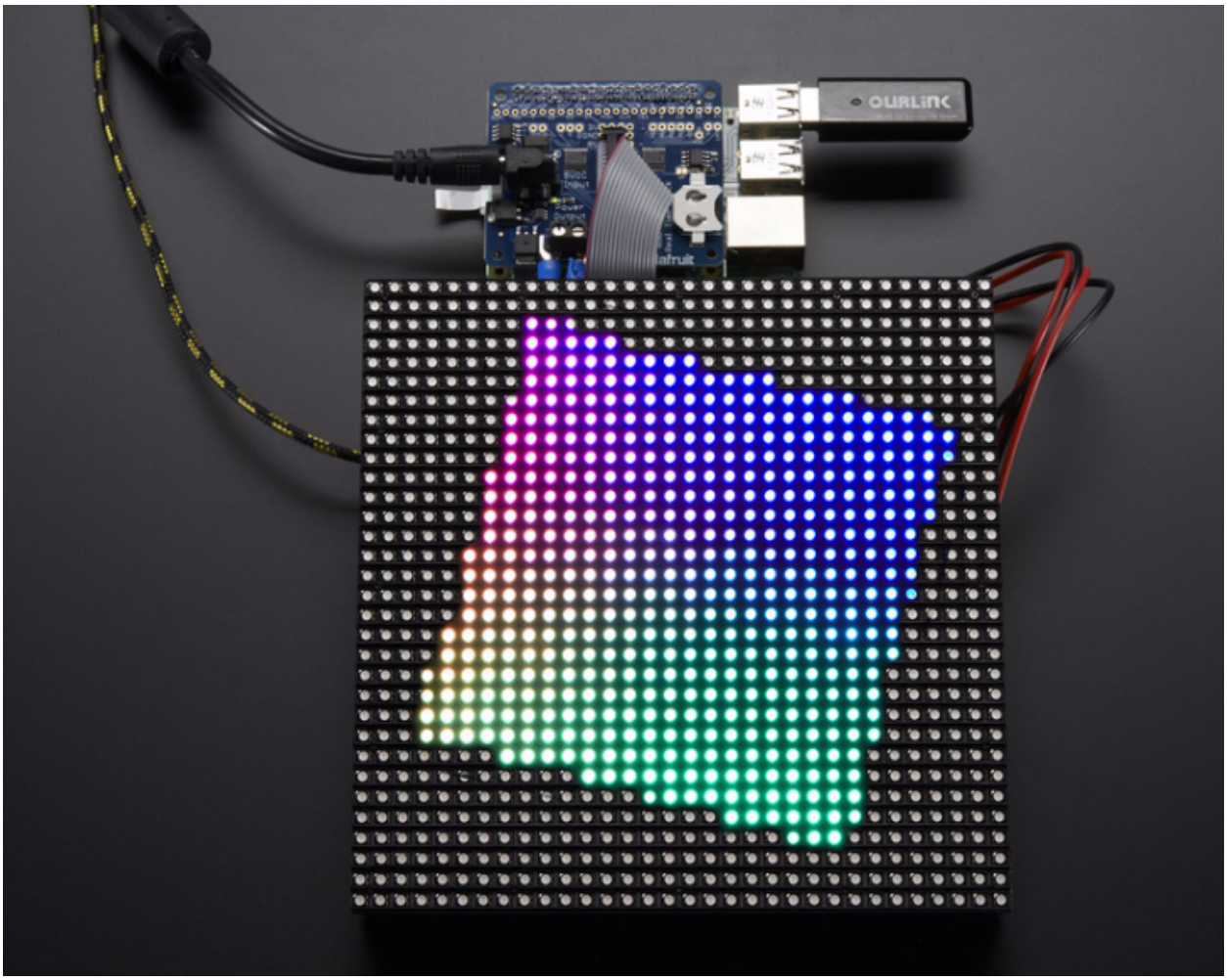
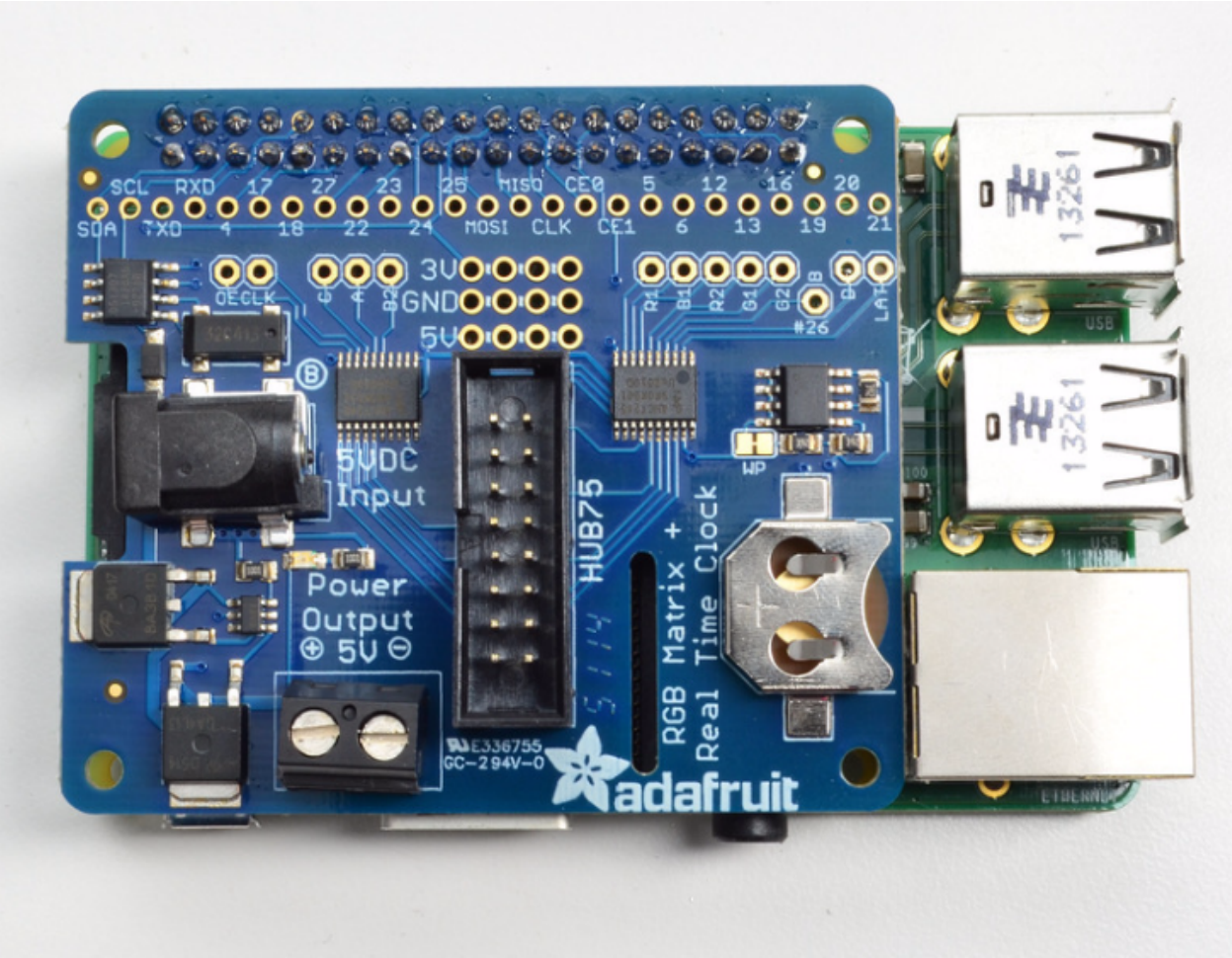
Driving Matrices | Adafruit RGB Matrix + Real Time Clock HAT for Raspberry Pi | Adafruit Learning System



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

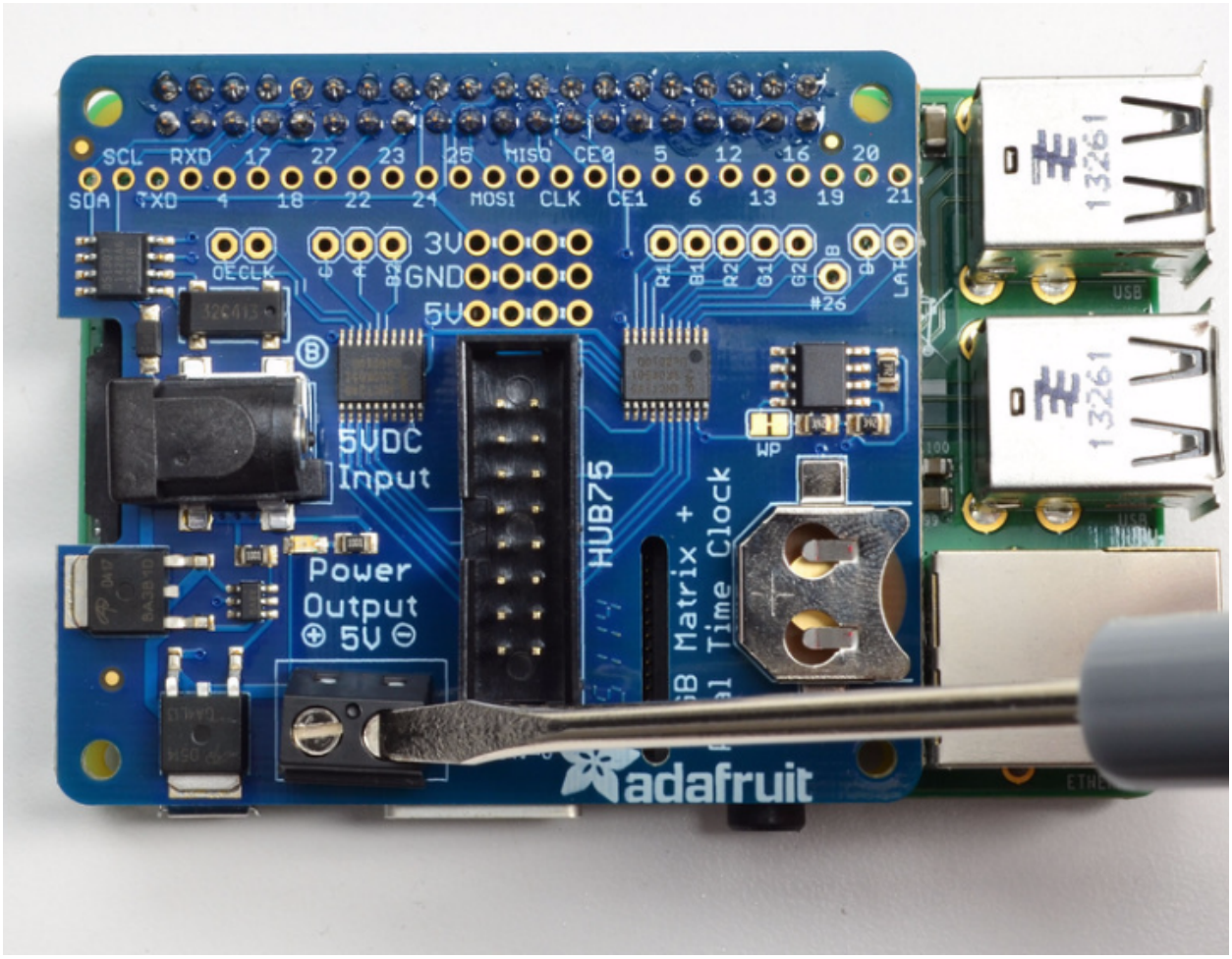
Shut down your Pi and remove power. Plug the HAT on so all the 2x20 pins go into the GPIO header.

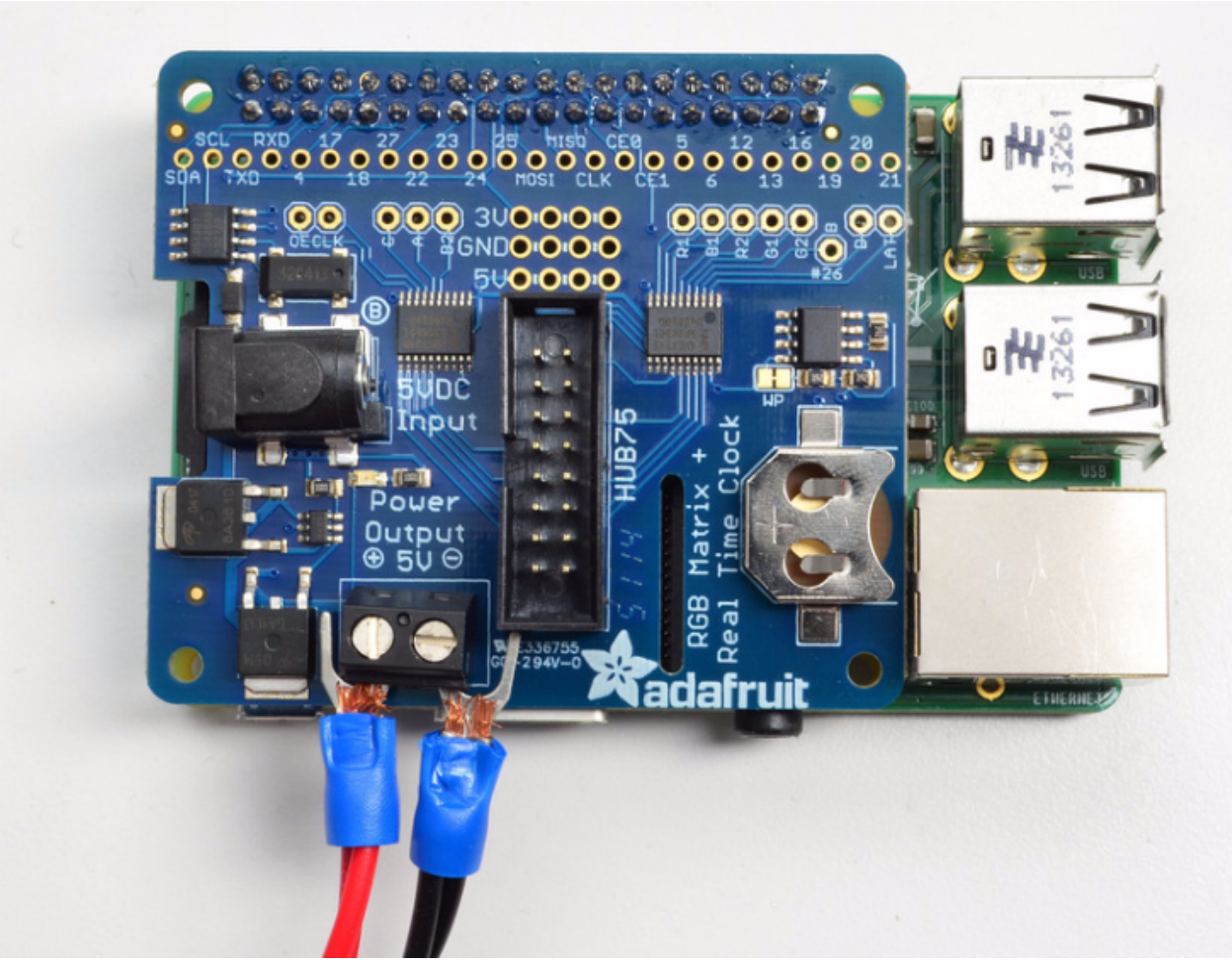


Step 2. Connect Matrix Power cable to terminal block

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.

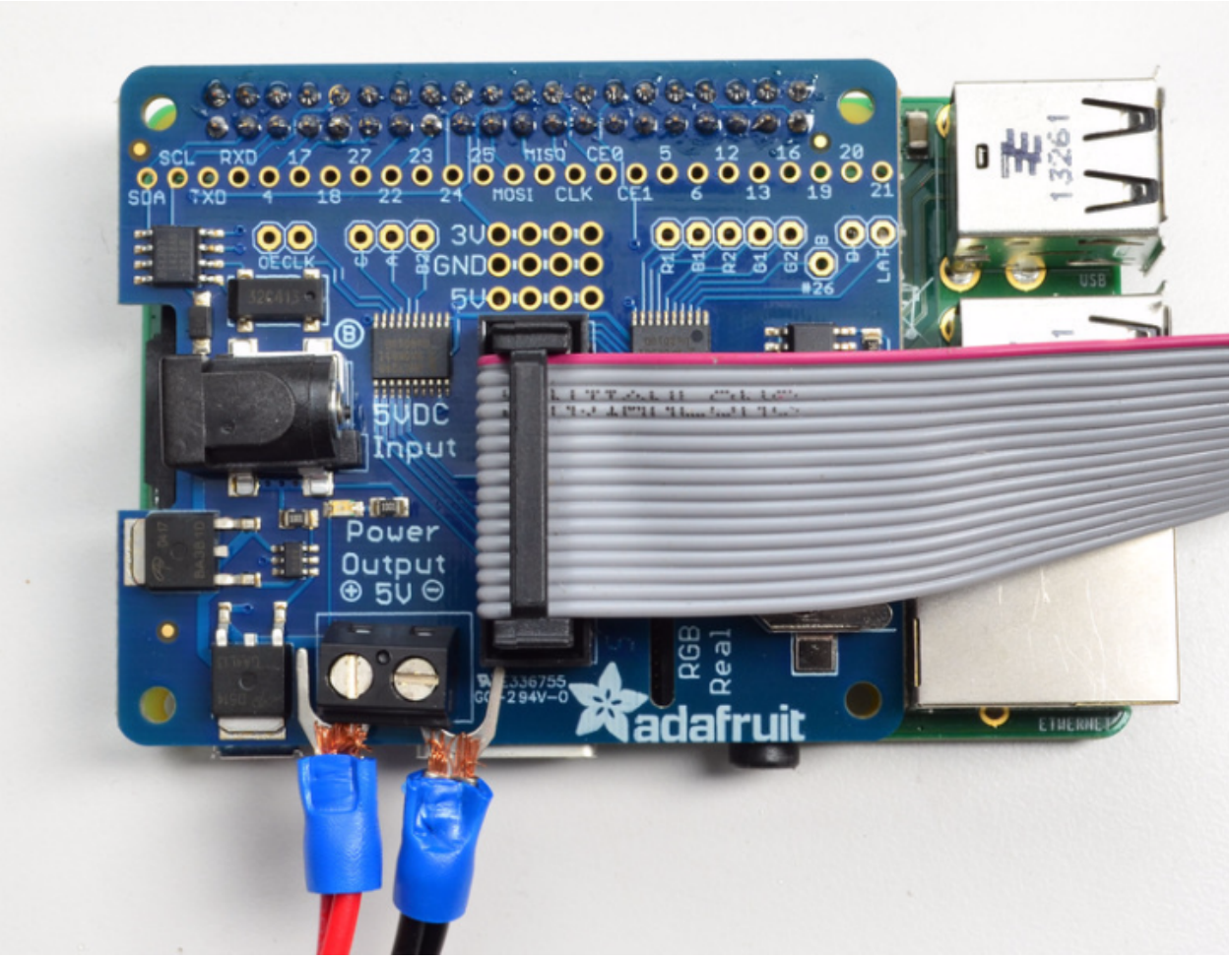

SCL RXD 17 27 23 25 MISO CE0 5 12 16 20
SDA TXD 4 18 22 24 MOSI CLK CE1 6 13 19 21
OECLK
3V
GND
5V
#26
5VDC
Input
Power
Output
⊕ 5V ⊖
HUB75
adafruit
13261
13261
USB
USB



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!



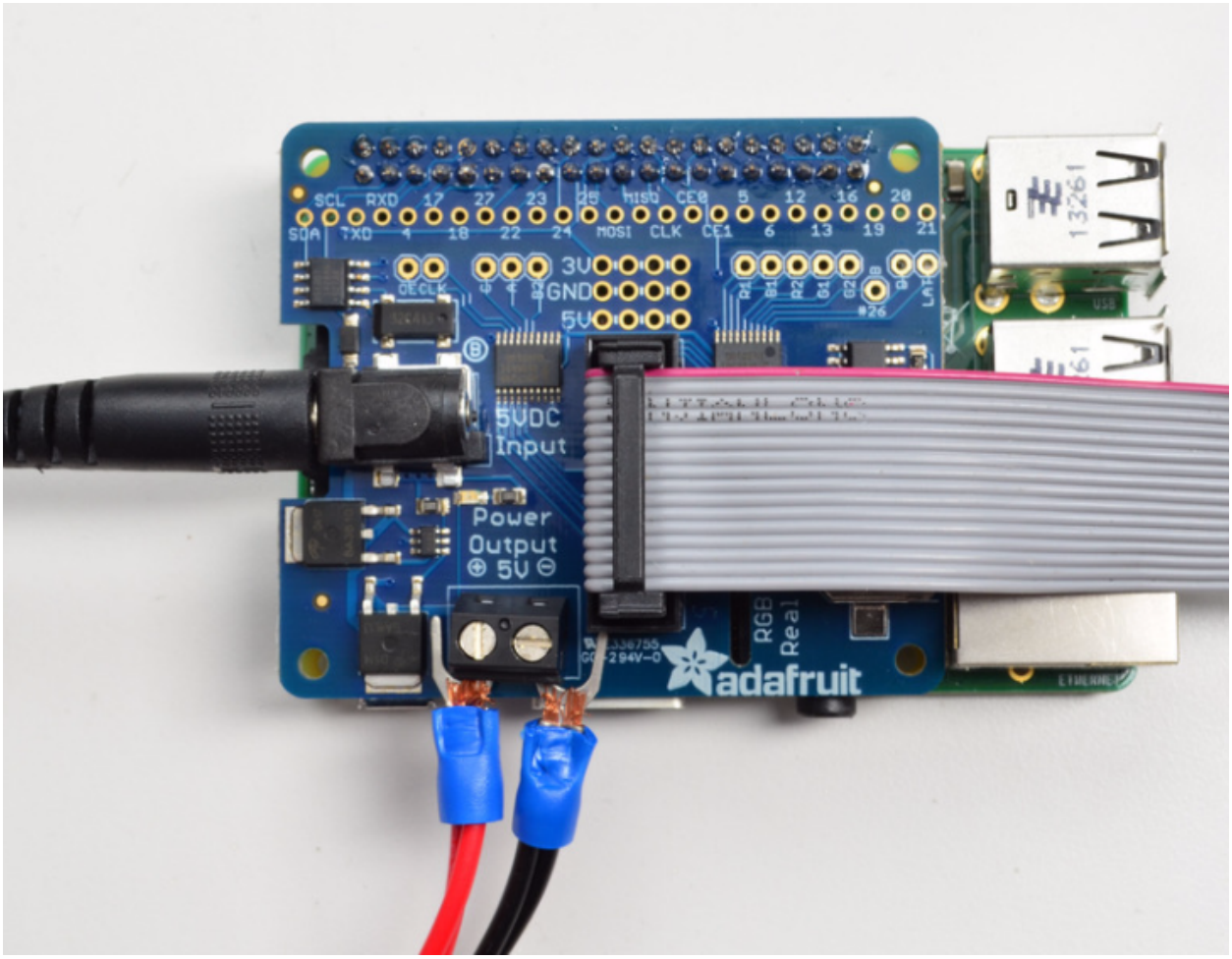
Step 4. Power up your Pi via MicroUSB (optional but suggested)

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

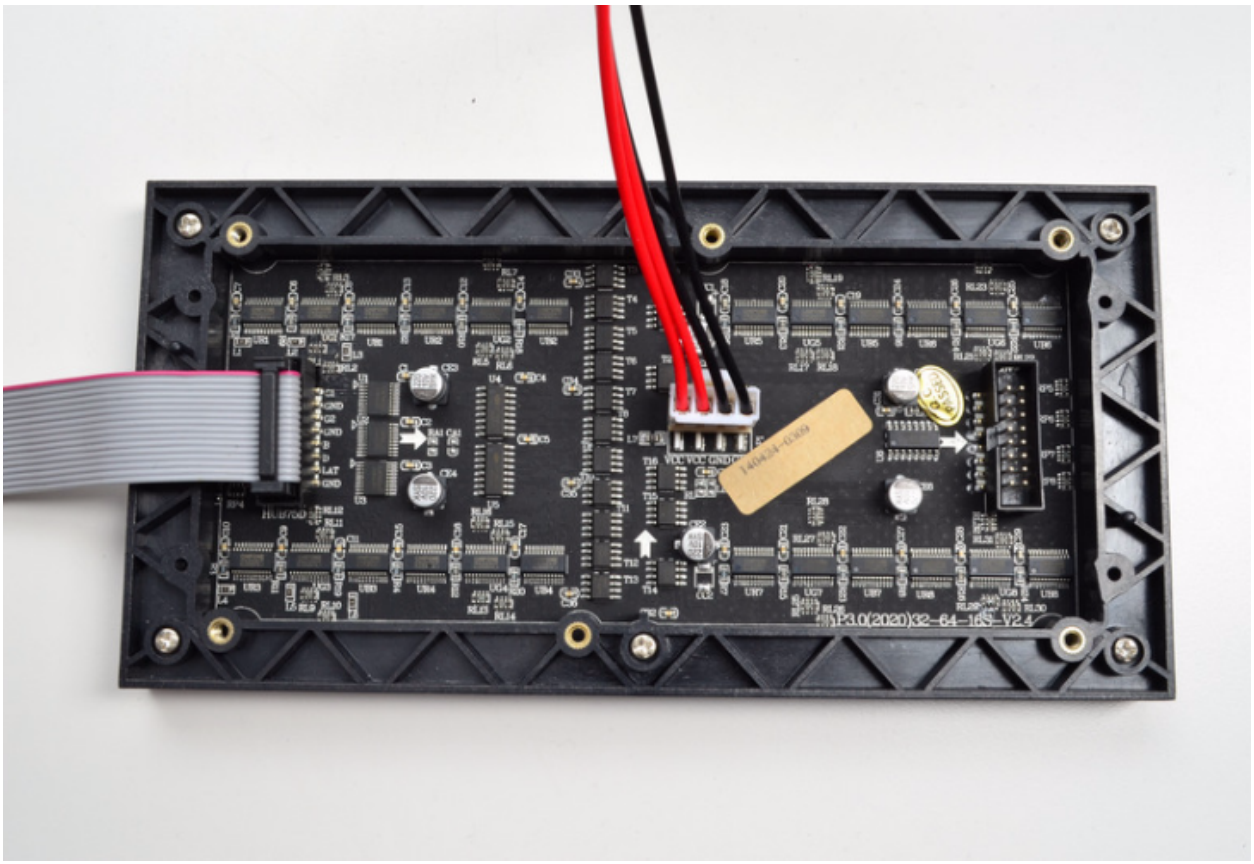
OK now you can plug in your 5V 2A or 4A or larger wall adapter into the HAT. This will turn the green LED on but nothing will display on your matrix yet because no software is running!



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:

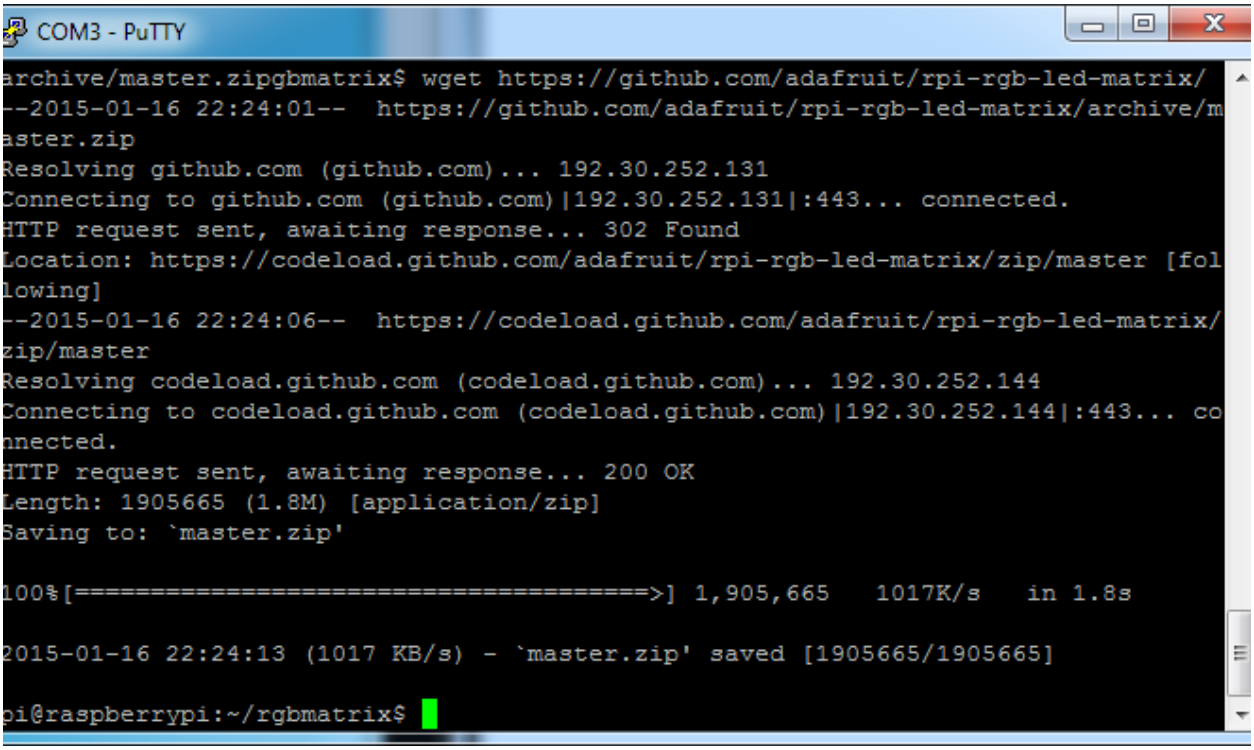
```
wget https://github.com/adafruit/rpi-rgb-led-matrix/archive/master.zip
unzip master.zip
```

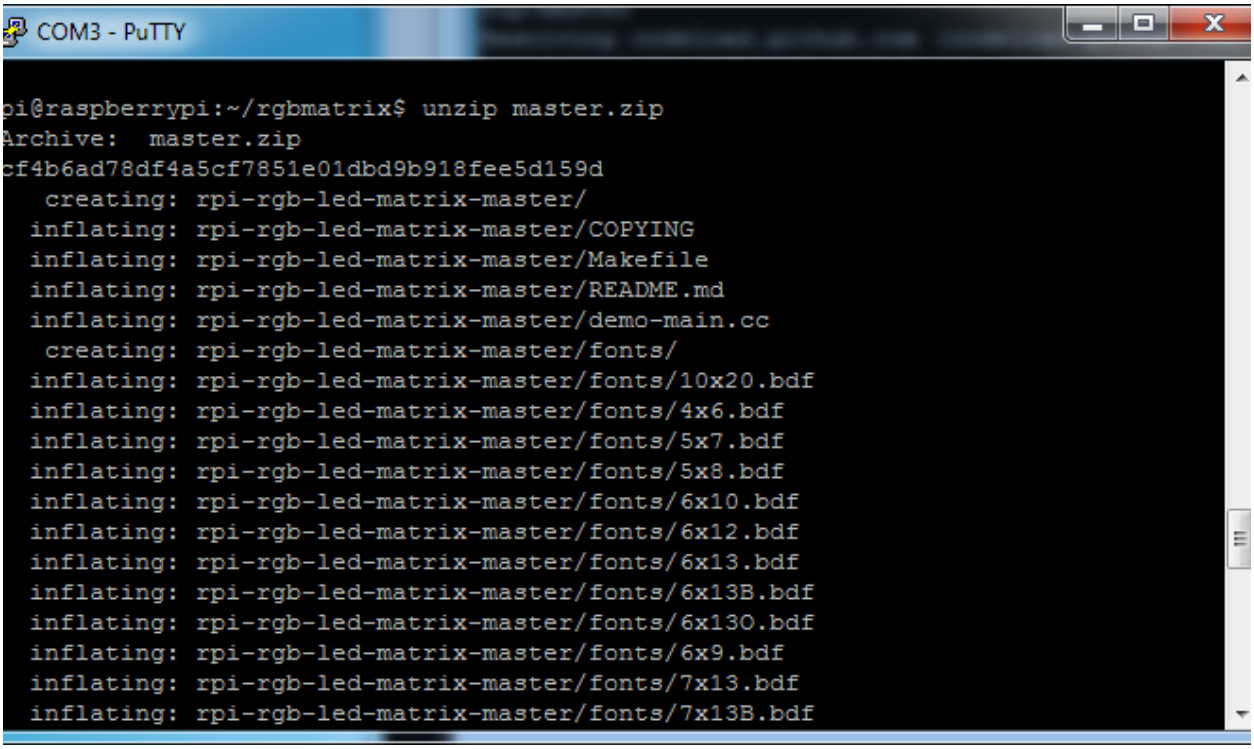
The LED-matrix library is (c) Henner Zeller h.zeller@acm.org with GNU General Public License

Version 2.0 http://www.gnu.org/licenses/gpl-2.0.txt

(Our fork adds HAT/B+/Pi2 support)

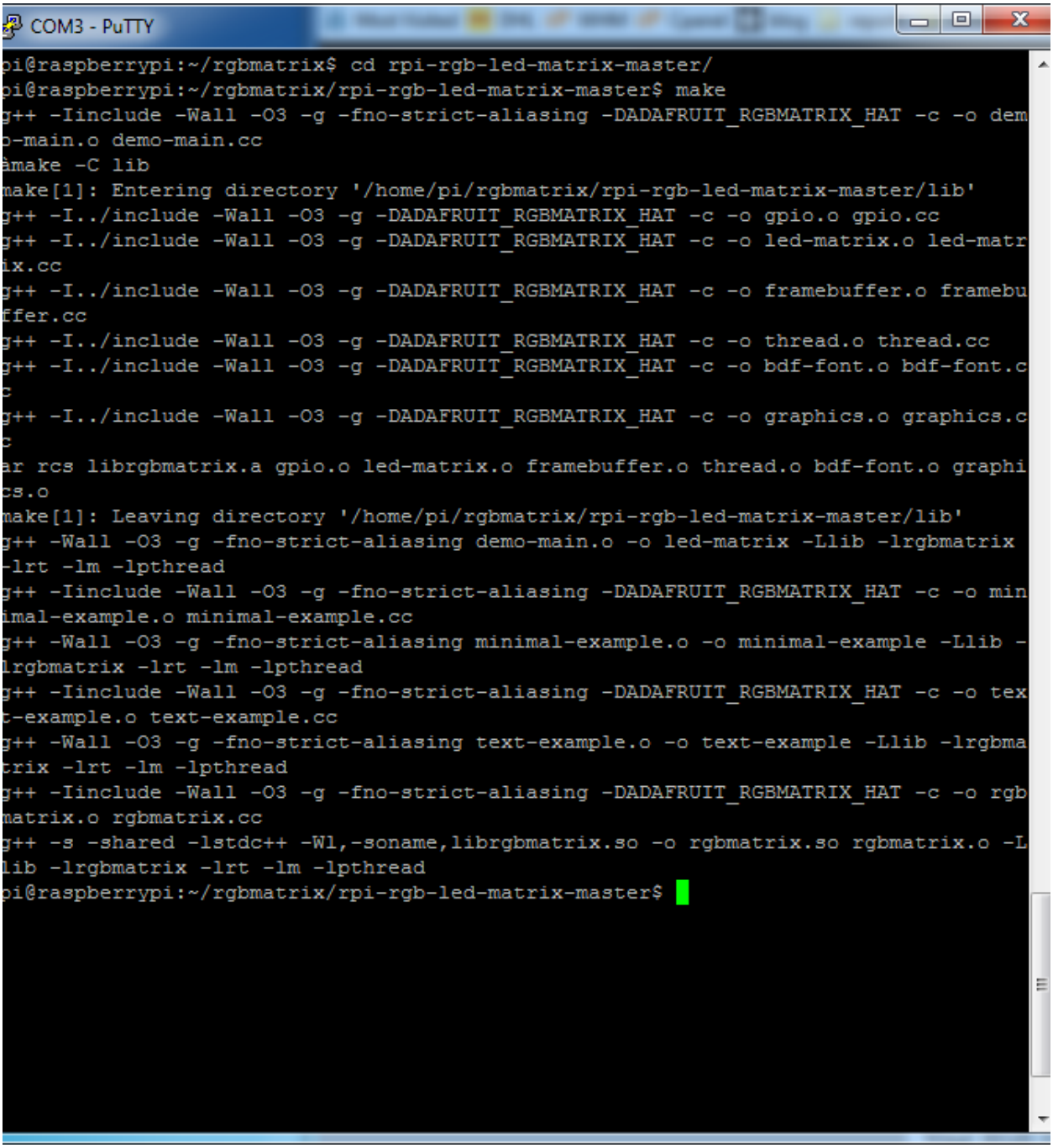
Overclocked Raspberry Pi boards may produce visual glitches on the LED matrix. If you encounter such trouble, first thing to try is to set the board to standard speed using raspi-config, then reboot and retest.





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

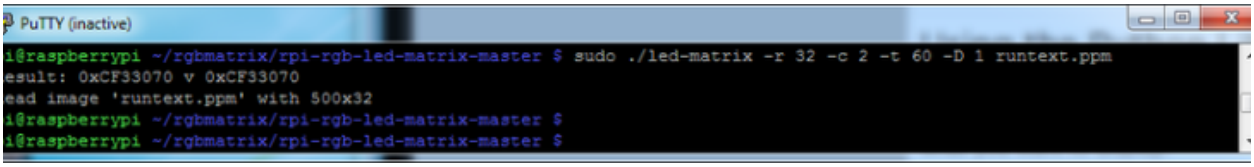
Each matrix is considered 32 pixels wide. If you have multiple matrices chained use **-c** to increase the width. If you have 3 chained together, use **-c 3** If you have a 64x32 matrix, it looks like 2 chained 32x32 so use **-c 2**

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.



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 * 0b000010001)

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

Graphics functions (lines, etc.) are here: The ImageDraw Module

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.

ASSEMBLY

USING THE RTC

Last updated on 2015-05-19 at 01.28.59 PM
Published on 2015-01-16 at 07.31.13 PM