DMA-Driven NeoPixels

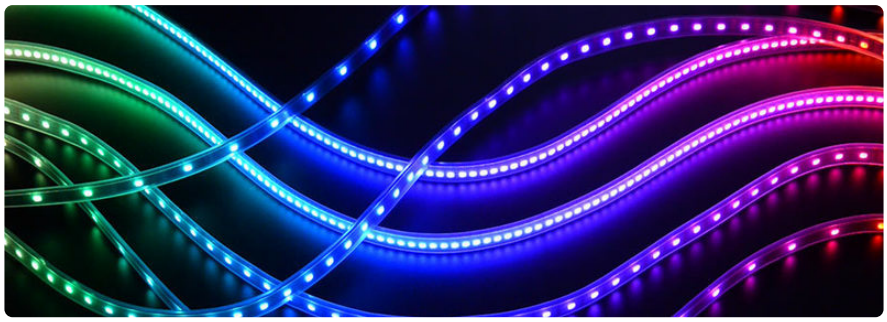
Created by Phillip Burgess

https://learn.adafruit.com/dma-driven-neopixels

Last updated on 2024-06-03 02:09:46 PM EDT

Table of Contents

Overview



If you've used **NeoPixels** extensively you've likely seen some issues in how they interact with other code. The Arduino millis() and micros() functions **lose accuracy**, the Servo library may **stutter**, or refreshing a very long NeoPixel strip can **slow a program down**.

Sometimes you can shift a task to another part of the microcontroller, as in this NeoPixels-and-Servos guide (https://adafru.it/xB0). That project's code is specific to the **8-bit AVR microcontroller** used in "classic" Arduino-type boards…it **won't work** on more **modern 32-bit boards**. But the core idea — exploiting a microcontroller's specific **peripherals** to handle a task — still applies. It's just a different microcontroller with different peripherals.

This guide solves the problem from the **other direction**: using device peripherals to drive the NeoPixels rather than the servos. But really it's applicable to all kinds of problems, **not just servos**…any NeoPixel-heavy task involving accurate timekeeping or handling interrupts.

The code here works only on boards based on **SAMD21** and **SAMD51** microcontrollers, such as the Adafruit Feather M0 and M4, Grand Central, Circuit Playground Express or Arduino Zero, using these devices' DMA-driven SPI capability. (Other boards, like the Teensy 3, may offer their own solutions…see the NeoPixel Überguide's "Advanced Coding" page (https://adafru.it/jFu).)

Setting Up

This requires installing three **libraries**. Two of these are new and experimental, so **you won't find them** in the normal Arduino Library Manager…you'll need to **download and install these manually** in your Documents/Arduino/Libraries folder. The third — Adafruit_NeoPixel — is in the Library Manager, so you can install that way (usually easier) or use the third link below.

Click to download Adafruit_ZeroDMA library

https://adafru.it/Ind

Click to download Adafruit_NeoPixel_ZeroDMA library

https://adafru.it/xAL

Click to download Adafruit_NeoPixel library

https://adafru.it/cDj

Using the Adafruit_NeoPixel_ZeroDMA Library

We'll assume some **familiarity with the regular Adafruit_NeoPixel library**. If you've never used NeoPixels before, it's best to start with the examples included with that library.

Usage is nearly identical to the regular Adafruit_NeoPixel library...only the #include line and the object type for the strip declaration are different (but the parameters are the same):

```
#include &lt;Adafruit_NeoPixel_ZeroDMA.h&gt;

Adafruit_NeoPixel_ZeroDMA strip(60, 5, NEO_GRB);
```

All other functions (begin(), show() and so forth) are invoked **exactly the same** as the regular Adafruit_NeoPixel library...so rather than repeat all that information here, just look at any existing NeoPixel example code to see how it works. Only the above two lines are different!

No Free Lunch

There are a few "gotchas" to be aware of. One...because hardware peripherals are used, and these are a finite resource, this alternate NeoPixel library **only works with specific pins:**

Board(s)	Supported Pins
Adafruit Metro M0 Arduino Zero	5, 12, MOSI*
Metro M4	6, 11, A3, MOSI*
Metro M4 AirLift	6, 11, MOSI*
Grand Central	11, 14, 23, MOSI*
Feather M0 Basic Proto	5, 6, 12, MOSI*
Feather M0 Express	6, 12, MOSI*
Feather M4	12, A2, A4, MOSI*
ItsyBitsy M0	5, 12, MOSI*
ItsyBitsy M4	2, 5, 12, MOSI*
HalloWing M0	4 (NEOPIX), 6, MOSI*
HalloWing M4	6, 8, A5, MOSI*
MONSTER M4SK	2

PyPortal (all versions)	3 (SENSE)
PyGamer (all versions)	12, A4
PyBadge	A4, MOSI*
Trellis M4	10 (keypad NeoPixels)
Circuit Playground Express	A2
Trinket M0	4**
Gemma M0	D0**
Arduino NANO 33 IoT	4, 6, 7, A2, A3, MOSI*

* If using the MOSI pin on these boards, the normal SPI peripheral can't be used for other things. Occasionally SPI is used for display add-ons, SD cards and so forth. But if you're not using any shields or FeatherWings, that pin is fair game.

** DMA NeoPixels on these boards can not be used simultaneously with I2C, Serial1 or SPI. Regular GPIO in/out is OK.

Also, the technique used in this library requires considerably **more RAM** for the NeoPixels...a little over **four times** as much. Where the "classic" library might use 180 bytes for 60 RGB NeoPixels, the new library consumes about 800 bytes for the same.

Keep in mind that these ARM chips are **3.3 Volt** devices, while NeoPixels want **5V logic**. As explained in the NeoPixel Überguide (https://adafru.it/xB7), a **logic level shifter** may be required, or you can use tricks such as a slightly lower supply voltage for the NeoPixels (e.g. 4.5V from three alkaline cells, or 3.7V from a LiPoly battery).

The Payoff

If you can work with the above constraints — compatible board, pin number and RAM usage — your code may run noticeably faster, plus you can now use the regular Arduino Servo library, and any code that relies on the millis() or micros() functions now keeps track of time correctly.

Additionally, getPixelColor() works correctly when used in combination with setBrightness(). In the classic NeoPixel library, changing brightness is a “destructive” operation and the value returned by getPixelColor() is only an approximation.

NeoMatrix Too...

There’s also a variant of the Adafruit_NeoMatrix library that uses the same methodology:

Click to download Adafruit_NeoMatrix_ZeroDMA library

https://adafru.it/xAZ

NeoMatrix is already explained in the NeoPixel Überguide (https://adafru.it/xB1), so we won’t cover the basics here.

Just like Adafruit_NeoPixel_ZeroDMA, the alternate NeoMatrix library requires just a couple changes to existing sketches — the #include line and object declaration:

```
#include &lt;Adafruit_NeoMatrix_ZeroDMA.h&gt;

Adafruit_NeoMatrix_ZeroDMA matrix(5, 8, PIN,
  NEO_MATRIX_TOP     + NEO_MATRIX_RIGHT +
  NEO_MATRIX_COLUMNS + NEO_MATRIX_PROGRESSIVE,
  NEO_GRB);
```

Everything else works the same as the regular library.

The same limitations that apply to Adafruit_NeoPixel_ZeroDMA also apply to Adafruit_NeoMatrix_ZeroDMA: it’s specific to SAMD21 and SAMD51 (aka M0 and M4) boards, works only on certain pins, and has a considerable RAM overhead (12 bytes per pixel for RGB NeoPixels, 16 bytes for RGBW pixels).

How Does It Work?

Magic!

...

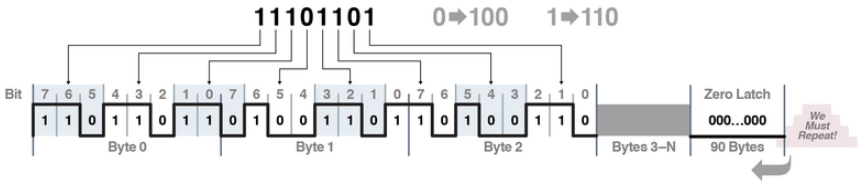
Okay, not really.

As the library name implies, **DMA** (direct memory access, the ability of this chip to move data directly between peripherals and memory without CPU intervention) is part of the solution. But this particular ARM chip does not support DMA to the GPIO PORT registers which set pins “high” or “low” (akin to Arduino’s digitalWrite() function).

The chip’s **SPI** peripherals (of which there are several, explained in this guide about SERCOMs (https://adafru.it/xB9)) do support DMA. By pulling some shenanigans, we can persuade SPI to issue a bit pattern that **resembles NeoPixel data**. The duty cycle doesn’t precisely match the datasheet, but it’s close enough to work. (The availability of SPI SERCOMs is why this library only works on specific pins.)

To do this, we expand every bit of NeoPixel color data by a factor of 3 — with a bit pattern of **100** to represent a NeoPixel “zero” bit, and **110** to represent a “one” bit. We then send this data out the SPI bus at **2.4 MHz**...with the 3:1 bit expansion, this matches the NeoPixels’ desired **800 KHz** data rate.

This 3:1 expansion explains the additional RAM use in this library. There’s the original NeoPixel color data in memory (3 or 4 bytes per pixel for RGB or RGBW), then a 3-fold bit-expanded copy to send over SPI.



Geting the end-of-data latch (300 microseconds of a LOW logic state) proved tricky, because the SPI idle state is a HIGH logic level. Switching quickly between SPI and GPIO at the beginning and end of each transaction didn’t quite work...this would always “glitch” the first bit out and mess up the first NeoPixel’s color. Solution was to end the NeoPixel data transfer with 300 microseconds worth of zero bits (about 90 bytes worth), then set DMA to run in an endless loop. So...even when the pixels aren’t changing, data is being sent again and again regardless. This doesn’t cost us

anything time-wise, there's zero impact on program speed, because this is all handled through DMA.

NeoPXL8

Sometimes even a DMA-driven strand of NeoPixels isn't enough. Sometimes you need a firehose of color!

NeoPXL8 (pronounced "NeoPixelate") is a more intense approach to driving NeoPixels on SAMD21 and SAMD51 (aka M0 and M4) microcontrollers, with **eight separate strands all running concurrently**. It shares the same benefits as NeoPixel_ZeroDMA — minimal CPU use, nondestructive brightness setting, plays well with timekeeping functions and servos — and then adds:

- **Faster overall refreshes.** NeoPixels aren't especially fast at receiving data, and sometimes programs must sit idle while data is issued to extremely long strands. By splitting the job across eight shorter strands, all writing at once, this wait time can be reduced significantly.
- **More manageable topology** for certain applications. Consider wearables: a costume incorporating NeoPixels in a single long strand may require lots of long "return wires" for data...pixels down one arm, return wire, pixels down another arm, return wire, legs and so forth...complicating the build and making it more prone to breakage. Separate strands for each area can make this easier and more robust!

Hardware

NeoPXL8 fares best on a SAMD21- or SAMD51-based board with **lots of I/O pins**. An **Adafruit Metro Express, Metro M4** or **Arduino Zero** is ideal, but some **Feather M0** and **M4** boards can join in the fun as well...even **ItsyBitsy M0** or **M4** or the tiny **QTPy M0**

board, though you won't get the full 8 outputs there. Don't bother on a Circuit Playground Express or Gemma M0...with only a few I/O pins, they're better served with a single strand and the Adafruit_NeoPixel_ZeroDMA library described on the prior page.

Keep in mind that these microcontrollers are **3.3 Volt** devices, while NeoPixels want **5V logic**. As explained in the NeoPixel Überguide (https://adafru.it/xB7), a **logic level shifter** may be required, or you can use tricks such as a slightly lower supply voltage for the NeoPixels (e.g. 4.5V from three alkaline cells, or 3.7V from a LiPoly battery).

Setting Up

This requires installing **three libraries**. Two of these are new and experimental, so **you won't find them** in the normal Arduino Library Manager...you'll need to **download and install these manually** in your Documents/Arduino/Libraries folder. The third — Adafruit_NeoPixel — is in the Library Manager, so you can install that way (usually easier) or use the third link below.

Click to download Adafruit_ZeroDMA library

https://adafru.it/Ind

Click to download Adafruit_NeoPXL8 library

https://adafru.it/y5e

Click to download Adafruit_NeoPixel library

https://adafru.it/cDj

Using the Adafruit_NeoPXL8 Library

We'll assume some **familiarity with the regular Adafruit_NeoPixel library**. If you've never used NeoPixels before, it's best to start with the examples included with that library.

Usage is nearly identical to the regular Adafruit_NeoPixel library...only the #include line and the object declaration are different:

```
#include &lt;Adafruit_NeoPXL8.h&gt;

Adafruit_NeoPXL8 strip(60, NULL, NEO_GRB);
```

The first argument — the NeoPixel strand length — is similar to the original NeoPixel library, but in the case of this library there are **eight times** this number of pixels overall. So in the above example there are 60 pixels per strand, or 480 pixels overall. If using strands of different lengths, set this to the longest one.

The second argument can be used for assigning different pins to the 8 outputs, but there are very specific rules for this...we'll cover that shortly. If you pass NULL here, this uses the library's default pin assignment, which is spread across digital pins 0 through 7 (you'll lose access to the Serial1 peripheral on pins 0 & 1 in this case).

Third argument is the pixel color order. Different generations of NeoPixel have used different byte ordering formats, and this lets you reassign them (including RGBW NeoPixels). All pixels on all strands must be the same type. You can leave this argument off if using the most common NeoPixel varieties (GRB byte order).

All other functions (begin(), show() and so forth) are invoked **exactly the same** as the regular Adafruit_NeoPixel library...so rather than repeat all that information here, just look at any existing NeoPixel example code to see how it works. Only the setup is different!

Pixels are addressed (via setPixelColor()) as a **single sequential strand**. Supposing the number-of-NeoPixels argument is 20 (meaning 160 pixels overall), the first physical strand would be pixels 0-19, second strand is 20-39, third is 40-59 and so forth, up to pixel #159 on the eighth strand.

Changing Pin Assignments

By default the library uses digital pins 0 through 7 for the eight concurrent NeoPixel strands. This works really well on the Metro Express and Arduino Zero, making the pin-to-strand wiring very intuitive. But sometimes you may have pins assigned to specific tasks (such as pins 0 & 1 for the Serial1 peripheral). Or on the Feather M0 boards, their small size means some of these pins aren't even available.

It's possible to reassign some of the output pins to other locations, but be aware that the options are very limited...most have one or possibly two alternate options. This is simply an artifact of how the microcontroller has been designed.

Default Pin	Alternate Pin(s)
0	12
1	10
2	MOSI or SDA
3	A4
4	A3
5	SCK or SCL
6	11 or MISO
7	13

To use an alternate pin layout, first declare an array of eight signed 8-bit values, one for each output pin. Then pass this array as the second argument to the Adafruit_NeoPXL8 constructor. Here's one possible pinout that might be used on a Feather M0 board:

```
int8_t pins[8] = { PIN_SERIAL1_RX, PIN_SERIAL1_TX, MISO, 13, 5, SDA, A4, A3 };
Adafruit_NeoPXL8 strip(60, pins, NEO_GRB);
```

This uses the SDA pin as a NeoPixel output, meaning that I2C can't be used (any of the Wire library functions). But a lot of sensors and displays need I2C! We might instead declare it as:

```
int8_t pins[8] = { PIN_SERIAL1_RX, PIN_SERIAL1_TX, MISO, 13, 5, MOSI, A4, A3 };
```

Now I2C will work, but this knocks out SPI. On a small board like the Feather this will always be a balancing act of what peripherals you absolutely need vs. what the NeoPXL8 library is hardwarily compatible with.

Sometimes, if you need both peripherals, there's no choice but to keep NeoPXL8 away from one or more pins entirely, in which case a value of -1 should be used. Here's an example with only 7 outputs enabled:

```
int8_t pins[8] = { PIN_SERIAL1_RX, PIN_SERIAL1_TX, 11, 13, 5, -1, A4, A3 };
```

Note that when you do this these pixels still consume memory and are still indexed sequentially in the overall pixel buffer. Only 7 pins are in use but it always has to store the full 8 pins worth of pixel data.

The order of elements in this array determines which is the considered the "first" strand, which is the second, and so forth. Functionally it makes no difference but you might find it easier to route or plan wiring with the pins in a specific order. For example, suppose we wanted all the pins in increasing order by number, and let's move that -1 to the end so there's not an awkward gap in our pixel numbering:

```
int8_t pins[8] = { PIN_SERIAL1_RX, PIN_SERIAL1_TX, 5, 11, 13, A3, A4, -1 };
```

Whatever your pin arrangement, even if one or more are unused, this array must have exactly 8 elements. Fill any unused pin positions with -1. Eight elements, always, or there will be...trouble.

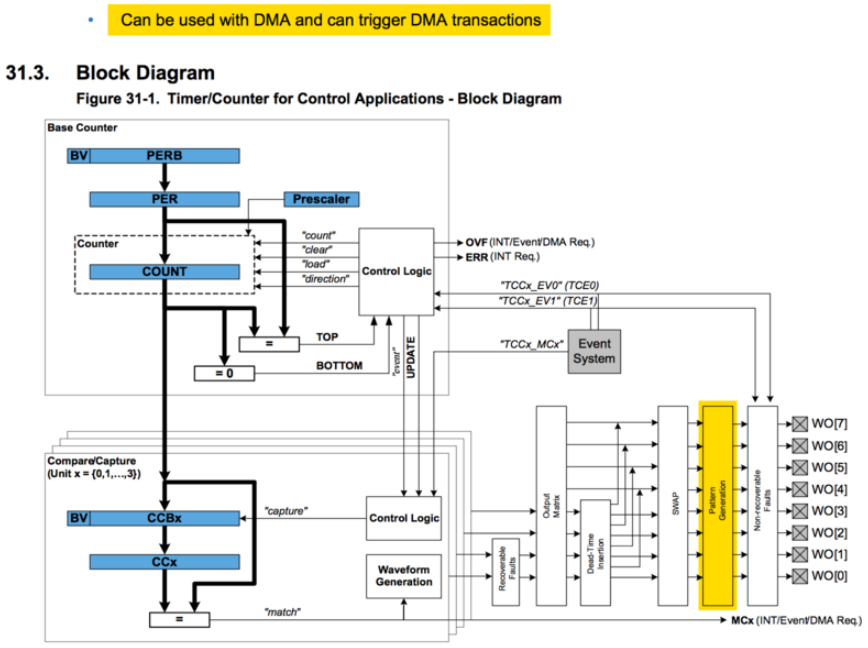
"Pin MUXing" is a complicated issue and over time we'll try to build up some ready-to-use examples for different boards and peripherals. You can also try picking your way through the SAMD21 datasheet or the NeoPXL8 source code for pin/peripheral assignments.

How Does it Work?

For all its nifty bells and whistles, one limitation of the SAMD21 microcontrollers that's frustrated us repeatedly is that there's **no DMA access to the GPIO ports**. There's DMA nearly everywhere else (ADC, DAC, SPI, etc.) but for whatever reason they decided on this chip series there'd be no fiddling of GPIO bits without the CPU's direct involvement.

Recently Ladyada spotted something in the datasheet, a side feature of one of the Timer/Counter peripherals called a "pattern generator." This apparently has applications in motor control and such...it's not documented very extensively, but is

definitely DMA-accessible. Could this provide the sort of parallel GPIO output we've been wanting for various projects?



After a lot of trial-and-error, we found that this was indeed a workaround for the DMA-less GPIO output! It only provides 8 concurrent output bits (vs. up to 32 bits for the GPIO ports), but for many applications that's sufficient.

The same Timer/Counter peripheral that provides the pattern generation also provides the 800 KHz timing required for NeoPixel data. We just need to reformat the NeoPixel data "sideways" in memory...as 8 bytes are issued concurrently, all the bit 7's are issued in one go, then all the bit 6's, bit 5's and so forth. This does require a moment to process, but combined with DMA (which then requires nothing from the CPU) the overall effect can be a dramatic reduction in both CPU load and waiting-around time.