Luma.LED$_{M}atrix Documentation$

Release 0.7.0

Richard Hull

Mar 04, 2017

Contents

CHAPTER 1

Introduction

Python library interfacing LED matrix displays with the MAX7219 driver (using SPI), WS2812 NeoPixels (using DMA) and TM1637 on the Raspberry Pi and other linux-based single board computers - it provides a Pillow-compatible drawing canvas, and other functionality to support:

- multiple cascaded devices
- LED matrix, seven-segment and NeoPixel variants
- scrolling/panning capability,
- terminal-style printing,
- state management,
- dithering to monochrome,
- Python 2.7 and 3.4+ are both supported

A LED matrix can be acquired for a few pounds from outlets like Banggood. Likewise 7-segment displays are available from Ali-Express or Ebay.

See also:

Further technical information for the specific devices can be found in the datasheets below:

- `MAX7219`
- `WS2812`
- `WS2812B`
- `TM1637`

CHAPTER 2

Installation

Note: The library has been tested against Python 2.7 and 3.4+.

For **Python3** installation, substitute the following in the instructions below.

- `pip pip3`,
- `python python3`,
- `python-dev python3-dev`,
- `python-pip python3-pip`.

Pre-requisites

MAX7219 Devices

By default, the SPI kernel driver is **NOT** enabled on the Raspberry Pi Raspian image. You can confirm whether it is enabled using the shell commands below:

```
$ lsmod | grep -i spi
spi_bcm2835             7424  0
```

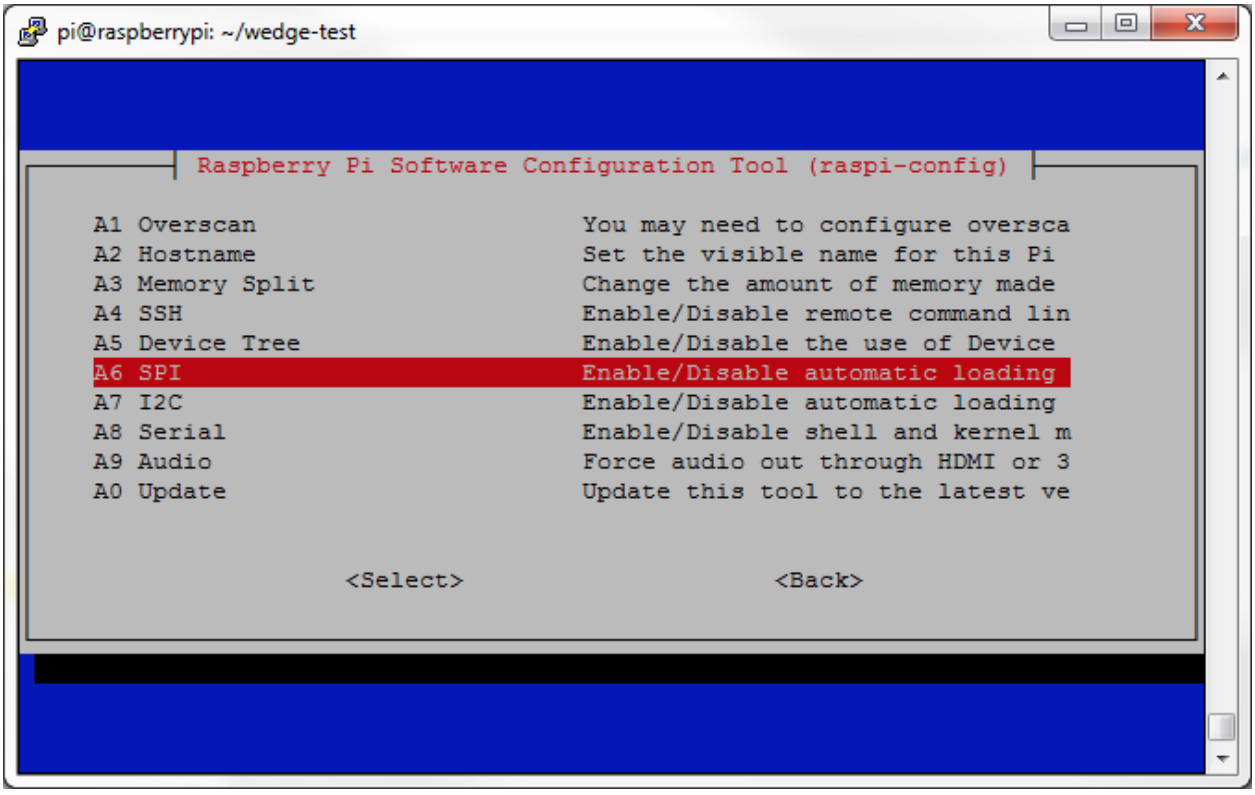
Depending on the hardware/kernel version, this may report **spi_bcm2807** rather than **spi_bcm2835** - either should be adequate.

And that the devices are successfully installed in `/dev`:

```
$ ls -l /dev/spi*
crw------- 1 root root 153, 0 Jan  1  1970 /dev/spidev0.0
crw------- 1 root root 153, 1 Jan  1  1970 /dev/spidev0.1
```

If you have no `/dev/spi` files and nothing is showing using `lsmod` then this implies the kernel SPI driver is not loaded. Enable the SPI as follows (steps taken from https://learn.sparkfun.com/tutorials/raspberry-pi-spi-and-i2c-tutorial#spi-on-pi):

1. Run `sudo raspi-config`
2. Use the down arrow to select `9 Advanced Options`
3. Arrow down to `A6 SPI`.
4. Select **yes** when it asks you to enable SPI,
5. Also select **yes** when it asks about automatically loading the kernel module.
6. Use the right arrow to select the **<Finish>** button.
7. Select **yes** when it asks to reboot.



After rebooting re-check that the `lsmod | grep -i spi` command shows whether SPI driver is loaded before proceeding. If you are stil experiencing problems, refer to the official Raspberry Pi SPI troubleshooting guide for further details, or ask a new question - but please remember to add as much detail as possible.

GPIO pin-outs

MAX7219 Devices (SPI)

The breakout board has two headers to allow daisy-chaining:

Board Pin	Name	Remarks	RPi Pin	RPi Function
1	VCC	+5V Power	2	5V0
2	GND	Ground	6	GND
3	DIN	Data In	19	GPIO 10 (MOSI)
4	CS	Chip Select	24	GPIO 8 (SPI CE0)
5	CLK	Clock	23	GPIO 11 (SPI CLK)

See also:

See notes section for cascading/daisy-chaining, power supply and level-shifting.

WS2812 NeoPixels (DMA)

Typically, WS2812 NeoPixels reqire VCC, VSS (GND) and DI pins connecting to the Raspberry Pi, where the DI pin is usually connected to a PWM control pin such as GPIO 18.

Board Pin	Name	Remarks	RPi Pin	RPi Function
1	DO	Data Out	•	•
2	DI	Data In	12	GPIO 18 (PWM0)
3	VCC	+5V Power	2	5V0
4	NC	Not connected	•	•
5	VDD	Not connected	•	•
6	VSS	Ground	6	GND

The DO pin should be connected to the DI pin on the next (daisy-chained) neopixel, while the VCC and VSS are supplied in-parallel to all LED's. WS2812b devices now are becoming more prevalent, and only have 4 pins.

TM1637

> TODO

Installing from PyPi

Install the dependencies for library first with:

```
$ sudo usermod -a -G spi,gpio pi
$ sudo apt-get install python-dev python-pip libfreetype6-dev libjpeg-dev
$ sudo -i pip install --upgrade pip
$ sudo apt-get purge python-pip
```

Warning: The default pip bundled with apt on Raspbian is really old, and can cause components to not be installed properly. Please ensure that **pip 9.0.1** is installed prior to continuing:

```
$ pip --version
pip 9.0.1 from /usr/local/lib/python2.7/dist-packages (python 2.7)
```

Proceed to install latest version of the library directly from PyPI:

```
$ sudo -H pip install --upgrade luma.led_matrix
```

Examples

Ensure you have followed the installation instructions above. Clone the repo from github, and run the example code as follows:

```
$ python examples/matrix_demo.py
```

The matrix demo accepts optional flags to configure the number of cascaded devices and correct the block orientation phase shift when using 4x8x8 matrices:

```
$ python examples/matrix_demo.py -h
usage: matrix_demo.py [-h] [--cascaded CASCADED]
                      [--block-orientation {horizontal,vertical}]

matrix_demo arguments

optional arguments:
-h, --help            show this help message and exit
--cascaded CASCADED, -n CASCADED
                      Number of cascaded MAX7219 LED matrices (default: 1)
--block-orientation {horizontal,vertical}
                      Corrects block orientation when wired vertically
                      (default: horizontal)
```

Similarly, there is a basic demo of the capabilities of the *luma.led_matrix.virtual.sevensegment* wrapper:

```
$ python examples/sevensegment_demo.py
```

and for the *luma.led_matrix.device.neopixel* device:

```
$ sudo python examples/neopixel_demo.py
```

Further examples are available in the luma.examples. git repository. Follow the instructions in the README for more details.

A small example application using ZeroSeg to display TOTP secrets can be found in https://github.com/rm-hull/zaup.

CHAPTER 3

Python Usage

8x8 LED Matrices

For the matrix device, initialize the *luma.led_matrix.device.max7219* class, as follows:

```
from luma.core.serial import spi, noop
from luma.core.render import canvas
from luma.led_matrix.device import max7219

serial = spi(port=0, device=0, gpio=noop())
device = max7219(serial)
```

The display device should now be configured for use. The specific *max7219* class exposes a *display()* method which takes an image with attributes consistent with the capabilities of the configured device's capabilities. However, for most cases, for drawing text and graphics primitives, the canvas class should be used as follows:

```
font = ImageFont.truetype("examples/pixelmix.ttf", 8)

with canvas(device) as draw:
    draw.rectangle(device.bounding_box, outline="white", fill="black")
```

The `luma.core.render.canvas` class automatically creates an `PIL.ImageDraw` object of the correct dimensions and bit depth suitable for the device, so you may then call the usual Pillow methods to draw onto the canvas.

As soon as the with scope is ended, the resultant image is automatically flushed to the device's display memory and the `PIL.ImageDraw` object is garbage collected.

Note: The default Pillow font is too big for 8px high devices like the LED matrices here, so the luma.examples repo inclues a small TTF pixel font called **pixelmix.ttf** (attribution: http://www.dafont.com/) which just fits.

Alternatively, a set of "legacy" fixed-width bitmap fonts are included in the luma.core codebase and may be used as follows:

```
from luma.core.legacy import text
from luma.core.legacy.font import proportional, CP437_FONT, LCD_FONT

with canvas(device) as draw:
   text(draw, text="A", fill="white", font=proportional(CP437_FONT))
```

The fixed-width fonts can be "converted" on-the-fly to proportionally spaced by wrapping them with the `luma.core.legacy.font.proportional` class.

Scrolling / Virtual viewports

A single 8x8 LED matrix clearly hasn't got a lot of area for displaying useful information. Obviously they can be daisy-chained together to provide a longer line of text, but as this library extends luma.core, then we can use the `luma.core.virtual.viewport` class to allow scrolling support:

```
import time

from luma.core.serial import spi, noop
from luma.core.render import canvas
from luma.core.virtual import viewport
from luma.led_matrix.device import max7219

serial = spi(port=0, device=0, gpio=noop())
device = max7219(serial)

virtual = viewport(device, width=200, height=100)

with canvas(virtual) as draw:
    draw.rectangle(device.bounding_box, outline="white", fill="black")
    draw.text((3, 3), text="Hello world", fill="white")

for offset in range(8):
    virtual.set_position((offset, offset))
    time.sleep(0.1)
```

Calling `set_position()` on a virtual viewport, causes the device to render what is visible at that specific position; altering the position in a loop refreshes every time it is called, and gives an animated scrolling effect.

By altering both the X and Y co-ordinates allows scrolling in any direction, not just horizontally.

Color Model

Any of the standard `PIL.ImageColor` color formats may be used, but since the 8x8 LED Matrices are monochrome, only the HTML color names `"black"` and `"white"` values should really be used; in fact, by default, any value *other* than black is treated as white. The `luma.core.render.canvas` constructor does have a `dither` flag which if set to `True`, will convert color drawings to a dithered monochrome effect.

```
with canvas(device, dither=True) as draw:
    draw.rectangle(device.bounding_box, outline="white", fill="red")
```

Landscape / Portrait Orientation

By default, cascaded matrices will be oriented in landscape mode. Should you have an application that requires the display to be mounted in a portrait aspect, then add a `rotate=N` parameter when creating the device:

```
from luma.core.serial import i2c
from luma.core.render import canvas
from luma.oled.device import ssd1306, ssd1325, ssd1331, sh1106

serial = i2c(port=1, address=0x3C)
device = ssd1306(serial, rotate=1)

# Box and text rendered in portrait mode
with canvas(device) as draw:
    draw.rectangle(device.bounding_box, outline="white", fill="black")
```

N should be a value of 0, 1, 2 or 3 only, where 0 is no rotation, 1 is rotate 90° clockwise, 2 is 180° rotation and 3 represents 270° rotation.

The `device.size`, `device.width` and `device.height` properties reflect the rotated dimensions rather than the physical dimensions.

Daisy-chaining

The MAX7219 chipset supports a serial 16-bit register/data buffer which is clocked in on pin DIN every time the clock edge falls, and clocked out on DOUT 16.5 clock cycles later. This allows multiple devices to be chained together.

If you have more than one device and they are daisy-chained together, you can initialize the library in one of two ways, either using `cascaded=N` to indicate the number of daisychained devices:

```
from luma.core.serial import spi, noop
from luma.core.render import canvas
from luma.led_matrix.device import max7219

serial = spi(port=0, device=0, gpio=noop())
device = max7219(serial, cascaded=3)

with canvas(device) as draw:
   draw.rectangle(device.bounding_box, outline="white", fill="black")
```

Using `cascaded=N` implies there are N devices arranged linearly and horizontally, running left to right.

Alternatively, the device configuration may configured with `width=W` and `height=H`. These dimensions denote the number of LEDs in the all the daisychained devices. The width and height *must* both be multiples of 8: this has scope for arranging in blocks in, say 3x3 or 5x2 matrices (24x24 or 40x16 pixels, respectively).

Given 12 daisychained MAX7219's arranged in a 4x3 layout, the simple example below,

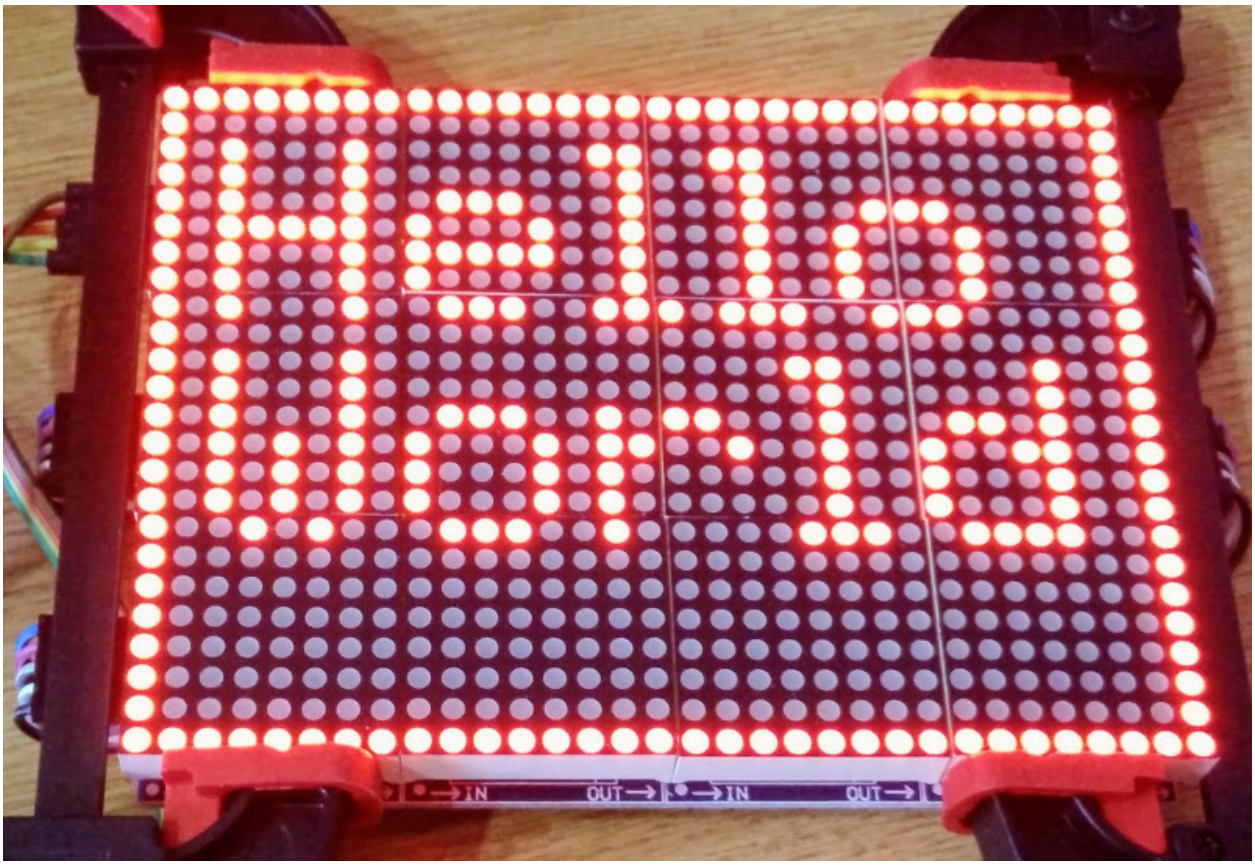
```
from luma.core.serial import spi, noop
from luma.core.render import canvas
from luma.led_matrix.device import max7219
from luma.core.legacy.font import proportional, LCD_FONT

serial = spi(port=0, device=0, gpio=noop(), block_orientation="vertical")
device = max7219(serial, width=32, height=24)

with canvas(device) as draw:
   draw.rectangle(device.bounding_box, outline="white")
```

```
    text(draw, (2, 2), "Hello", fill="white", font=proportional(LCD_FONT))
    text(draw, (2, 10), "World", fill="white", font=proportional(LCD_FONT))
```

displays as:



Trouble-shooting / common problems

Some online retailers are selling pre-assembled '4-in-1' LED matrix displays, but they appear to be wired 90° out-of-phase such that horizontal scrolling appears as below:

This can be rectified by initializing the *max7219* device with a parameter of `block_orientation="vertical"`:

```
from luma.core.serial import spi, noop
from luma.core.render import canvas
from luma.led_matrix.device import max7219

serial = spi(port=0, device=0, gpio=noop())
device = max7219(serial, cascaded=4, block_orientation="vertical")
```

Every time a display render is subsequenly requested, the underlying image representation is corrected to reverse the 90° phase shift.

7-Segment LED Displays

For the 7-segment device, initialize the *luma.led_matrix.virtual.sevensegment* class, and wrap it around a previously created *max7219* device:

```
from luma.core.serial import spi, noop
from luma.core.render import canvas
from luma.led_matrix.device import max7219
from luma.led_matrix.virtual import sevensegment

serial = spi(port=0, device=0, gpio=noop())
device = max7219(serial, cascaded=2)
seg = sevensegment(device)
```

The **seg** instance now has a *text* property which may be assigned, and when it does will update all digits according to the limited alphabet the 7-segment displays support. For example, assuming there are 2 cascaded modules, we have 16 character available, and so can write:

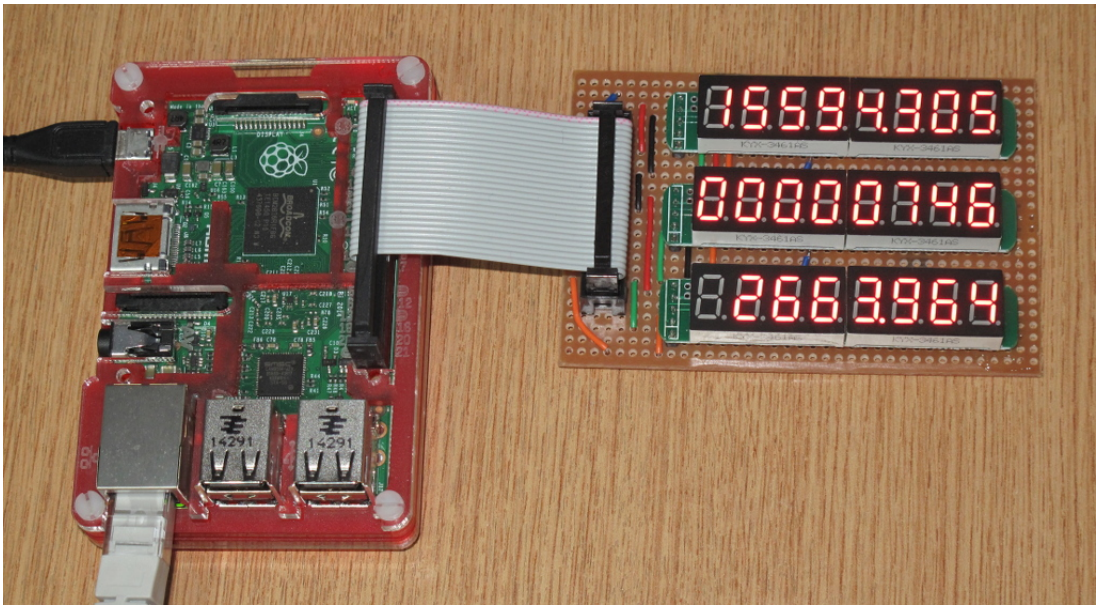
```
seg.text = "Hello world"
```

Rather than updating the whole display buffer, it is possible to update 'slices', as per the below example:

```
seg.text[0:5] = "Goodbye"
```

This replaces `Hello` in the previous example, replacing it with `Gooobye`. The usual python idioms for slicing (inserting / replacing / deleteing) can be used here, but note if inserted text exceeds the underlying buffer size, a `ValueError` is raised.

Floating point numbers (or text with '.') are handled slightly differently - the decimal-place is fused in place on the character immediately preceding it. This means that it is technically possible to get more characters displayed than the buffer allows, but only because dots are folded into their host character.



WS2812 NeoPixels

For a strip of neopixels, initialize the *luma.led_matrix.device.neopixel* class, supplying a parameter `cascaded=N` where *N* is the number of daisy-chained LEDs. This creates a drawing surface 100 pixels long, and lights up three specific pixels, and a contiguous block:

```
from luma.core.render import canvas
from luma.led_matrix.device import neopixel

device = neopixel(cascaded=100)

with canvas(device) as draw:
    draw.point((0,0), fill="white")
    draw.point((4,0), fill="blue")
    draw.point((11,0), fill="orange")
    draw.rectange((20, 0, 40, 0), fill="red")
```

If you have a device like Pimoroni's Unicorn pHat, initialize the device with `width=N` and `height=N` attributes instead:

```
from luma.core.render import canvas
from luma.led_matrix.device import neopixel

# Pimoroni's Unicorn pHat is 8x4 neopixels
device = neopixel(width=8, height=4)

with canvas(device) as draw:
    draw.line((0, 0, 0, device.height), fill="red")
    draw.line((1, 0, 1, device.height), fill="orange")
    draw.line((2, 0, 2, device.height), fill="yellow")
    draw.line((3, 0, 3, device.height), fill="green")
    draw.line((4, 0, 4, device.height), fill="blue")
    draw.line((5, 0, 5, device.height), fill="indigo")
    draw.line((6, 0, 6, device.height), fill="violet")
    draw.line((7, 0, 7, device.height), fill="white")
```

Note: The neopixel driver uses the ws2812 PyPi package to interface to the daisychained LEDs. It uses DMA (direct memory access) via `/dev/mem` which means that it has to run in privileged mode (via `sudo` root access).

The same viewport, scroll support, portrait/landscape orientation and color model idioms provided in luma.core are equally applicable to the neopixel implementation.

Pimoroni Unicorn HAT

Pimoroni sells the Unicorn HAT, comprising 64 WS2812b NeoPixels in an 8x8 arrangement. The pixels are cascaded, but arranged in a 'snake' layout, rather than a 'scan' layout. In order to accomodate this, a translation mapping is required, as follows:

```
import time

from luma.led_matrix.device import neopixel, UNICORN_HAT
from luma.core.render import canvas

device = neopixel(width=8, height=8, mapping=UNICORN_HAT)
```

```
for y in range(device.height):
    for x in range(device.width):
        with canvas(device) as draw:
            draw.point((x, y), fill="green")
        time.sleep(0.5)
```

This should animate a green dot moving left-to-right down each line.

Emulators

There are various display emulators available for running code against, for debugging and screen capture functionality:

- The `luma.emulator.device.capture` device will persist a numbered PNG file to disk every time its `display()` method is called.
- The `luma.emulator.device.gifanim` device will record every image when its `display()` method is called, and on program exit (or Ctrl-C), will assemble the images into an animated GIF.
- The `luma.emulator.device.pygame` device uses the `pygame` library to render the displayed image to a pygame display surface.

Invoke the demos with:

```
$ python examples/clock.py -d capture --transform=led_matrix
```

or:

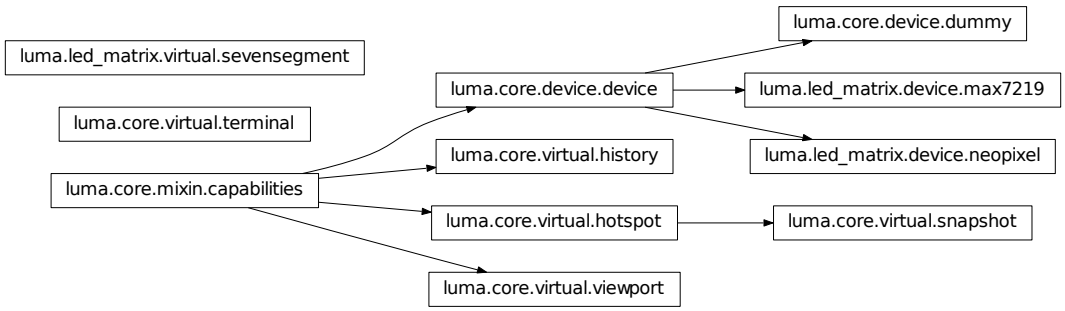
```
$ python examples/clock.py -d pygame --transform=led_matrix
```

Note: *Pygame* is required to use any of the emulated devices, but it is **NOT** installed as a dependency by default, and so must be manually installed before using any of these emulation devices (e.g. `pip install pygame`). See the install instructions in luma.emulator for further details.

CHAPTER 4

API Documentation

LED matrix display driver for max7219 devices.



Breaking changes

Warning: Version 0.3.0 was released on 19 January 2017: this came with a rename of the project in github from **max7219** to **luma.led_matrix** to reflect the changing nature of the codebase. It introduces a complete rewrite of the codebase to bring it in line with other 'luma' implementations.

There is no direct migration path, but the old documentation and PyPi packages will remain available indefinitely, but that deprecated codebase will no longer recieve updates or fixes.

This breaking change was necessary to be able to add different classes of devices, so that they could reuse core components.

luma.led_matrix.device

class luma.led_matrix.device.**max7219**(*serial_interface=None*, *width=8*, *height=8*, *cascaded=None*, *rotate=0*, *block_orientation='horizontal'*, ***kwargs*)

Bases: luma.core.device.device

Encapsulates the serial interface to a series of 8x8 LED matrixes daisychained together with MAX7219 chips. On creation, an initialization sequence is pumped to the display to properly configure it. Further control commands can then be called to affect the brightness and other settings.

capabilities(*width*, *height*, *rotate*, *mode='1'*)

Assigns attributes such as `width`, `height`, `size` and `bounding_box` correctly oriented from the supplied parameters.

Parameters

- **width** (*int*) – the device width
- **height** (*int*) – the device height
- **rotate** (*int*) – an integer value of 0 (default), 1, 2 or 3 only, where 0 is no rotation, 1 is rotate 90° clockwise, 2 is 180° rotation and 3 represents 270° rotation.
- **mode** (*str*) – the supported color model, one of "1", "RGB" or "RGBA" only.

cleanup()

Attempt to switch the device off or put into low power mode (this helps prolong the life of the device), clear the screen and close resources associated with the underlying serial interface.

This is a managed function, which is called when the python processs is being shutdown, so shouldn't usually need be called directly in application code.

clear()

Initializes the device memory with an empty (blank) image.

command(**cmd*)

Sends a command or sequence of commands through to the delegated serial interface.

contrast(*value*)

Sets the LED intensity to the desired level, in the range 0-255.

Parameters **level** (*int*) – Desired contrast level in the range of 0-255.

data(*data*)

Sends a data byte or sequence of data bytes through to the delegated serial interface.

display(*image*)

Takes a 1-bit `PIL.Image` and dumps it to the LED matrix display via the MAX7219 serializers.

hide()

Sets the display mode ON, waking the device out of a prior low-power sleep mode.

preprocess(*image*)

Performs the inherited behviour (if any), and if the LED matrix is declared to being a common row cathode, each 8x8 block of pixels is rotated 90° clockwise.

show()

Switches the display mode OFF, putting the device in low-power sleep mode.

class luma.led_matrix.device.**neopixel**(*dma_interface=None*, *width=8*, *height=4*, *cascaded=None*, *rotate=0*, *mapping=None*, ***kwargs*)

Bases: luma.core.device.device

Encapsulates the serial interface to a series of RGB neopixels daisy-chained together with WS281x chips. On creation, the array is initialized with the correct number of cascaded devices. Further control commands can then be called to affect the brightness and other settings.

Parameters

- **dma_interface** – The WS2812 interface to write to (usually omit this parameter and it will default to the correct value - it is only needed for testing whereby a mock implementation is supplied)
- **width** (*int*) – The number of pixels laid out horizontally
- **height** – The number of pixels laid out vertically
- **cascaded** – The number of pixels in a single strip - if supplied, this will override `width` and `height`.
- **rotate** (*int*) – Whether the device dimenstions should be rotated in-situ: A value of: 0=0°, 1=90°, 2=180°, 3=270°. If not supplied, zero is assumed.
- **mapping** (*int[]*) – An (optional) array of integer values that translate the pixel to physical offsets. If supplied, should be the same size as `width * height`

capabilities(*width*, *height*, *rotate*, *mode='1'*)

Assigns attributes such as `width`, `height`, `size` and `bounding_box` correctly oriented from the supplied parameters.

Parameters

- **width** (*int*) – the device width
- **height** (*int*) – the device height
- **rotate** (*int*) – an integer value of 0 (default), 1, 2 or 3 only, where 0 is no rotation, 1 is rotate 90° clockwise, 2 is 180° rotation and 3 represents 270° rotation.
- **mode** (*str*) – the supported color model, one of "1", "RGB" or "RGBA" only.

cleanup()

Attempt to reset the device & switching it off prior to exiting the python process.

clear()

Initializes the device memory with an empty (blank) image.

command(**cmd*)

Sends a command or sequence of commands through to the delegated serial interface.

contrast(*value*)

Sets the LED intensity to the desired level, in the range 0-255.

Parameters **level** (*int*) – Desired contrast level in the range of 0-255.

data(*data*)

Sends a data byte or sequence of data bytes through to the delegated serial interface.

display(*image*)

Takes a 24-bit RGB `PIL.Image` and dumps it to the daisy-chained WS2812 neopixels.

hide()

Not supported

preprocess(*image*)

Provides a preprocessing facility (which may be overridden) whereby the supplied image is rotated according to the device's rotate capability. If this method is overridden, it is important to call the super

Parameters **image** (*PIL.Image.Image*) – An image to pre-process

Returns A new processed image

Return type PIL.Image.Image

show()
Not supported

luma.led_matrix.virtual

class luma.led_matrix.virtual.**sevensegment**(*device*, *undefined='_'*, *mapper=<function dot_muncher>*)

Bases: object

Abstraction that wraps a MAX7219 device, this class provides a `text` property which can be used to set and get a value, which is propagated onto the underlying device.

Parameters

- **device** – A MAX7219 device instance
- **undefined** (*char*) – The default character to substitute when an unrenderable character is supplied to the text property.
- **mapper** – A function that maps strings into the correct representation for the 7-segment physical layout. By default, a "dot" muncher implementation is used which places dots inline with the preceeding character.

text
Returns the current state of the text buffer. This may not reflect accurately what is displayed on the seven-segment device, as certain alpha-numerics and most punctuation cannot be rendered on the limited display

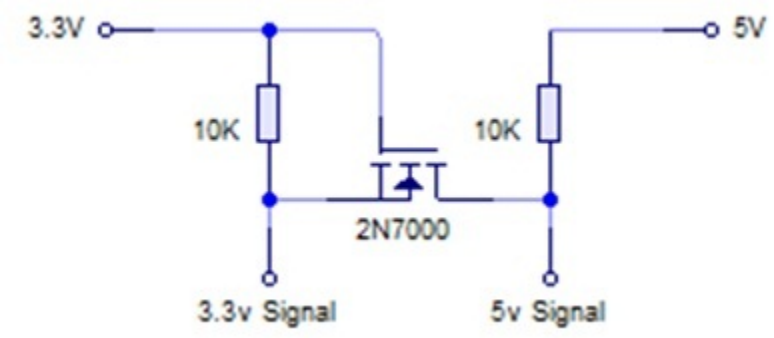
CHAPTER 5

Notes

Cascading, power supply & level shifting

The MAX7219 chip supports cascading devices by connecting the DIN of one chip to the DOUT of another chip. For a long time I was puzzled as to why this didnt seem to work properly for me, despite spending a lot of time investigating and always assuming it was a bug in code.

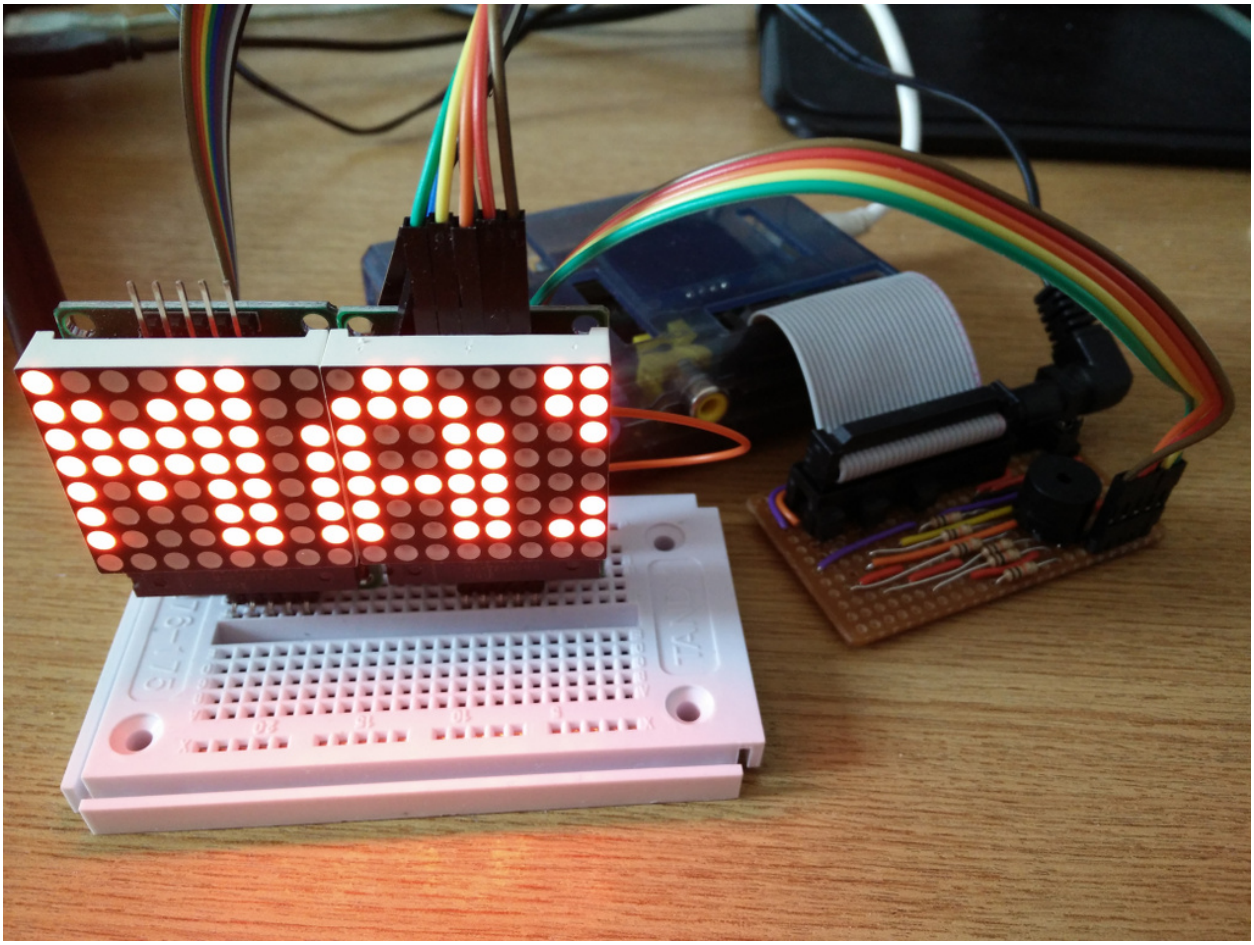
- Because the Raspberry PI can only supply a limited amount of power from the 5V rail, it is recommended that any LED matrices are powered separately by a 5V supply, and grounded with the Raspberry PI. It is possible to power one or two LED matrices directly from a Raspberry PI, but any more is likely to cause intermittent faults & crashes.
- Also because the GPIO ports used for SPI are 3.3V, a simple level shifter (as per the diagram below) should be employed on the DIN, CS and CLK inputs to boost the levels to 5V. Again it is possible to drive them directly by the 3.3V GPIO pins, it is just outside tolerance, and will result in intermittent issues.



Despite the above two points, I still had no success getting cascaded matrices to work properly. Revisiting the wiring, I had connected the devices in serial connecting the out pins of one device to the in pins of another. This just produced garbled bit patterns.

Connecting all the CS lines on the input side together and CLK lines on the input side all together worked. The same

should probably apply to GND and VCC respectively: Only the DOUT of one device should be connected to the next devices DIN pins. Connecting through the output side, never worked consistently; I can only assume that there is some noise on the clock line, or a dry solder joint somewhere.



CHAPTER 6

References

- http://hackaday.com/2013/01/06/hardware-spi-with-python-on-a-raspberry-pi/
- http://gammon.com.au/forum/?id=11516
- http://louisthiery.com/spi-python-hardware-spi-for-raspi/
- http://www.brianhensley.net/2012/07/getting-spi-working-on-raspberry-pi.html
- http://raspi.tv/2013/8-x-8-led-array-driven-by-max7219-on-the-raspberry-pi-via-python
- http://quick2wire.com/non-root-access-to-spi-on-the-pi

CHAPTER 7

Contributing

Pull requests (code changes / documentation / typos / feature requests / setup) are gladly accepted. If you are intending some large-scale changes, please get in touch first to make sure we're on the same page: try and include a docstring for any new methods, and try and keep method bodies small, readable and PEP8-compliant.

GitHub

The source code is available to clone at: http://github.com/rm-hull/luma.led_matrix

Contributors

- Thijs Triemstra (@thijstriemstra)
- Jon Carlos (@webmonger)
- Unattributed (@wkapga)
- Taras (@tarasius)
- Brice Parent (@agripo)
- Thomas De Keulenaer (@twdkeule)

CHAPTER 8

ChangeLog

Version	Description	Date
Upcoming	• TM1637 4-character seven-segment display driver	
0.7.0	• **BREAKING CHANGE:** Move sevensegment class to `luma.led_matrix.virtual` package • Documentation updates & corrections	2017/03/04
0.6.2	• Allow MAX7219 and NeoPixel driver constructors to accept any args	2017/03/02
0.6.1	• Restrict exported Python symbols from `luma.led_matrix.device`	2017/03/02
0.6.0	• Add support for arbitrary MxN matrices rathern than a single chain	2017/02/22
0.5.3	• Huge performance improvements for cascaded MAX7219 devices • Documentation updates	2017/02/21
0.5.2	• Add apostrophe representation to seven-segment display	2017/02/19
0.5.1	• Deprecate `luma.led_matrix.legacy` (moved to `luma.core.legacy`)	2017/02/19

CHAPTER 9

The MIT License (MIT)

Python Module Index

l

Index