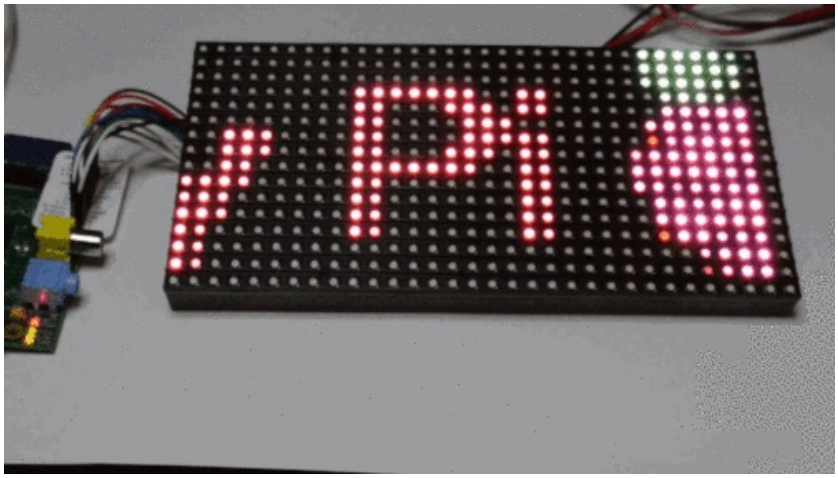
adafruit learning system

Connecting a 16x32 RGB LED Matrix Panel to a Raspberry Pi

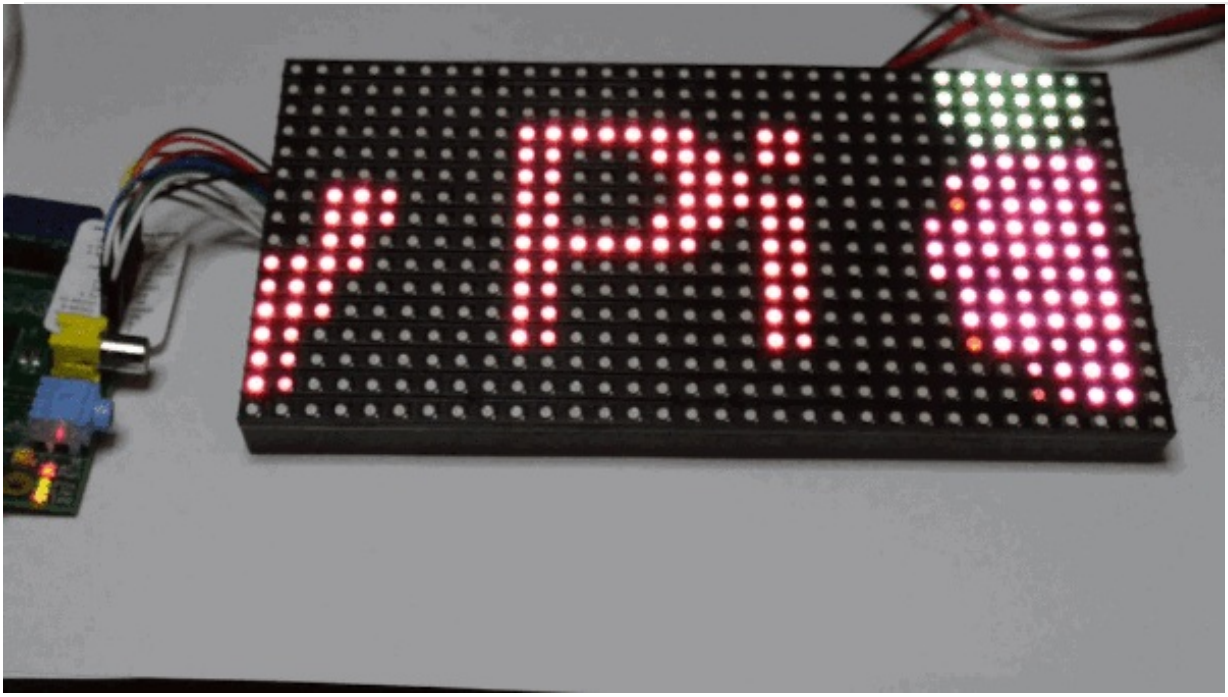
Created by Simon Monk



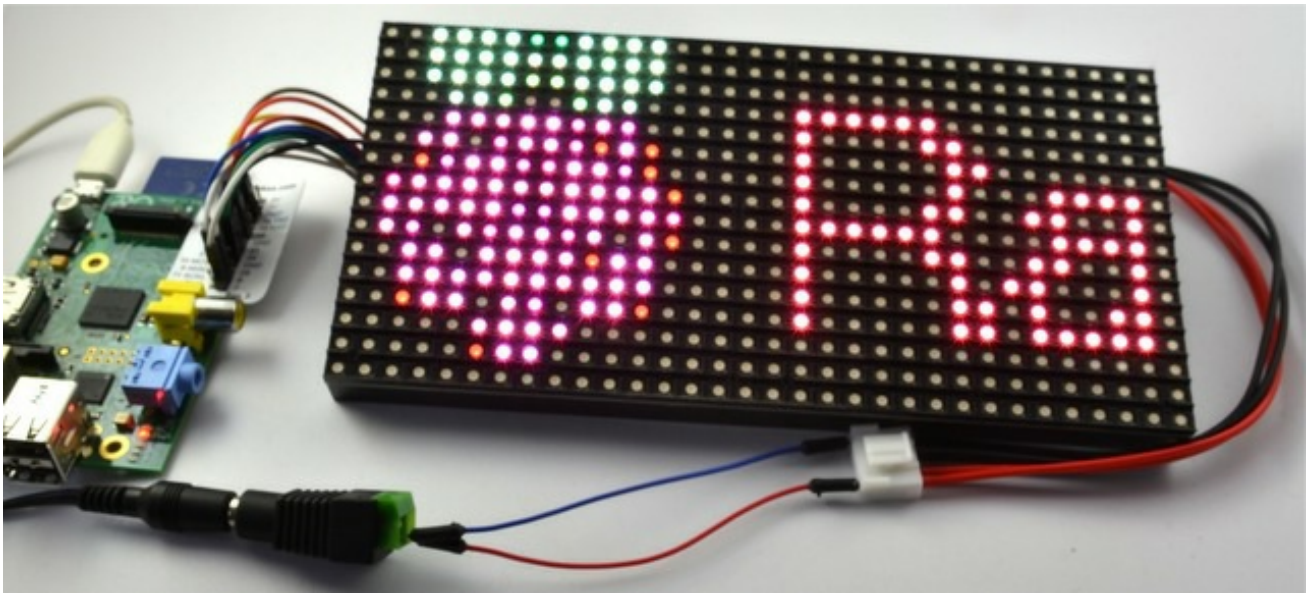
Last updated on 2015-07-31 12:20:12 AM EDT

Guide Contents

Overview



Everyone loves a colorful LED screen! Our 16x32 RGB LED display is a low cost, and easy-to-use arrangement of bright LEDs - just like the ones used in Times Square! This display makes a fantastic addition to your Raspberry Pi. It is pretty easy to wire up, but the display is quite demanding when it comes to getting it to display something...luckily this tutorial has all the code you need to get blinky



This tutorial is based on this one (http://adafru.it/dcp) by Henner Zeller but as well as using Henner's C code to drive the display, we will also experiment (with limited success) in driving the display using Python.

You Will Need

To connect up and use this display, you will need the following items.

	16x32 RGB LED Matrix Panel
	5V 2A Power Supply
	Female/Female Jumper Wires

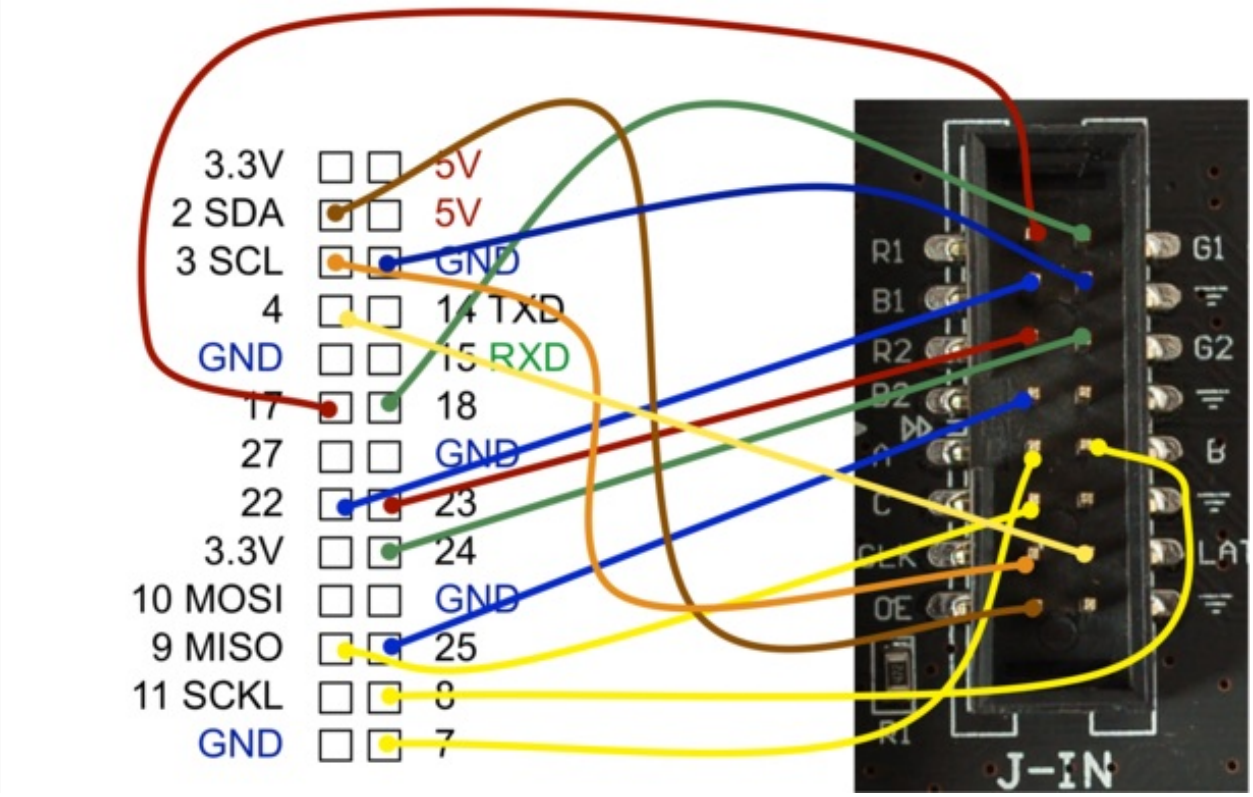
	Male/Male Jumper Wires
	2.1mm to Screw Jack Adapter

If you don't mind cutting the connector off one end of the power cable for the display, and then stripping two of the wires, then you can do without the male to male headers.

Wiring the Display

I noticed that the display is able to pull some power from the GPIO pins when it is not powered up. So I would suggest that, to be on the safe side, you wire up the display to the Pi, with both the Pi and the display powered off.

Connect up the display to the GPIO connector using the female to female jumper wires as shown below:



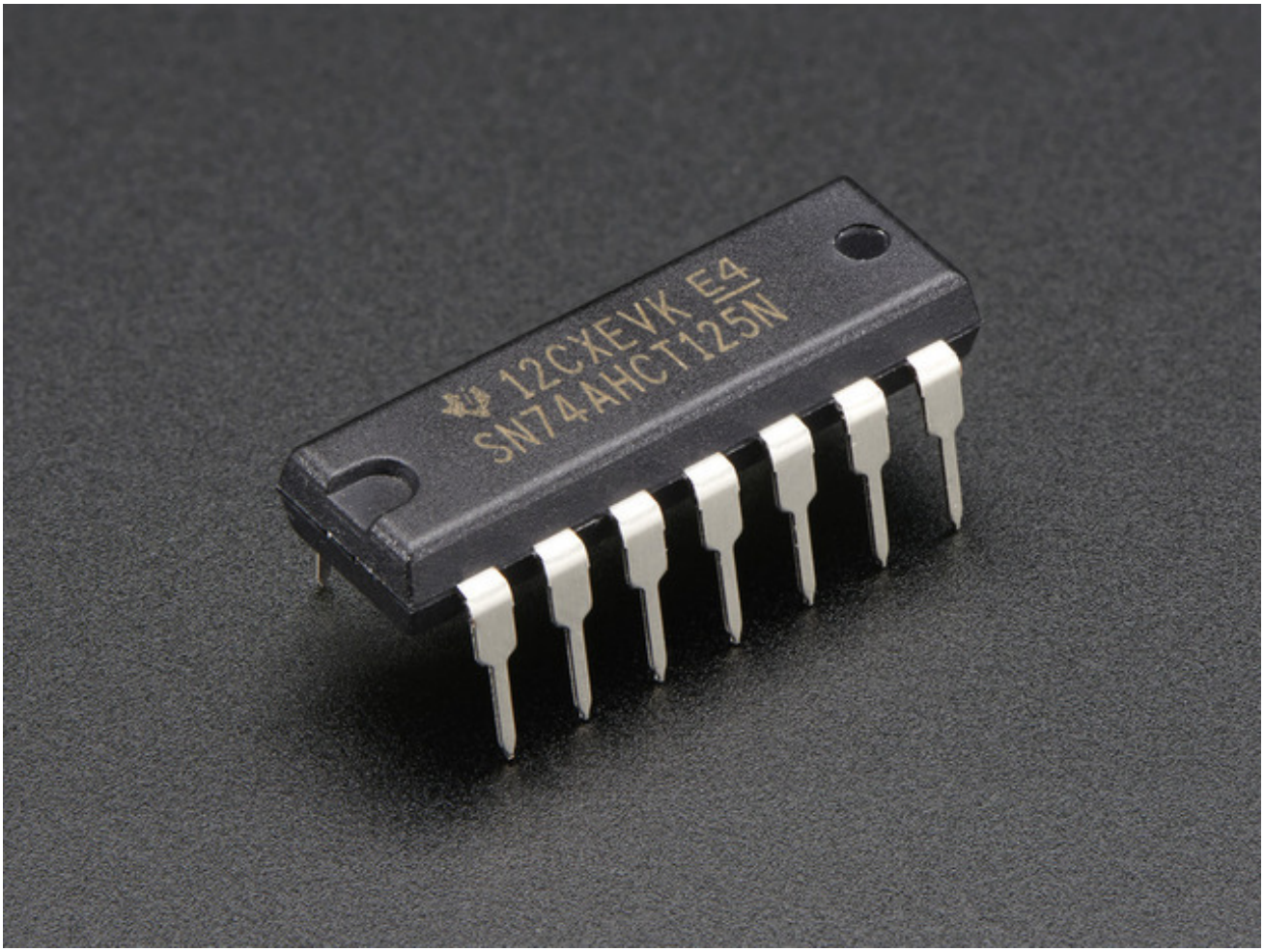
Note that the 3.3V and 5V pins of the Raspberry Pi are on the edge of the Raspberry Pi where the SD card fits!

If you prefer to make the connections using a list, they are as follow:

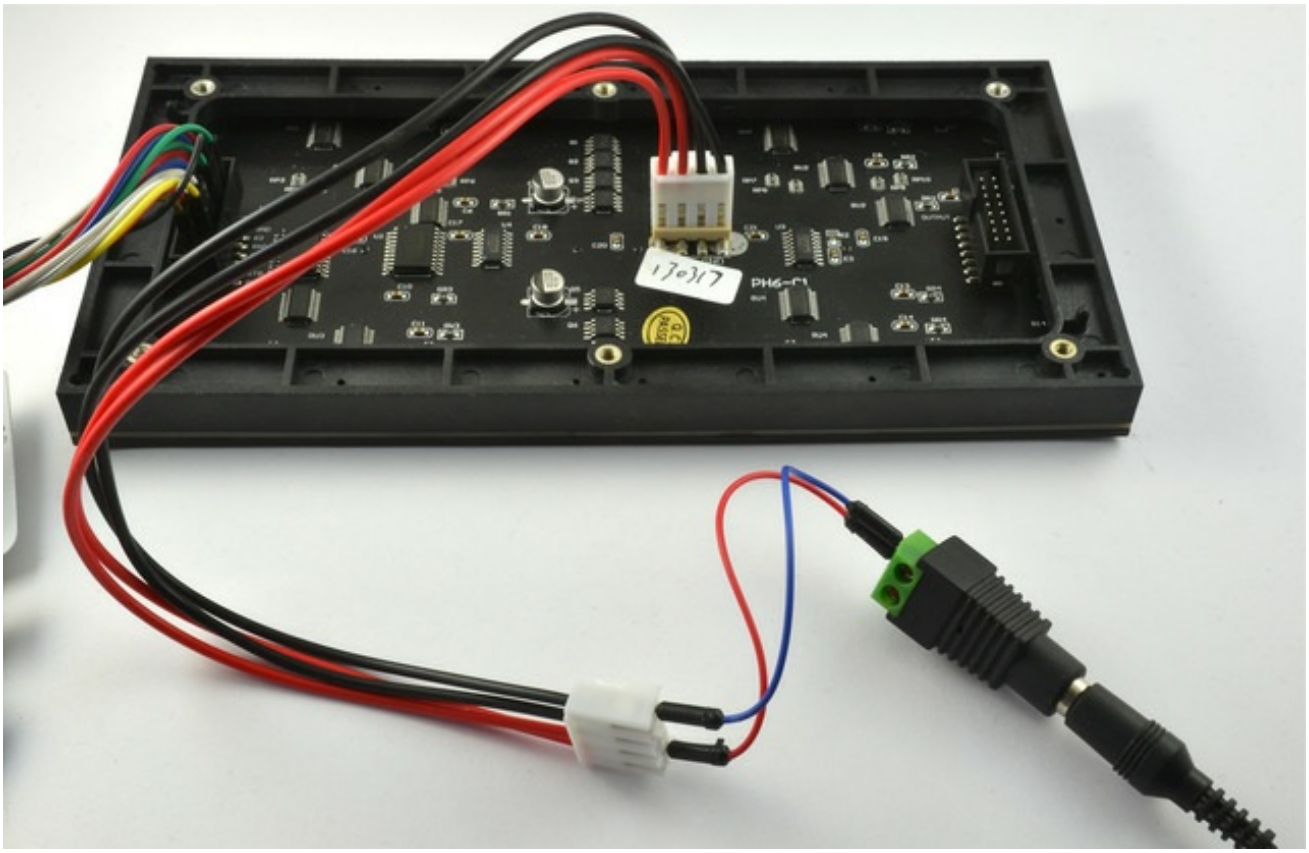
- **GND** on display to GND on the Pi (blue or black)
- **R1** on display to GPIO 17 on the Pi (red)
- **G1** on display to GPIO 18 on the Pi (green)
- **B1** on display to GPIO 22 on the Pi (blue)
- **R2** on display to GPIO 23 on the Pi (red)
- **G2** on display to GPIO 24 on the Pi (green)
- **B2** on display to GPIO 25 on the Pi (blue)
- **A** on display to GPIO 7 on the Pi (yellow or white)
- **B** on display to GPIO 8 on the Pi (yellow or white)
- **C** on display to GPIO 9 on the Pi (yellow or white)

- **OE** on display to GPIO 2 on the Pi (brown)
- **CLK** on display to GPIO 3 on the Pi (orange)
- **LAT** on display to GPIO 4 on the Pi (yellow)

These panels are designed to take 5V logic, and the Pi uses 3.3V logic. For some panels, the 3V logic is not high enough to trigger. If you're having difficulty you may need to use level shifting chips such as https://www.adafruit.com/product/1787 to shift the 3.3V up to 5V!



Using some kind of template like the Raspberry Leaf (http://adafru.it/dcq), soon also to be available ready made from Adafruit, will simplify the process of finding the right pins on the Raspberry Pi. When all the data pins are connected, we can wire up the power supply.

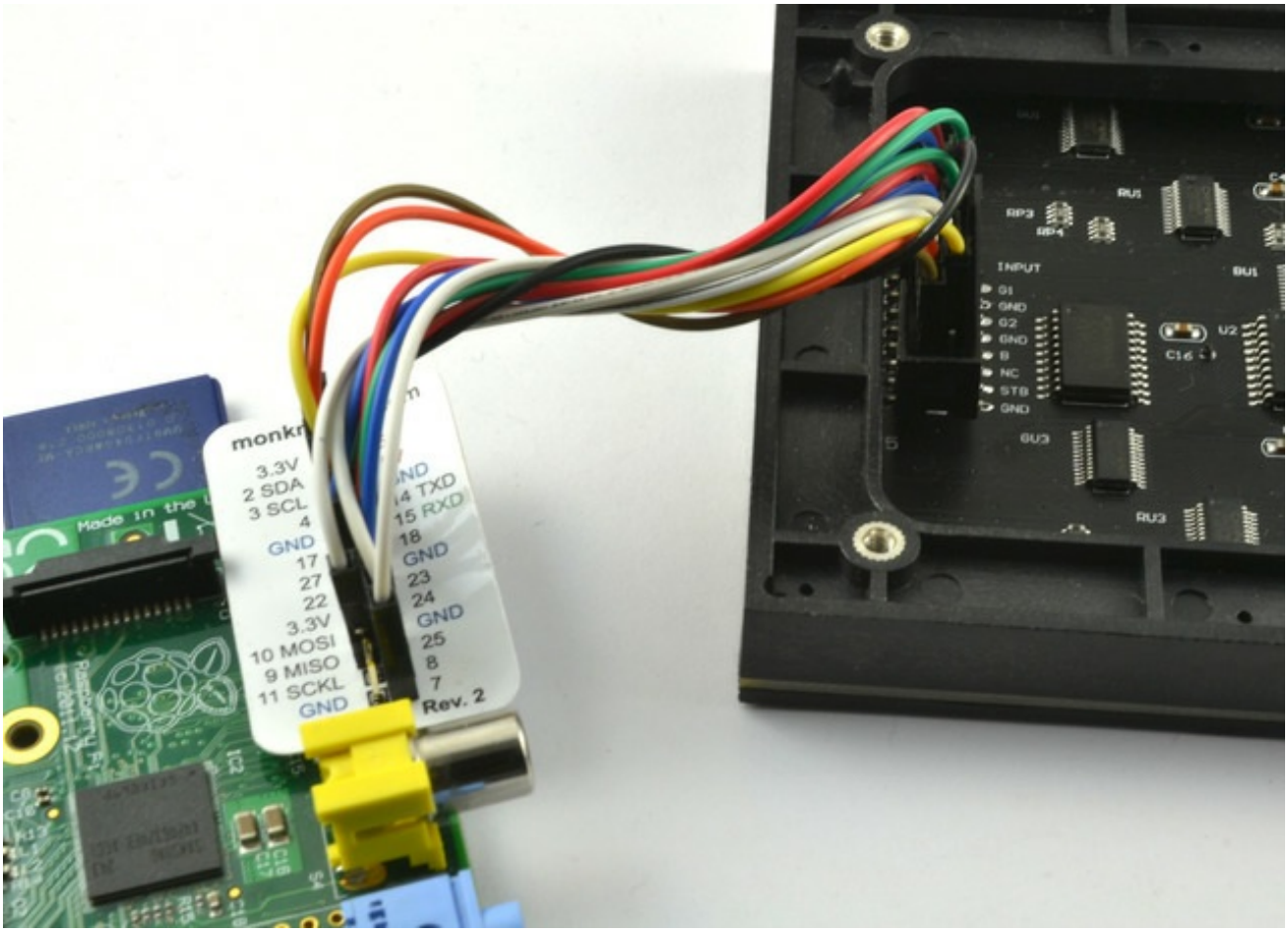


Attach the header pins to the 2.1mm to Screw Jack Adapter. Make sure that the red lead goes to the + mark on the screw terminal. Place the other ends of the jumper wires into the power cable. The power cable has double positive and negative leads. It does not matter which of the pairs of wires you connect to as long as you make sure that positive goes to one of the red leads and negative to one of the black leads.

It is very important that the power supply that you use is 5V. Many power supplies that look like the one used are 12V and if you use a 12V power adapter it may destroy your display.

Testing

Before powering up the Display and Raspberry Pi, check the wiring over carefully and make sure everything is connected as it should be.



Power the display up first and then the Raspberry Pi.

Lets start by creating a directory in which to keep the code. So open an LX terminal session (you can use SSH with the Pi (http://adafru.it/aWc)) if you prefer and enter the following command.

```
$ mkdir display16x32
$ cd display16x32
```

We now need to fetch the code from Henner Zeller's original project using the command:

```
$ git clone https://github.com/hzeller/rpi-rgb-led-matrix/
```

This code is C source code, so we need to compile it before we can run it. It also requires a small

change for the wiring layout we're using. Enter the following commands:

```
$ cd rpi-rgb-led-matrix/lib
$ nano Makefile
```

(Or substitute your editor of preference.)

Enable this line in the Makefile (it's commented out by default):

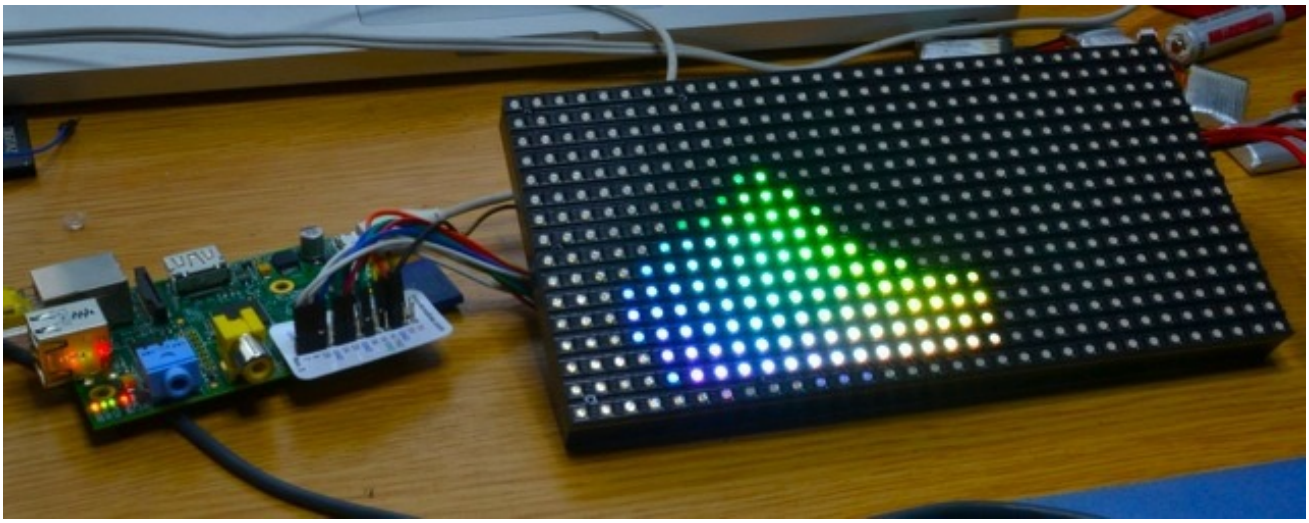
```
DEFINES+=-DRGB_CLASSIC_PINOUT
```

Save the changes to the file, then exit the editor and type...

```
$ cd ..
$ make
```

To run the test program, enter the command:

```
$ sudo ./led-matrix
```



The test program can also display a scrolling message. Infortunately, this message was designed for a 32x32 display and so you will only see the top half of the message. In the next section I will show you how to create your own message image using GIMP.

Try out the oversized message anyway using:

```
$ sudo ./led-matrix 1 runtext.ppm
```

A Scrolling Message

The images must be in a format called ppm (portable pixel format). Many image editors support this format, including the free open source application Gimp.

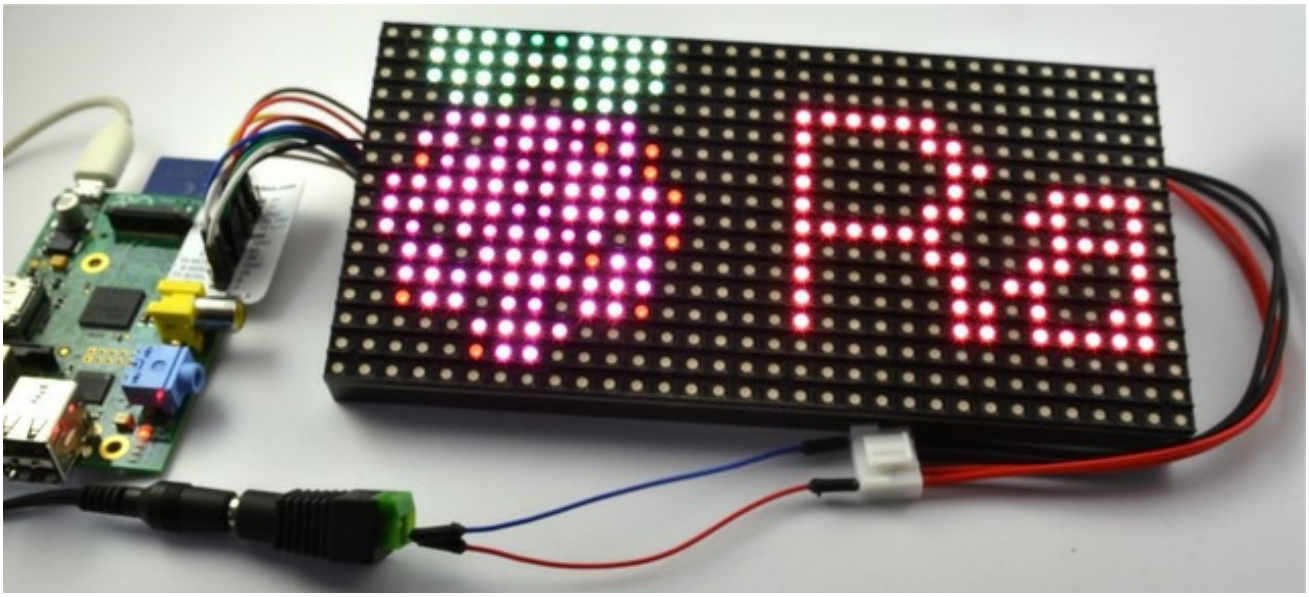
The image needs to be 16 pixels high, but can be wider than the 32 pixels of the display, as the text will scroll horizontally.

This image was created in GIMP. Notice that the background needs to be black.

Use the **File->Export** command on GIMP to export the image as ppm and then transfer the file into the directory **/home/pi/display16x32/rpi-rgb-led-matrix** on your Raspberry Pi. You can just use a flash drive to copy the file over.
Run the command again, this time using the name that you gave the file:

```
$ sudo ./led-matrix 1 my_image.ppm
```

The image should then appear on the display.

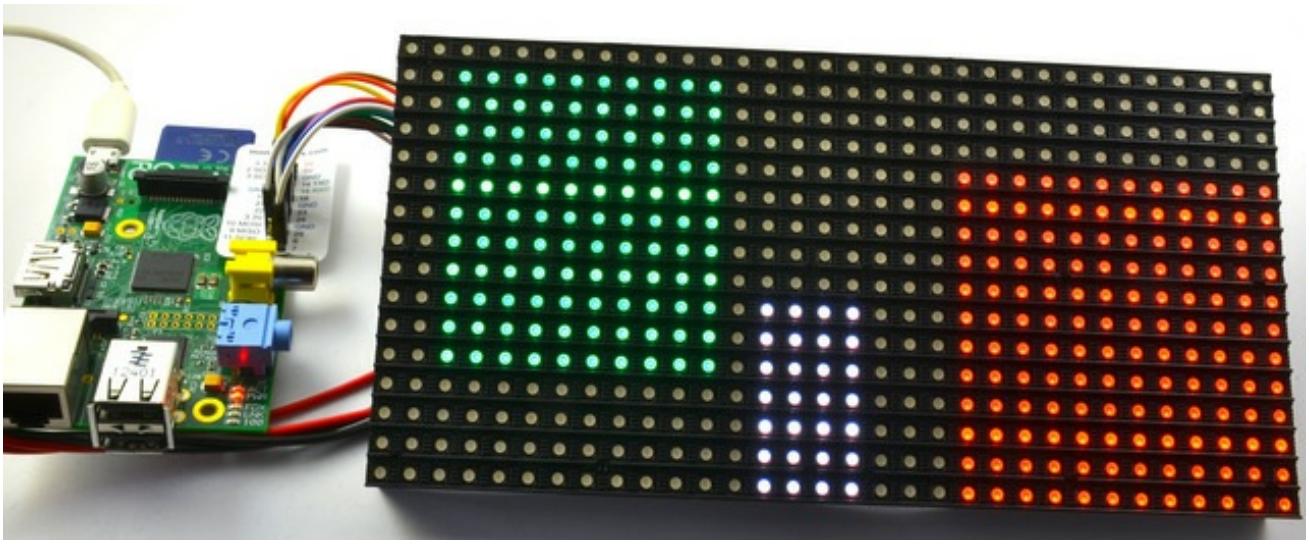


Experimental Python Code

The display uses shift registers, those are all the chips on the back of the screen. This requires the Pi to do a lot of work bit-banging the pixels onto the screen. You can read more about how this display works here (http://adafru.it/dcr).

As an experiment, I decided to see how practical it would be to try and drive the display from Python code, rather than using the much faster C code.

I had fairly limited success in this. The display refreshes with a bit of flicker, is rather uneven in light intensity and dimmer than the C code. So if anyone can improve the code and make it work better, we would love to hear from you and be happy to update this section with due credit.



The code is relatively compact. It is crude, doing nothing in the way of PWM and can therefore only display eight colors.

If you want to give it a go, the wiring is exactly the smae as before, and you will just need to paste the following code into a file called test_display.py

```
import RPi.GPIO as GPIO
import time

delay = 0.000001

GPIO.setmode(GPIO.BCM)
red1_pin = 17
green1_pin = 18
blue1_pin = 22
red2_pin = 23
green2_pin = 24
blue2_pin = 25
clock_pin = 3
a_pin = 7
b_pin = 8
```

```
b_pin = 8
c_pin = 9
latch_pin = 4
oe_pin = 2

GPIO.setup(red1_pin, GPIO.OUT)
GPIO.setup(green1_pin, GPIO.OUT)
GPIO.setup(blue1_pin, GPIO.OUT)
GPIO.setup(red2_pin, GPIO.OUT)
GPIO.setup(green2_pin, GPIO.OUT)
GPIO.setup(blue2_pin, GPIO.OUT)
GPIO.setup(clock_pin, GPIO.OUT)
GPIO.setup(a_pin, GPIO.OUT)
GPIO.setup(b_pin, GPIO.OUT)
GPIO.setup(c_pin, GPIO.OUT)
GPIO.setup(latch_pin, GPIO.OUT)
GPIO.setup(oe_pin, GPIO.OUT)

screen = [[0 for x in xrange(32)] for x in xrange(16)]

def clock():
    GPIO.output(clock_pin, 1)
    GPIO.output(clock_pin, 0)

def latch():
    GPIO.output(latch_pin, 1)
    GPIO.output(latch_pin, 0)

def bits_from_int(x):
    a_bit = x & 1
    b_bit = x & 2
    c_bit = x & 4
    return (a_bit, b_bit, c_bit)

def set_row(row):
    #time.sleep(delay)
    a_bit, b_bit, c_bit = bits_from_int(row)
    GPIO.output(a_pin, a_bit)
    GPIO.output(b_pin, b_bit)
    GPIO.output(c_pin, c_bit)
    #time.sleep(delay)

def set_color_top(color):
    #time.sleep(delay)
    red, green, blue = bits_from_int(color)
    GPIO.output(red1_pin, red)
    GPIO.output(green1_pin, green)
```

```
    GPIO.output(blue1_pin, blue)
    #time.sleep(delay)

def set_color_bottom(color):
    #time.sleep(delay)
    red, green, blue = bits_from_int(color)
    GPIO.output(red2_pin, red)
    GPIO.output(green2_pin, green)
    GPIO.output(blue2_pin, blue)
    #time.sleep(delay)

def refresh():
    for row in range(8):
        GPIO.output(oe_pin, 1)
        set_color_top(0)
        set_row(row)
        #time.sleep(delay)
        for col in range(32):
            set_color_top(screen[row][col])
            set_color_bottom(screen[row+8][col])
            clock()
        #GPIO.output(oe_pin, 0)
        latch()
        GPIO.output(oe_pin, 0)
        time.sleep(delay)

def fill_rectangle(x1, y1, x2, y2, color):
    for x in range(x1, x2):
        for y in range(y1, y2):
            screen[y][x] = color


def set_pixel(x, y, color):
    screen[y][x] = color

fill_rectangle(0, 0, 12, 12, 1)
fill_rectangle(20, 4, 30, 15, 2)
fill_rectangle(15, 0, 19, 7, 7)

while True:
    refresh()
```

You can run the program using the command:

```
$ sudo python test_display.py
```

Looking at the end of the file, you can see the "fill_rectangle" and "set_pixel" functions that you can use to draw on the screen. The final argument of both of these functions is the color that should be a number between 0 and 7.

The display refreshing should really take place in a separate thread of execution, but as I said, this is more of an experiment than anything else.

Next Steps

Using a scrolling image file is the easiest way to get your raspberry Pi displaying something fancy. You could create a whole series of images, or even create images on the fly and then send them to the test program to be displayed.

You may also like to continue the Python experimentation!

About the Author.

As well as contributing lots of tutorials about Raspberry Pi, Arduino and now BeagleBone Black, Simon Monk writes books about open source hardware. You will find his books for sale here (http://adafru.it/caH) at Adafruit.