Intelligent 8x8 RGB LED matrix

This month we're looking at a module with an 8x8 matrix of 64 'intelligent' RGB LEDs. Each LED can display over 16 million different colours, or primary colours at 256 brightness levels. The LEDs are controlled serially via a single wire, and multiple modules can be cascaded to build a much larger display. That makes for all sorts of useful applications!

We looked at some 8x8 LED display modules in an earlier article in this series, back in the June 2017 issue (siliconchip.com.au/Article/10680). We thought it was worth writing this one up too, as it is significantly more flexible and just generally more useful.

It uses RGB (red/green/blue) LEDs rather than monochrome (single colour) LEDs. Each LED can display up to 256 brightness levels for each of the three colours, to give a total of 16,777,216 (256 × 256 × 256) different colours.

In this module, each RGB LED has its own built-in serial data register, latch register and decoder/driver, so no separate controller is needed.

All 64 LEDs of the module are connected in sequential (daisy-chain) fashion, so that serial data can be fed into the first LED of the module and passed through to the other LEDs in turn.

If you want to use multiple modules, the data output from the 64th LED on the first module can be fed to the first LED of the next module to program its LEDs as well. And so on.

This module is based on an impressive device: the WS2812B intelligent control LED made by WorldSemi, based in Dongguan, Guangdong province, China (between Guangzhou and Shenzhen, and near Hong Kong).

I should note that some of the modules currently available use a 'clone' of the WS2812B device, the SK6812, made by another Chinese firm: Shenzhen Sikewei Electronics. Although the timing specs for the SK6812 differ a little from those of the WS2812B,

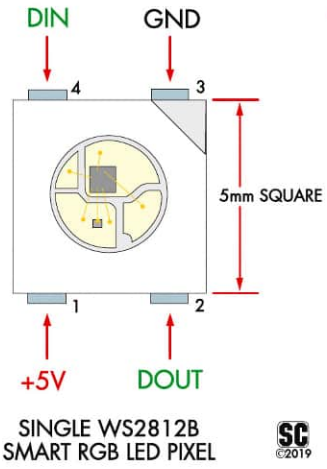
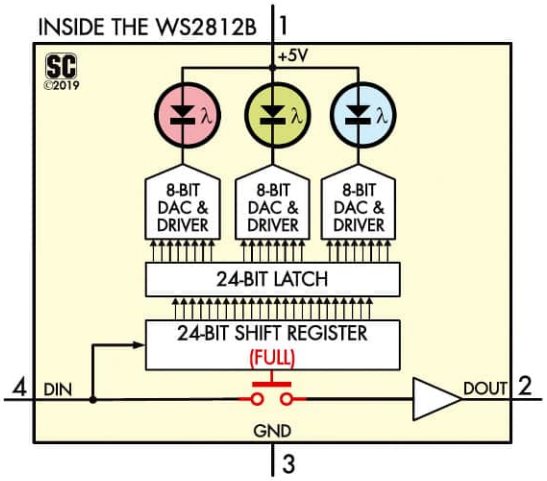


Fig.1: the SMD package size and pinout of the WS2812B (and equivalent) chips. Internally, it's made from multiple semiconductor dies, tied together with bond wires and encapsulated with a plastic lens on top. Note that the package orientation marking is located on pin 3, rather than pin 1.

Fig.2: as well as the red, green and blue LED dies, the WS2812B incorporates a controller/driver IC, which includes a serial latch plus three linear LED drivers with 8-bit DACs.



they are quite compatible with most of the available software.

You can find these WS2812B/SK6812-based 8x8 RGB LED modules on the internet from various vendors, many of them available via sites like eBay or AliExpress (www.aliexpress.com/item/32671025605.html). The prices vary quite a bit. You can find them for between $8 and $26 each. So it pays to search around!

Now let us look at the WS2812B IC to see how it works. This description applies to the SK6812 as well.

The WS2812B LED chip

Inside its small (5 x 5 x 1.6mm) four-lead SMD package, shown in Fig.1, this device houses a trio of LEDs as well as a serial controller IC. It looks deceptively simple, but you can see from the block diagram (Fig.2), there's quite a lot inside.

It includes a 24-bit shift register, a 24-bit latch, three eight-bit DACs (digital-to-analog converters) coupled to a driver for each LED and even a buffer amplifier to boost and reshape the serial data output, ready for feeding to the next WS2812B.

Fig.3 shows how a string of 64 WS2812B devices are connected to make up the module. This is simplified by showing just three of the 64 devices. The data stream from the MCU is fed into pin 4 (DIN) of the first device, while the output from pin 2 (DOUT) is connected to pin 4 of the next device, and so on.

One of the slightly interesting features of this chip is that unlike other daisy-chained shift registers, it doesn't feed the top-most 'overflow' bit of the shift register to the output, for feeding into the next device.

Rather, the output is held in a static state until all 24 bits have been shifted into the register (presumably, tracked via a counter register), at which point it no longer shifts in any new bits. The input is then connected to the output buffer via an internal switch.

This means that the first 24 bits of data shifted into the daisy chain determine the state of the first device. With the more typical (and simpler) shift-through design, the first bits of data end up in the last device – ie, you have to shift in the data in reverse order.

So, presumably the reason for this unusual scheme is to avoid the need to reverse the order of data being sent to an array of these devices.

The only other components are the 100nF bypass capacitors on the +5V supply line, with one next to each device. The 1000µF reservoir capacitor is external to the module.

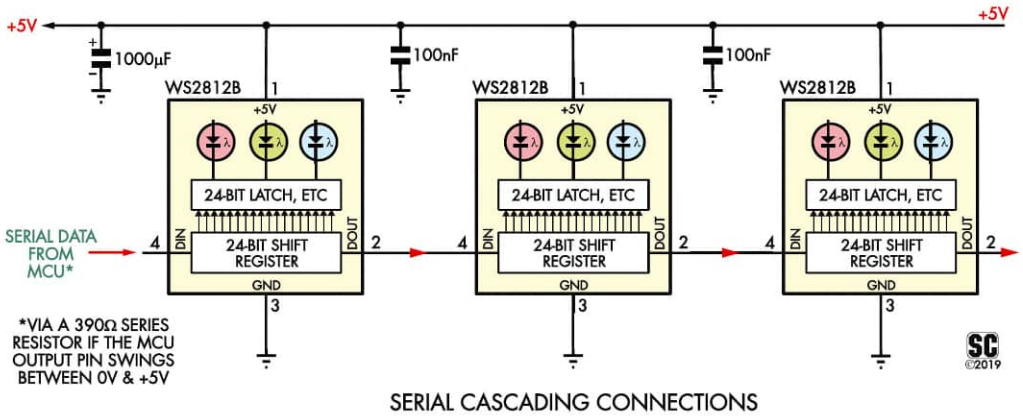
The physical layout of the 64-LED array, which measures 65 x 65mm, is shown in Fig.4. The input connections for the module are at lower left, while the output connections are at upper right.

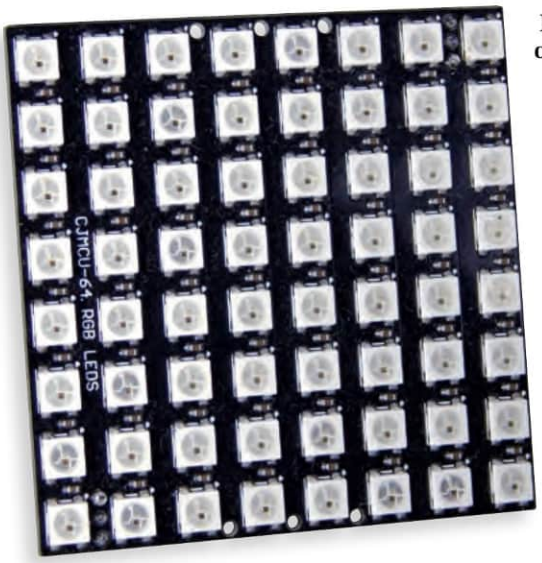
Each WS2812B device can draw up to 18mA from the +5V supply during operation, so a single 64-LED module can draw as much as 1.152A.

That's why it's recommended that even using a single module, the +5V supply for the module should not come from your MCU (Arduino or Micromite, etc), but from a separate DC supply.

It's even more important to do this when you're using several modules in cascade. This is also why that 1000µF capacitor is needed on the +5V supply line.

Fig.3: cascading multiple WS2812B devices is simple. The DOUT (data out) pin of one device is simply connected to the DIN (data in) pin of the next device. The 5V and GND pins are all connected in parallel, with a 100nF bypass capacitor close to each device.





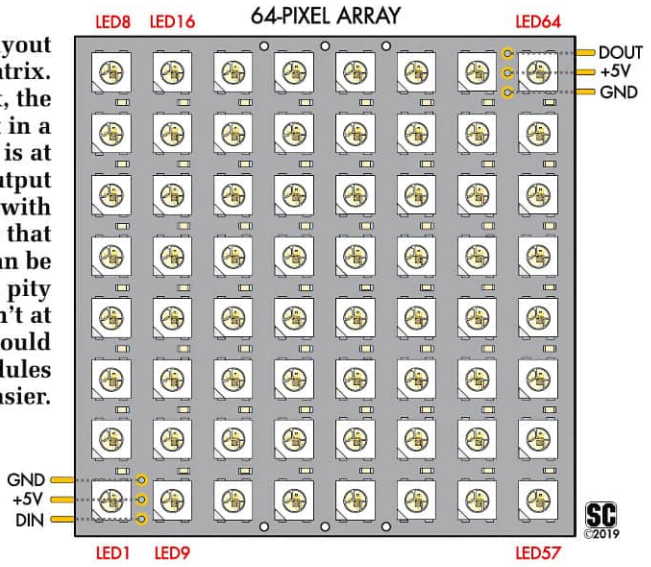


Fig.4: this shows the layout of the 8x8 RGB LED matrix. As you would expect, the LEDs are laid out in a grid. The data input is at lower left and data output at upper right (along with the supply pins), so that multiple modules can be daisy-chained. It's a pity that the output isn't at lower right, as that would make chaining modules considerably easier.

Driving the module

The LEDs in these modules are programmed serially via a single wire, as mentioned earlier. But they use a special pulse width modulation (PWM) coding system for the data, shown in Fig.5.

The timing for a zero bit, a one bit and the RESET/LATCH pulse for a basic WS2812B device are shown at the top of Fig.5; this is used in most of the currently-available 8x8 modules. The corresponding timings for the latest WS2812B-V4 version of the device are shown adjacent.

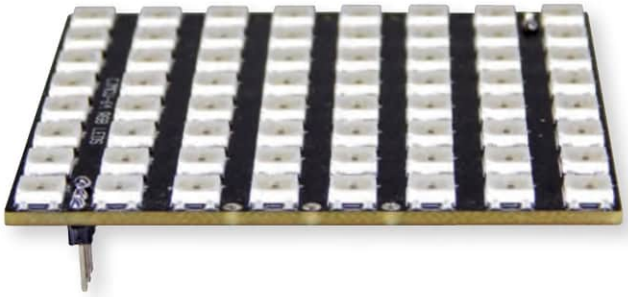
There are subtle differences in data bit timing between the two versions. The main difference is that the WS2812B needs a RESET/LATCH pulse lasting more than 50µs, while the WS2812B-V4 needs a longer pulse of more than 280µs.

Timing for the SK6812 device is similar to that for the WS2812B, with a zero bit composed of a 300ns high followed by a 900ns low, a one bit composed of a 600ns high followed by a 600ns low, and the RESET/LATCH pulse needing to be 80µs or more.

The centre section of Fig.5 shows the 24-bit data packet used to program a single WS2812B LED. There are eight bits for each of the three colours, with each colour's data byte sent MSB (most-significant-bit) first. So the total time needed to refresh one LED is either 30µs or 26.4µs, depending on the version of the WS2812B chip.

Fig.5 also shows the colour data being sent in GRB (green-red-blue) order, but some of the WS2812B or equivalent

This 8x8 RGB LED module uses WS2812B ICs. The data and power connections are made via two 3-pin male headers on the underside of the PCB.



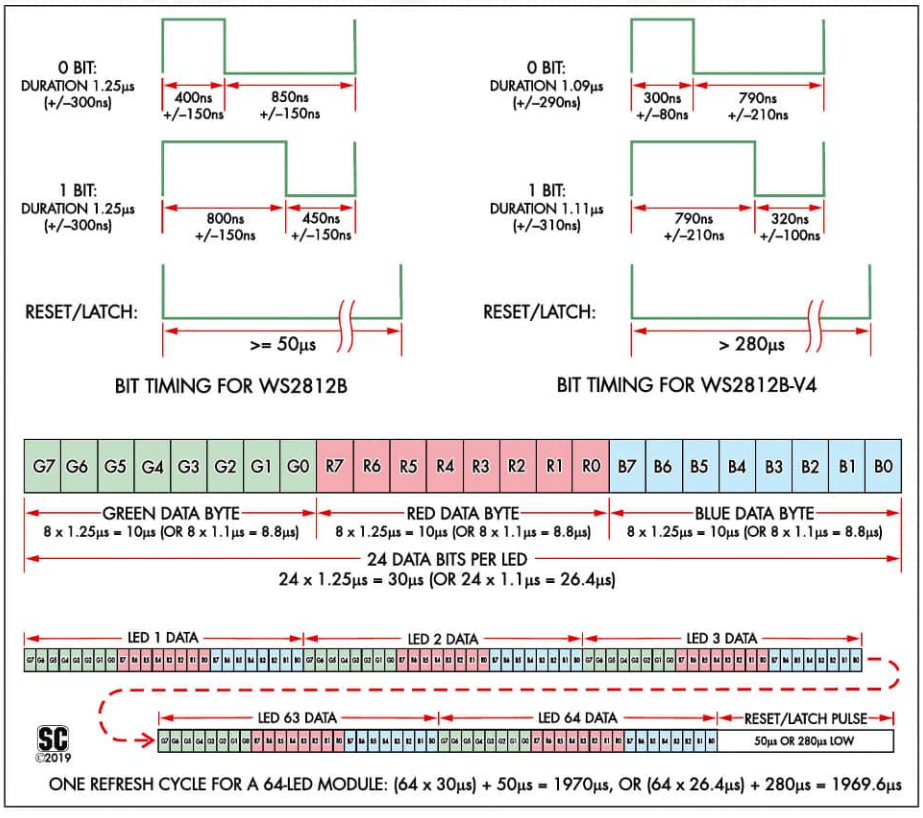


Fig.5: the WS2812B uses a custom 1-wire serial protocol, with the duration of the positive pulse distinguishing between a zero and one bit. Unfortunately, different versions of the chip require different timings, although it is possible to choose timings which will suit all versions. Note the much longer latch pulse required for the V4 chips. Also, while many chips expect colour data in the green, red, blue order shown here, some use the more standard red, green, blue order.

devices used in these modules require the data to be sent in RGB order. As a result, much of the software written for these modules allow the colour byte order to be changed to suit the specific devices being used.

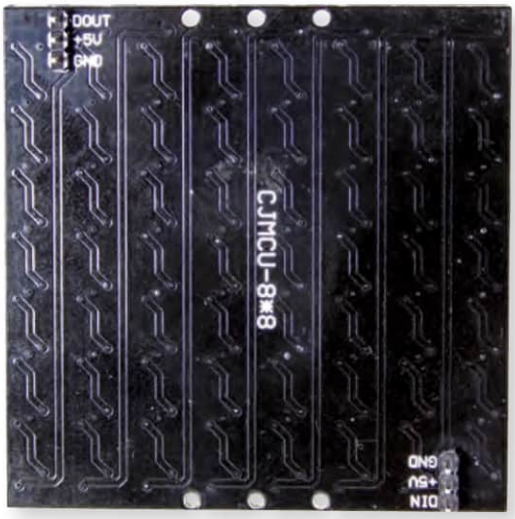
The 64-LED data stream used to program all of the WS2812B LEDs in a single 8x8 module is shown at the bottom of Fig.5. As you can see, the 24 bits of data for each of the 64 LEDs are sent in turn, followed by a RESET/LATCH pulse. This pulse instructs all of the WS2812Bs to transfer the data in their shift register into the latch register, changing the colour and brightness of its LEDs to the new values.

So one complete refresh cycle for an 8x8 module takes very close to 1970µs (1.970ms) or 1969.6µs (1.969ms), depending on which version of the WS2812B is being used. As a result, the display can be refreshed up to 500 times each second (or a fraction of this with multiple modules, eg, 100 times per second for five modules daisy-chained).

Driving it from an Arduino

Thanks to the single-wire data programming system used by the WS2812B device, it's physically quite easy to drive this module from an Arduino.

As shown in Fig.6, all that's needed is a wire connecting the module's GND pin to one of the Arduino GND pins, together with a wire with a 390Ω series resistor connecting the module's DIN pin to one of the Arduino's digital I/O pins.



While the underside of this module uses headers for external connections, some modules provide SMD pads rather than holes. It can be worthwhile to shop around, but there is a risk that you may come across clones which are not fully compatible.

Wires from the module's +5V and GND pins are then used to supply it with 5V power, with a 1000µF capacitor used as a reservoir to ensure that the 5V power remains constant.

Writing the required Arduino 'sketch' (program) is a little complicated due to the unusual PWM coding system used. Luckily, several Arduino software libraries have been written to drive a string of WS2812B/SK6812 devices.

You'll find suitable programs in various places on the Web, most of them fairly simple and straightforward. Many of them make use of a library of routines for the Arduino written by the Adafruit people and called "Adafruit_NeoPixel".

To get you started, I've written a sketch called "RGBLED_Matrix_sketch.ino", which is available for download from the SILICON CHIP website. It uses the Adafruit_NeoPixel library, which can be downloaded from https://github.com/adafruit/Adafruit_NeoPixel (or via the Arduino IDE's Library Manager).

This sketch allows you to produce one of nine different patterns on the module, simply by sending a digit (from 1 to 9) to the Arduino from your PC's serial port (eg, via the IDE's Serial Monitor). For example, sending a "1" produces a changing rainbow pat-

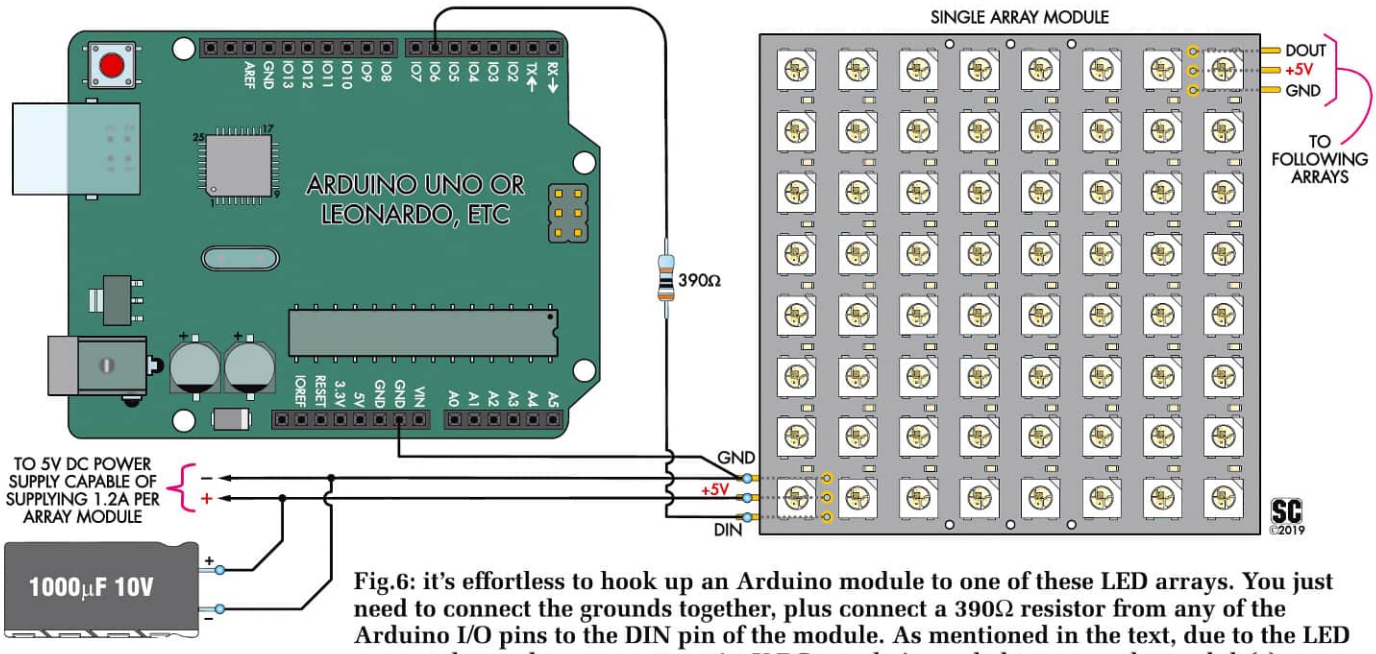


Fig.6: it's effortless to hook up an Arduino module to one of these LED arrays. You just need to connect the grounds together, plus connect a 390Ω resistor from any of the Arduino I/O pins to the DIN pin of the module. As mentioned in the text, due to the LED current demands, a separate >1A 5V DC supply is needed to power the module(s).

tern, sending a "3" produces a display of all LEDs glowing mid-green, sending a "6" produces a pattern of white dots 'chasing' each other, etc.

While this may not sound terribly exciting, it should give you a good idea of what's involved in driving these modules from an Arduino.

Driving it from a Micromite

Driving one of the modules from a Micromite again isn't easy, mainly because of the PWM bit encoding scheme.

After trying to make unorthodox use of MMBasic's built-in SPI communications protocol (with no luck), I realised that I would need an embedded C function similar to Geoff Graham's SerialTX module.

CFUNCTIONs allow native 'machine language' code to be added to an MMBasic program. This would let me send the serial data streams to the LED module with the right encoding and at the right speed.

I was rather daunted at the prospect of writing this CFUNCTION. But Geoff Graham advised me that a suitable function had already been created by Peter Mather, one of the Micromite 'gurus' on The BackShed Forum (siliconchip.com.au/link/aavx).

I eagerly downloaded Mr Mather's CFUNCTION, and tried using it with a small MMBasic program to drive a module with 64 WS2812B LEDs. The results were a bit disappointing, with a variety of unexpected errors. This prompted me to try using my DSO to check the pulse timing of the bitstream being sent to the WS2812B LEDs, to compare it to the required timing shown in Fig.5.

I subsequently found a few differences, which seemed likely to explain the problems I was having.

After an exchange of emails with Mr Mather, I learned that his CFUNCTION had been written about four years ago to suit the original WS2812 LEDs.

He suggested a couple of changes to it to make the pulse timing more compatible with the WS2812B, SK6812 and WS2812B-V4 devices, and also guided me regarding how to make the changes easily without having to recompile his code.

I made the suggested changes and tried it all again. Now the timing of the pulse stream was much closer to that needed by the WS2812B/SK6812 devices, and, lo and behold, the modules gave the correct displays from my test program.

I then proceeded to write an expanded version of my original MMBasic test program to provide readers with a suitable demo program to run on a Micromite. This program is called "RGB LED matrix test program.bas", and again you can download it from: siliconchip.com.au/Shop/6

This program displays a 'rainbow' of coloured stripes on the 64-LED SW2812B/SK6812 module, then clears the display for another five seconds before repeating the cycle. While simple, again I hope it will give you a good idea as to how a Micromite can be used to drive these modules.

To achieve different kinds of display (including dynamic displays), all you need to do is use the MMBasic part of the program to change the 'pixel' data stored in the colours() array.

You can find some useful links on this module below:

Documentation: siliconchip.com.au/link/aavv

Datasheets: siliconchip.com.au/link/aavy and siliconchip.com.au/link/aavw SC

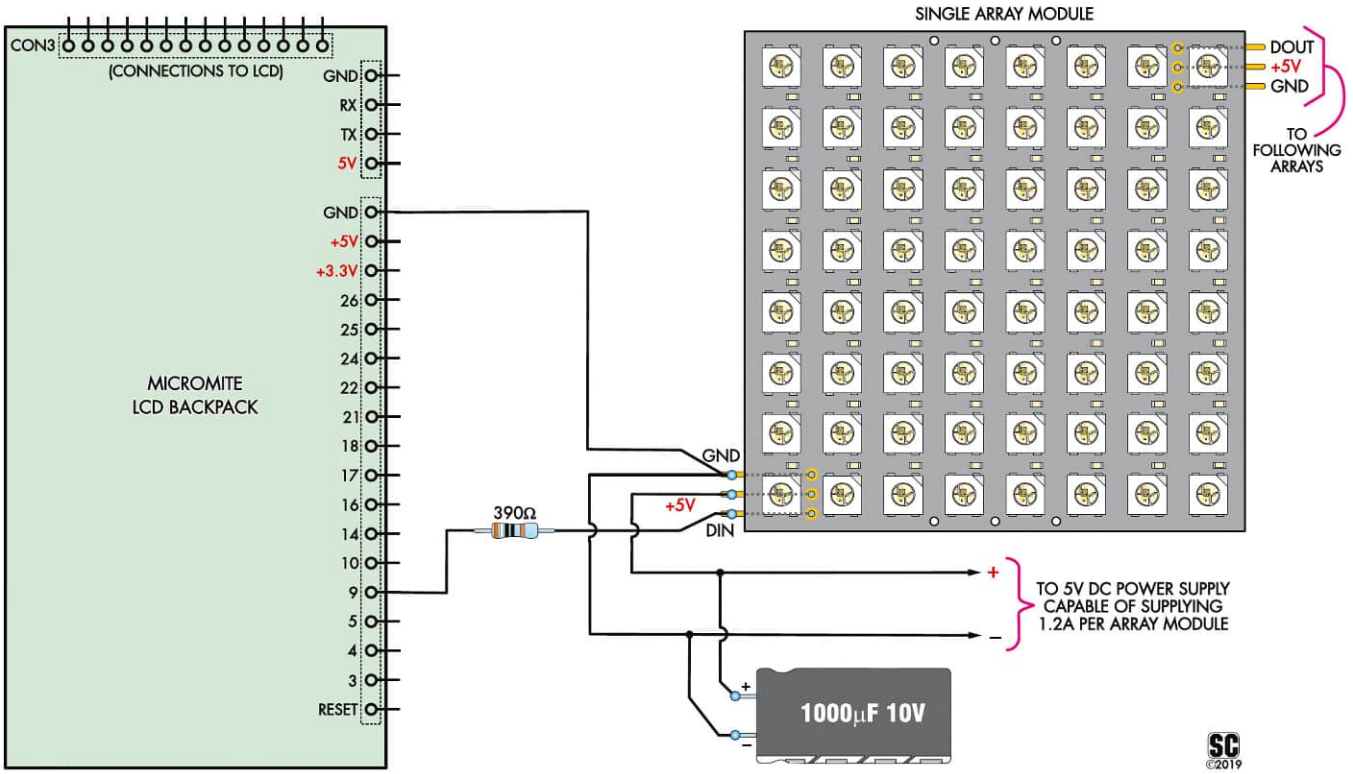


Fig.7: driving a "neopixel" LED array from a Micromite is nearly identical to an Arduino: the two grounds connected together, and a 390Ω resistor (or just a direct connection) from one of the Micromite's I/O pins to the LED array DIN pin. The software is a bit more complicated, but if you start with our sample code, it should work straight away.