NeoPixels Controller

Jiacheng Zhang

Submitted

April 26, 2022

Introduction

This document is a general overview of the technical details of the NeoPixels controller, with the aim of clarifying how this peripheral's functions are implemented. Our peripheral can set 16-bit, 24-bit, gradient colors, set a color with a matrix system, read information of the color from the pixel, replace a user-specified color with another target color, and use a timer to turn off the NeoPixels on time. All of the above functions can be used with simple assembly code. Our design priority is user-friendly. So in our peripheral, users can change the mode according to their needs. After each mode ends, it will return to the default mode. This makes it easier for users to track the behavior of Neopixels. Our peripheral implements everything on the previous proposal, so it's done pretty well.

Device Functionality

A general structure of asserting OUT to PXL_M, PXL_A, and PXL_D is for the most functionalities to set mode, address, and data respectively. At the pre-execution stage within a mode, a new OUT to PXL_M will abort the current process and transfer to the new mode. After the execution of each mode, the mode changed back to the default mode.

TABLE 1
NEOPIXEL CONTROLLER I/O ADDRESSES.

	PXL_A	PXL_D	PXL_M
I/O address	&HB0	&HB1	&HB2

The functionalities include

1. setting a 16-bit color at any pixel with a default address increment,
2. setting a 24-bit color using three consecutive data OUTs concatenating the lower 8-bit in the AC to make up the 24 bits,

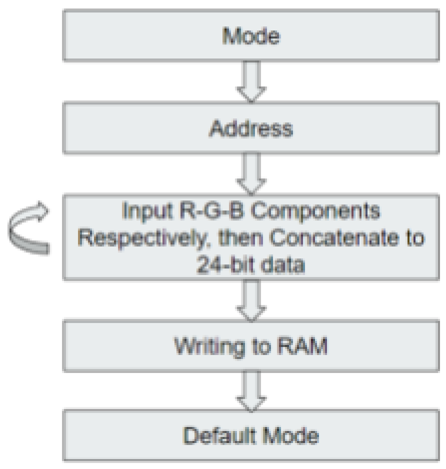
Mode
Address
Input R-G-B Components Respectively, then Concatenate to 24-bit data
Writing to RAM
Default Mode

Figure 2. Process of 24-bit color mode.

3. setting all pixels to 16-bit color using a single OUT,
4. setting the gradient color for all pixels,

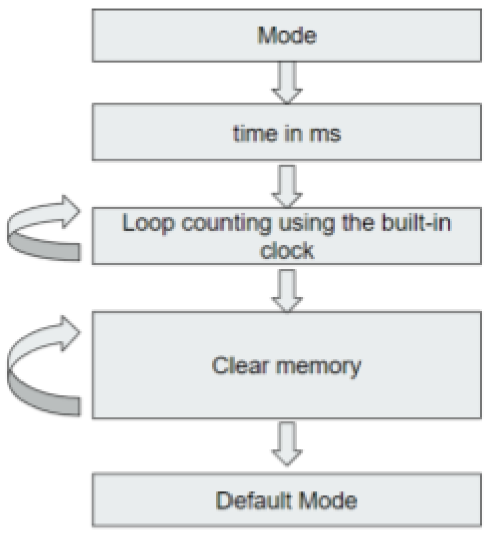

Figure 3. Process of timer mode.

5. setting the timer to set a timeout without a discrete timer peripheral,
6. reading the color to the AC of a pixel using a single IN after selecting the address,
7. replacing all colors of one kind with another target color,
8. coloring a specific pixel in a matrix system by a packed row-column coordinate.

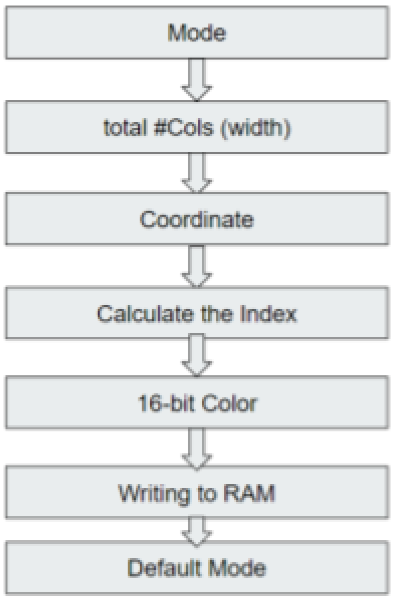

Figure 4. Process of matrix mode.

&B0000010100010101

0 **5** **1** **5**

5th Row from bottom

15th Pixel from left

Figure 5. Structure of packed coordinate in Matrix mode.

Design Decisions and Implementations

Decision is made on switching between functionalities easily. A state machine was used to handle this as it provided a robust, non-sticky framework. It also enables parallel working without interfering with each other’s code because they are in different states. For specific functionality of matrix, as the large Neopixel String is in a zig-zag pattern, to enable the matrix function, we use the last significant figure of the row index to determine whether the pixels on that row are aligned left to right or in reverse order. Based on this information, we apply different math formulas to ensure the function is working correctly.

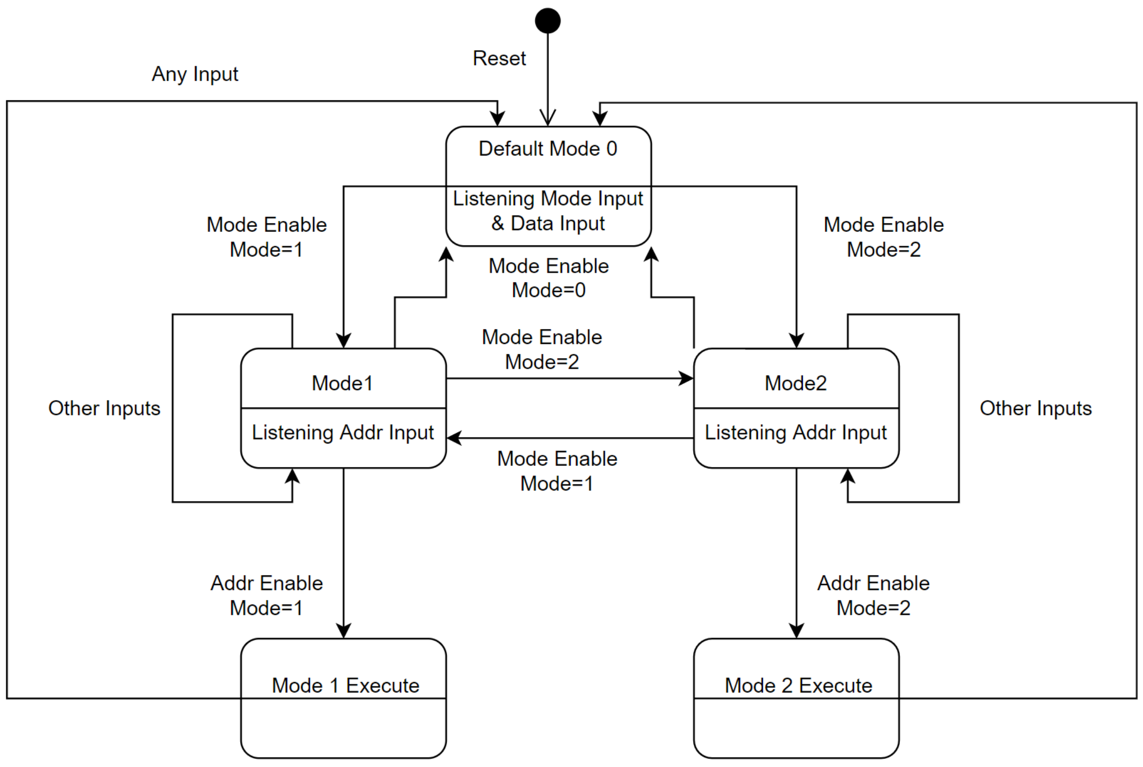

Figure 6. Sample diagram of modes as an FSM.

Plus, to better manipulate the data memory, a new signal is mapped to the altsyncram megafunction enabling dual access to the memory. This makes the features of reading color data and find-and-replace possible, without intervening in the memory access by the controller itself.

Conclusions

As we originally envisioned, our NeoPixels controller has a variety of functions and is easy to use. Users can access six different modes through simple assembly code. These modes provide convenience for setting functions such as 16-bit, 24-bit color, etc. In addition, the state machines we designed for these six modes are also non-sticky, robust, and user-friendly. As a designer, I suggest that you be familiar with project's source code before starting work and pay attention to the debugging process to avoid detours. Suppose I revisit this project in the future. In that case, I will refer more to the VHDL library during the design process to become more familiar with each port and signal and avoid getting entangled in a problem for too long. And I will think about making more interesting patterns.