Design of the NeoPixel API

Nicholas Polimeni

Submitted
April 26, 2022

Introduction

This document details the design and instructions for use of the NeoPixel API. The API provides an interface between Assembly and NeoPixel hardware, providing many easy-to-use functionalities.

The design centers on built-in modes for the NeoPixel which the user can select. Each mode has a distinct functionality and may be combined for complex creations, as demonstrated with the smiley-face demo. This design simplifies the most common use cases while creating a framework for additional functionality.

The following sections detail mode instructions, the design process, and lessons learned through the API's development.

Device Functionality

Our peripheral uses the following I/O addresses:

I/O Address	0xB0	0xB1	0xB2	0xB3	0xB4	0xB8
Function	Pixel Address	Pixel Data	Mode	Set All	Refresh	Pixel Count

Our peripheral uses the following modes:

Single Pixel	**Multiple Pixel**	**Pattern Generation**	**Manual Refresh**
16-bit color (Mode 1)	Set All Pixels (Any Mode)	Xmas Lights (Mode 7)	Manual Refresh (Any Mode)
16-bit auto-increment (Mode 3)	Set Group of Pixels (Mode 4)	Color Fade (Mode 8)	
24-bit color (Mode 2)			

Single Pixel

To set a pixel, first specify the pixel address and then the pixel data. 16-bit auto-increment mode will also increment the address after storing data. Any 16-bit mode sets the color with one OUT, while 24-bit color sets the color with two OUTs.

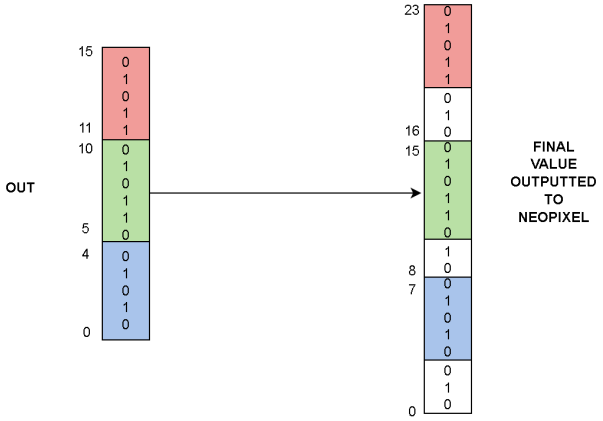

Figure 1. 16-bit color specification.

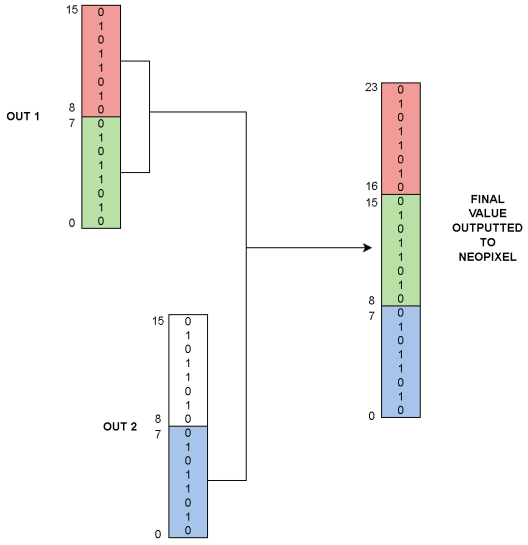

Figure 2. 24-bit color specification.

Multiple Pixels

To set a group of pixels, set a starting address, the number of pixels to set, and a color. Once in mode 4, these are sent, in order, with 3 consecutive OUTs from SCOMP. Set all pixels uses a single OUT with the 16-bit color. Set all pixels has a dedicated I/O address and no mode requirements.

Pattern Generation

Pattern generation is activated through the mode. The pattern begins when the peripheral enters the mode and will end once the peripheral exits the mode. Christmas Lights Mode will display a pattern of flashing red, white, and green pixels. Color Fade mode will take each pixel currently on the NeoPixels, and barrel shift it through different colors.

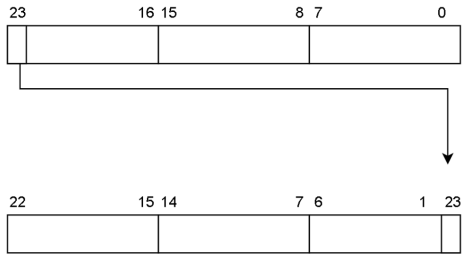

Figure 3. Color Fade's method of shifting pixel colors.

Manual Refresh

Manual refresh pauses the update of the NeoPixel array at the end of the current frame. During this time, the user may still update the NeoPixel's internal storage, but it will not appear on the pixels until the peripheral receives another OUT.

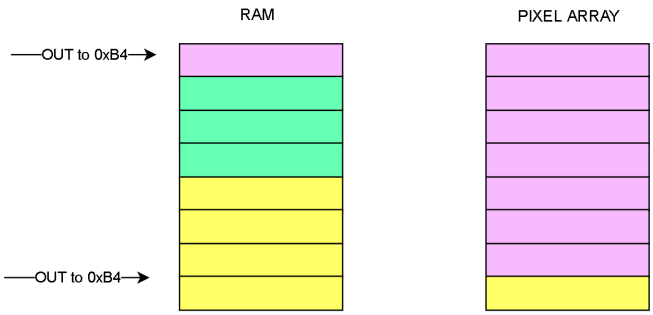

Figure 4. Visualization of manual refresh functionality for NeoPixel peripheral.

Design Decisions and Implementation

The design took inspiration from two primary places: screens and pixel light strings. Some of our features mimic those of a normal screen interface, like the ability to set several consecutive pixels and manual refresh mode. These allow the user to quickly make multiple changes and variably control the frame rate, necessary for many graphical applications. Our pattern generation features were inspired by other pixel strings. While not providing much flexibility to the user, they create interesting patterns which are simple to activate. One design decision we made was to include a mode register. The mode register enabled reuse of I/O addresses which had the same purpose. For example, we were able to use the same data address for 16-bit mode and set group, making our peripheral more intuitive.

Conclusion

The design excels in ease-of-use with consideration to the initial project, which outlined basic functionality to be implemented.

If the design process was to be completed again, more time should be spent perfecting the framework – in this case, the mode system – before working on individual functionalities. Any changes to this framework throughout development caused issues in every functionality.

However, having a strong framework was also a strength; it was very clear where and how additional functionalities should be implemented, and how these would be interfaced with the others.

One key improvement could have been removing repetitive code, improving code clarity. Regardless, the project was a success, as demonstrated by engaging examples in the demo.