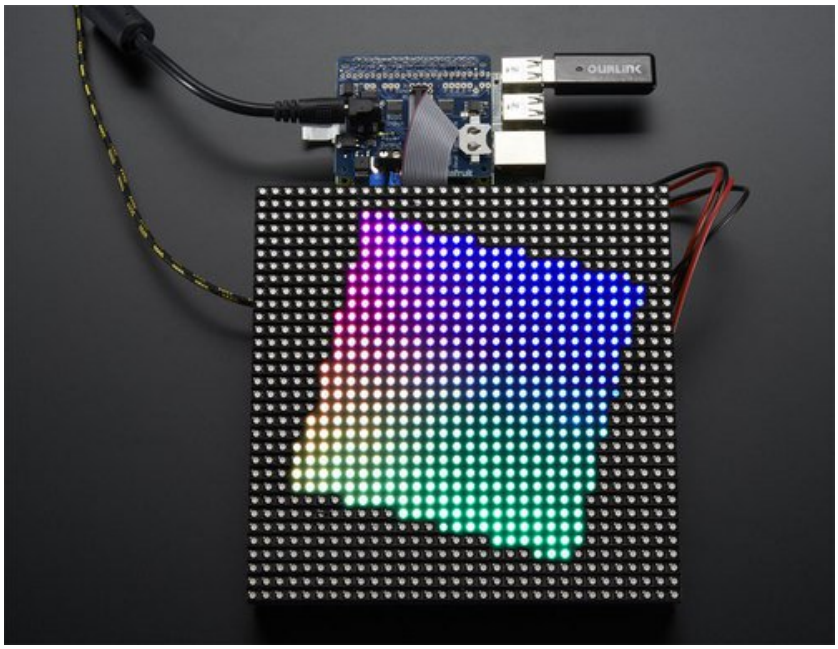
Adafruit RGB Matrix + Real Time Clock HAT for Raspberry Pi

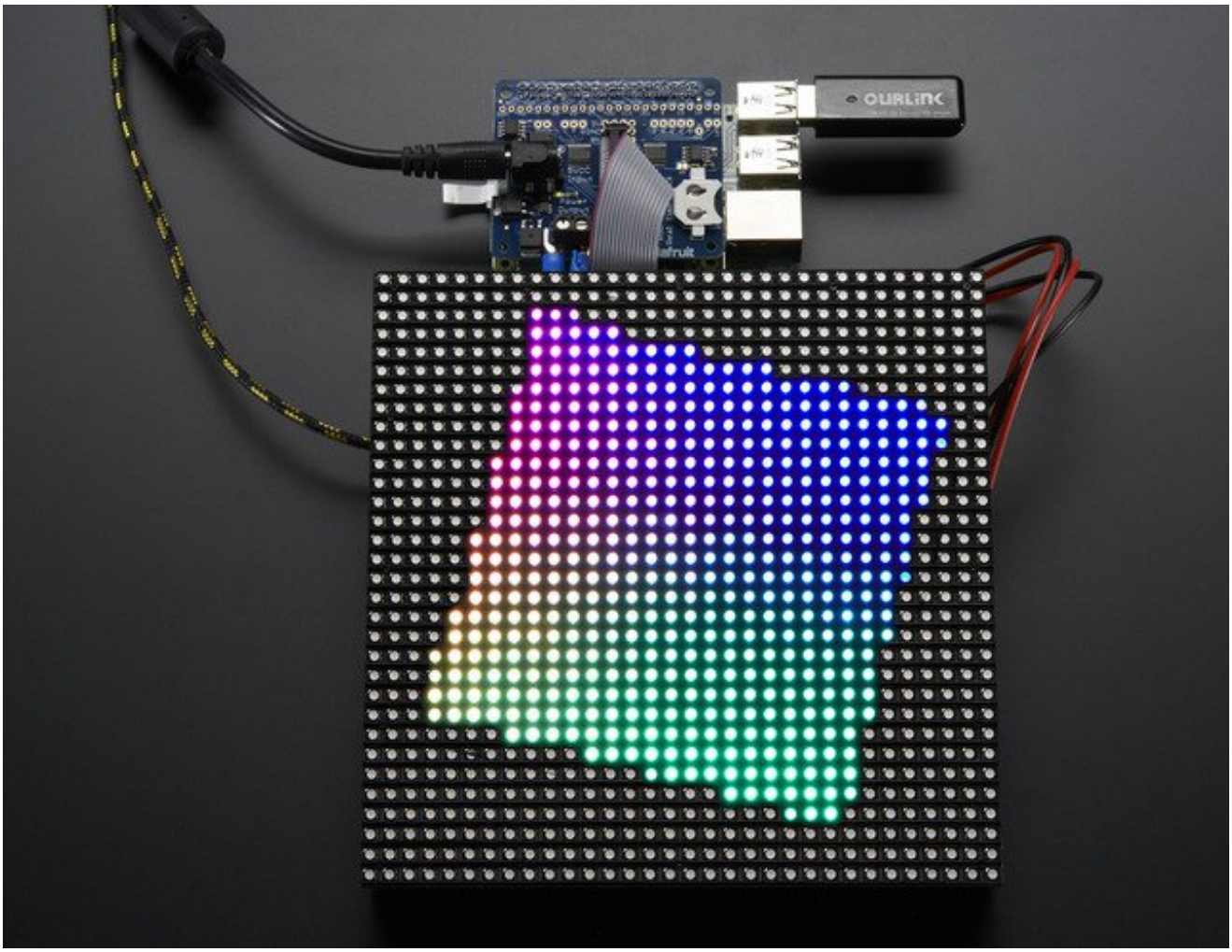
Created by lady ada



Last updated on 2016-10-04 10:31:30 PM UTC

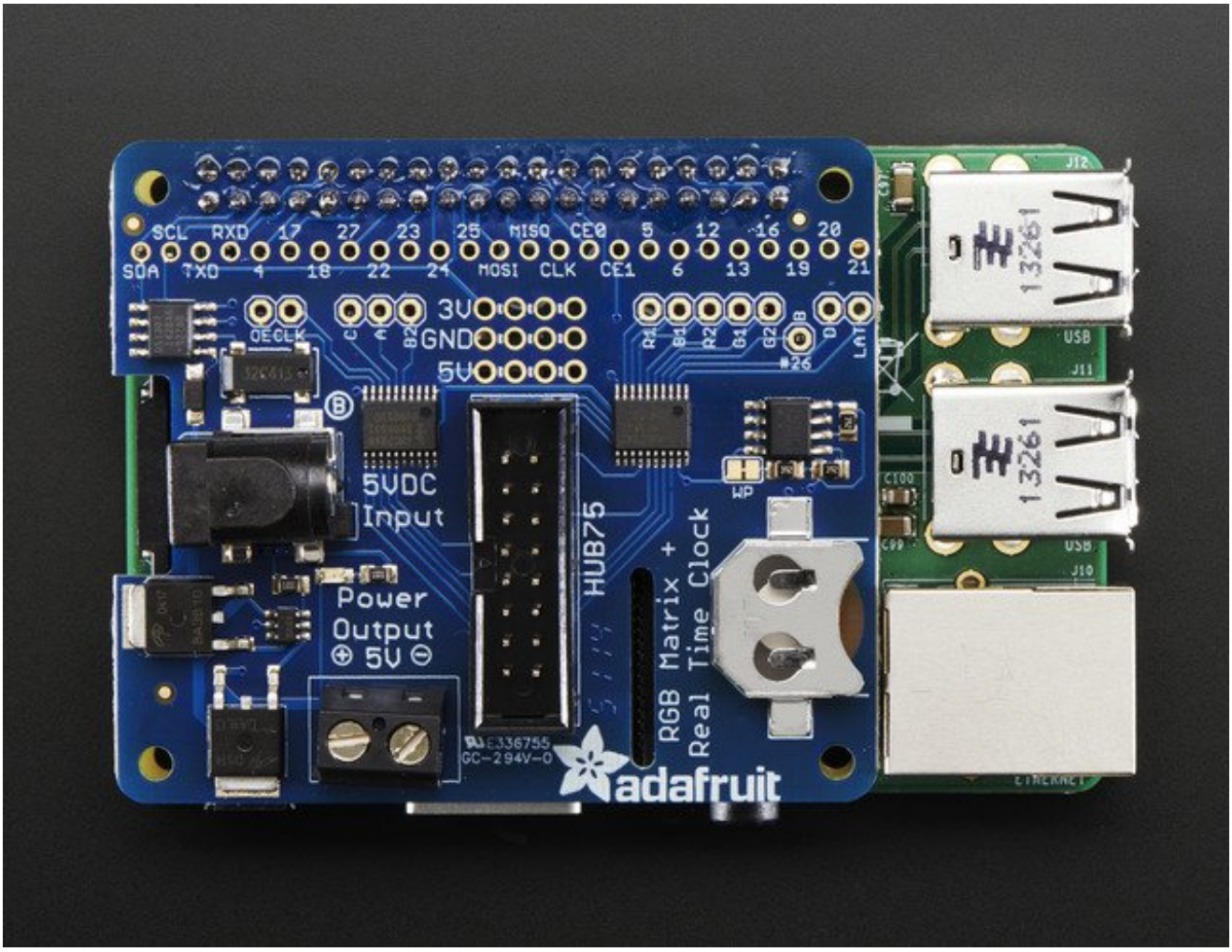
Guide Contents

Overview



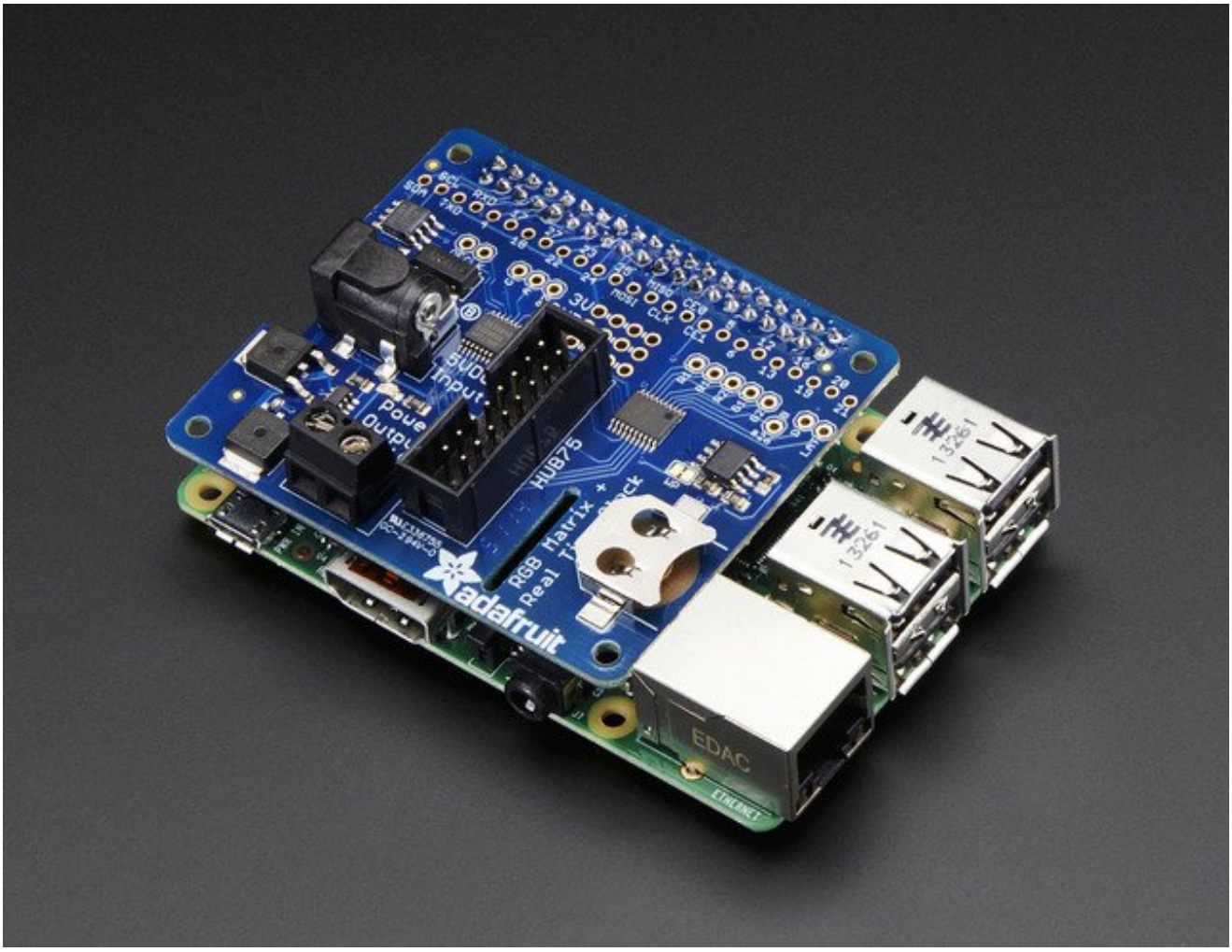
You can now create a dazzling display with your Raspberry Pi with the Adafruit RGB Matrix HAT. This HAT plugs into your Pi and makes it super easy to control RGB matrices such as those we stock in the shop and create a colorful scrolling display or mini LED wall with ease.

“HAT” boards work on any Raspberry Pi with a 40-pin GPIO header — Model A+, B+, Pi 2 and Pi 3. They do not work with older 26-pin boards like the original Model A or B. Pi Zero is possible *if* you solder a header on the Pi board; it’s normally unpopulated on that model.



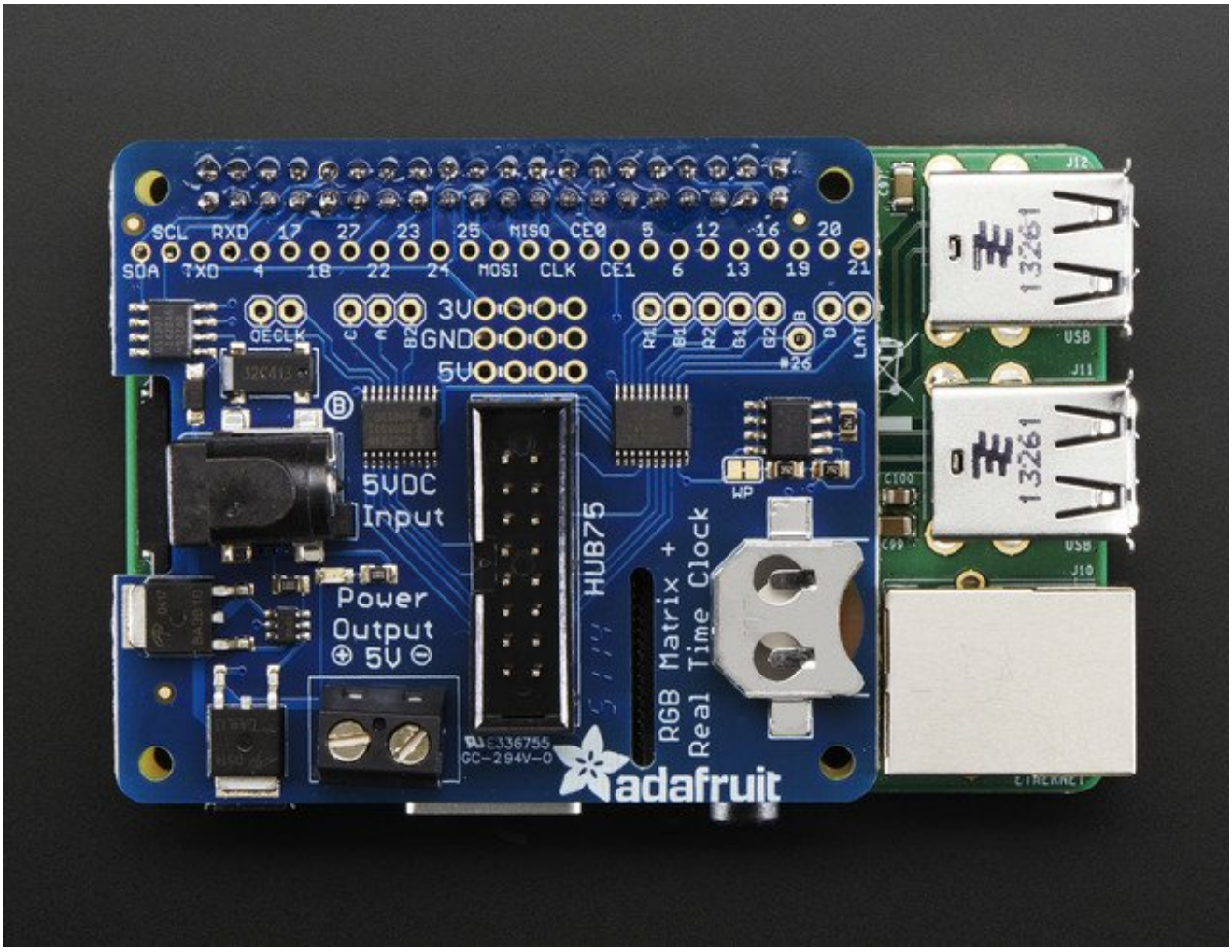
This HAT is our finest to date, full of some really great circuitry. Let me break it down for you:

- **Simple design** - plug in power, plug in IDC cable, run our Python code!
- **Power protection circuitry** - you can plug a 5V 4A wall adapter into the HAT and it will automatically protect against negative, over or under-voltages! Yay for no accidental destruction of your setup.
- **Onboard level shifters** to convert the RasPi's 3.3V to 5.0V logic for clean and glitch free matrix driving
- **DS1307 Real Time Clock** can keep track of time for the Pi even when it is rebooted or powered down, to make for really nice time displays



Works with any of our [16x32, 32x32 or 32x64 RGB LED Matrices with HUB75 connections](http://adafru.it/emd). You can even chain multiple matrices together for a longer display - we've only tested up to 32x128, the bigger the display the harder it is on the Pi so keep that in mind!

Please note: this HAT is only for use with HUB75 type RGB Matrices. Not for use with NeoPixel, DotStar, or other 'addressable' LEDs.



Each order comes with a HAT PCB with all surface mount parts assembled, a 2x20 female socket connector, a 2 pin terminal block, and a 2x8 IDC socket connector. A CR1220 coin cell is not included to make air shipping easier, please order one seperately (http://adafru.it/em8) if you do not have one and would like to use the real time clock.

RGB Matrix is not included, please check out our fine selection (http://adafru.it/emd)!

A 5V power supply is also required, not included, for power the matrix itself, the Pi cannot do it, to calculate the power, multiply the width of all the chained matrices * 0.12 Amps : A 32 pixel wide matrix can end up drawing 32*0.12 = 3.85A so pick up a 5V 4A power supply (http://adafru.it/e50).

Raspberry Pi not included (but we have 'em in the shop so pick one up, Model A+, B+, Pi 2 or Pi 3) (http://adafru.it/eme)

Some light soldering is required to attach the headers to your Pi. A soldering iron and solder are required, but it's a simple soldering job and most beginners can do it in about 15

minutes.

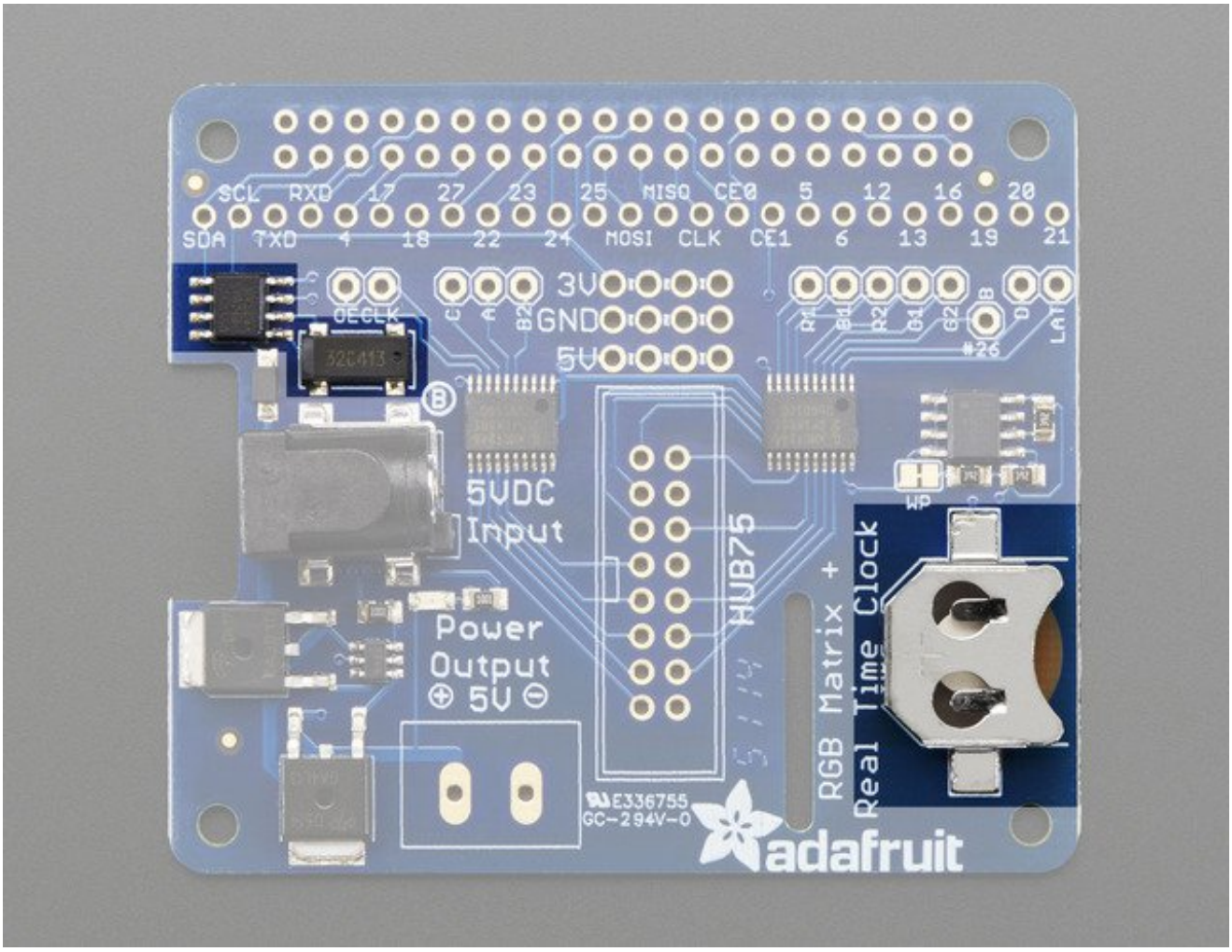
Pinouts

This HAT uses **a lot** of pins to drive the RGB Matrix. You'll still have a couple left over but just be aware a majority are in use by the matrix. **Unused GPIO pins include:** RX, TX, 18, 24, 25, MOSI, MISO, SCLK, CE0, CE1, 19

I2C / RTC pins

The DS1307 Real Time Clock soldered onboard is connected to the I2C pins**SDA** and **SCL** - these can still be used for other I2C sensors and devices as long as they are not on address 0x68

To use the Real Time Clock, a CR1220 3V lithium battery is required.

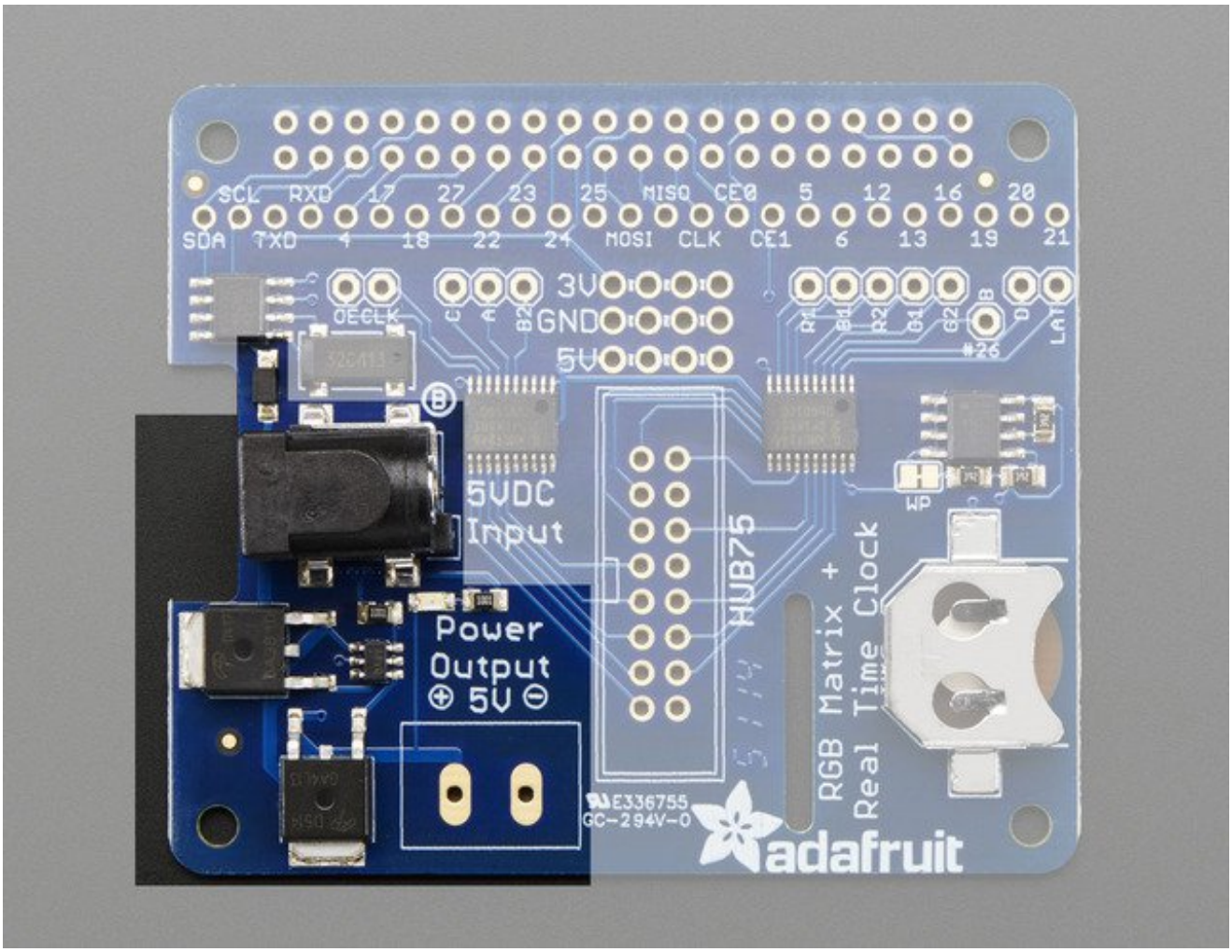


5V protection circuitry and backpower diode

LED matrix panels require 5V power and **a lot of it!** 5V 2A at a minimum and you can easily need a 5V 4A or 5V 10A supply for big stretches of panels!

Each matrix has 64 pixels (16x32 or 32x32 panels) or 128 pixels (for the 32x64 panels) lit at one time. Each pixel can draw up to 0.06 Amps each if on full white. The total max per panel is thus **64 * 0.06 = 3.95 Amps** or **128 * 0.06 = 7.68 Amps**

That's if all the LEDs are on at once, which is not likely - but still, its good to have at least half for the power supply in case you get bright!



5V power from a wall plug goes into the DC jack on the HAT which then goes through a fancy protection circuit that makes sure the voltage is not higher than 5.8V - this means that

if you accidentally grab a 9V or 12V plug or a reverse polarity plug you will not damage the HAT, Pi and panels. (**Please note, this does not protect against extreme damage**, if you plug in a 120VAC output into the DC jack or continuously try to plug in the wrong voltage you could still cause damage so please do be careful!)

We recommend powering your driving Raspberry Pi from the Pi's microUSB port but we do have a 1A diode on board that will automatically power the Pi if/when the voltage drops. So if you want, just plug in the 5V wall adapter into the HAT and it will automagically power up the Pi too!

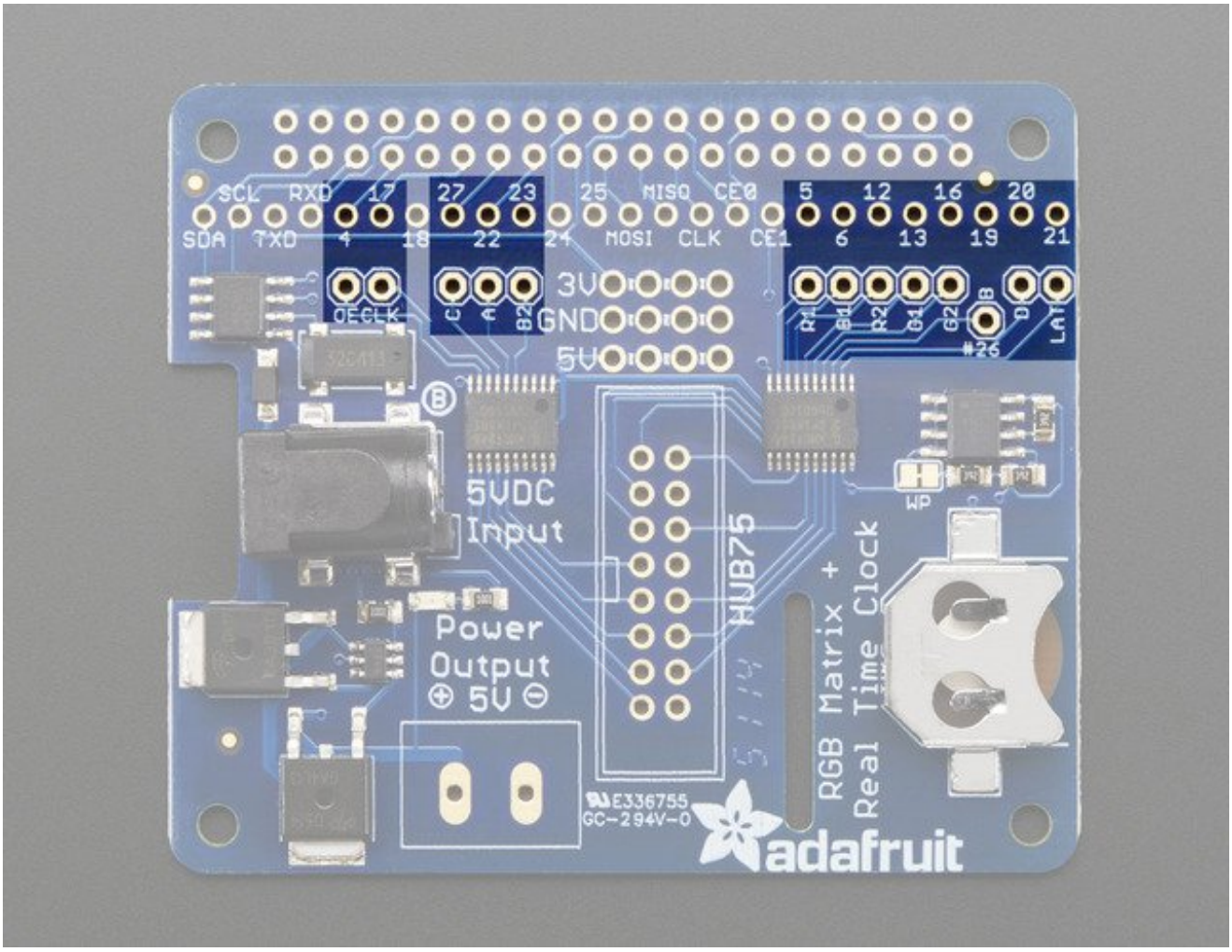
The green LED next to the DC jack will indicate that the 5V power is good, make sure it is lit when trying to use the HAT!

Matrix Drive pins

The matrix does not work like 'smart' pixels you may have used, like NeoPixels or DotStars or LPD8806 or WS2801 or what have you. The matrix panels are very 'dumb' and have no memory or self-drawing capability.

Data must be constantly streamed to the matrix for an image to display! So all of these pins are always used when drawing to the display

All these pins go thru a 74AHCT145 level shifter to convert the 3.3V logic from the Pi to the 5V logic required by the panels



Matrix Color Pins

- Pi GPIO **#5 - Matrix R1** (Red row 1) pin
  This pin controls the red LEDs on the top half of the display
- Pi GPIO **#13 - Matrix G1** (Green row 1) pin
  This pin controls the green LEDs on the top half of the display
- Pi GPIO **#6 - Matrix B1** (Blue row 1) pin
  This pin controls the blue LEDs on the top half of the display
- Pi GPIO **#12 - Matrix R2** (Red row 2) pin
  This pin controls the red LEDs on the bottom half of the display
- Pi GPIO **#16 - Matrix G2** (Green row2) pin
  This pin controls the green LEDs on the bottom half of the display
- Pi GPIO **#23 - Matrix B2** (Blue row 2) pin
  This pin controls the blue LEDs on the bottom half of the display

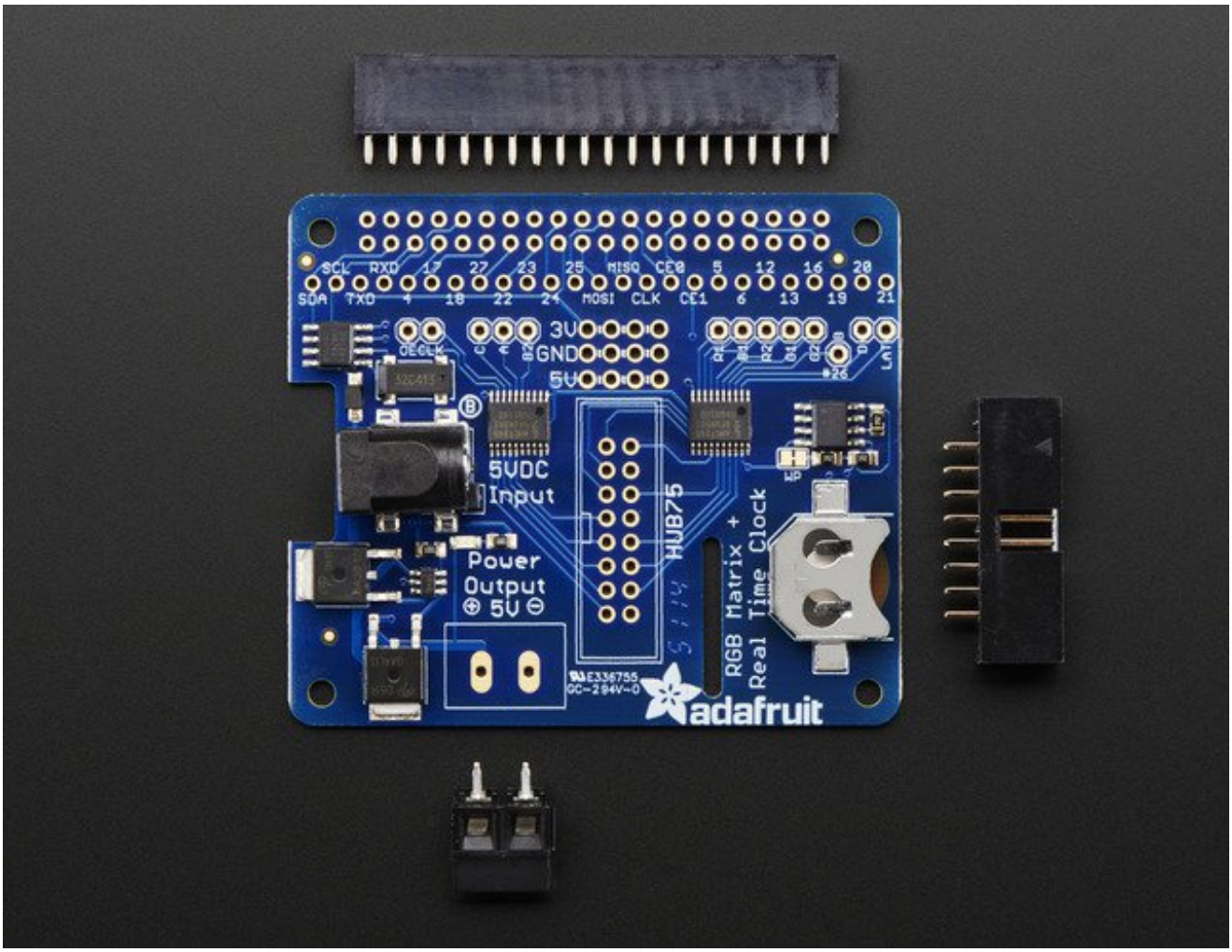
Matrix Control pins

- Pi GPIO **#4 - Matrix OE** (output enable) pin
  This pin controls whether the LEDs are lit at all
- Pi GPIO **#17 - Matrix CLK** (clock) pin
  This pin is the high speed clock pin for clocking RGB data to the matrix
- Pi GPIO **#21 - Matrix LAT** (latch) pin
  This pin is the data latching pin for clocking RGB data to the matrix

RGB Matrix Address pins

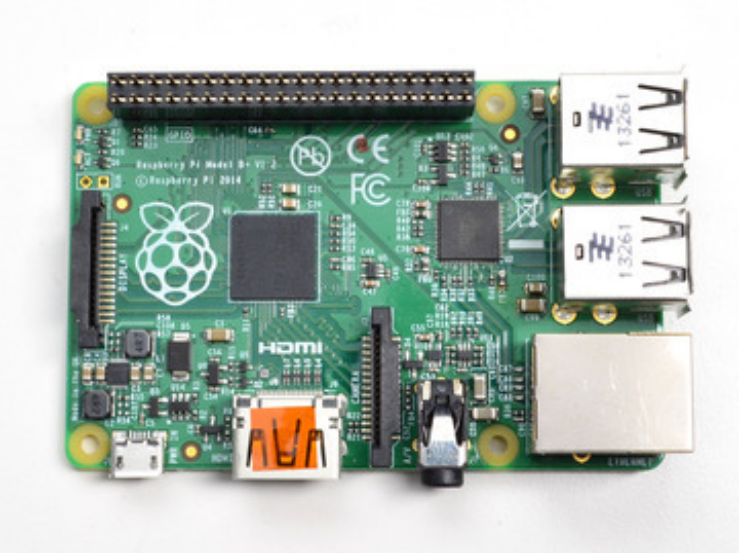
- Pi GPIO **#22 - Matrix A** (address A) pin
  This pin is part of the 1->16 or 1->8 multiplexing circuitry.
- Pi GPIO **#26 - Matrix B** (address B) pin
  This pin is part of the 1->16 or 1->8 multiplexing circuitry.
- Pi GPIO **#27 - Matrix C** (address C) pin
  This pin is part of the 1->16 or 1->8 multiplexing circuitry.
- Pi GPIO **#20 - Matrix D** (address D) pin
  This pin is part of the 1->16 multiplexing circuitry. Used for 32-pixel tall displays only

Assembly

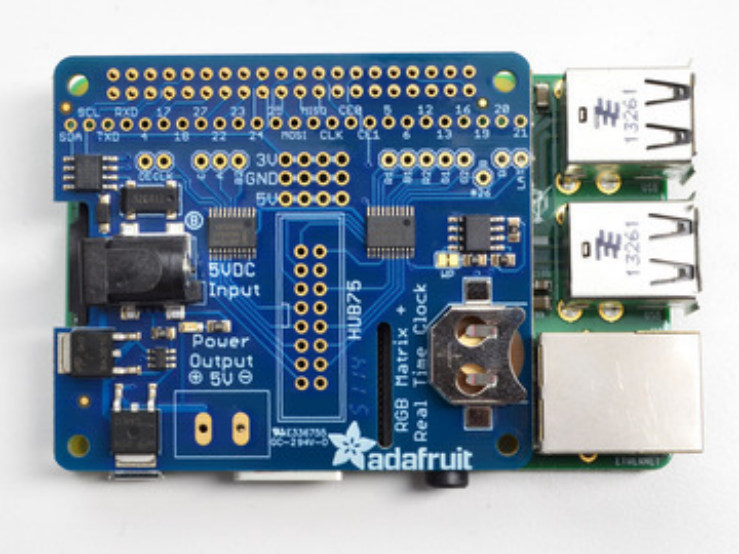


Solder on Headers and Terminal Block

Before we can a-blinkin' 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, the 2x8 header so we can plug the RGB matrix into the HAT, and a terminal block so you can power the matrix through the HAT.

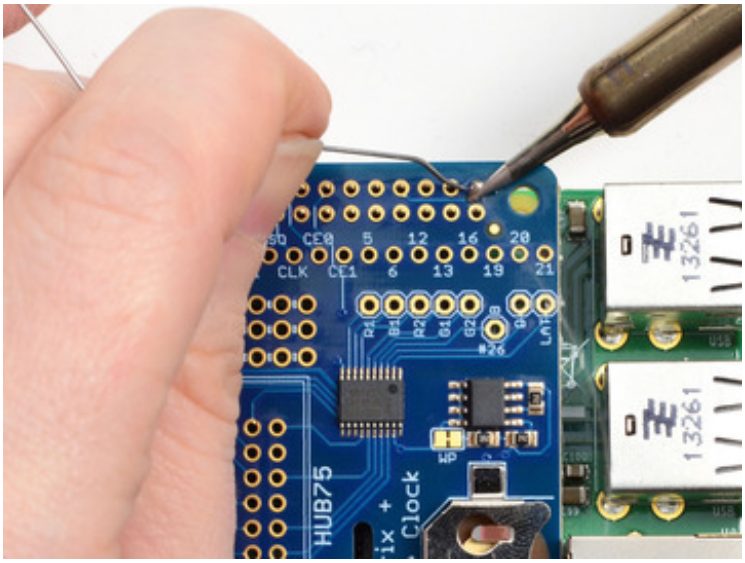
- 

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 off!

- 

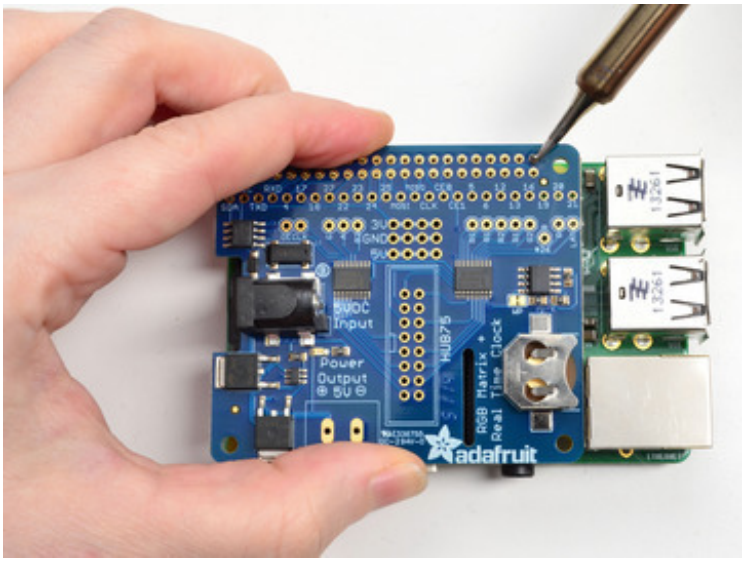
Place the HAT on top so that the short pins of the 2x20 header line up with the pads on the HAT

And Solder!

- 

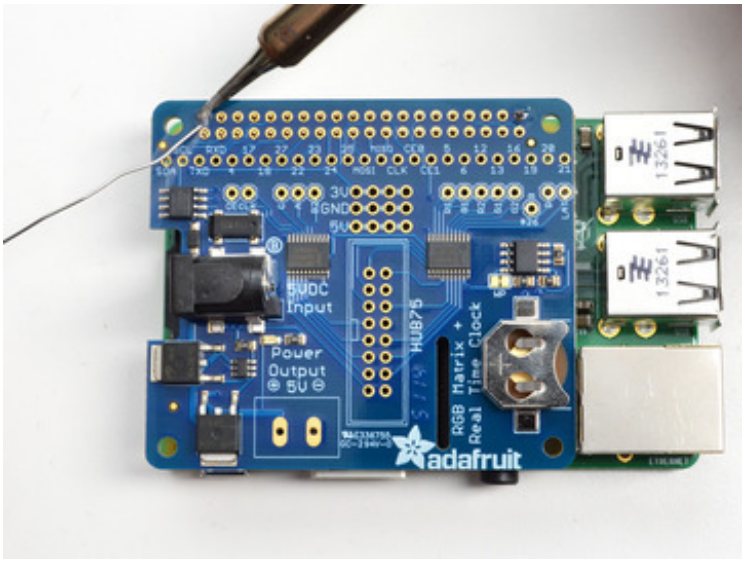
  Heat up your iron and solder in one header connection on the right.

  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

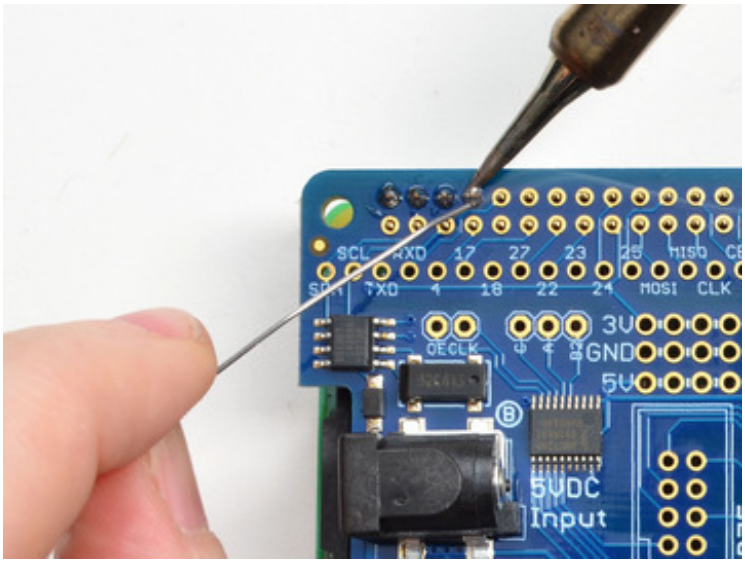
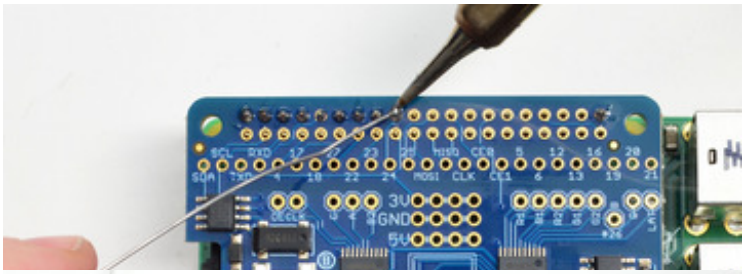
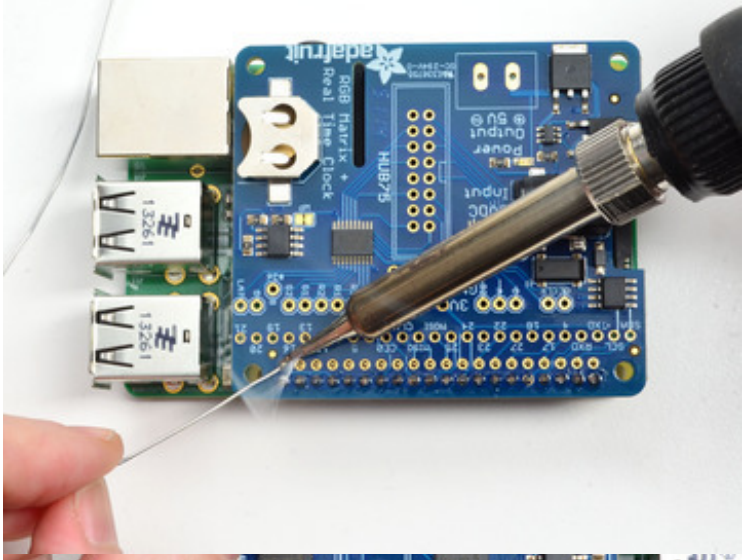
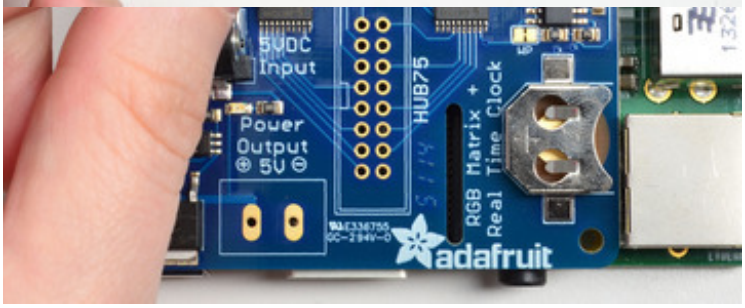
- 

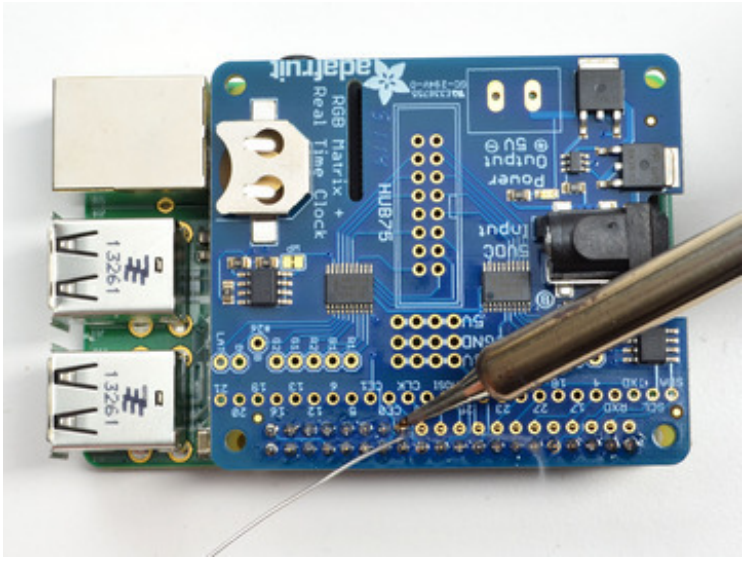
  (For tips on soldering, be sure to check out our [Guide to Excellent Soldering](http://adafru.it/aTk)).

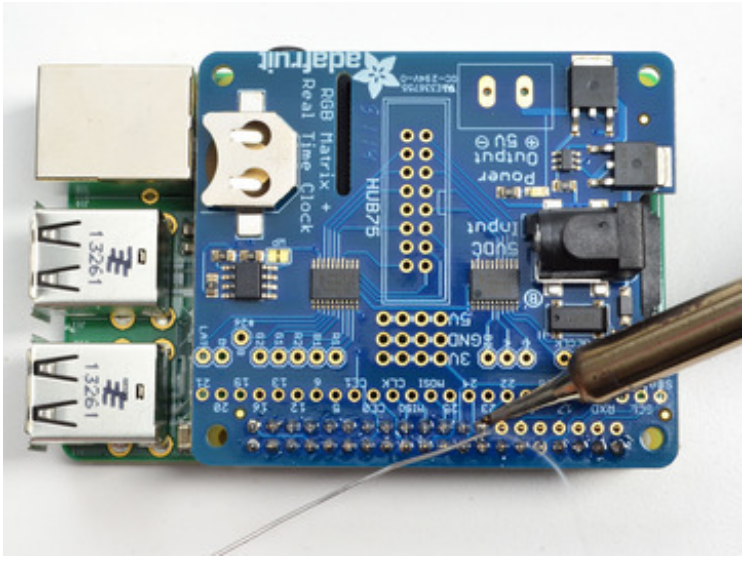
Solder one point on the opposite side of the connector

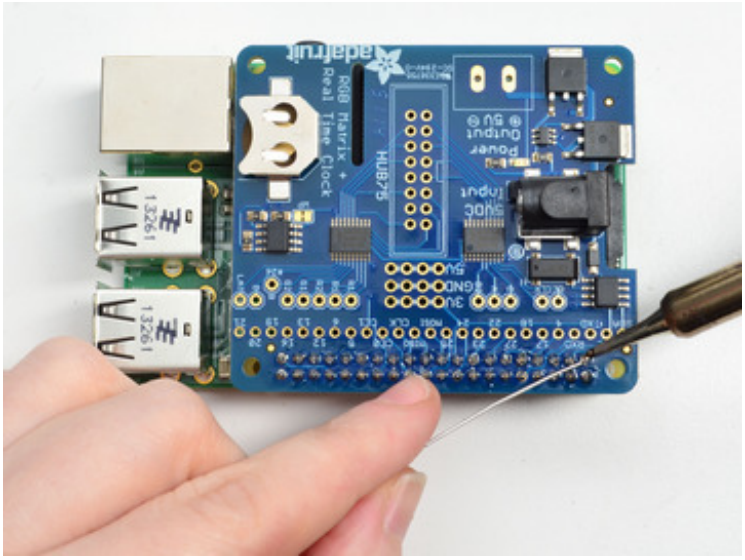
- 

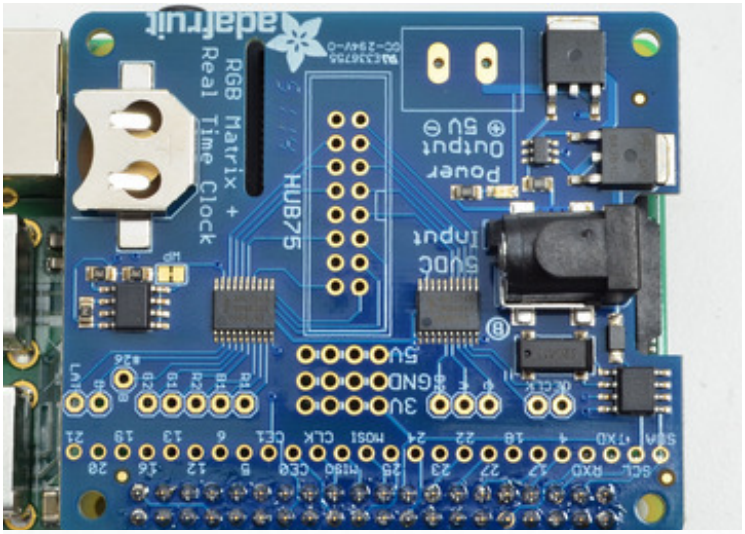
Solder each of the connections for the top row

- 
SCL RXD 17 27 23 25 MISO
SDA TXD 4 18 22 24 MOSI CLK
OECLK
3V
GND
5V
5VDC
Input
- 
- 
RGB Matrix +
Real Time Clock
HUB75
Power
Output
5V
- 
5VDC
Input
Power
Output
5V
HUB75
RGB Matrix +
Real Time Clock
adafruit

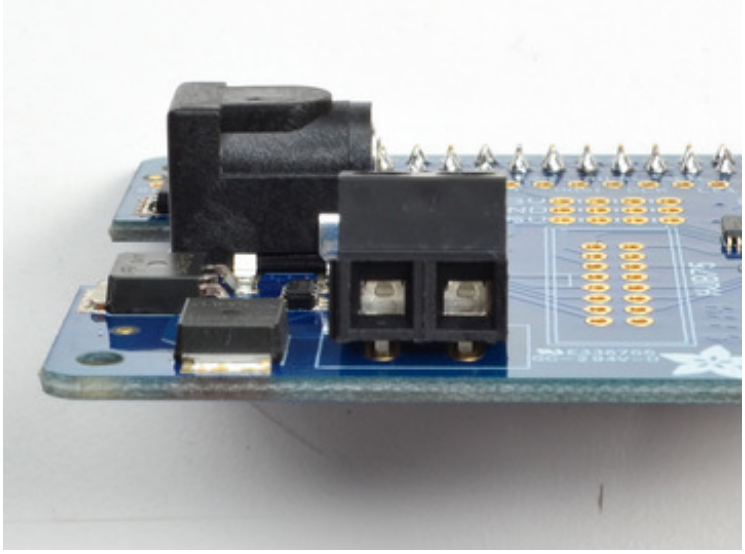
- 

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

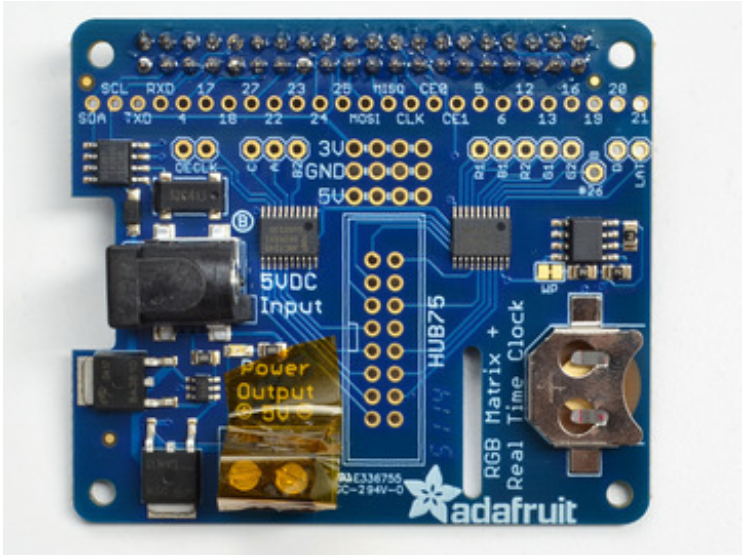
- 



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

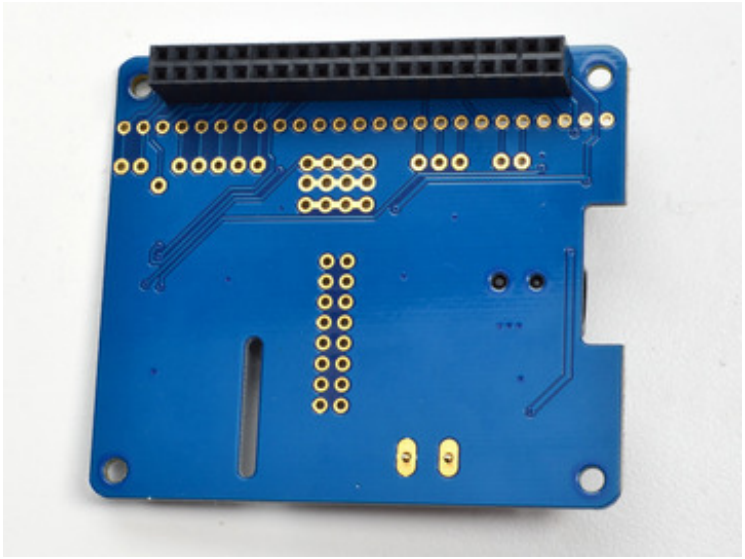
- 

Place the 2 pin terminal block first, make sure the two 'mouths' are facing outwards

- 

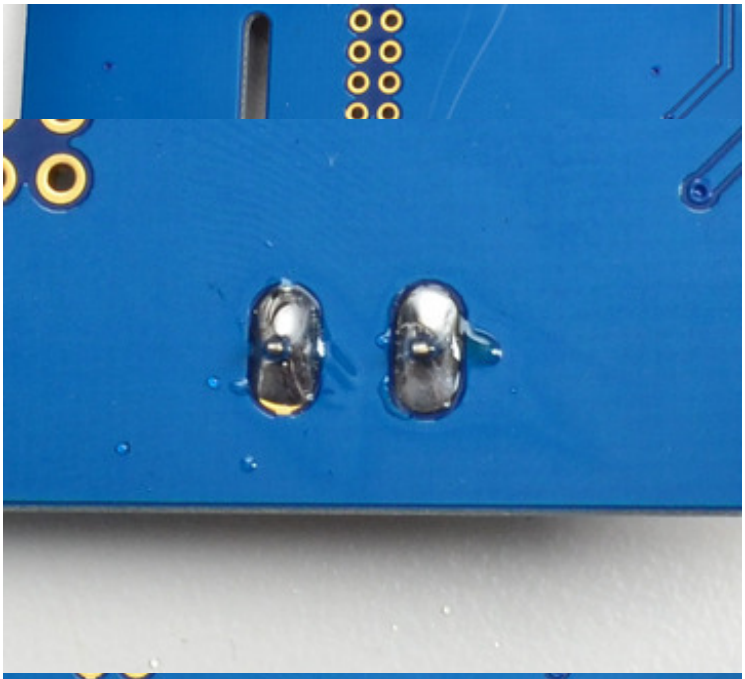
Use some tape to stick the terminal down in place

-

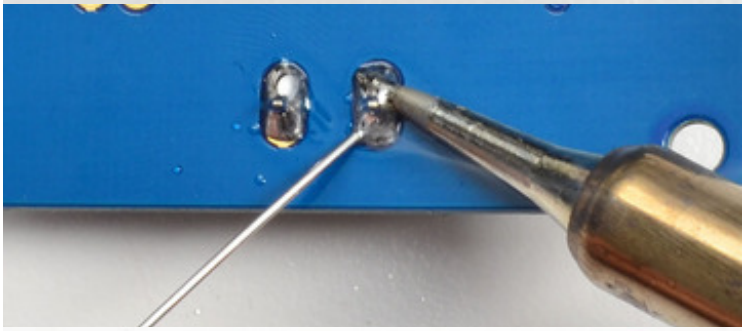
- 

Flip the board over, the tape should keep the terminal block in place

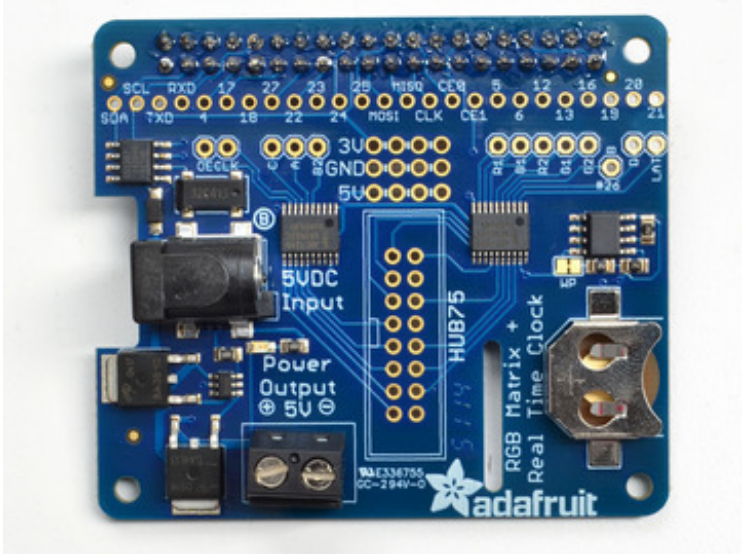
Solder the two big connections, use plenty of solder!



Check your work, the connections should be solid and shiny

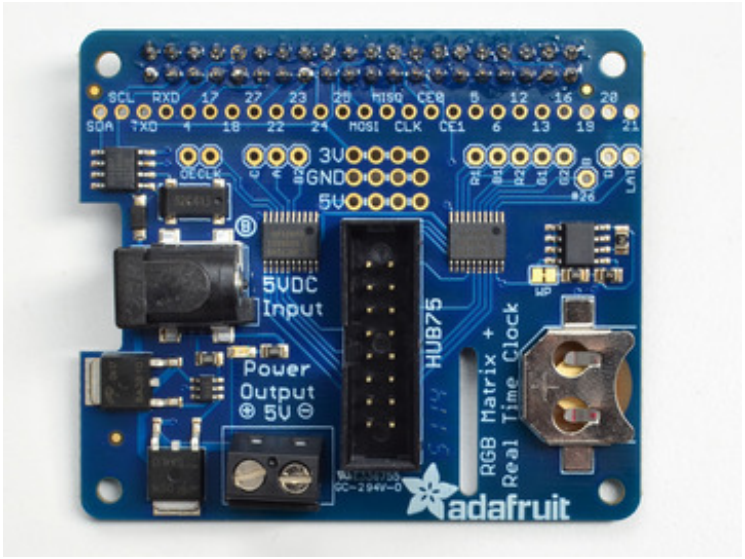


Next up we will attach the 2x8 IDC header. Unlike the 2x20 header, **this connector has a direction!**

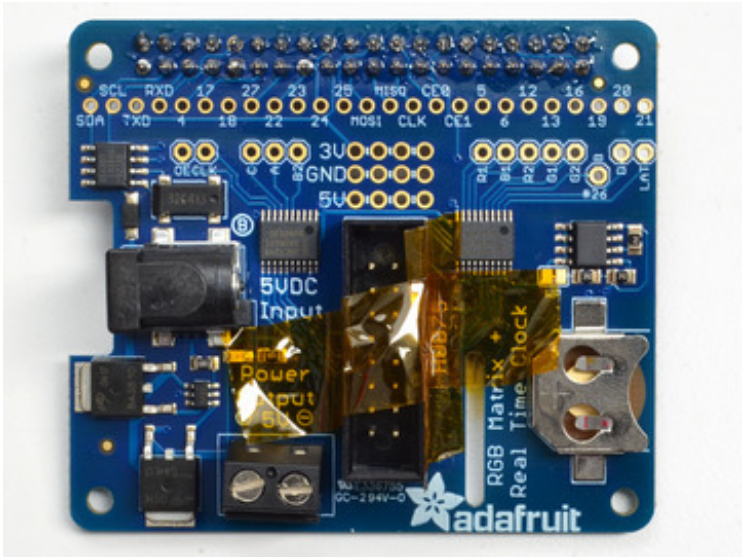


Notice in the middle there's an outline for the connector in the middle. On the right it says **HUB75** and on the left of the connector there is a little 'cutout' shape. This cutout shape must match up with the cut out on the connector.

If you solder it in backwards, its not a huge deal, you can use diagonal cutters to cut out a notch on the opposite side, but if you get it right then you will never have to worry about plugging in your matrix data cable the wrong way

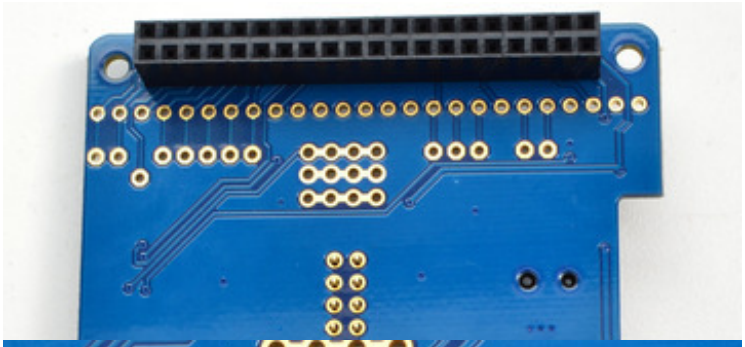
- 

Place the connector in the slot so that the notched side is on the **left**

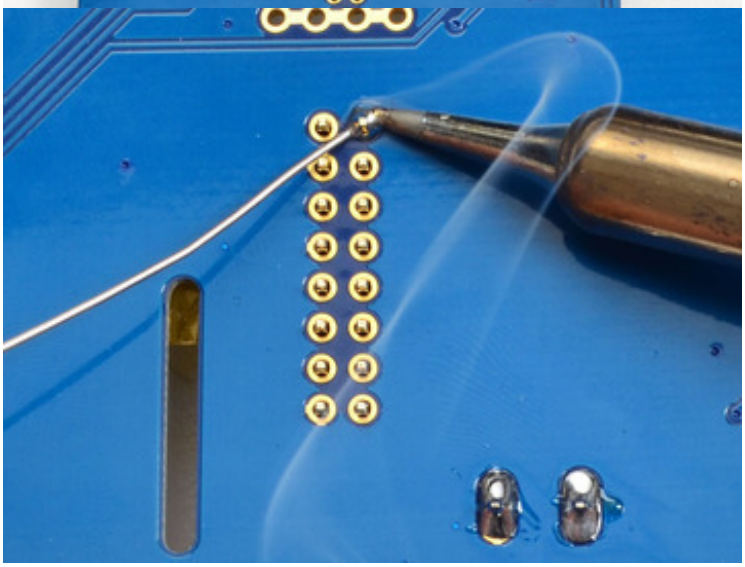
- 

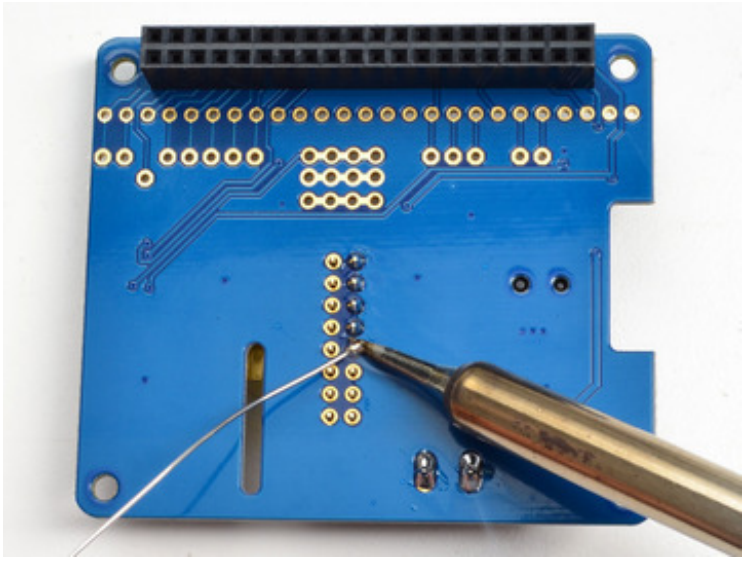
Use some tape to hold the IDC connector in place

Flip the board over, the tape

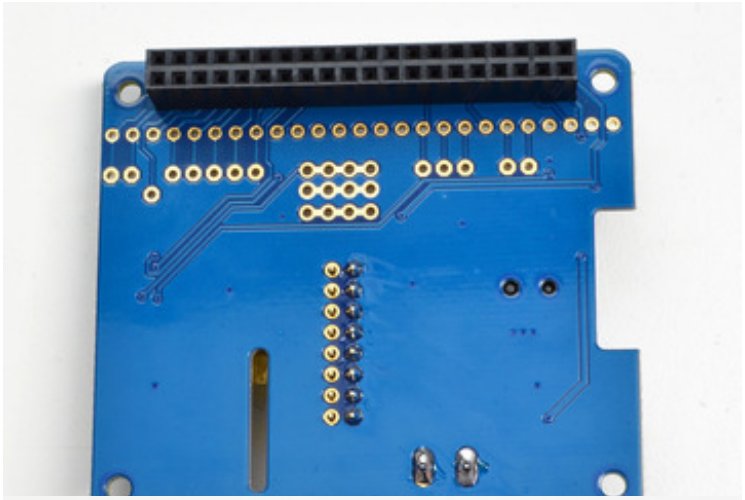


should keep the connector from falling out

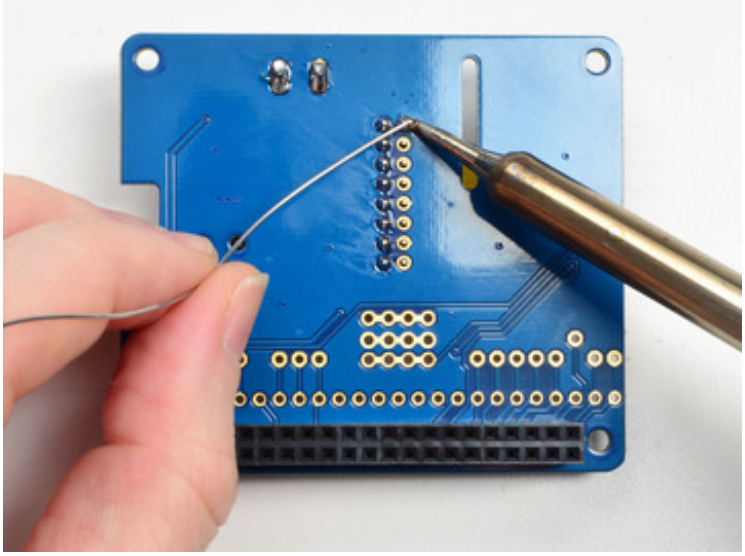




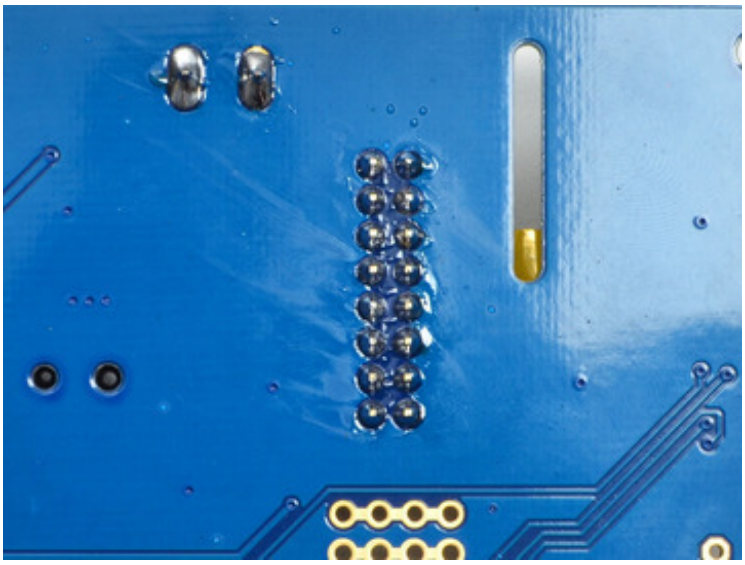
Solder in all the pins like you did with the 2x20 connector



Check your work! Make sure all the solder points are clean and not shorted or cracked or dull

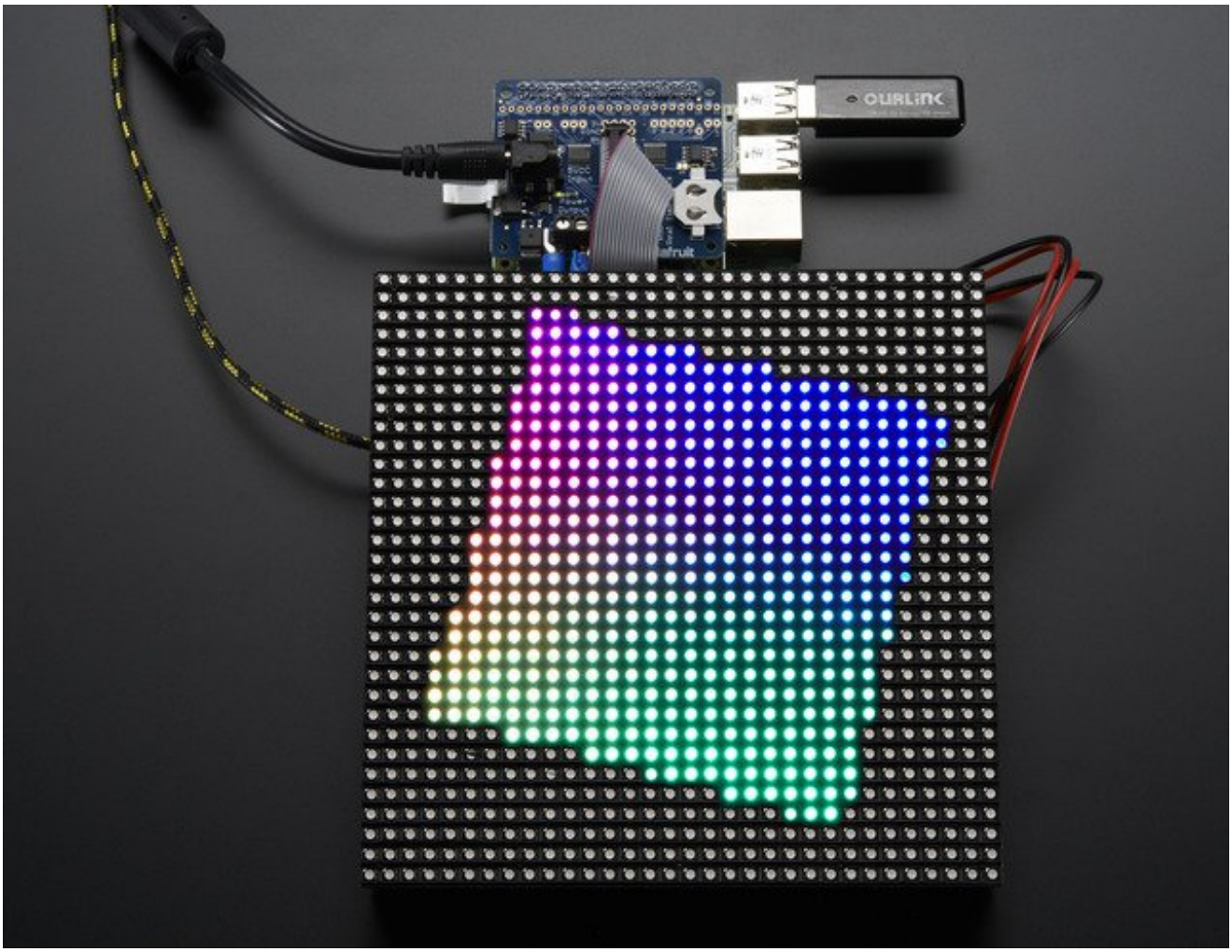
- 

Flip the board around & solder up the other half!

- 

Check your work one last time...now continue to testing!

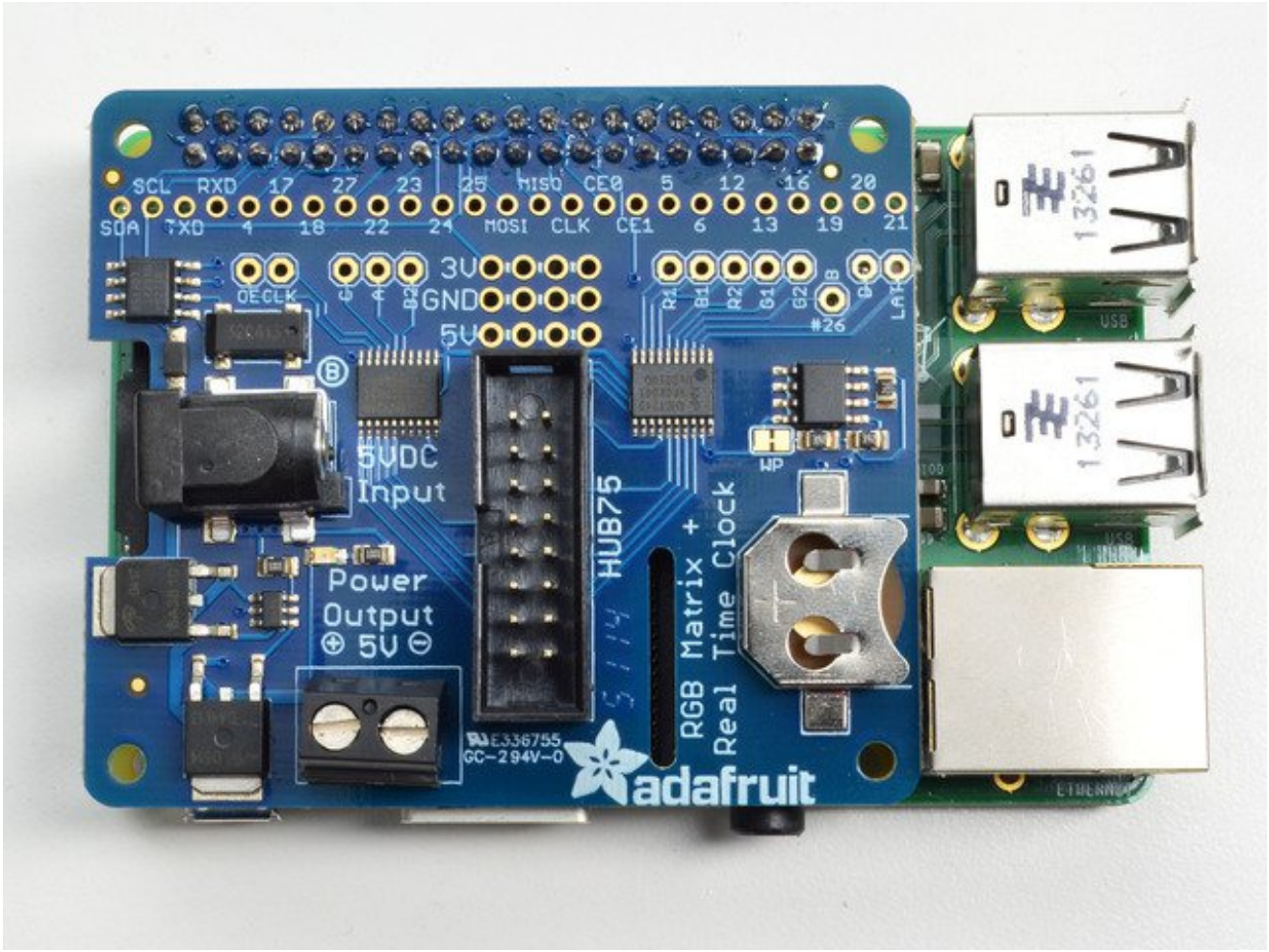
Driving Matrices



OK we're onto the fun part now! Be sure you have completed the Assembly step before continuing, the soldering is **not optional**

Step 1. Plug HAT into Raspberry Pi

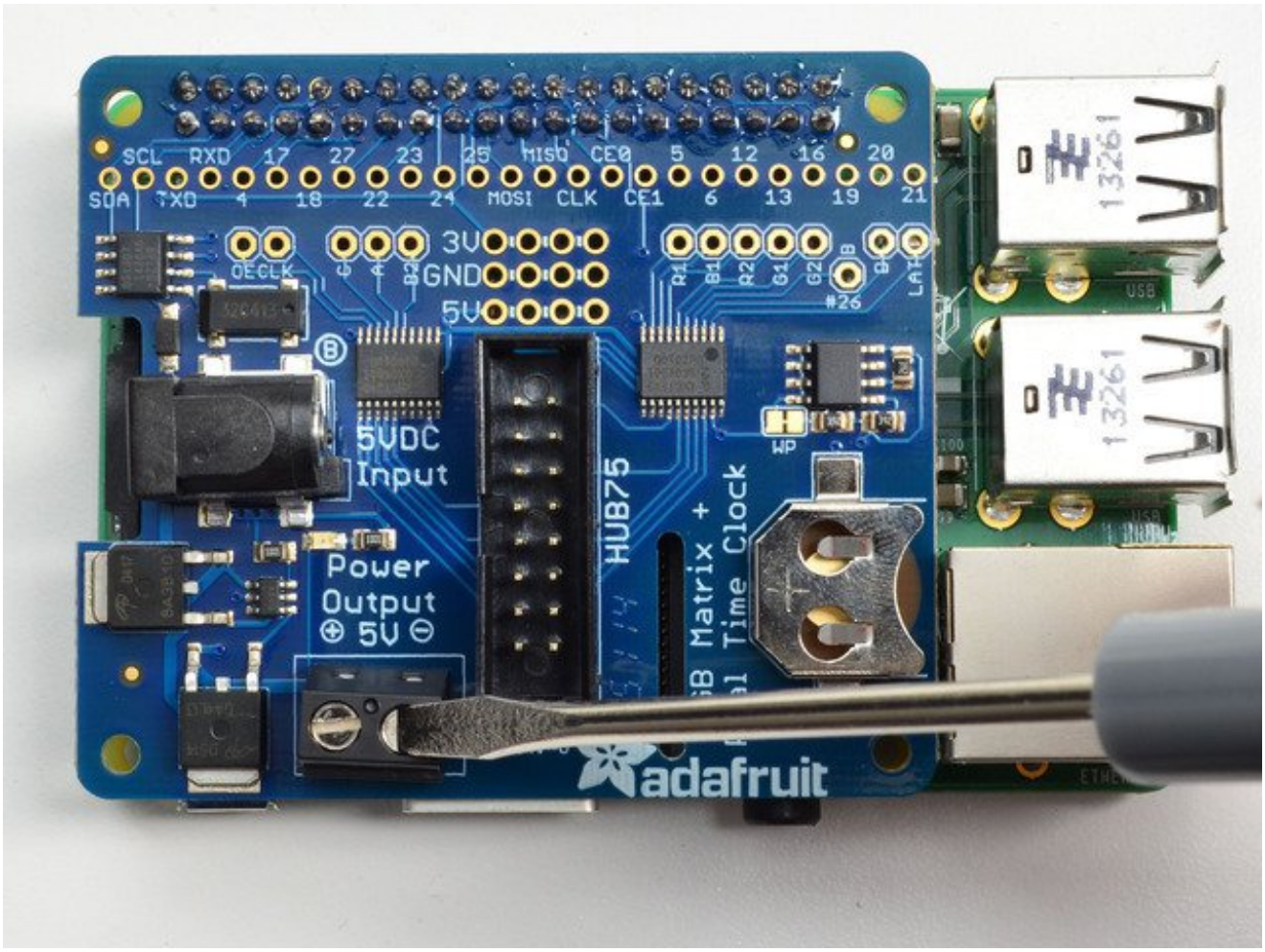
Shut down your Pi and remove power. Plug the HAT on so all the 2x20 pins go into the GPIO header.

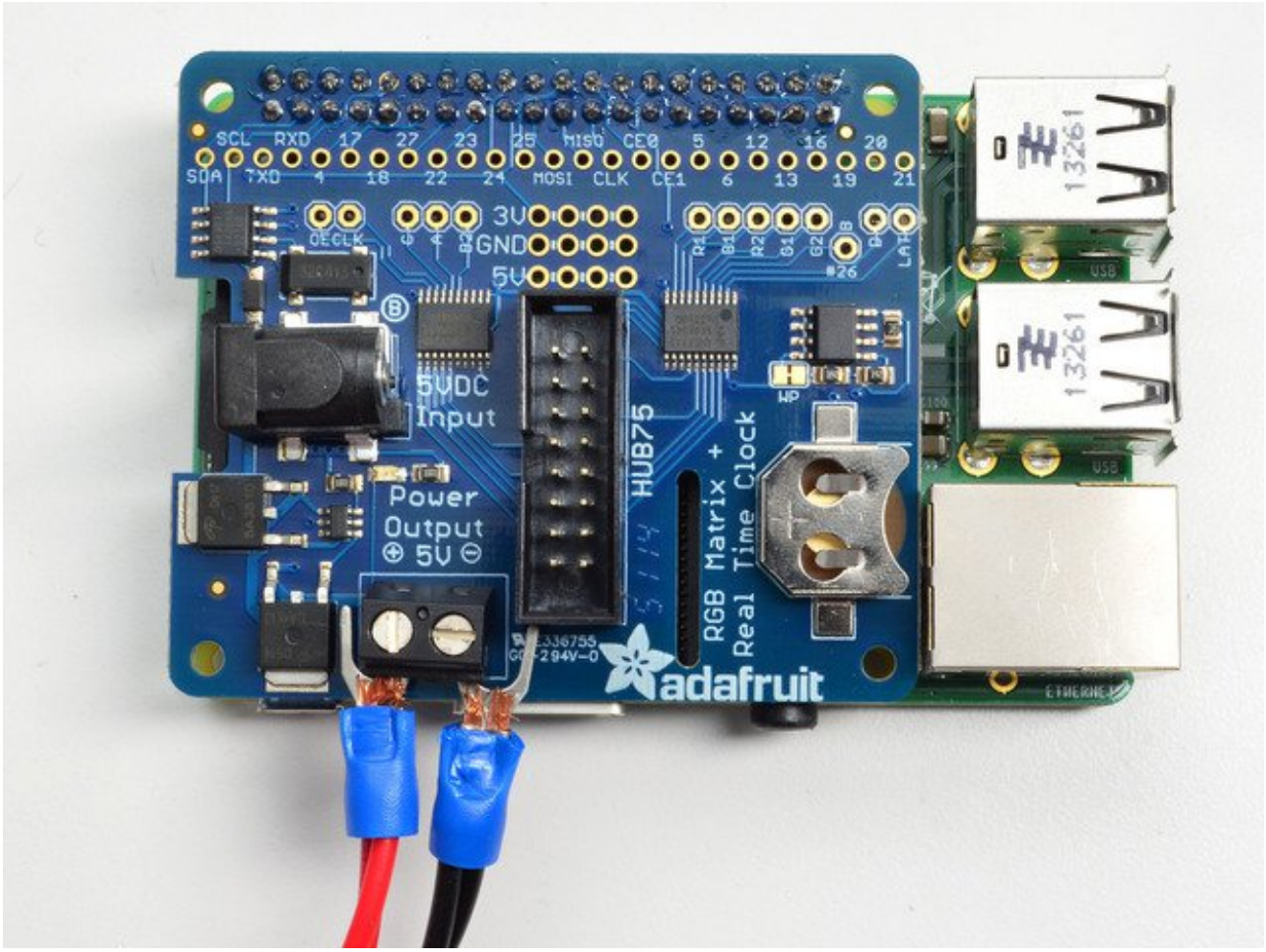


Step 2. Connect Matrix Power cable to terminal block

Your RGB matrix came with a red & black power cable. One end has a 4-pin MOLEX connector that goes into the matrix. The other end probably has a spade connector. If you didn't get a spade connector, you may have to cut off the connector and tin the wires to plug them into the terminal block

Either way, unscrew the terminal blocks to loosen them, and plug the red wire into the **+** side, and the black wire into the **-** side.

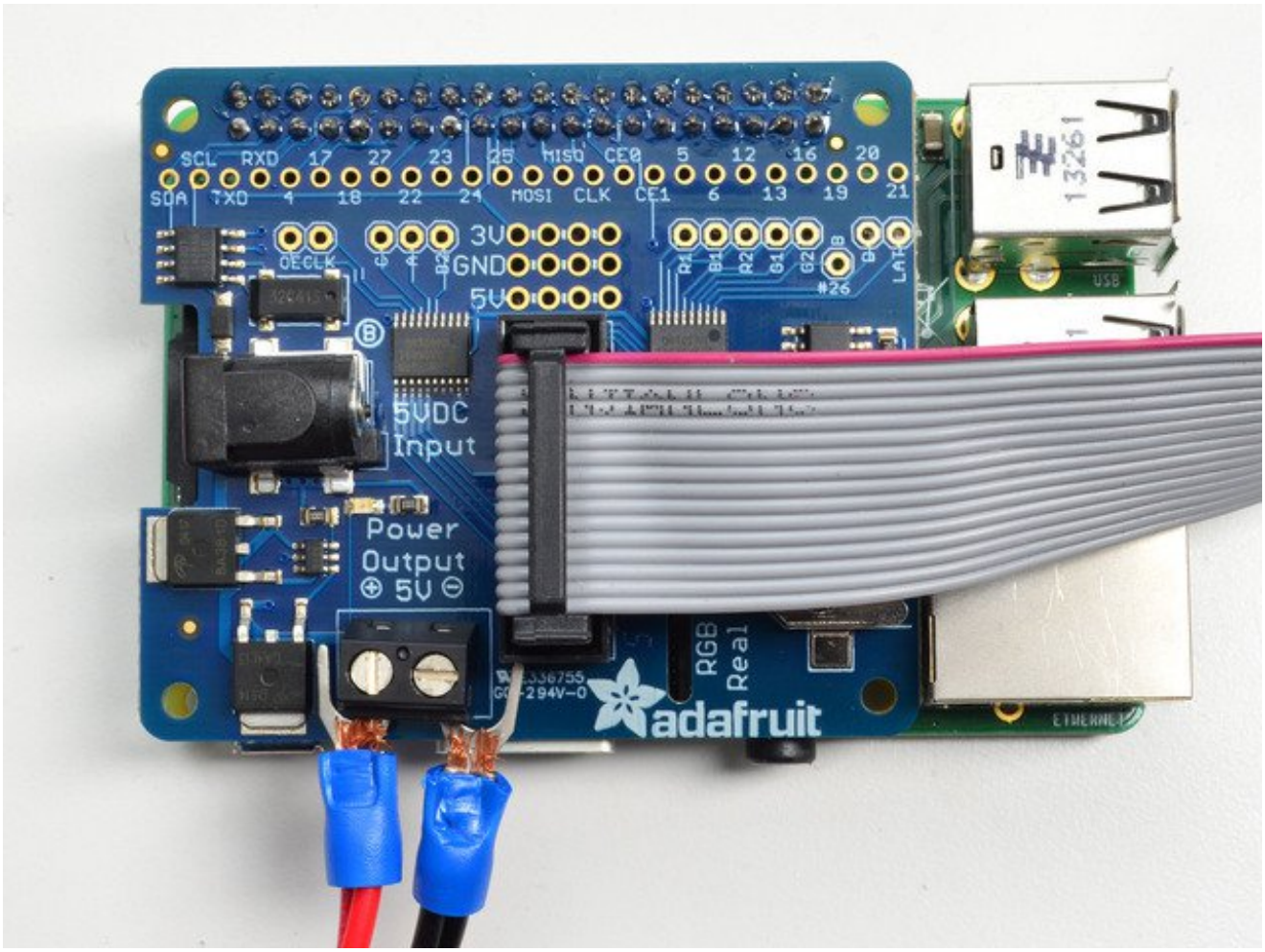

SCL RXD 17 27 23 25 MISO CE0 5 12 16 20
SDA TXD 4 18 22 24 MOSI CLK CE1 6 13 19 21
3V
GND
5V
OECLK
5VDC
Input
Power
Output
5V
HUB75
WP
adafruit



Step 3. Connect RGB Matrix Data cable to IDC

The RGB matrix also came with a 2x8 data cable. Connect one end to the matrix's INPUT side and the other end to the IDC socket on the HAT.

It wont damage the matrix if you accidentally get the cable connected to the output end of the matrix but it wont work so you might as well get it right first time!



Step 4. Power up your Pi via MicroUSB (optional but suggested)

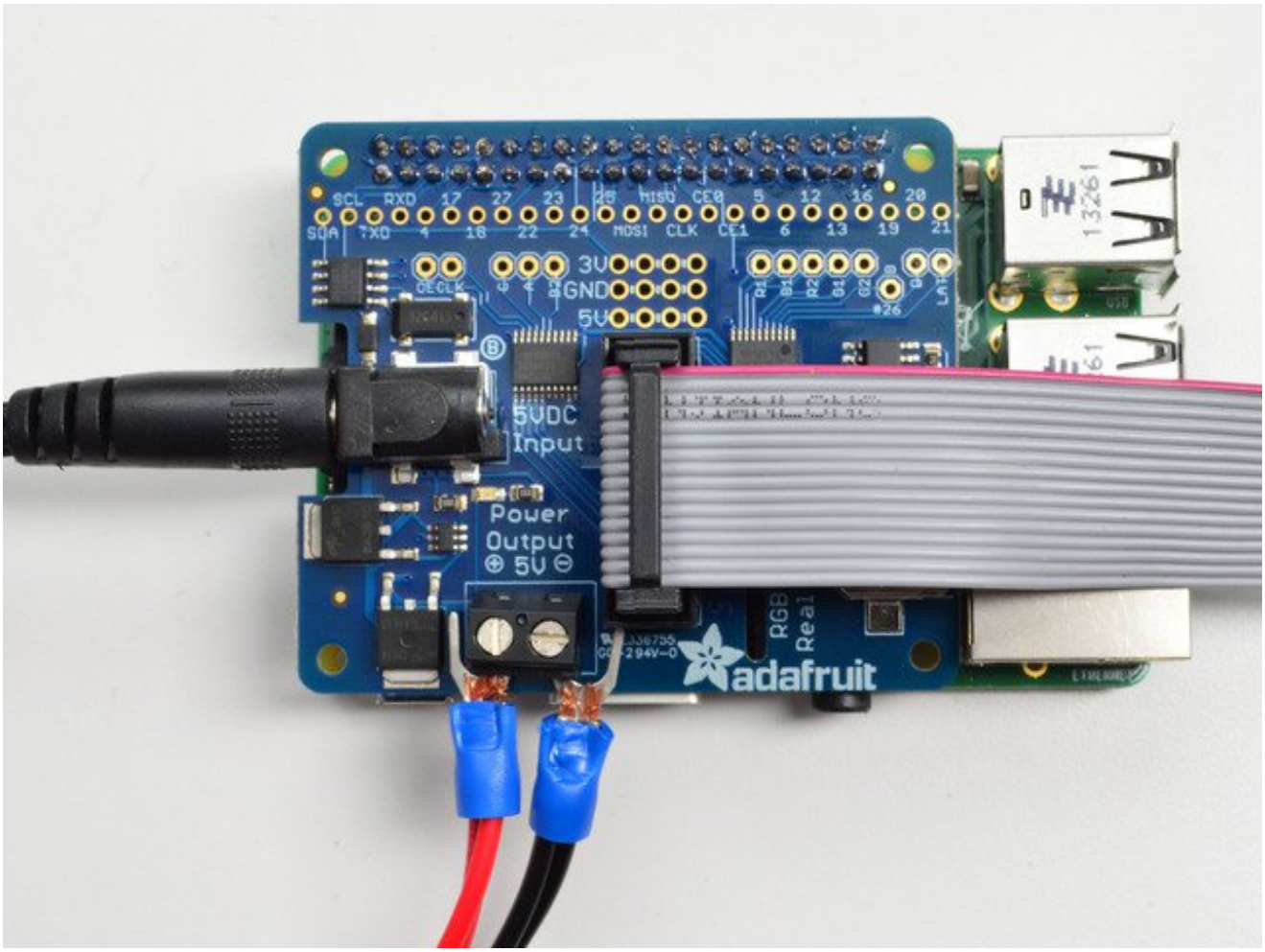
Connect your Raspberry Pi to power via the microUSB cable, just like you normally would to power it up.

You **can** power the Pi via the 5V wall plug that is also used for the Matrix but its best to have it powered seperately

Step 5. Plug in the 5V DC power for the Matrix

OK now you can plug in your 5V 2A or 4A or larger wall adapter into the HAT. This will turn the green LED on but nothing will display on your matrix yet because no software is

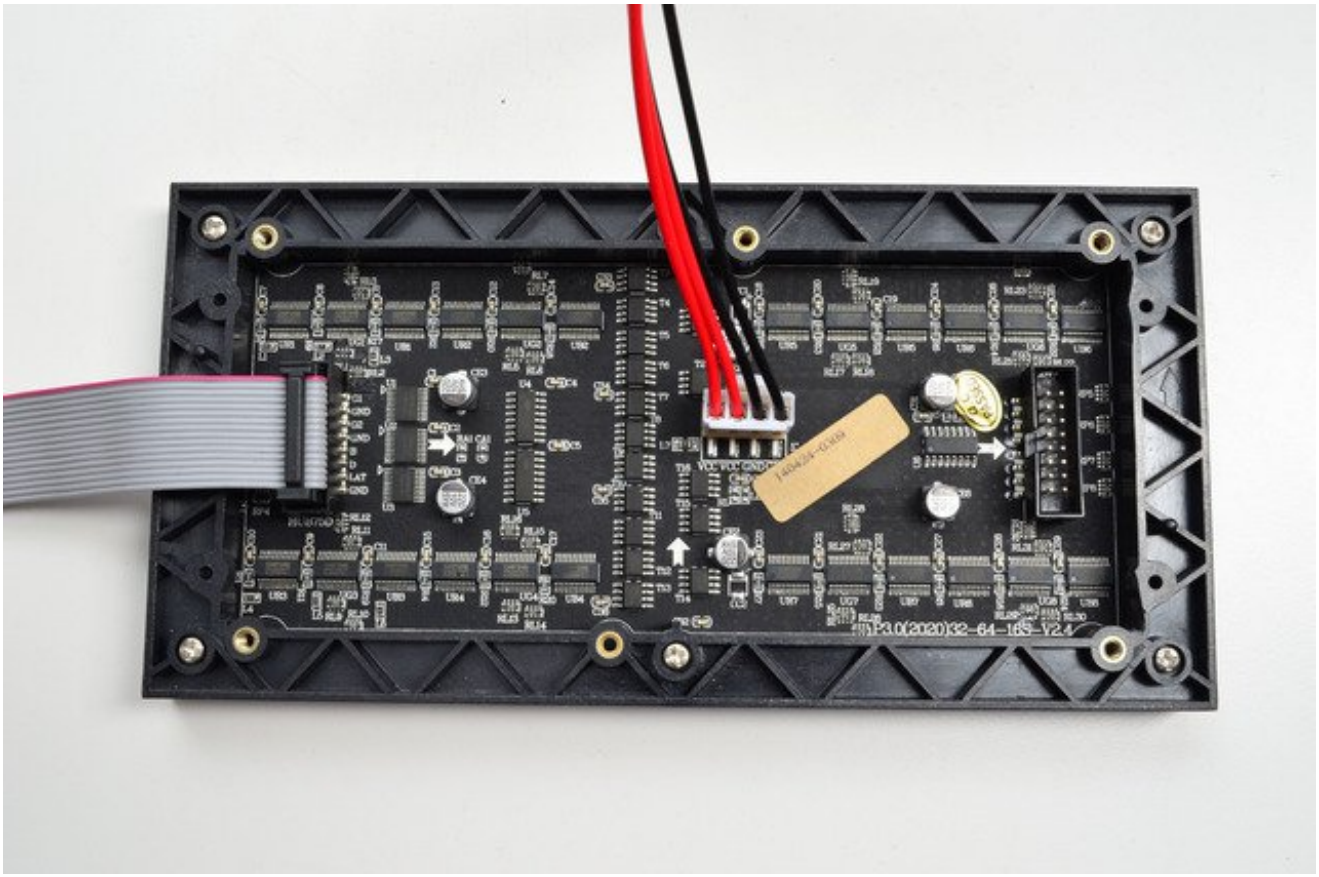
running!



Check that the Matrix plugs are installed and in the right location

IDC goes into the INPUT side (look for any arrows, arrows point from INPUT side to OUTPUT)

Power plug installed, red wires go to VCC, black wires to GND



Step 6. Log into your Pi to install and run software

OK now you are ready to run the Pi software. You will need to get into a command line via the HDMI monitor, ssh or console cable. You will also need to make sure your Pi is on the Internet via a WiFi or Ethernet connection.

First, let's install some prerequisite software for compiling the code:

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

Then download and uncompress the matrix code package from github (http://adafru.it/ewy):

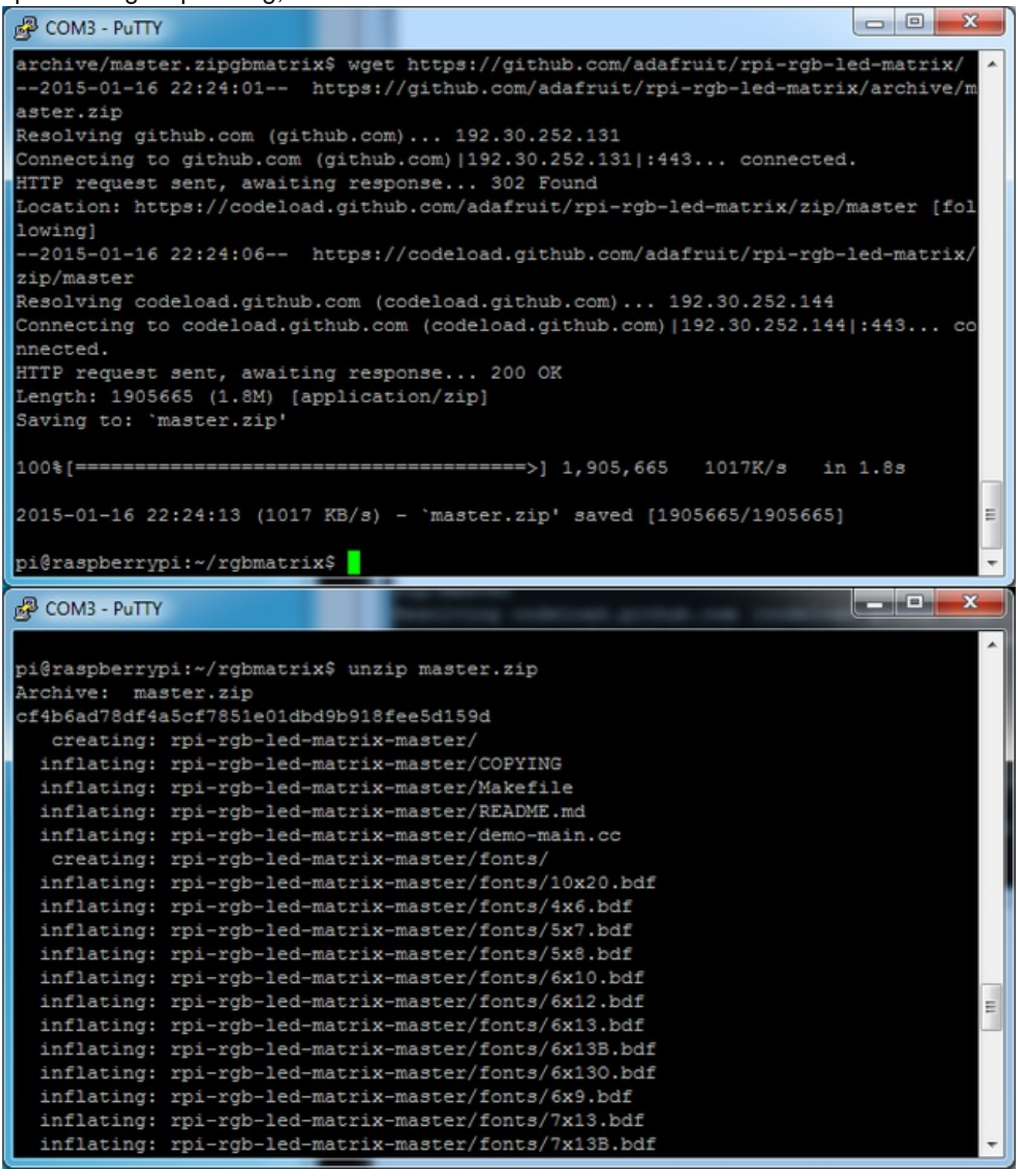
```
wget https://github.com/adafruit/rpi-rgb-led-matrix/archive/master.zip
unzip master.zip
```

The LED-matrix library is (c) Henner Zeller h.zeller@acm.org with GNU General Public License Version 2.0 http://www.gnu.org/licenses/gpl-2.0.txt (http://adafru.it/ewN)

(Our fork adds support for the HAT and the latest Raspberry Pi

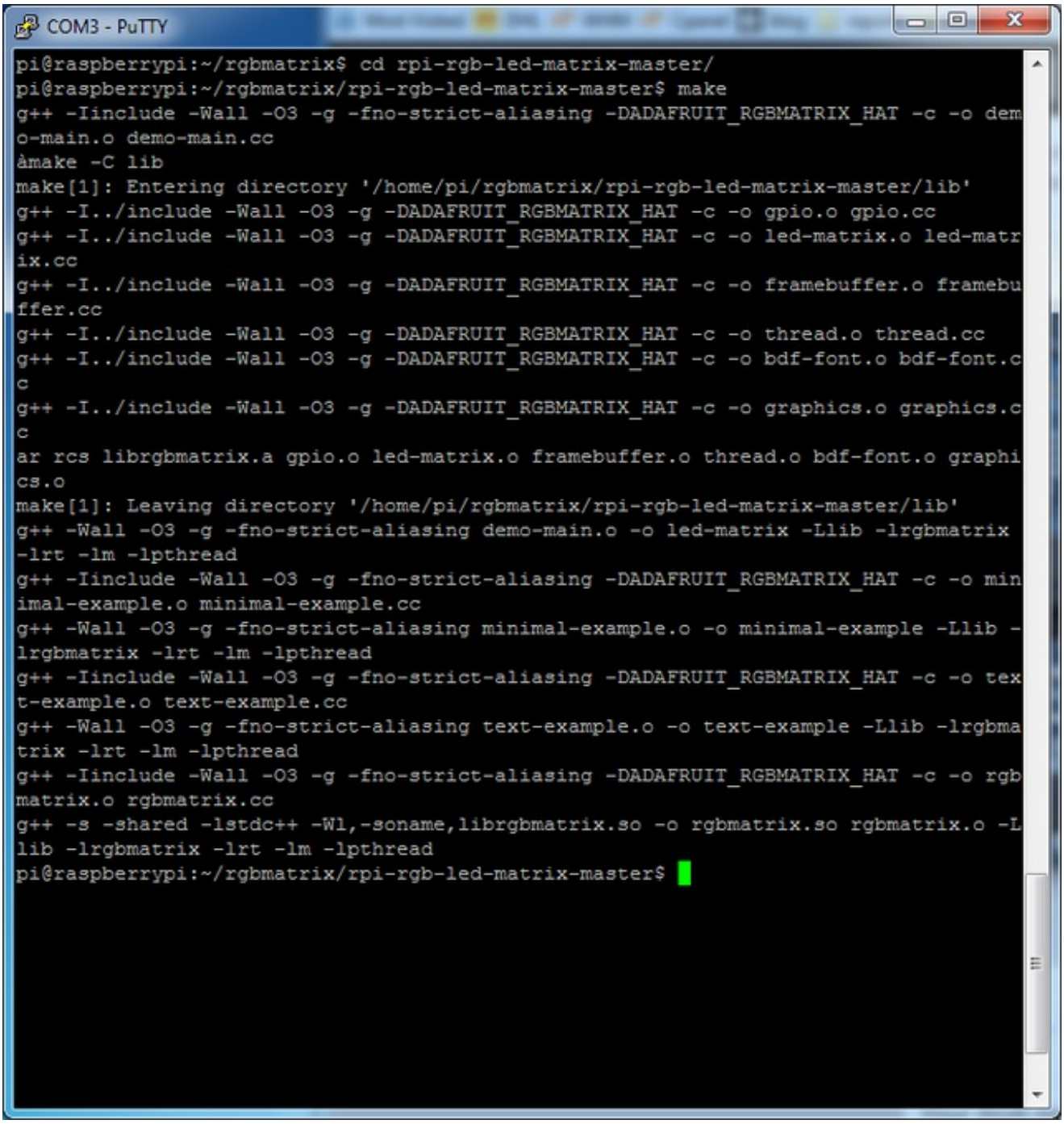
models (http://adafru.it/ewy))

Overclocked Raspberry Pi boards may produce visual glitches on the LED matrix. If you encounter such trouble, first thing to try is to set the Pi to the default (non-overclocked) speed using raspi-config, then reboot and retest.



Now you can **cd** to the folder of source code and compile the demo with**make**

```
cd rpi-rgb-led-matrix-master/
make
```



Now you can run the test/demo software**led-matrix**

You'll want to change up the command flags based on how many matrices you have

Rows

The # of rows is indicated with **-r**

If you are using a 16 pixel tall matrix (a 16x32) use **-r 16**

If you are using 32 pixel tall matrix (64x32 or 32x32) use **-r 32**

Chained

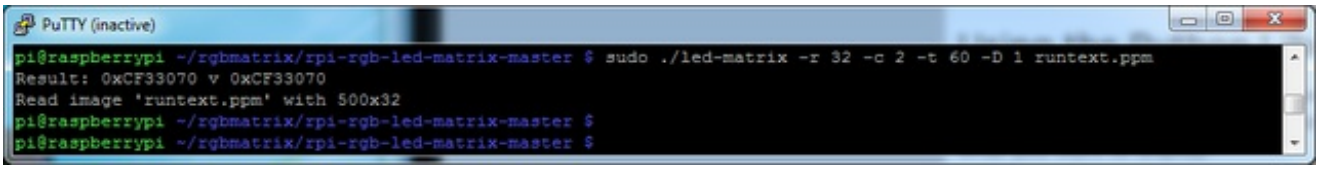
Each matrix is considered 32 pixels wide. If you have multiple matrices chained use **-c** to increase the width. If you have 3 chained together, use **-c 3** If you have a 64x32 matrix, it looks like 2 chained 32x32 so use **-c 2**

Time Running

You can run the demo for a given amount of time with **-t** e.g. **-t 60** is 60 seconds

Demo

There's a bunch of built-in demos you can run to test out the matrix, start with **-D 0** which will show a spinning rainbow square or you can run the **-D 1** scrolling image demo, just give it a ppm image to display.



The demos kinda run, but I'm seeing weird rectangles and glitches.

If your Pi is overclocked, or if you're using a Raspberry Pi 2, you may need to dial back the matrix control speed slightly. This can be done with the **-w** option, which sets the number of write cycles (higher numbers are slower). The default is **1** cycle for Raspberry Pi Model A, B and similar, and **2** cycles for Pi 2 or 3. If you see glitches, try entering a higher value such as **-w4** and test again, up or down until you arrive at a value that works reliably.

Using the Python Library

We have a Python library for drawing on the display. To use this, you'll need the **rgbmatrix.so** file (created with the 'make' command earlier). Since it's still in an early state, we've not made this an installable Python module; you'll need that .so file in the same directory as any matrix code you write.

There are two examples. The first, **matrixtest.py**, shows how to clear the display, fill it with a solid color or plot individual pixels. These functions all have an immediate effect on the display…simple to use, but not great for smooth animation.

```
#!/usr/bin/python

# Simple RGBMatrix example, using only Clear(), Fill() and SetPixel().
# These functions have an immediate effect on the display; no special
# refresh operation needed.
# Requires rgbmatrix.so present in the same directory.

import time
from rgbmatrix import Adafruit_RGBmatrix

# Rows and chain length are both required parameters:
matrix = Adafruit_RGBmatrix(32, 1)

# Flash screen red, green, blue (packed color values)
matrix.Fill(0xFF0000)
time.sleep(1.0)
matrix.Fill(0x00FF00)
time.sleep(1.0)
matrix.Fill(0x0000FF)
time.sleep(1.0)

# Show RGB test pattern (separate R, G, B color values)
for b in range(16):
	for g in range(8):
		for r in range(8):
			matrix.SetPixel(
			  (b / 4) * 8 + g,
			  (b & 3) * 8 + r,
			  (r * 0b001001001) / 2,
			  (g * 0b001001001) / 2,
			   b * 0b00010001)

time.sleep(10.0)
matrix.Clear()
```

A second example uses the **Python Imaging Library** (PIL) to add support for drawing shapes (lines, circles, etc.) and loading images (GIF, PNG, JPEG, etc.). PIL is not always installed by default on some systems, so let's start with:

```
sudo apt-get install python-imaging
```

Unlike the prior example, **PIL graphics do not have an immediate effect on the display.** The image is drawn into a separate buffer, which is then copied to the matrix. On the plus side, this extra step affords us the opportunity for smooth animation and scrolling.

The rgbmatrix.so library only supports these image modes: RGB (full color) (RGBA is also supported but the alpha channel is ignored); color palette (such as GIF images use); and bitmap (black/white). Colorspaces like CMYK and YCbCr are not directly handled, but you might be able to convert these to RGB through other PIL functions.

Core PIL image functions are explained here: The Image Module (http://adafru.it/dvE)

Graphics functions (lines, etc.) are here: The ImageDraw Module (http://adafru.it/dfH)

matrixtest2.py demonstrates simple drawing, image loading and scrolling. **Super important:** notice that the PIL *Image id* (not the Image object itself) is passed to SetImage(). Also, if you're loading an image file both the open() and load() functions need to be called before invoking SetImage(). All this can be seen in the example…

```
#!/usr/bin/python

# A more complex RGBMatrix example works with the Python Imaging Library,
# demonstrating a few graphics primitives and image loading.
# Note that PIL graphics do not have an immediate effect on the display --
# image is drawn into a separate buffer, which is then copied to the matrix
# using the SetImage() function (see examples below).
# Requires rgbmatrix.so present in the same directory.

# PIL Image module (create or load images) is explained here:
# http://effbot.org/imagingbook/image.htm
# PIL ImageDraw module (draw shapes to images) explained here:
# http://effbot.org/imagingbook/imagedraw.htm

import Image
import ImageDraw
import time
from rgbmatrix import Adafruit_RGBmatrix

# Rows and chain length are both required parameters:
matrix = Adafruit_RGBmatrix(32, 1)

# Bitmap example w/graphics prims
image = Image.new("1", (32, 32)) # Can be larger than matrix if wanted!!
draw  = ImageDraw.Draw(image)    # Declare Draw instance before prims
# Draw some shapes into image (no immediate effect on matrix)...
draw.rectangle((0, 0, 31, 31), fill=0, outline=1)
draw.line((0, 0, 31, 31), fill=1)
draw.line((0, 31, 31, 0), fill=1)
# Then scroll image across matrix...
for n in range(-32, 33): # Start off top-left, move off bottom-right
        matrix.Clear()
        # IMPORTANT: *MUST* pass image ID, *NOT* image object!
        matrix.SetImage(image.im.id, n, n)
        time.sleep(0.05)
```

```
# 8-bit paletted GIF scrolling example
image = Image.open("cloud.gif")
image.load()          # Must do this before SetImage() calls
matrix.Fill(0x6F85FF) # Fill screen to sky color
for n in range(32, -image.size[0], -1): # Scroll R to L
        matrix.SetImage(image.im.id, n, 0)
        time.sleep(0.025)

# 24-bit RGB scrolling example.
# The adafruit.png image has a couple columns of black pixels at
# the right edge, so erasing after the scrolled image isn't necessary.
matrix.Clear()
image = Image.open("adafruit.png")
image.load()
for n in range(32, -image.size[0], -1):
        matrix.SetImage(image.im.id, n, 1)
        time.sleep(0.025)

matrix.Clear()
```

I'm drawing shapes but nothing's appearing on the matrix!

PIL graphics draw into an Image buffer, not directly to the display. Call SetImage() (passing the Image id as a parameter) each time you want the matrix updated.

It mostly works but I'm seeing sparkles and glitches!

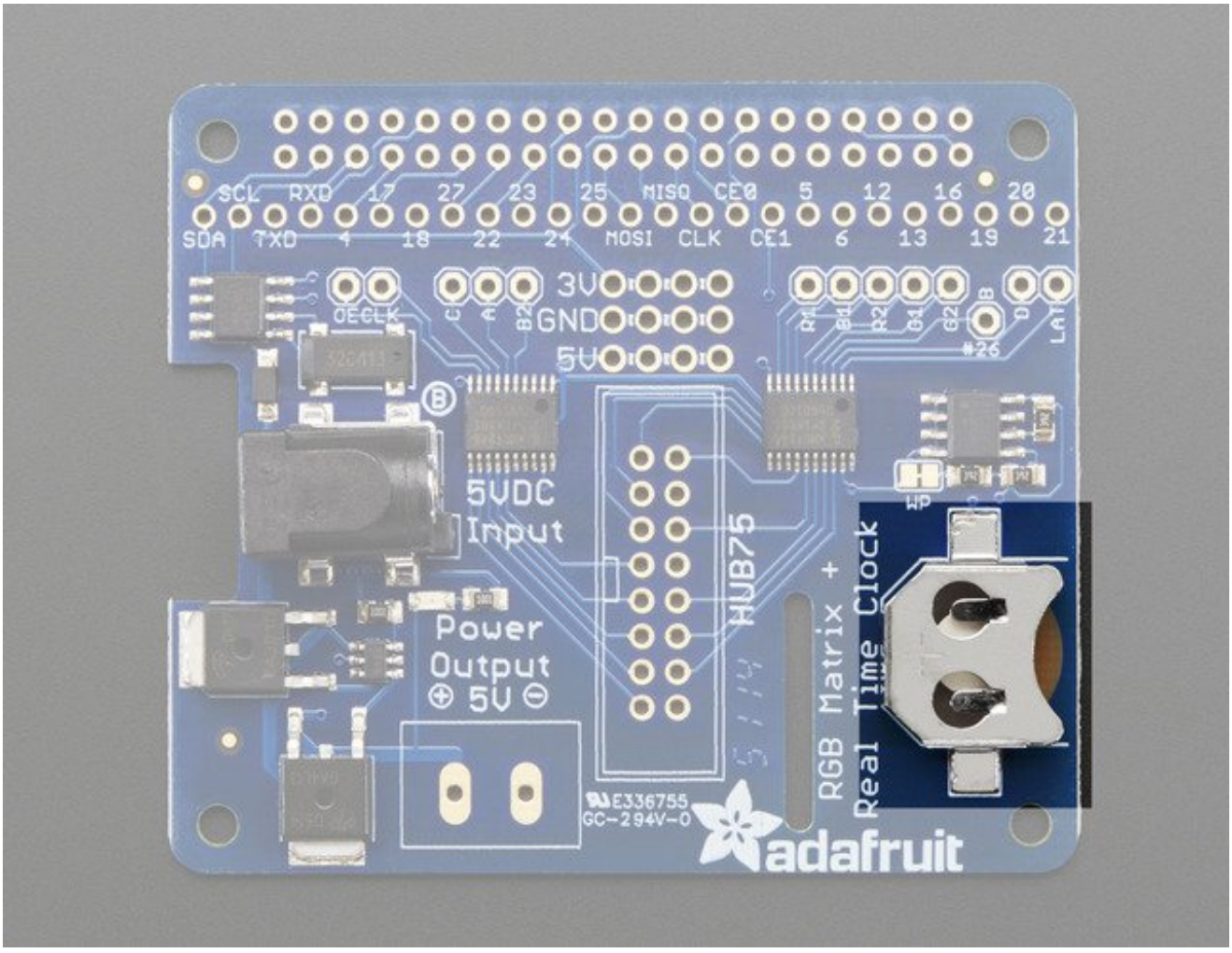
If your Pi is overclocked, or using a Pi 2, it may be necessary to dial back the I/O speed slightly, just like the “-w” option for the led-matrix software previously described. The SetWriteCycles() function sets the I/O speed, higher values are slower. For example:

```
matrix.SetWriteCycles(4)
```

Using the RTC

We had a little space and thought a real time clock would be a nice pairing for this HAT so we tossed on a DS1307 real time clock (RTC). This clock uses a 32.768KHz crystal and backup battery to let the HAT & Pi keep track of time even when power is lost and there's no network access. This makes it great for time displays!

A 12mm 3V Lithium Coin Cell (CR1220) is REQUIRED to use the RTC! It will not work without one! (http://adafru.it/em8)



Once you have installed the coin cell into the HAT, go ahead and[follow this fine Pi+DS1307 tutorial which will show you how to program the current time into the RTC, then have the Pi automatically get the time from it on bootup (http://adafru.it/lF1)

HELP!

I'm using a Raspberry Pi 2 and things are all not working right!

Run **sudo raspi-config** and in the “Overclock” options set the core frequency to 350 MHz or less. Reboot and see if the image is stable. There seems to be an issue when toggling GPIO too quickly.

Also see the “Driving Matrices” page for notes about dialing back the GPIO speed in software.

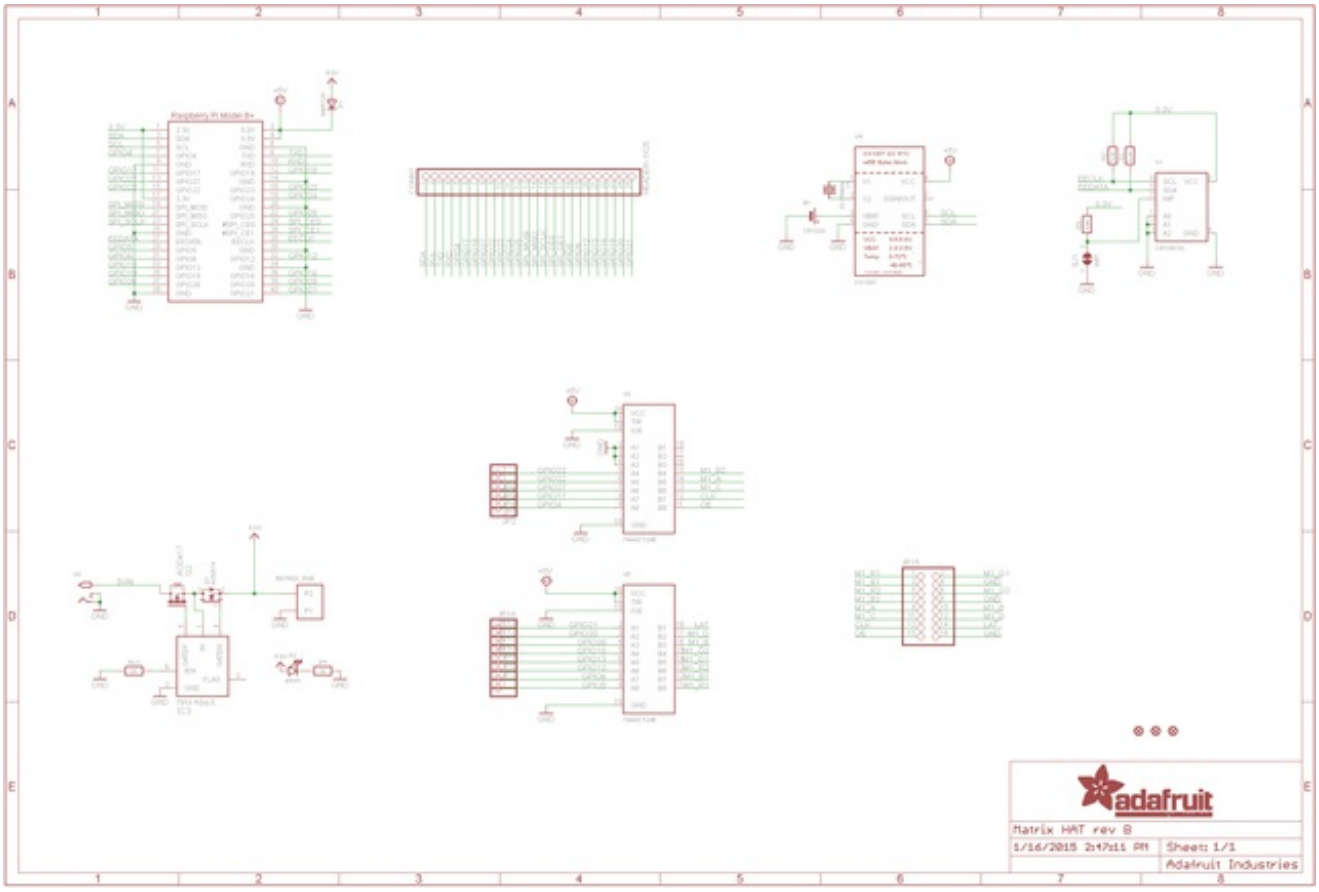
Downloads

Datasheets

- DS1307 Real Time Clock (http://adafru.it/em5)
- MAX4866 5V protection chip (http://adafru.it/em6)
- Fritzing object in the Adafruit Fritzing Library (http://adafru.it/aP3)
- EagleCAD PCB files on GitHub (http://adafru.it/qHc)

Schematic

Click to embiggen



Fabrication Print

Here's the fabrication print with dimensions in inches. This HAT is compatible with the Raspberry Pi mechanical HAT specification!