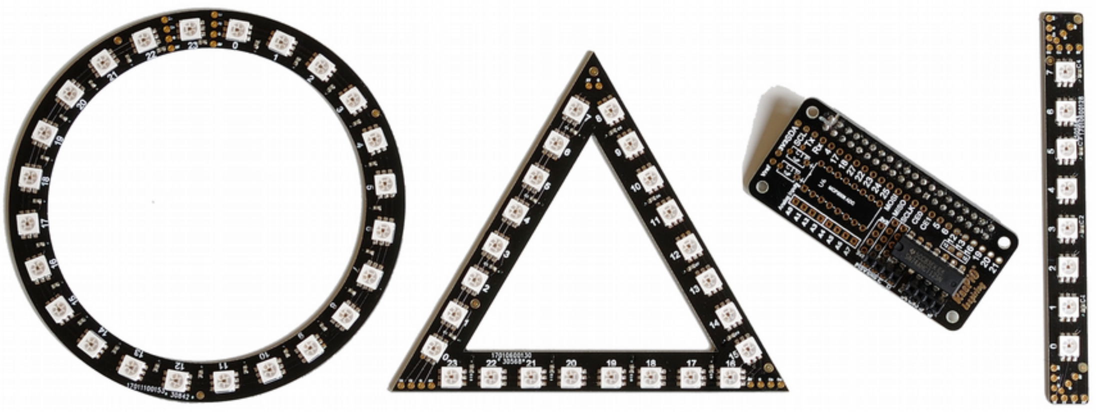
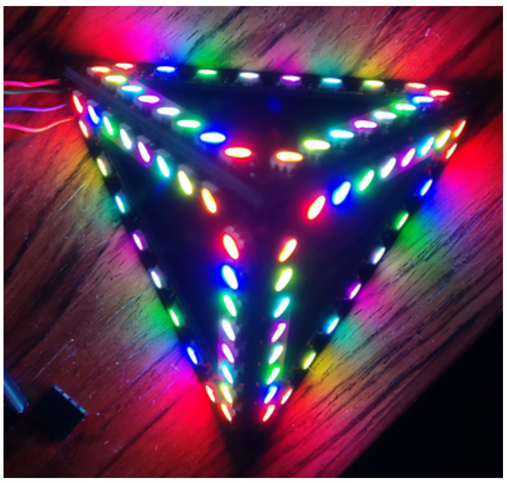
Using Your

RasPiO® InsPiRing





by Alex Eames

Introduction

Experiment with full colour LEDs

The RasPiO®[1] InsPiRing system offers a compact, affordable, easy, modular way to experiment with programmable full colour (RGB) LEDs. It works on Raspberry Pi, Arduino, Micro:Bit, ESP8266, ATtiny or pretty much any micro-controller.

The boards can connect together to make 2D or 3D objects. You can make all sorts of things. A few ideas are...

- Light Pyramid (Make a Pyramid)
- Internet NTP Clock (NTP Clock)
- TV simulator security device (Make a TV Simulator)
- Animated signage
- Dials & gauges to show output from (analog) sensors
- Persistance of vision projects
- Natural light alarm clock
- Mood light
- Controllable light source for macro photography/videography
- Christmas or other holiday decorative lighting
- Eye-catching wearable

All Python code in this guide and in the RasPiO InsPiRing GitHub Repository is in Python 3. All scripts may be run with...

```
python3 scriptname.py
```

...but you will need to download the scripts and make sure SPI is enabled before you start. (see Pi Configuration & Software Installation)

1 RasPiO is a trademark of Alex Eames. Raspberry Pi is a trademark of the Raspberry Pi Foundation

RasPiO InsPiRing Instructions

RasPiO InsPiRing LEDs are incredibly bright. Please be sensible (or wear shades).

The RasPiO InsPiRing driver pHAT uses the SPI ports on the Raspberry Pi. All other GPIO ports[2] are broken out and available for use in your projects. The BCM GPIO port numbering scheme is used. This is a perfect match for GPIO Zero.

Hardware Technical Overview

SPI

The RasPiO InsPiRing driver pHAT uses hardware SPI on the Raspberry Pi. CE0 is connected to the MCP3008 analog to digital converter (if fitted). CE1 is connected to the 74AHCT125N buffer chip (1OE & 4OE). This mechanism allows simultaneous use of both SPI channels. MOSI and SCLK are shared, but the buffer chip prevents the 'wrong' signals from reaching the LEDs. MISO is not used by the LEDs (but is used by the optional MCP3008).

Buffer Chip – Why?

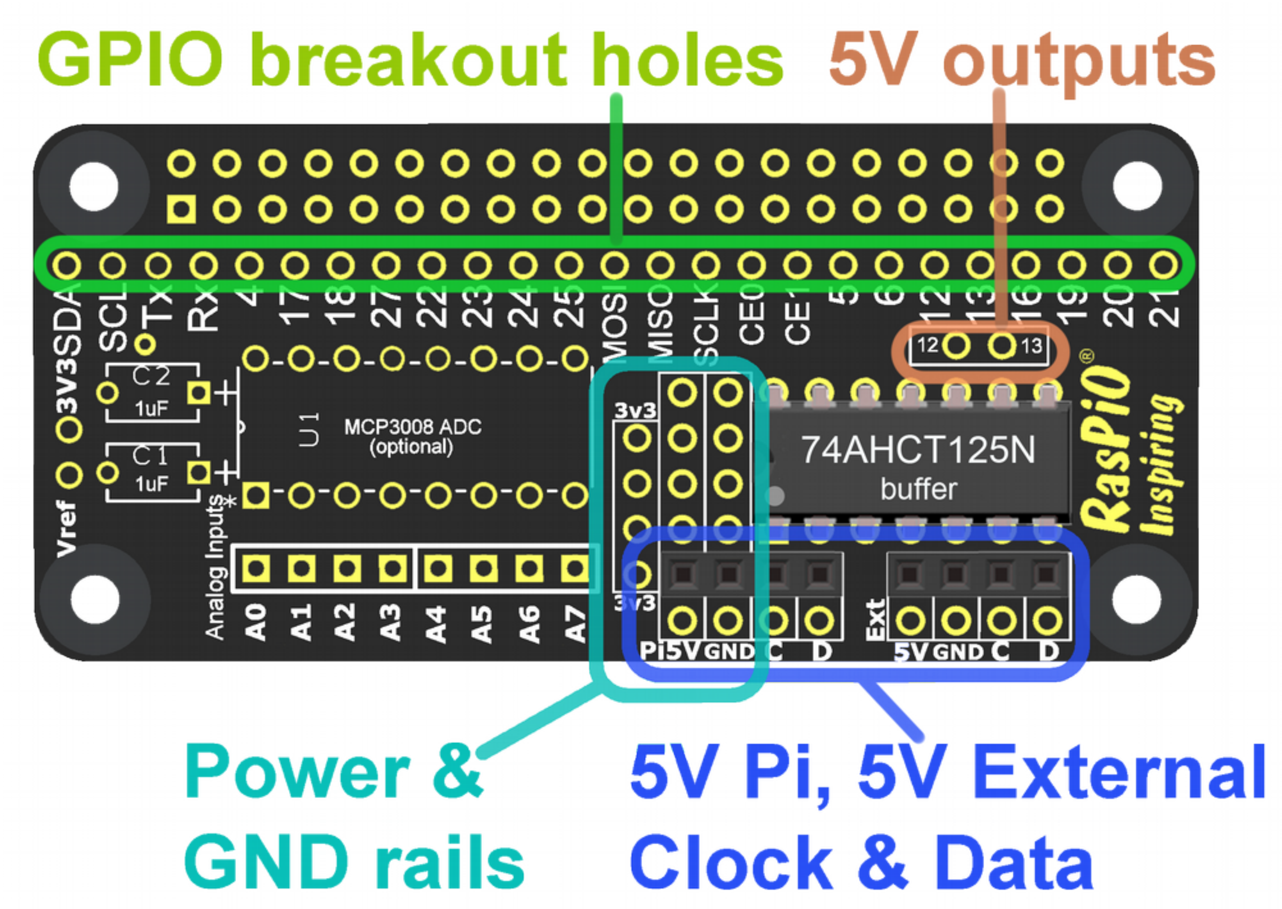
A 74AHCT125N buffer chip was selected for a couple of reasons...

1. It enables the LEDs to be powered at 5V and driven by 5V logic. The Pi can only drive 3V3 logic on its own. This is considered a robust method which also protects the Pi and enables the LEDs to run at maximum brightness.
2. SK9822 LEDs don't have a chip-select mechanism built-in. So if you want to use more than one device on the Pi's hardware SPI, the shared MOSI and SCLK channels can cause the LEDs to behave unpredictably. Tying Pi's CE1 to buffer's 1OE & 4OE prevents unwanted signals reaching the LEDs and causing uncontrolled output.

2 Except GPIO26 – pHAT is one hole too short to fit them all ☹

Know Your InsPiRing Boards

InsPiRing Driver



RasPiO InsPiRing driver pHAT (basic)

GPIO Breakouts (Green)

All GPIO ports[3] on the Raspberry Pi are broken out in a single column across the pHAT. All apart from SPI are available for use.

Two 5V Outputs (Brown)

Holes labelled 12 & 13 next to the buffer chip are able to be used as 5V outputs on GPIOs 12 & 13. There were two unused outputs on the 74AHCT125N buffer chip, so it was decided to make them available for use. However, you can still use GPIO12/13 directly on the Pi (as well or instead) by using the breakout holes in the single column.

3 Except GPIO26

LED Board Connections (Blue)

There are two sets of connections for RasPiO InsPiRing LED boards.

The left-hand set (labelled Pi5V) uses the 5V rail from the Raspberry Pi. For smallish numbers of LEDs (*e.g.* <50), the Pi can normally deliver enough current - as long as your power supply is a good one and you don't have power-hungry devices connected to USB.

The right-hand set is for use with an external 5V source. Connect your 5V supply positive lead to 'Ext 5V' and negative to GND. Do not connect your external PSU to 'Pi5V' on the wrong side or the two power supplies will fight each other (it could end badly *e.g.* fire). The two 'Ext 5V' holes are only connected to each other. They are not connected to anything else on the board.

C is the clock (SCLK) output after it has passed through the buffer chip and been level-shifted to 5V.

D is the data (MOSI) output after it has passed through the buffer chip and been level-shifted to 5V.

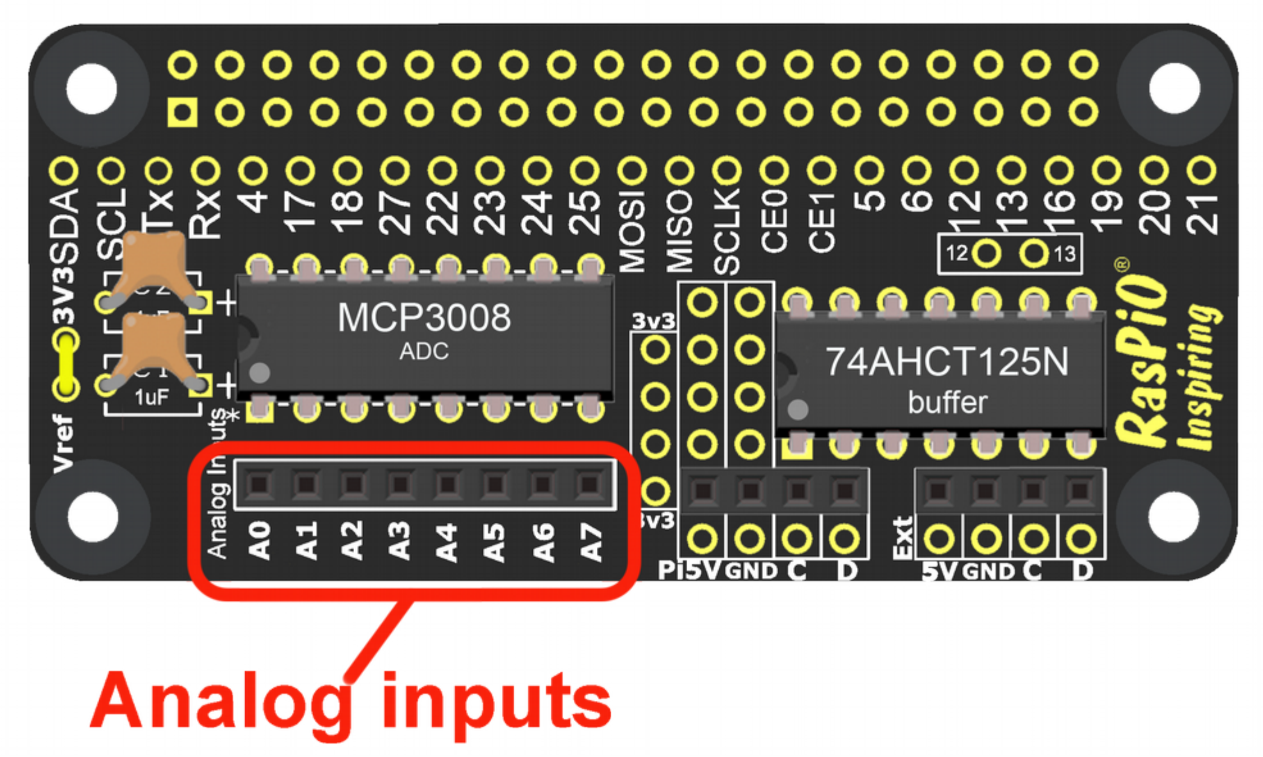
C & D are identical on both sides and are connected to each other. If you want to run two strips in parallel (both doing the same thing) you can do this as long as you provide power to 'Ext 5V'. It's not absolutely guaranteed to work, but it usually does.

Power and Ground Rails (Turquoise)

3V3, Pi5V and GND rails are provided. Once the female headers for the LED boards are soldered in place, there should be four available holes for each. The GND holes are a little tight up against the buffer chip socket, but careful assembly enables their full use.

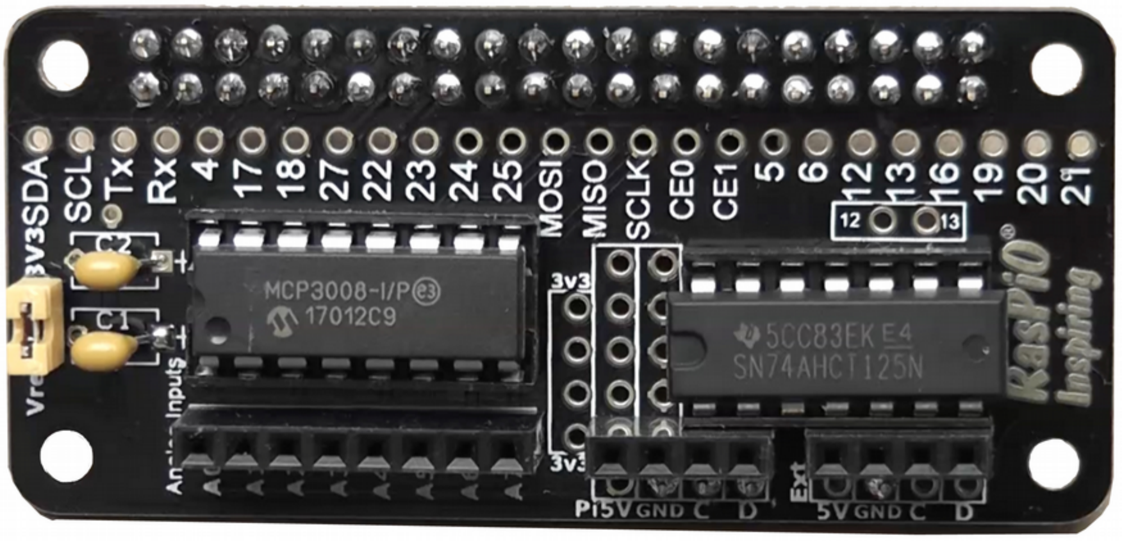
Optional Analog inputs (Red)

If you purchased the optional ADC upgrade parts you will have eight analog inputs shown in the red area below...

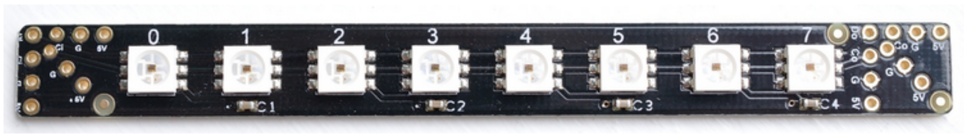


RasPiO InsPiRing driver pHAT with optional Analog (ADC) components

V_{ref} can be tied to 3V3 (default if jumper fitted) or anything less than 3V3 if you provide an alternative stable voltage reference. The MCP3008 is powered by 3V3 from the Pi, so cannot have V_{ref} higher than 3.3V

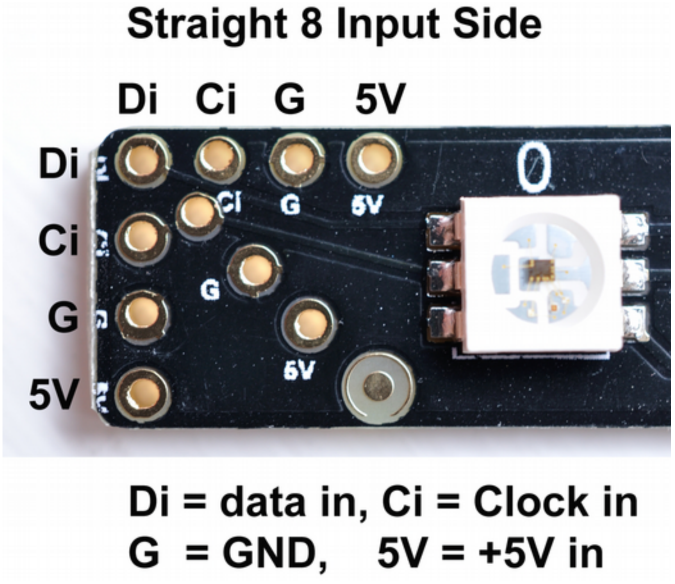


Straight-8

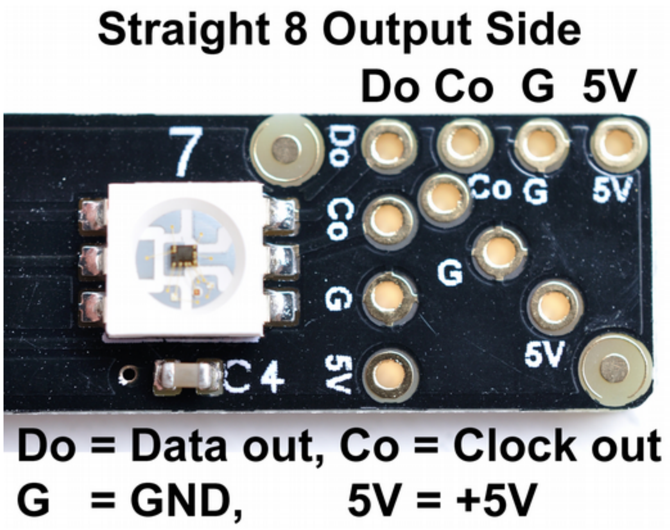


The Straight-8 has eight LEDs and three choices of inputs and outputs so it's possible to join them together at 180°, 90° or 45° directly. But you can join them at any other angle you like using wires.

The labels on the input side (next to LED 0) are a bit hard to read as the V-groove from manufacturing has partially obliterated them ☹. But it's easy when you know how. Essentially we have D for data, C for clock, G for Ground and 5V for +5V. Inputs are on the left and outputs on the right. You can plug the input side straight into the driver board or connect wires or headers to it. The choice is yours.

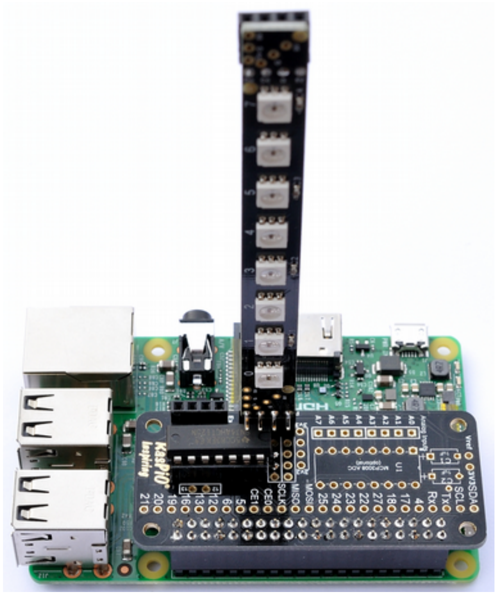


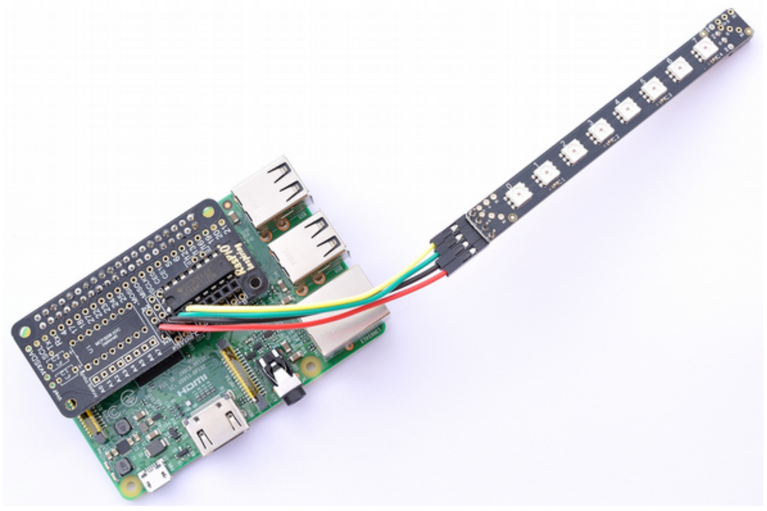
The output side (next to LED 7) is used for connecting to the input side of another RasPiO InsPiRing board to make a longer chain...

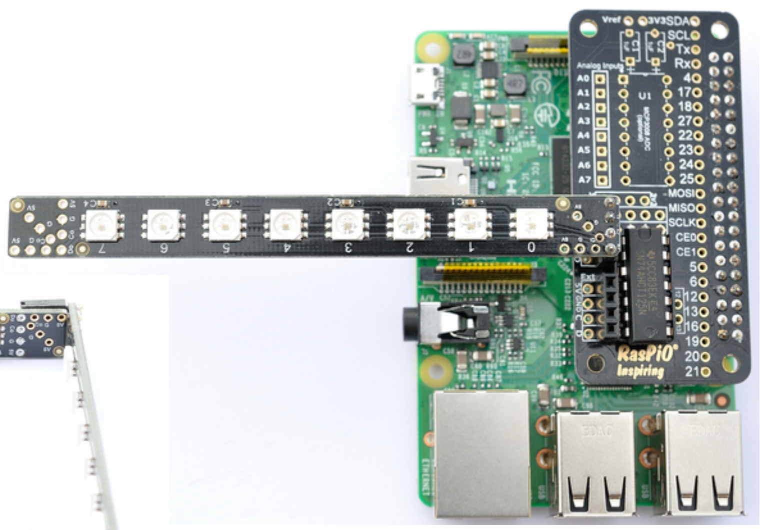


Some Connection Options/Ideas

Two Straight-8s chained together


Direct connection to driver


Connected to driver with jumper wires


Direct driver 90°

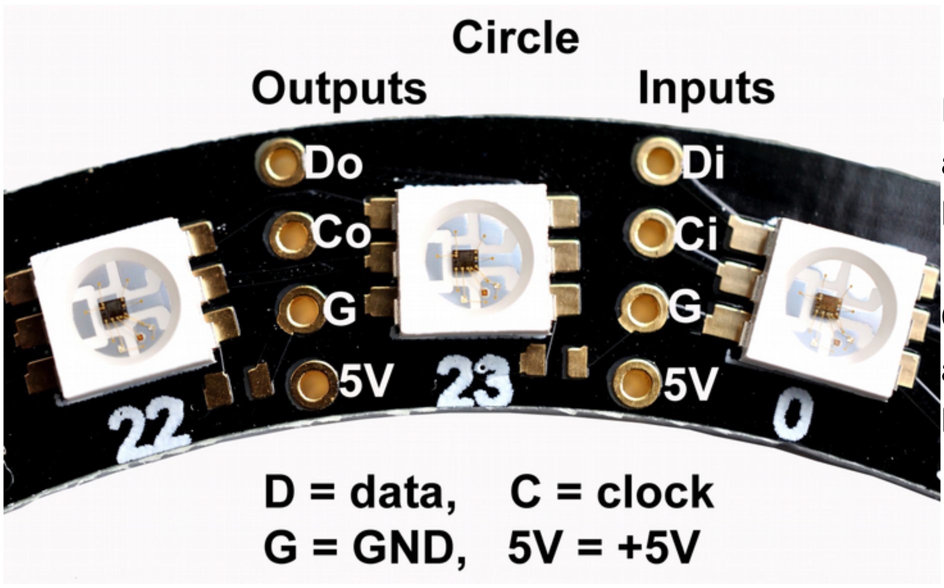
Chain of 3 at 90° to each other

Circle

The RasPiO InsPiRing circle has 24 LEDs. This makes it suitable for use as a clock.

Its circular nature makes it suitable for use as a 'ring light' or ring flash for lighting up small to medium-sized items in photography or videography.

It's also great for patterns of light and many other applications.



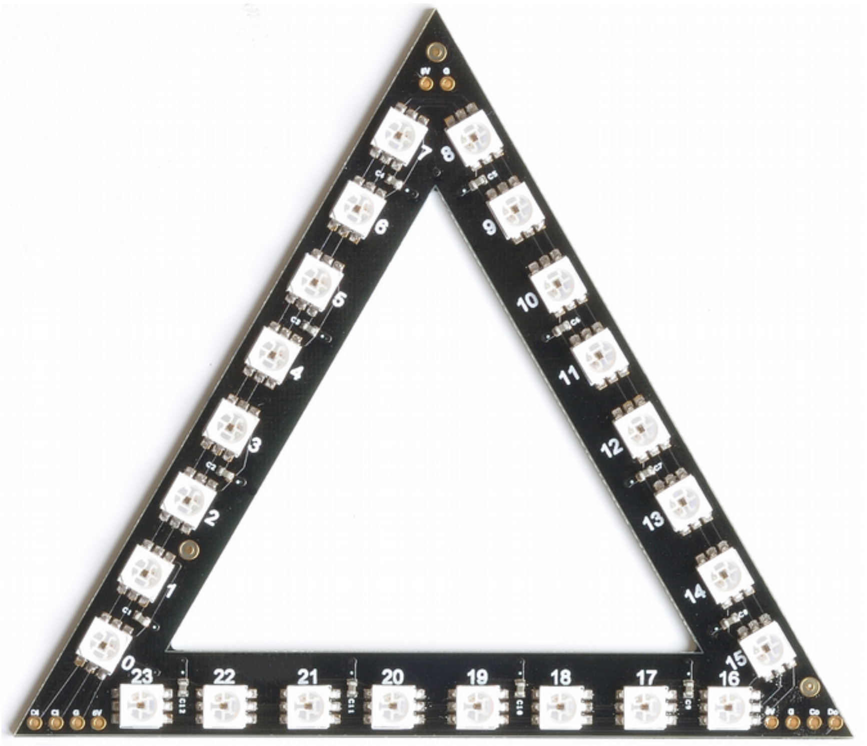
Input connections are to the right of LED 23.

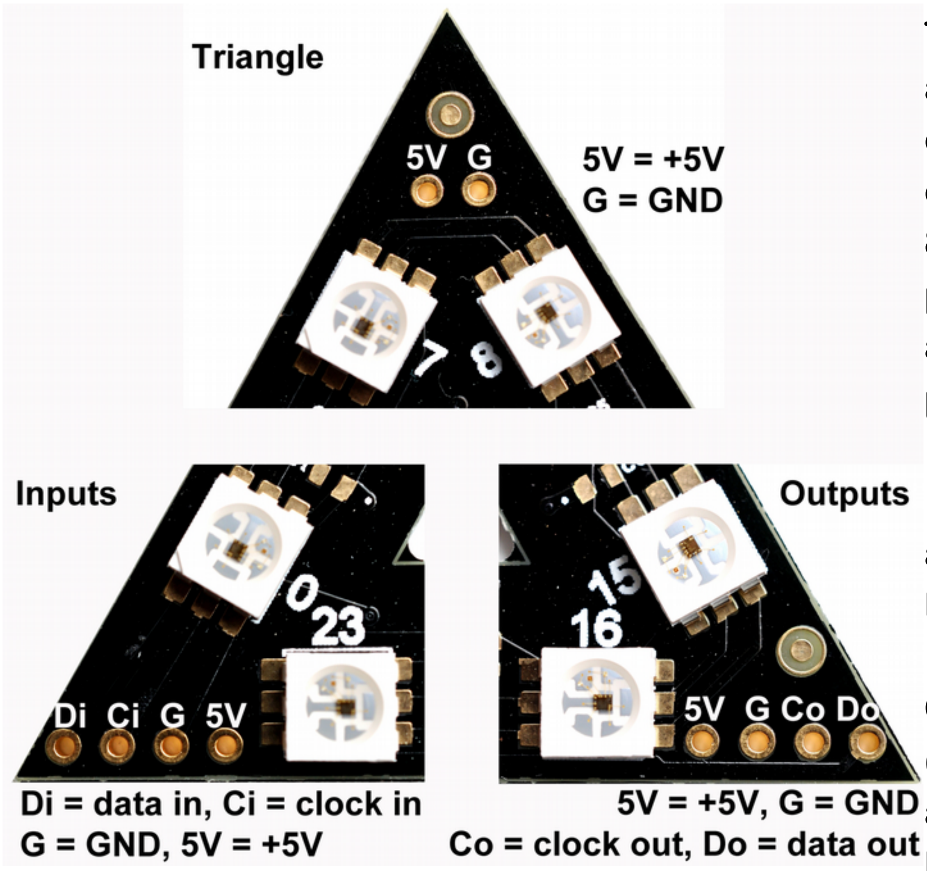
Output connections are to the left of LED 23.

Triangle

The RasPiO InsPiRing triangle also has 24 LEDs so could be used for a quirky-shaped clock or as three separate 8-segment indicators.

Combining three triangles makes a triangular pyramid shape. You could even combine four to make a square pyramid.





There are extra 5V and GND connections at the corner of LEDs 7 & 8. (Useful for joining pyramids and if additional external power is required.)

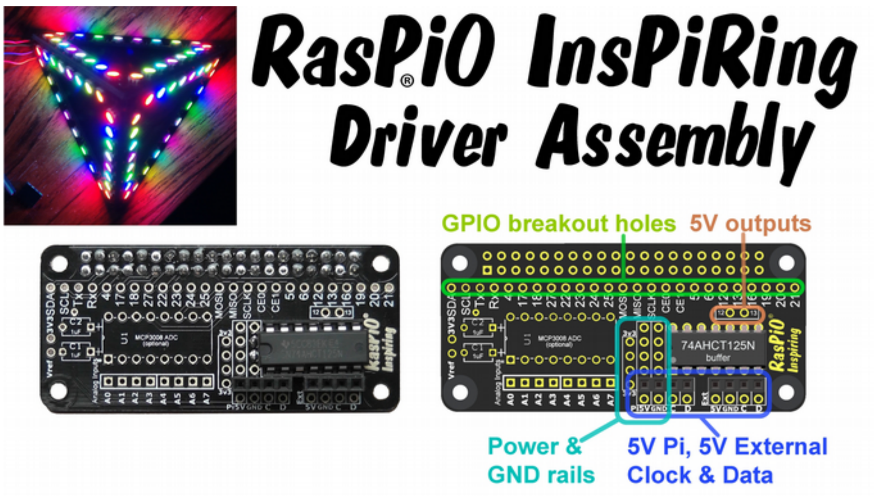
Input connections are at the corner of LEDs 0 & 23.

Output connections (order reversed) are at the corner of LEDs 15 & 16.

Assembly Instructions

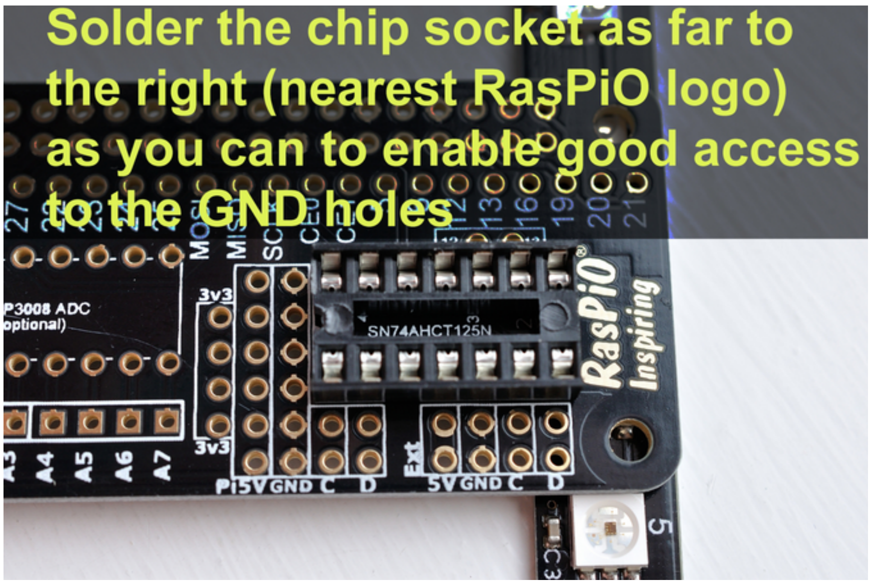
Driver Board Assembly

RasPiO InsPiRing driver board assembly is pretty straightforward. If you prefer written instructions, they start on this page. For everyone else, there's a complete step by step soldering video...



https://youtu.be/TyhNENlXlYk

The buffer chip and socket slightly overlap four GND pins. If you want full access to these, solder the chip socket as close as you can to the RasPiO logo. (A small blob of Blu-tack can be used to ensure the chip socket location is controlled.)



Solder the chip socket as far to the right (nearest RasPiO logo) as you can to enable good access to the GND holes

Suggested assembly order (basic board, no ADC extras)...

1. **14-pin chip socket for 74AHCT125N (as close to RasPiO logo as you can – use Blu-tack)**
2. ***optional if you want to fit them*** **2x 10k resistors joining both C and D to GND**
3. **2x 4-way female headers**
4. **40-way female header on underside of board**
5. **74AHCT125N chip**

Suggested assembly order (with ADC extras)...

1. **14-pin chip socket for 74AHCT125N (as close to RasPiO logo as you can – use Blu-tack)**
2. **16-pin chip socket for MCP3008**
3. ***optional if you want to fit them*** **2x 10k resistors joining both C and D to GND**
4. **2x 1μF capacitor**
5. **2x 4-way female headers**
6. **2-way male header between V_{ref} and 3v3 + jumper joining these pins**
7. **8-way female header**
8. **40-way female header on underside of board**
9. **74AHCT125N & MCP3008 chips**

Adding Headers to Straight-8, Triangle or Circle

Two headers are provided with each shape...

1. a 4-way right-angled male
2. a 4-way straight female

But that doesn't mean you have to use them. Use whatever suits your project. In some cases you may wish to solder wires directly to the through-holes as it may suit your project better. I've made a video showing how I used the supplied headers to connect straight-8 boards nicely. Have a look and then decide how you want to do it...

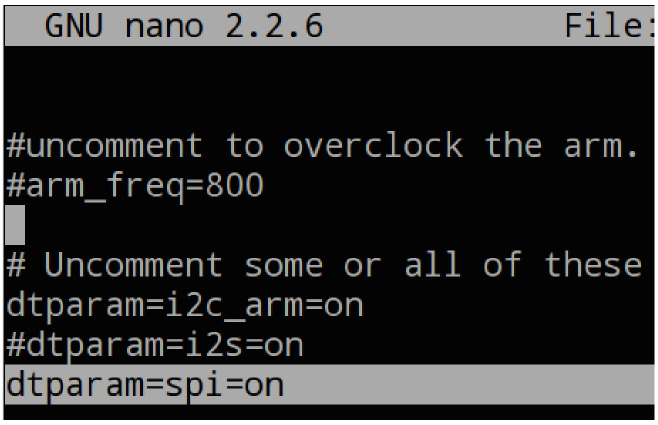
https://youtu.be/ySVLg9muw9c

If you are consistent, all your boards can fit together. I generally use a male header on the input side and a female header on the output side. It enables a board to be plugged straight into the driver board. The important thing is to think about what you want to do and choose the most suitable method.

Pi Configuration & Software Installation

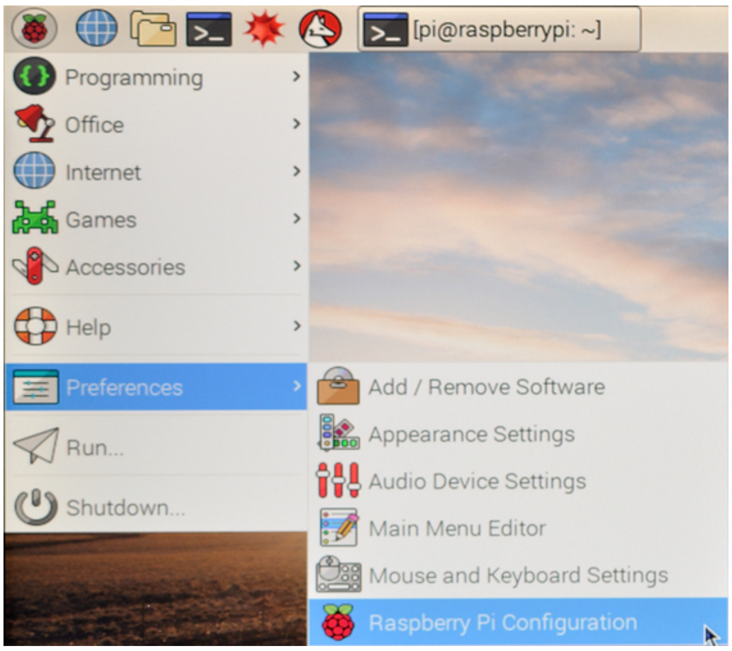
Enable SPI, then Reboot

There are a couple of ways to enable SPI on the Pi.

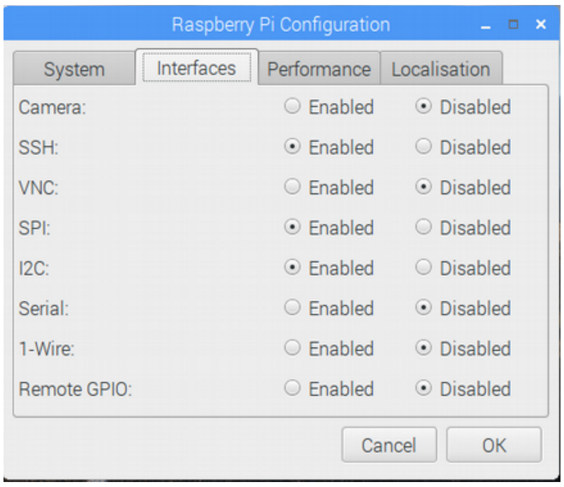


```
  GNU nano 2.2.6            File:

#uncomment to overclock the arm.
#arm_freq=800

# Uncomment some or all of these
dtparam=i2c_arm=on
#dtparam=i2s=on
dtparam=spi=on
```

If you're happy directly editing `/boot/config.txt` **you can ensure that** `dtparam=spi=on` **is not commented out with a #.**



If you prefer using the GUI, click the raspberry icon (top-left), then choose `Preferences > Raspberry Pi Configuration`



Camera:	Enabled	[✓ Disabled]
SSH:	[✓ Enabled]	Disabled
VNC:	Enabled	[✓ Disabled]
SPI:	[✓ Enabled]	Disabled
I2C:	[✓ Enabled]	Disabled
Serial:	Enabled	[✓ Disabled]
1-Wire:	Enabled	[✓ Disabled]
Remote GPIO:	Enabled	[✓ Disabled]

Then click the `interfaces` **tab,**
ensure that `SPI` **is set to** `'enabled'` **and click** `OK`.

Whichever way you choose, you will need to reboot the Pi either with `sudo reboot` **or from the** `shutdown` **menu.**

Install spidev for Python

So now we've enabled SPI, we need to be able to talk to it from Python. We use a package called python3-spidev which we might need to install (if you're not using a recent Raspbian). An internet connection is required for this. Here are the commands...

```
sudo apt-get update
sudo apt-get install python3-spidev python-spidev
```

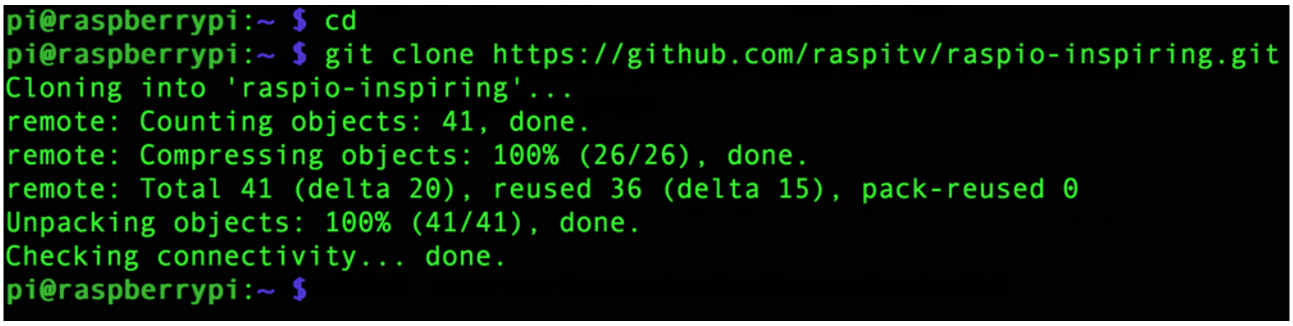
If you have a recent Raspbian, you may find this is already installed. If it is you'll get this message...

```
python-spidev is already the newest version.
python3-spidev is already the newest version.
0 upgraded, 0 newly installed, 0 to remove and 0 not
upgraded
```

Download Python Class & Examples

You should now be ready to grab the Python examples from GitHub. From a terminal window, enter...

```
cd
git clone https://github.com/raspitv/raspio-inspiring.git
```

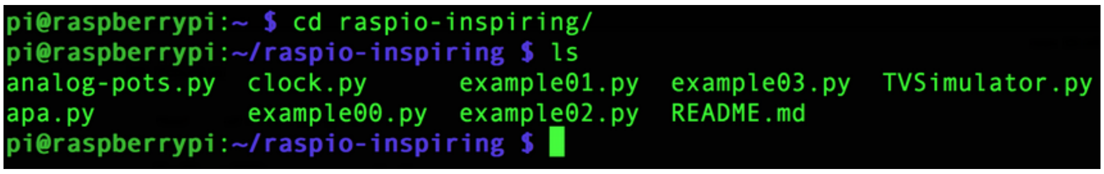


```
pi@raspberrypi:~ $ cd
pi@raspberrypi:~ $ git clone https://github.com/raspitv/raspio-inspiring.git
Cloning into 'raspio-inspiring'...
remote: Counting objects: 41, done.
remote: Compressing objects: 100% (26/26), done.
remote: Total 41 (delta 20), reused 36 (delta 15), pack-reused 0
Unpacking objects: 100% (41/41), done.
Checking connectivity... done.
pi@raspberrypi:~ $
```

The number of objects will increase over time as more projects are added.

This should create a directory called `raspio-inspiring` under `/home/pi` All of the examples can be found there. Go there now...

```
cd /home/pi/raspio-inspiring
ls
```

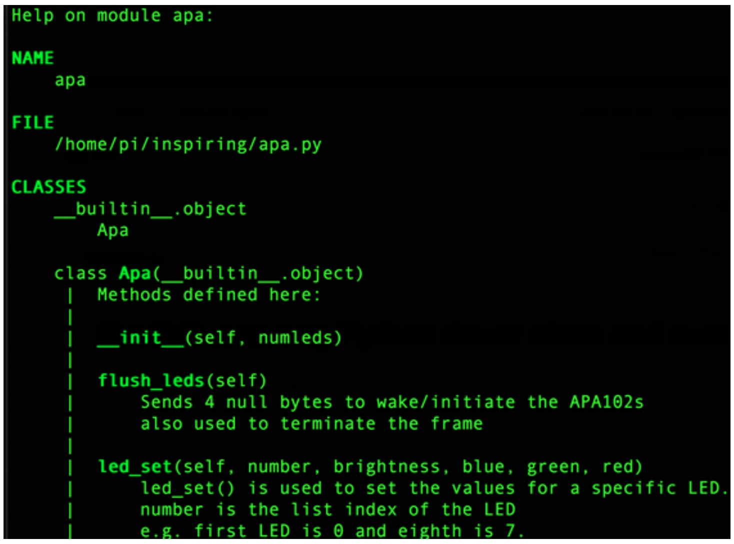


```
pi@raspberrypi:~ $ cd raspio-inspiring/
pi@raspberrypi:~/raspio-inspiring $ ls
analog-pots.py  clock.py       example01.py  example03.py  TVSimulator.py
apa.py          example00.py   example02.py  README.md
pi@raspberrypi:~/raspio-inspiring $
```

The number of scripts will increase over time as more projects are added.

Pydoc Documentation

Before we begin, if you ever forget how to use the `apa` class you can find some help in `pydoc`. From within the `/home/pi/raspio-inspiring` directory, Just type `pydoc apa` and you will see the on-screen help...



```
Help on module apa:

NAME
    apa

FILE
    /home/pi/inspiring/apa.py

CLASSES
    __builtin__.object
        Apa

    class Apa(__builtin__.object)
     |  Methods defined here:
     |
     |  __init__(self, numleds)
     |
     |  flush_leds(self)
     |      Sends 4 null bytes to wake/initiate the APA102s
     |      also used to terminate the frame
     |
     |  led_set(self, number, brightness, blue, green, red)
     |      led_set() is used to set the values for a specific LED.
     |      number is the list index of the LED
     |      e.g. first LED is 0 and eighth is 7.
```

There are several example scripts to illustrate how to use the apa Python class and some useful programming principles.

Python apa Class Explanation

The most important concept to understand is that the 'current' values for each LED are stored in a list variable (array) which itself contains a list variable `[brightness, blue, green, red]` for each LED. So it's a 'list of lists' (called `led_values`) looking something like this...

```
[[brightness, blue, green, red], [brightness, blue, green, red],
 [brightness, blue, green, red], ...]
```

If you want to animate the LEDs you have to think about 'frames' the same way you would in video. Each time we write to the LEDs with `ledstrip.write_leds()` all of the LEDs are written to with their current values. There is no way around this because they are chained together in series. You cannot just 'write' to one LED (except LED 0). But because SPI is so fast, it doesn't matter. You can easily change the values for one or more LEDs then write to them all, wait a bit, then write the next 'frame'. This all happens invisibly fast.

The important point is, nothing is actually sent (written) to the LEDs until you call `ledstrip.write_leds()` and then the entire set is written to. With 24 LEDs and Python, you can get about 100 frames per second (fps). So with a pyramid (72 LEDs) you should still comfortably get >30 fps.

Once you understand how it works, you realise that your main program needs to have a looping rhythm something like this...

1. Determine LED(s) values for next frame
2. Change stored LED(s) values for the next frame
3. Wait until ready
4. Write the LED values
5. Back to 1

Everything you do to decide what the next 'frame' should be comes between 1 & 2. Now let's look at some examples.

example00.py

This is the bare minimum code you need to do something. These 4 lines will set LED #3 (the 4th LED) to full brightness, full green. Then it will exit without switching off the LED.

```
import apa
ledstrip = apa.Apa(24)
ledstrip.led_set(3, 31, 0, 255, 0)
ledstrip.write_leds()
```

This is example00.py from GitHub but with the comments removed.

Line 1: brings in the apa class so we can use it.
Line 2: sets up an object `ledstrip` with 24 LEDs (all OFF initially).
Line 3: sets LED 3's values to full brightness and full green.
Line 4: writes the current values to all the LEDs.

example01.py

The following simple script shows the use of each available method...

```
import apa, time
numleds = 24
ledstrip = apa.Apa(numleds)
ledstrip.flush_leds()
brightness = 31
ledstrip.led_set(0, brightness, 0, 0, 255)
ledstrip.write_leds()
time.sleep(2)
ledstrip.zero_leds()
ledstrip.led_set(4, brightness, 255, 255, 255)
ledstrip.write_leds()
time.sleep(2)
ledstrip.reset_leds()
```

This is example01.py from GitHub but with the comments removed.

Your Python script needs to be in the same directory as `apa.py`, which will normally be `/home/pi/raspio-inspiring`

At the top of your script, import the apa class

```
  import apa
```

Set the number of LEDs in our display. The class needs this so it can set up a list variable `led_values` that will contain the brightness, blue, green and red values for each LED.

```
  numleds = 24
```

Initialise the LEDs (this creates an instance of the class – `ledstrip`)

```
  ledstrip = apa.Apa(numleds)
```

`numleds` can be an integer variable or a hard-coded integer.

Now we can manipulate our LEDs by using the object `ledstrip` that we just created.

You have the following methods you can play with to control your LEDs...

`flush_leds()`	'wakes up' the LEDs ready for action
`led_set()`	sets values of an LED's Brightness, Blue, Green, Red
`write_leds()`	sends the LED values to all the LEDs via SPI
`zero_leds()`	zeroes all stored `led_values` without writing to LEDs
`reset_leds()`	switches all LEDs off without changing stored values

In line 4 we `ledstrip.flush_leds()` This is not strictly necessary as the LEDs are flushed when `write_leds()` is called. But I wanted to show how to use the method. It flushes the LEDs by sending several bytes of zeros through SPI. This prepares them to receive data.

`brightness = 31` **sets the global brightness value to maximum.**

Next we use...
`ledstrip.led_set(number, brightness, blue, green, red)`
specifically
`ledstrip.led_set(0, brightness, 0, 0, 255)`
to set the values of LED number 0's brightness, Blue, Green, Red

`number` **is the LED position number starting at 0**

`brightness` **values are integers 0-31 (or 224-255 or 0xE0-0xFF).**
0 or 224 or 0xE0 = OFF
31 or 255 or 0xFF = FULL brightness.
This can be used to set the 'global' brightness of each RGB LED.
Any integer value within these ranges will work.

`blue, green` **and** *`red`* **values are integers 0-255 (or hex 0x00-0xFF)**

If you want "black" or "OFF", you can simply use brightness of 0, 224 or 0xE0, then the LED will be OFF regardless of what colour values you give it.

Line 7: `ledstrip.write_leds()` **initiates the LEDs, then writes to SPI the contents of** `led_values`**, namely 4 data bytes for each LED:** *`brightness, blue, green, red`*. **This sets the brightness and colours of each RGB LED.**

Then, after a 2 second sleep, all the values in list variable `led_values` **are zeroed without sending (writing) them to the LEDs**

```
time.sleep(2)
ledstrip.zero_leds()
```

Then we set LED 4 to be full brightness, white (all 3 colours FULL) and write the values of the LEDs to SPI...

```
ledstrip.led_set(4, brightness, 255, 255, 255)
ledstrip.write_leds()
```

After another 2 second wait,

```
time.sleep(2)
ledstrip.reset_leds()
```

switches all LEDs off manually without changing stored `led_values`**, and the program exits with all LEDs off.**

So we've seen simple code examples of how to control one LED at a time. Now let's do something a bit more interesting and control them all using loops to change their values quickly.

example02.py

```
from time import sleep
import apa
numleds = 24                # number of LEDs in our display
brightness = 11             # 0 = OFF, 31 = FULL
ledstrip = apa.Apa(numleds)

try:
    while True:
        for x in range(numleds):
            ledstrip.led_set(x, brightness, 0, 0, 255)  # red
        ledstrip.write_leds()
        sleep(1)
        for x in range(numleds):
            ledstrip.led_set(x, brightness, 0, 255, 0)  # green
        ledstrip.write_leds()
        sleep(1)
        for x in range(numleds):
            ledstrip.led_set(x, brightness, 255, 0, 0)  # blue
        ledstrip.write_leds()
        sleep(1)
finally:
    print("/nAll LEDs OFF - BYE!/n")
    ledstrip.zero_leds()
    ledstrip.write_leds()
```

This is example02.py from GitHub.

This whole main loop is wrapped in a `try: finally:` block which allows us to control what happens on program exit. No matter how the program exits, the LEDs will be switched off.

The rest of the time, the `while True:` loop will run forever.

The interesting code here is...

```
    for x in range(numleds):
        ledstrip.led_set(x, brightness, 0, 0, 255)  # red
    ledstrip.write_leds()
    sleep(1)
```

...in which we are using a loop to iterate through all of our LEDs and change their values in turn. Once we've changed them all, we write to them, pause for 1 second (s), then move on to the next section.

Our code cycles through red, green and blue, waiting 1s between each colour change.

So that's shown you how to use a loop to manipulate multiple LED values. Now let's do something a bit faster and more fun and show you a few more tricks.

In the next script (example03.py), we'll use a function to do most of the repetitive work. We're going to cycle up (lines 9-12) through all of our LEDs, switching them on in turn, waiting 0.04s between each one.

Once we've cycled up through switching them on, we'll cycle back down switching them off (lines 14-17).

Then we'll do it all again with a different colour. This will also give you an idea of how to mix the colours.

example03.py

```
from time import sleep
import apa
numleds = 24           # number of LEDs in our display
delay = 0.04           # seconds between frames
brightness = 6         # 0=OFF (224 or 0xE0), 31=FULL (255 or 0xFF)
ledstrip = apa.Apa(numleds)

def updown(b,g,r):
    for led in range(numleds):
        ledstrip.led_set(led, brightness, b, g, r)
        ledstrip.write_leds()
        sleep(delay)

    for led in range(numleds -1, -1, -1):
        ledstrip.led_set(led, brightness, 0, 0, 0)
        ledstrip.write_leds()
        sleep(delay)
try:
    while True:
        updown(255,0,0)        # Blue
        updown(0,255,0)        # Green
        updown(0,0,255)        # Red
        updown(255,150,0)      # Cyan
        updown(0,150,255)      # Yellow
        updown(150,0,255)      # Magenta
        updown(255,255,255)    # White
        #delay = delay * 0.9    # speeds up each iteration

finally:
    print("/nAll LEDs OFF - BYE!/n")
    ledstrip.reset_leds()
```

This is example03.py from GitHub. You now have pretty much all the tools and programming constructs that you need to make your own animations. All you need is your imagination and some experimentation.

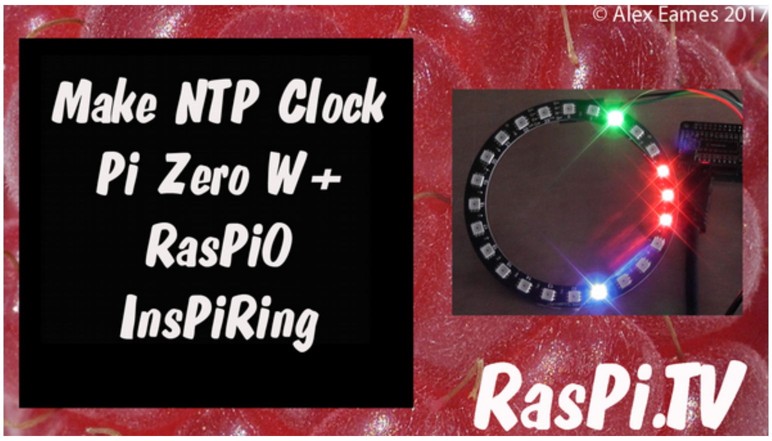
Programs in the GitHub Repo

There are plenty more examples in the GitHub repo, but the above should be enough to get you started on "doing your own thing". Here is a full list.

example00.py	Bare minimum code to make an LED do something (see example00.py)
example01.py	Demonstrates all the methods available in the apa class (see example01.py)
example02.py	Shows using loops to set all the LED values efficiently (see example02.py)
example03.py	Combines loops and a function to make multi-coloured 'wipes'. Also RGB colour mixing (see example03.py)
analog-pots.py	Control LED colours with potentiometers and an optional ADC (see Analog Potentiometer Control)
clock.py	NTP clock (see NTP Clock)
TVSimulator.py	Make it look (from outside at night) like someone is watching TV (see Make a TV Simulator)
apasnake.py	Snakes around a pyramid increasing the colour brightness around each face. Introduction to using a mathematical formula & loop to control the LEDs. Will also work with circle and triangle with a tweak (see Make a Pyramid)
updown-pyramid.py	Scrolling up and down controlling each 'level' of the LEDs on a 3-sided 72-LED pyramid. Rather an eye-catching visual effect (see Make a Pyramid)

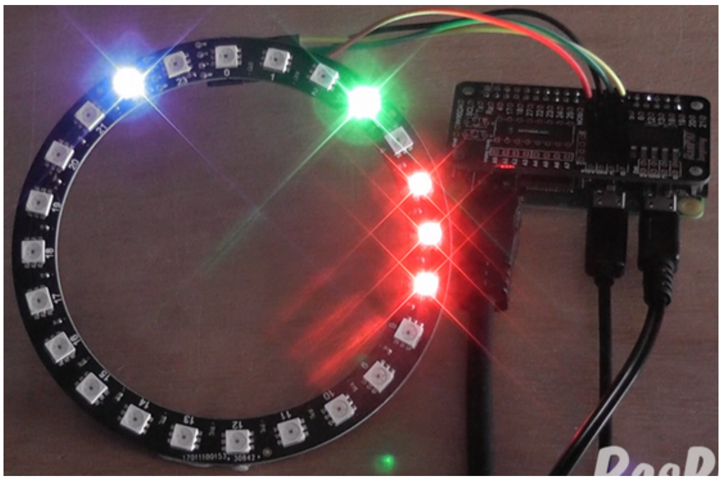
NTP Clock

You can use a RasPiO InsPiRing circle or triangle to make a clock. A triangle clock would be 'quirky' but could be lots of fun. Both shapes have 24 LEDs. If you want to see what the circular NTP clock looks like, I made a video of it during the KickStarter campaign...



https://youtu.be/pHPj0LVPgxE

It's quite an easy project. All you have to do is assemble your driver board (see Driver Board Assembly), then solder a header or wires to your RasPiO InsPiRing circle or triangle's inputs (see Triangle & Adding Headers to Straight-8, Triangle or Circle)



Then connect your driver board to the Raspberry Pi and your circle or triangle to the driver board. (Directly, with wires or jumper wires.)

Make sure you've configured your Pi and downloaded the software from github (see Pi Configuration & Software Installation and Download Python Class & Examples)

Check you have an internet connection. This is needed so you can grab the correct time (unless you have a realtime clock on your Pi).

Then run the script from a terminal window with...

```
  cd /home/pi/raspio-inspiring
  python3 clock.py
```

This clock script is a 12-hour version. It uses 2 LEDs for each hour (3 LEDs are shown for the 'hour' in the video, but I eventually decided I didn't like that, so changed it to 2). Minutes and seconds update every 2.5 (60/24=2.5). Since we don't have 60 LEDs we can't have one for each minute/second. (But you could possibly do something with blinking or colour fading if you felt like it.)

There's a walk-through of the code in this video here. The code has been tweaked slightly to improve it since then, but the ideas explained are still the same.

You could tweak the script clock.py to make it 24-hour or change the hourly, half-hourly and quarter-hourly animations. Either way, it's lots of fun to include in your projects. You could also make a nice case or mount to present it in.

Make a TV Simulator

What is a TV simulator? It's a device which can be used to simulate the lighting effect of watching TV. So when you're out, if someone is watching your house, you can make it look like you're at home.

Of course you can leave your TV on, but that wastes quite a bit of power. Also, if you build this with a Raspberry Pi connected to the internet, you can control it from anywhere in the world with your mobile phone. You could also program it to run at pre-set or random times.

When you're walking down the street at night you can often see the light flickering from people's TV sets. Depending on the show, the flickering effect can vary quite considerably, as can brightness level. So in my simple Python-based TV simulation script I included some random elements to give a mixture of short duration and long duration "scene changes". Those are, of course, completely tweakable.

The advantage of doing this on an internet connected Raspberry Pi is that you could switch it on and off remotely from anywhere in the world. How awesome is that? Here's a screenshot from a video clip. Obviously a still photo doesn't show the flickering, but unless you knew, or looked straight through the window, you could never tell it wasn't a TV.

This is a really easy project and it works with any of the RasPiO InsPiRing boards. I used a circle, but 8 LEDs on a straight-8 would also do just fine.

Just connect the driver board to your Pi, a RasPiO InsPiRing board to the driver (either directly or with wires) and run the code TVSimulator.py from the GitHub repo. If you ran through the standard installation procedures in this guide (see Pi Configuration & Software Installation) you will already have it at...

```
/home/pi/raspio-inspiring/TVSimulator.py
```

The code is very tweakable. You can tweak...

- interval times between 'scene changes' (lines 44 & 48)
- colour intensities (lines 51-53)
- maximum overall brightness (line 7)

You'll notice that the red (line 51) has been limited deliberately because the output didn't look 'realistic' enough with too much red in.

The possibilities here are great. You could expand the project by...

- using a light sensor to determine whether or not it's dark enough to bother lighting the LEDs
- switch it on or off via internet connection
- have different 'styles' of light patterns for different types of TV show
- switch it on at specific or random times of day/night (like a time-switch)
- add a PIR sensor or microphone trigger
- add sound output

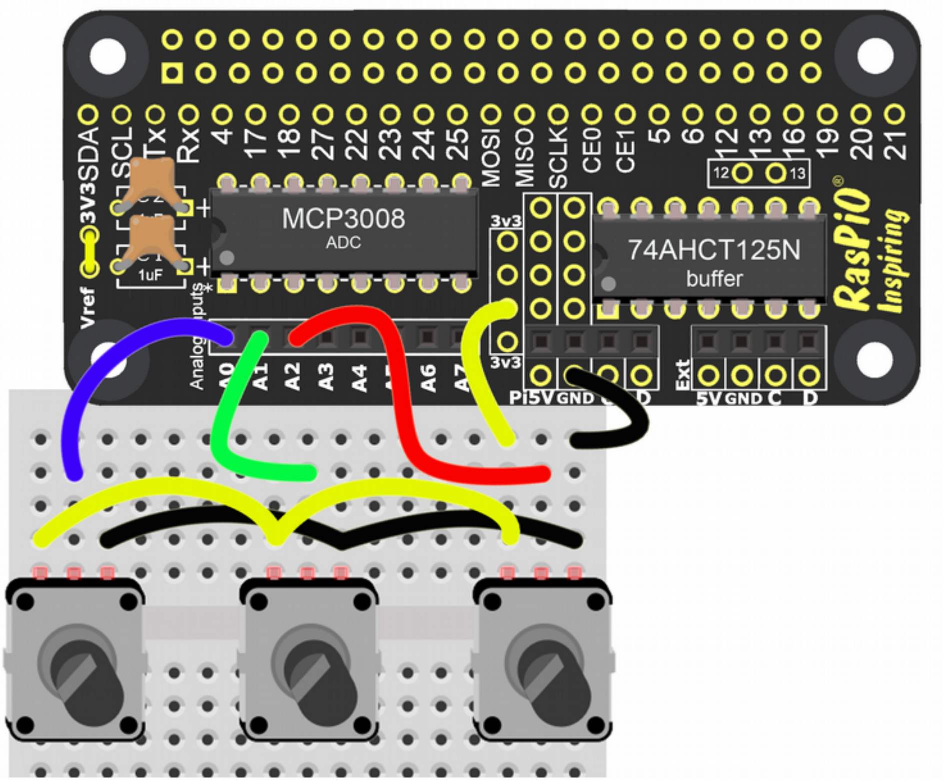
But whatever you do with it, have fun!

Analog Potentiometer Control

This project uses the optional MCP3008 components[4]. It enables you to control the brightness of each LED colour using potentiometers in the following circuit...

Circuit Diagram

Connect a 10k potentiometer so that the middle terminal goes to an ADC channel. The other two terminals go to GND and 3V3. Do this for channels 0-2 (connections A0, A1, A2).



Build the circuit, connect your RasPiO InsPiRing board, then run the script analog-pots.py

```
cd /home/pi/raspio-inspiring
python3 analog-pots.py
```

4 You will also need three 10k potentiometers, a breadboard/stripboard and some wires

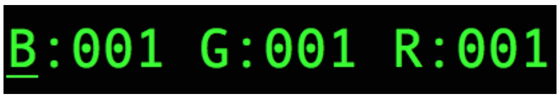
How it Works

The analog-pots.py script reads the input voltage on channels 0-2 of the MCP3008 and converts it to a number between 0 and 255. This is then used to control the LEDs.

Channel 0 = blue, 1 = green, 2 = red

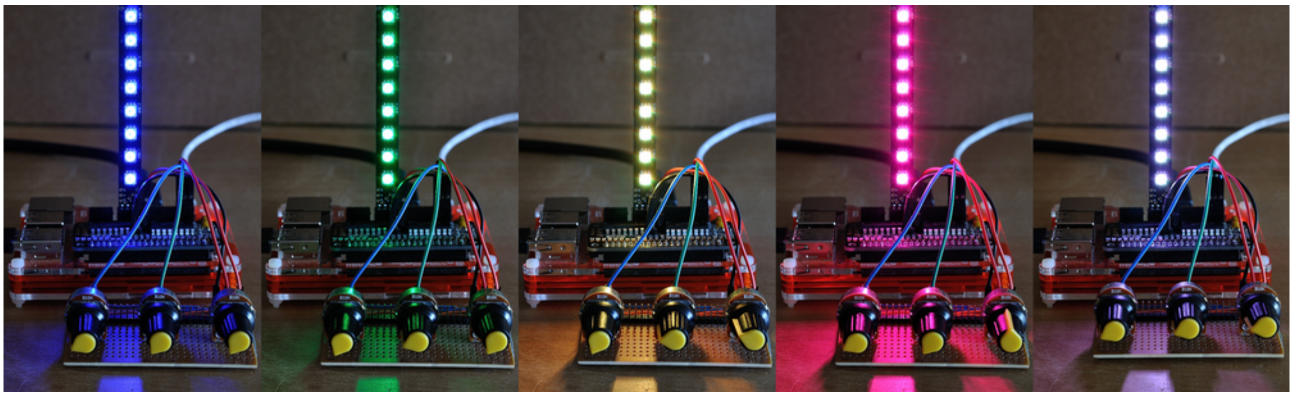
The LEDs are lit with the intensity determined by the analog inputs. You can mix any colour on the RGB light spectrum by twiddling the potentiometers (pots).

You should get realtime on-screen output, which will change – along with the LEDs' intensity as you twiddle the pots.



```
B:001 G:001 R:001
```

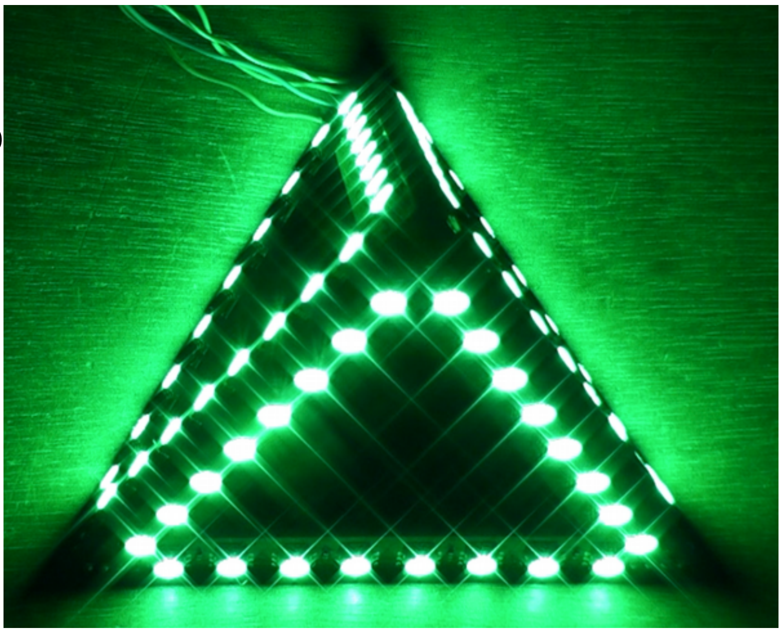
Check out the colours and the relative positions of the potentiometers in the photographs below. Bear in mind these were shot at fairly low brightness (it's really hard to photograph bright LEDs well).



Blue	Green	Yellow	Magenta	White
B:15 G:0 R:0	B:0 G:15 R:0	B:0 G:32 R:56	B:15 G:0 R:121	B:15 G:15 R:15

Have fun! And don't forget to tweak the script to make it your own. Happy colour mixing.

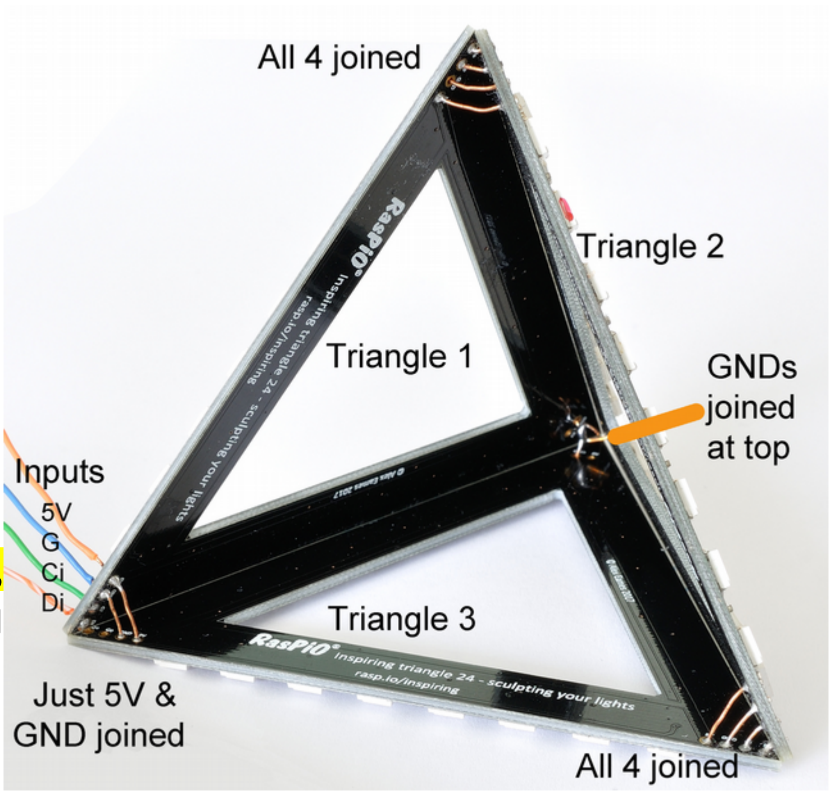
Make a Pyramid



One of my favourite RasPiO InsPiRing demonstrations is the pyramid. Made from three triangles joined together (I can't wait to make a square pyramid with four) it can be quite mesmerising to watch.

Pyramid Assembly

The pyramid comprises 3 RasPiO InsPiRing triangles 'daisy-chained' together. The inputs of triangle 1 connect to your Pi or micro-controller.



The outputs (Do & Co) of triangle 1 connect to the inputs (Di & Ci) of triangle 2 *(top of photo)*.

The outputs of triangle 2 connect to the inputs of triangle 3 *(bottom right)*.

The outputs (Do & Co) of triangle 3 are unconnected, *(bottom left)* but the GND and 5V of all three triangles are connected together.

It's also a good idea to connect together the spare GNDs at the top of each triangle. This helps structurally and electrically. You can ignore the top 5V holes.

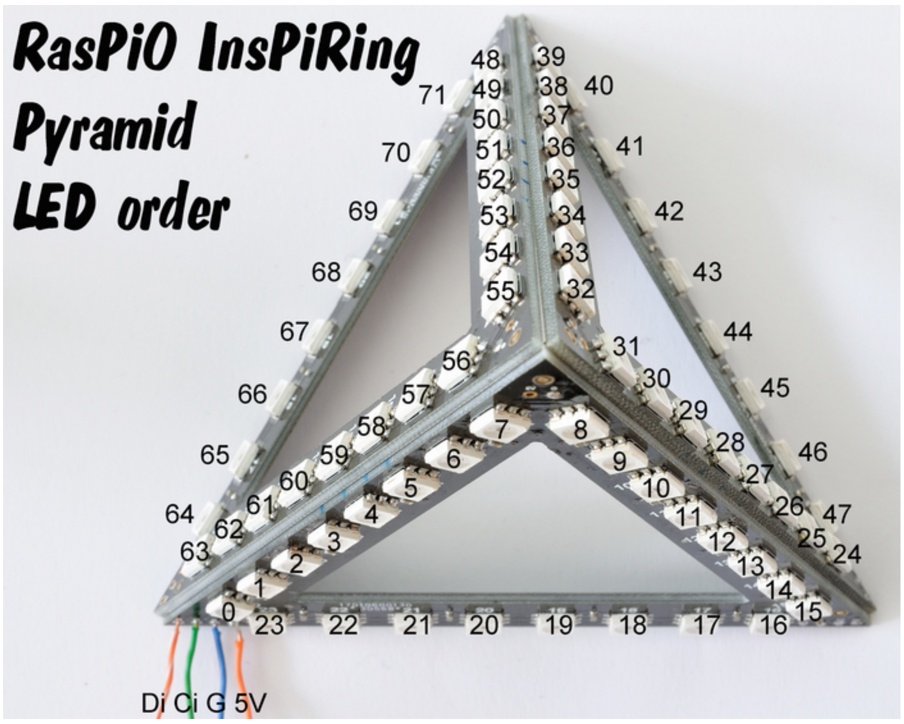
The above photo shows the connections, but if you want a full walk-through of the procedure, there is a YouTube video showing how to put the pyramid together...

RasPiO InsPiRing
Pyramid Assembly
All 4 joined
Triangle 2
Triangle 1
GNDs joined at top
Inputs
5V
G
Ci
Di
Triangle 3
Just 5V & GND joined
All 4 joined
Di Ci G 5V

https://youtu.be/Yt9tsg5DmtM

Pyramid Anatomy

The LEDs are numbered as follows for programming purposes. This is the path the signals follow around the pyramid...



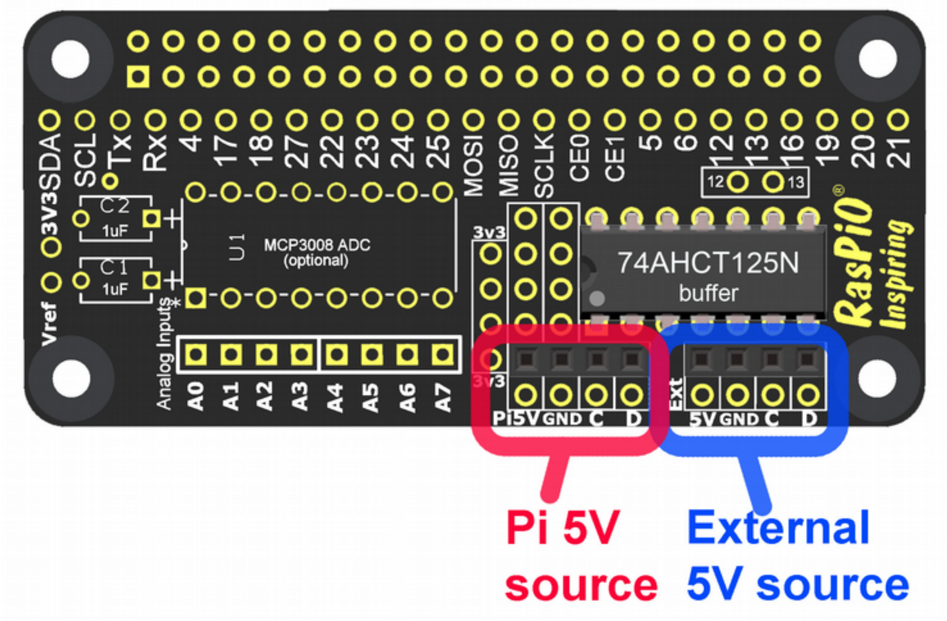
Connect your pyramid wires Di to driver board D, Ci to driver board C, G to driver board GND and 5V to driver board 5V. Use the appropriate set of connections for your choice of power supply (explained in the next section Pyramid Power).

Pyramid Power

If you are powering your pyramid from the Pi's 5V rail, you may well find that it cannot deliver enough current unless you tweak the brightness down below 20 (brightness can be 0-31). At full brightness and full Red/Green/Blue, the pyramid's 72 LEDs can draw more current than most Pi power supplies can deliver. See SK9822 Power Usage for more detailed information about current requirements.

External 5V DC PSU

Apart from reducing global LED brightness, another way round this is to use the second "Ext" set of connectors on the driver board, along with an external 5V power supply



There are two sets of LED connectors on the driver board. Those marked in red and labelled "Pi" feed power to the LEDs from the Pi's 5V rail.

Those marked in blue and labelled “Ext” feed power to the LEDs from an external 5V source. Your external 5V source must be connected to the spare Ext 5V through-hole and GND.

GND, C and D are identical on both sides (connected to each other)

You could also connect an external 5V supply directly to any of the 5V connections on the the pyramid (3 per triangle). But if you do that, make sure that you don't connect the LEDs' 5V to the Pi 5V rail or the PSUs will fight each other and it could potentially end badly (*e.g.* fire).

Pyramid Code

Here are a couple of demonstration scripts to get you started with the pyramid...

updown-pyramid.py

updown-pyramid.py cycles up and down the pyramid, changing colours for each iteration.

Things of note about the program are...

`levels` **(line 33) is a list of lists.**
Each element in this main list contains a list of LED numbers on the same level.
e.g. `levels[0]` **contains 24 elements because there are 3 strips of 8 LEDs on the bottom level.**
All the other levels have just 6 elements.

By grouping them in levels, we are able to control each level separately.

We do this by iterating through the levels and through the LEDs on each level using nested loops...

```
for level in levels:
    for led in level:
        ledstrip.led_values[led] = [brightness, b, g, r]
```

Then we manipulate `ledstrip.led_values` **directly as this makes it easy for us to directly over-write the values for the LEDs without using** `ledstrip.led_set()`

The rest of the program is very similar to example03.py
(I ripped code from `updown_pyramid.py` **when I wrote** `example03.py`**)**

apasnake.py snakes its way round the pyramid from LED 0 to LED 71 increasing the intensity by 10 for each successive LED on each triangle. Once it gets to the end, it repeats the process, cycling through a set sequence of colours.

You can do some very complex and flashy things with these LEDs. Once you've decided what you want to do, you have to devise some code to be able to do it.

apasnake.py starts to show you how you can use a mathematical formula, to loop through all the LEDs, making them change the way that you want.

What I wanted to do was vary the colour intensity of the LEDs on each triangle identically. The first LED should have intensity of 10 and each successive LED should increment 10 more.
***e.g.* LED0=10, LED5=60, LED23=240**

But I want the second triangle to be exactly the same.
So LED25 should be the same as LED0, LED47 same as LED23 etc.
This is what makes it slightly complex, but I figured it out.

The main 'guts' of this code is this bit here...

```
for y in range(numleds):                # (          B            ), G, R
    ledstrip.led_values[y] = [brightness, ((y+1)*10 - (y // 24)*240), 0, 0]
    ledstrip.write_leds()
    sleep(delay)
```

So we're looping through 0 to `numleds`, (0 to 72, stops at 72 so 71 is the last number processed) on a pyramid.
The loop iterator `y` is the LED we're currently working on.

In the first loop iteration, `y`=0.
This gives us...

```
((y + 1) * 10 - (y // 24) * 240)
(1 * 10 - (0 // 24) * 240)
(10 - 0) = 10
```

The `//` integer division makes this work the same for each triangle.
For triangle 1, LEDs 0-23, `(y // 24) * 240 = 0 * 240 = 0`
For triangle 2, LEDs 24-47, `(y // 24) * 240 = 1 * 240 = 240`
For triangle 3, LEDs 48-71, `(y // 24) * 240 = 2 * 240 = 480`

for LED5 we get...

```
((y + 1) * 10 - (y // 24) * 240)
(6 * 10 - (6 // 24) * 240)
(60 - 0) = 60
```

for LED47 (the last LED on triangle 2) it should give 240

```
((y + 1) * 10 - (y // 24) * 240)
(48 * 10 - (47 // 24) * 240)
(480 - (1 * 240) = 240
```

It can be quite difficult (if you're not a mathematician) to work out what your formula should be, but just take it one step at a time, testing as you go.

The more you do it, the better you get at it too. This is where all that nasty algebra we did in school (and never understood what it was useful for) shows its true worth.

Specifications

Board Dimensions

LEDs are SK9822 (click here for the datasheet) spaced 1cm apart.

RasPiO InsPiRing Product Name	Board Dimensions	Number of SK9822 LEDs
Straight-8	10 x 1cm	8
Triangle 24	Equilateral triangle of side 10cm. External to internal edge 1cm	24
Circle 24	Circle of external diameter 10cm, internal diameter 8cm	24

SK9822 Power Usage

Based on actual measurements using RasPiO InsPiRing driver board and 24 SK9822 LEDs, all powered from the 5V rail on a Raspberry Pi 3B with official 2.5A Pi power supply.

Single SK9822 LED	Current Consumption
At rest (224,0,0,0)	0.28 mA
At full RGB brightness (255,255,255,255)	51 mA
Full brightness, full Red (255,0,0,255)	17.7 mA
Full brightness, full Green (255,0,255,0)	17.7 mA
Full brightness, full Blue (255,255,0,0)	17.5 mA

Summing Full brightness R, G & B gives 52.9 mA but there may some slight current limiting from the Pi at that level (24 LEDs pulling >1.2 A). In the data sheet the LEDs are rated at 20 mA current limited for each colour. So the maximum you could possibly pull would be 60 mA per RGB LED. The SK9822 uses current limiting rather than PWM for global brightness.

So if you want maximum brightness white, budget for 60 mA per LED. For most applications, you can likely manage with about half of that as you don't usually have all the LEDs at full brightness at the same time.

For example, with a pyramid (72 LEDs) the theoretical maximum power draw is 72 x 0.06A = 4.32A

Powered from the 5V rail of a Raspberry Pi 3B, the updown-pyramid.py program starts 'misfiring' (with my setup and PSU) at global brightness levels >20-25. If you try to power too much from the Pi's 5V rail you can even reset the Pi.

But the good news is that full brightness is insanely bright. I run my pyramid on brightness setting 5 and it's enough for me, but up to 20 works well.

Heat Dissipation

The back of each board is essentially one large ground plane intended to help dissipate heat from the LEDs. Running constantly at maximum power, the back of the board can get warm (~45°C – which is unpleasantly warm to the touch, but will not likely cause burns).

It does mean that if you want to mount the boards in a project, hot-glue guns might not be adequate. The glue might soften or melt.

Control

SK9822 are controlled by SPI up to a maximum speed of 30 MHz. See SPI for further information.

Sections Yet to Add...

It is hoped to add the following sections with time...

Make an Arrow Project

Using RasPiO InsPiRing with Arduino & FastLED

Using RasPiO InsPiRing with Wemos D1 & FastLED

Using RasPiO InsPiRing with Micro:Bit

Further project ideas as they arise

Final Word

You should now have a thorough overview of how to use the RasPiO InsPiRing system.

I hope you have a lot of fun with it. There is always more to learn and further to go. As time goes by, I hope to add more to this guide with more ideas for projects.

You can check for the latest version at http://rasp.io/inspiring

And if you haven't yet got yourself a RasPiO InsPiRing kit, or need another, you can get that from here as well...

http://rasp.io/inspiring