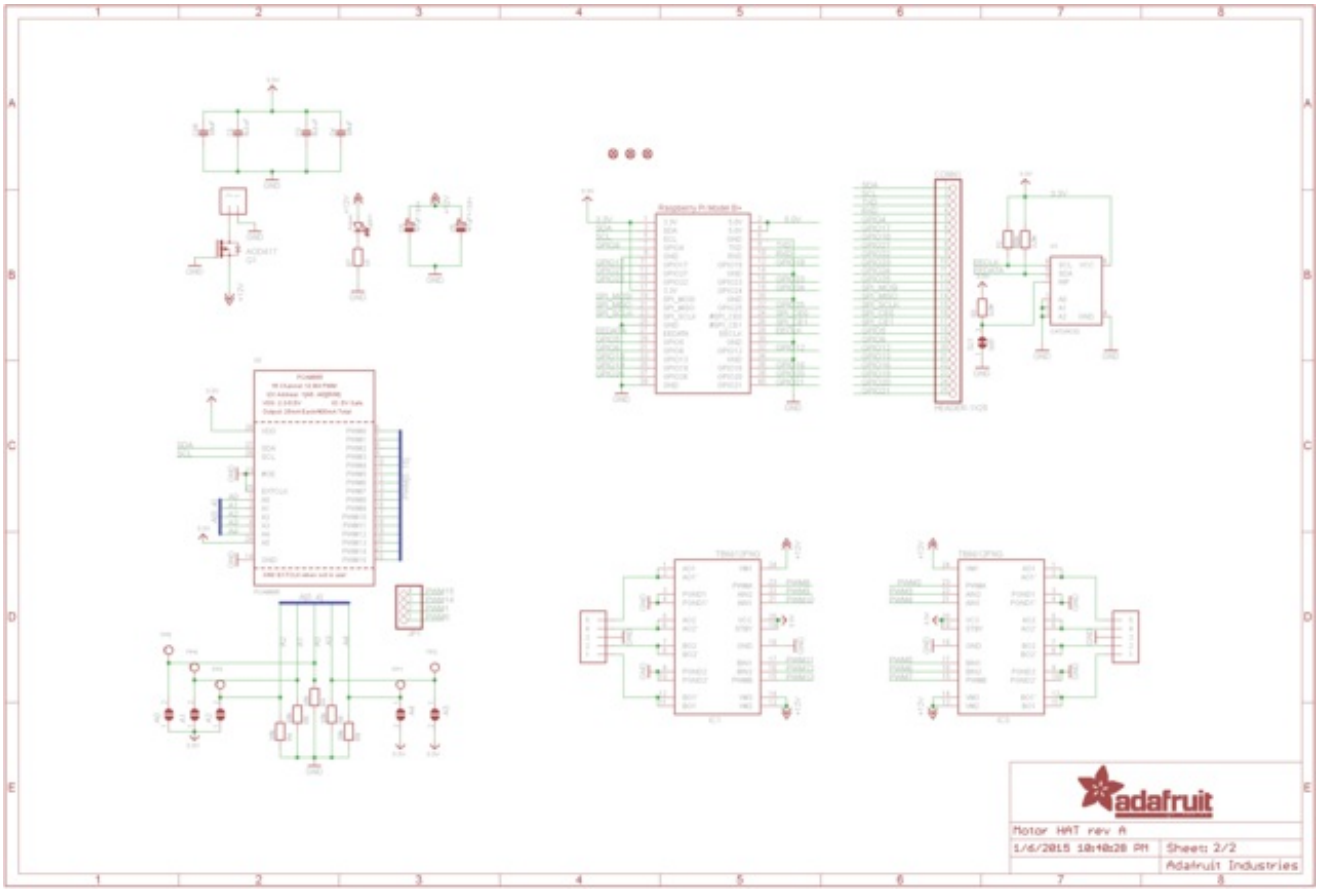
- Wikipedia has tons of information (http://adafru.it/aOF) on steppers
- Jones on stepper motor types (http://adafru.it/aOH)
- Jason on reverse engineering the stepper wire pinouts (http://adafru.it/aOI)

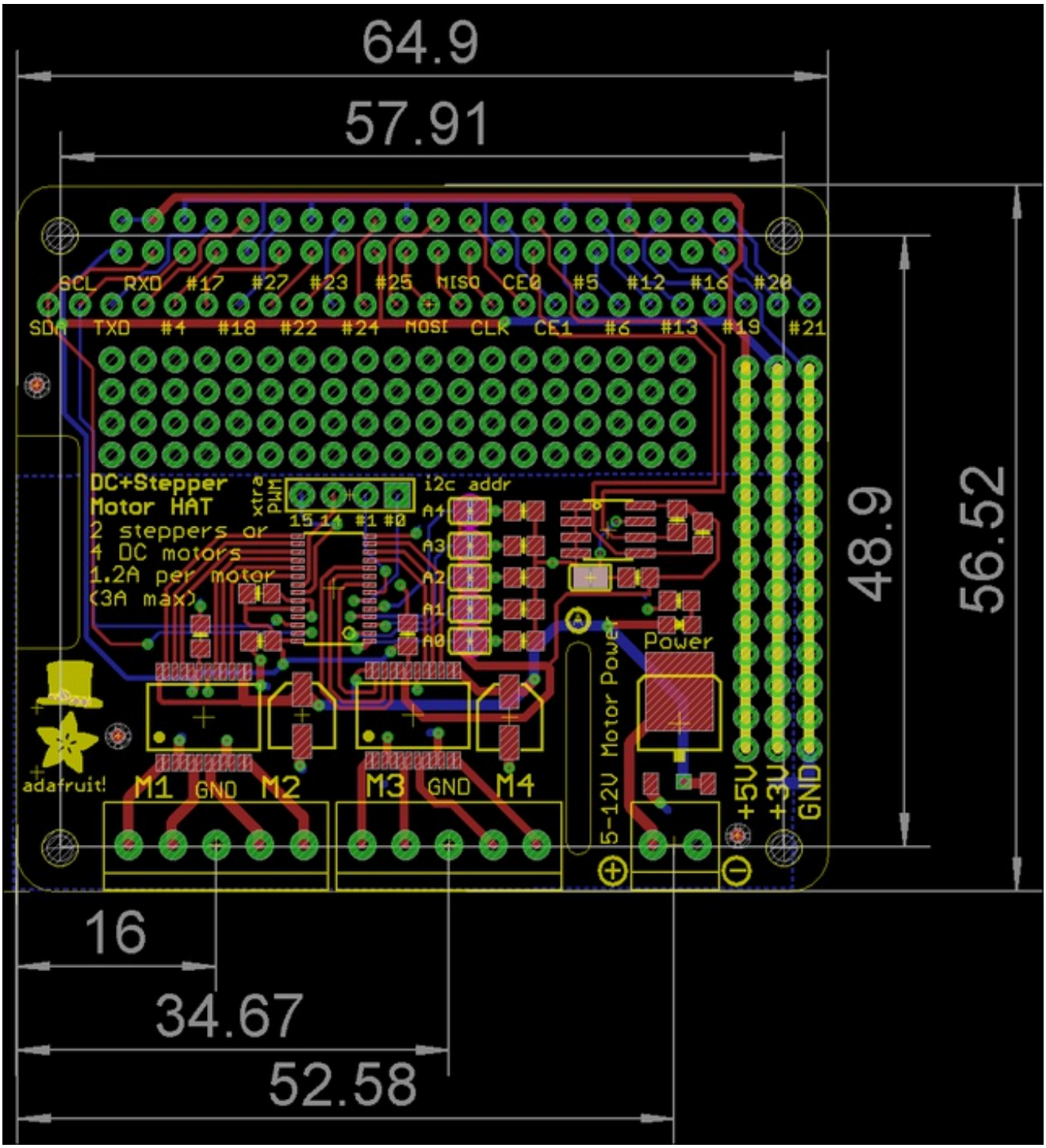
Datasheets

- PCA9685 PWM driver (http://adafru.it/emJ)
- TB6612 Motor driver (http://adafru.it/emK)

Schematic

Click to embiggen



Fabrication Print

This is a 'HAT mechanical standard' compatible!