CSI 370 Computer Architecture Research Project

Arduino WS2812B Addressable LEDs

Jake Rose
jacob.rose@mymail.champlain.edu

December 14, 2020

1 Introduction

In my apartment, I have a large array of lights that are Addressable LEDs. Addressable LEDs allow the color to be changed on a per-light basis, allowing for unique effects that can be much more unique than a standard LED strip.

I have achieved this already using an Arduino and the FastLED library, but I am hoping to gain more control and get faster speeds than the library can provide. That requires programming the protocol based directly off of the data sheet specification for WS2812B LED standard, the type of LED strip I am using for this project. Knowing the hardware is capable of this output is useful, but I want to see how fast I can make the refresh rate to create unique animations that are satisfying to the eye.

I want to create a very fluid bouncing dot using these LEDs. I also will be creating an iteration with the similar array for color structure like FastLED.

Full WS2812B Datasheet

I will include direct images from this document to explain the process for the Arduino when useful.

2 Enviorment

The light array is currently up to 300 LEDs. This has some unfortunate side effects, such as power consumption going over the USB 5v connector, and I do not have a 12v connector.

The specific micro-controller I will be using will be an Arduino Uno running at 16MHz. 16MHz when calculated out equals:

$$16MHz = 62.5\, nanoseconds\, per\, cycle \tag{1}$$

The WS2812B requires only three cables connected to the strip. The pin layout will be shown is the next section. The JST JM connector is the standard connector type used for these LED strips.

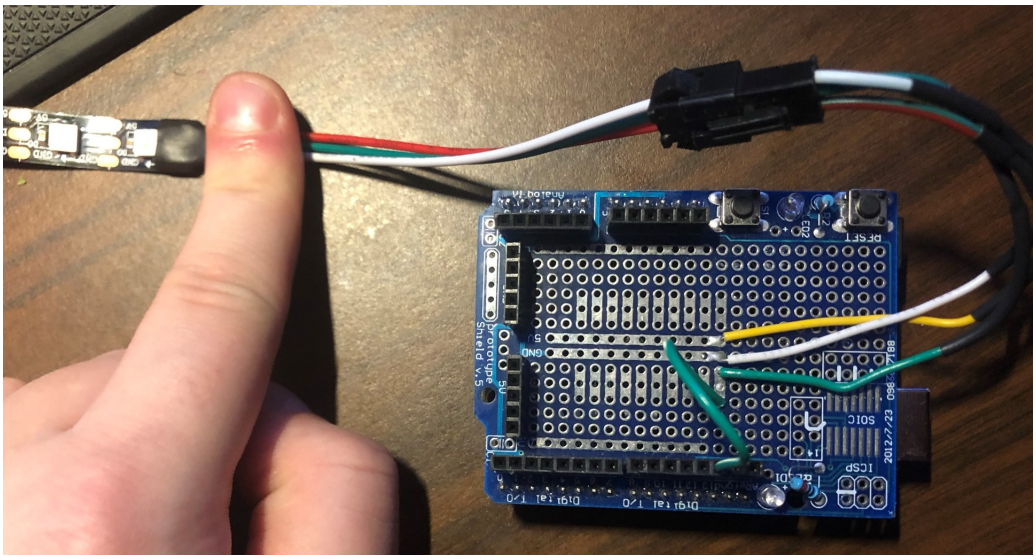
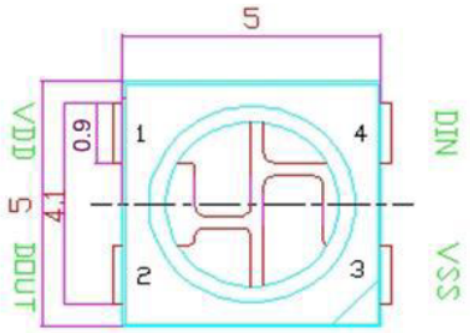


Figure 1: Arduino Layout for Project — Using an extension board that routes the positive to the red/yellow and ground to green, with the white wire routed to pin 13.

3 WS2812B Specifications

The WS2812B LED Strip uses only three wires, and each LED has four pins as shown below:

PIN configuration



PIN function

NO.	Symbol	Function description
1	VDD	Power supply LED
2	DOUT	Control data signal output
3	VSS	Ground
4	DIN	Control data signal input

Figure 2: WS2812B Pin Configuration Specifications — The data is transferred through to each LED by daisy chaining off of the previous lights, with one input and one output per light. The light modifies the output based on the input as well to allow the separation of color per light.

Using this singular wire, the specification gives guidelines on how to send color values on a per light basis. This is defined in the next three figures below.

Data transfer time(TH+TL=1.25μs±600ns)

T0H	0 code ,high voltage time	0.4us	±150ns
T1H	1 code ,high voltage time	0.8us	±150ns
T0L	0 code , low voltage time	0.85us	±150ns
T1L	1 code ,low voltage time	0.45us	±150ns
RES	low voltage time	Above 50μs	

Figure 3: WS2812B Byte Time Specifications — Proper timing is required as we are sending a stream of data instead of a singular bit. These are the timings that will be used in figure 4. Notice that we have a error leniency of 150ns, or around two full cycles of leeway in timing per variable.

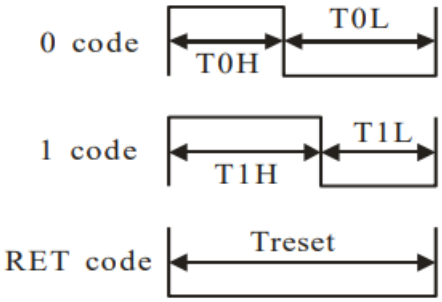


Figure 4: WS2812B Timing Order — The values from 3 are used here to show how a singular bit value of zero or one will be sent. It will be useful to note here that for both of these situations, one bit sent equals 1.25μs.

G7	G6	G5	G4	G3	G2	G1	G0	R7	R6	R5	R4	R3	R2	R1	R0	B7	B6	B5	B4	B3	B2	B1	B0

Figure 5: WS2812B Bit Order per LED — each color can have 7 bits, thus each light has 255 possible values, similar to most RGB standards.

4 Development Cycle

4.1 First Approach — Start High Level

4.1.1 Strategy

The first approach for this project was to start high level and to add in assembly after the fact to improve optimization. That proved to not be the case, as the smallest delay that can be created without assembly is one microsecond, which is just slightly too large to get the time differences required for the WS2812B data transfer.

4.1.2 Progress Made

We got the lights to turn on at full brightness, but that is the only brightness currently. The number of lights we control is also easily set, as there is a 50microsecond delay between each LED's bits being sent as specified in the sheet.

Assembly is going to be required, and using the nop command, the no-operation command, or any other operations that require one cycle, will be the only way to properly send the data.

4.2 Second Approach — Some Assembly Required

4.2.1 Strategy

The next approach was to create my own functions that would write zero and write one in a properly timed fashion. I chose this method as I was taking into considerations the few conclusions I made in the previous step.

1. Each Bit can have a bit of space between them, and this can be useful for the for loops and subroutine calls required as I am still using high-level code for the majority of this.

2. I could not create a custom delay as the calling of a function itself would use clock cycles.

3. The time per clock cycle allows two full cycles before we are outside the error leniency.

Using these findings, I figured that creating a writeZero() and writeOne() function

in the code was going to be the best bridge between the high-level code and the low-level assembly.

4.2.2 Progress Made

Using these properly timed delays, I was able to get the lights to be fully controllable on a per-LED basis. This allowed any RGB value to be inputted into the lights and it would reliably show up, and it also could work on any amount of LEDs.

One of the largest setbacks and time sinks of this project occured at this step. The lights were completely addressable, but it seems there was an issue with the writeZero() function. The writeZero() function works about 95% of the time from my estimation. This is clearly the right step, but if 5% of the bits are ones instead of zero, there is a notable flicker among random lights, I would say around 10 per second. Since this is updating so quickly, I would assume for each loop through the LEDs, around 1 or two LED's have a one where a zero should have been. The flicker is thus less noticeable when the light is set to a higher brightness.

There must be some issue either with the math behind my code, or a misreading of the specifications. I attempted every mix and match of nop operators on the writeZero() function, and even tried messing with the writeOne() function as well to see if it was having an effect, but this came to no noticeable cause.

If I were to hypothesize, I would guess that the higher level code is throwing off some timing in sending the bits, and that it can add up every now and then and the lights interpret the data as a one instead of a zero. However, after many hours attempting fixes, I wanted to finish the project with this flaw instead of getting obsessed with fixing the flicker.

4.3 Final Step — Fluid Light Bounce

It took me a while to come to the conclusion that I must move on with my life and that I needed to show off the control that was there in the program. I started by creating a simple light bouncing, setting an integer increment per LED, and it looked exactly as expected, not good.

The second attempt I used a float and incremented that by a smaller value. The benefit of doing this is that you don't need to use a delay on a per-frame basis, and that lets you get higher frame rates than the previous method.

4.4 If I Had More Time

If I had to guess, my next step would be to put the whole program in assembly. I also think I could rearrange the delays and put some before the Arduino starts outputting the bits, but the real slowdown I believe is in the readBit() function, which I think I would need to rewrite in assembly.

I also did not complete the full project in assembly, and I know half of those operations could be minimized if written in assembly, and those noops I use in the writeZero() and writeOne() could be used for other purposes instead of wasting time. This would be the best option, but I do not have time to go to such extremes. I could have also only updated the pixels that actually change per frame, which is reasonable but it gets harder and harder for each animation you add.

5 Limitations

5.1 Timing

Arduino has a very poor sense of time, as well as a slow refresh rate. They are cheap, and the fact that I can push out these frame rates is insane, however any additional math equations for other patterns is going to slow down the frame rate considerably. There is also no proper time check equation, making counting how much time has passed is impossible to a reliable degree. The onkly way we can measure time is from the clock speed, and keeping track of the number of clock cycles is the only way to properly measure time in this system. The problem is, counting the number of clock cycles dynamically requires using clock cycles, so we need to create these sections for writeZero() and writeOne() in order to get one full bit sent in the proper timing, with it ensuring that it leaves the pin power set to low when it ends the function. Any time the pin is turned on, there is a set number of clock cycles written in Arduino to clear the bit again.

5.2 Max Refresh Rate

125ms per byte is required according to figure 3

$$\begin{gathered} Time\ per\ LED = (1.25s\ to\ send\ byte * 24bits\ per\ led) \\ +50s\ per\ rest\ between\ LEDs = \\ 80\ microseconds\ per\ LED \end{gathered}$$

However, that is just the maximum theoretical input that these LEDs could handle, and that is just the sending of the color. We also need to assume that per LED, there are going to be clock cycles dedicated to calculating the color for the pixel (for the animation we program). These are just some approximations of the number of cycles for certain processes, but this will give a good sense of the magnitude of this project.

$$\begin{gathered} \textit{Time per LED Calculation} = 62.4\ ns * (10\ \textit{cycles per bit check} + \\ 12\ \textit{cycles per increment and checks} + \\ (12 * 3)\ \textit{cycles to calculate light values}) = \\ 3.619ms \end{gathered}$$

This is just for checking and calculating the light values. It is hard to account for the high-level code written and the number of clock cycles, but each of these added checks, for loops, and calculations are going to affect the maximum framerate.

6 Conclusion

I have a theory that we need to send the data and handle the loops using an array and assembly to get rid of the flickering issue, and each send we need to send all the colors at the same time, but that would require a substantial amount of assembly knowledge that was not possible in the time. However, I am very happy that I got a large amount of control with these lights and that they actually work almost perfectly except for the flicker. I got a very fluid animation out of the Arduino using floats as well, further improving the results.

Video Example Of LEDs Working