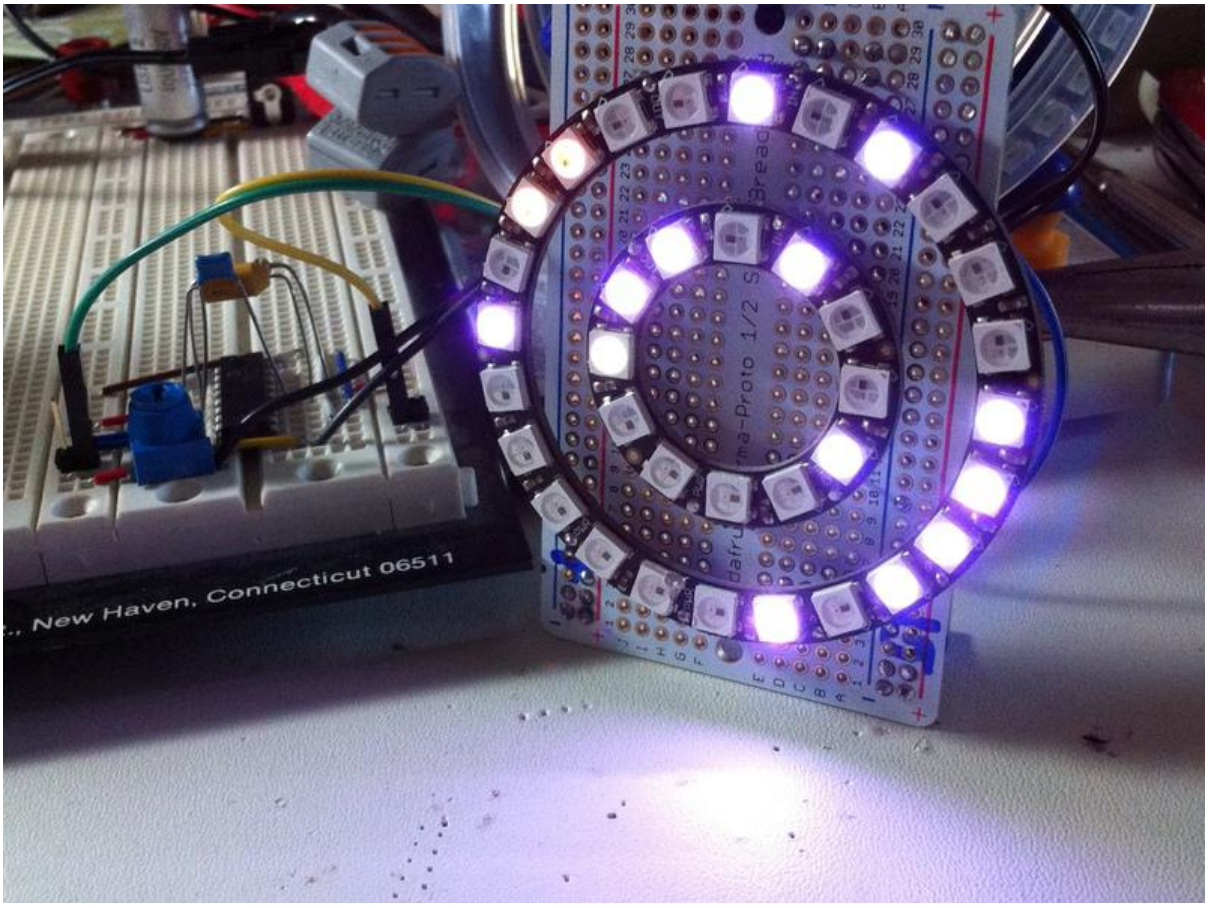
The PICsellator

Created by David Littell



https://learn.adafruit.com/the-picsellator

Last updated on 2023-08-29 02:39:51 PM EDT

Table of Contents

Introduction

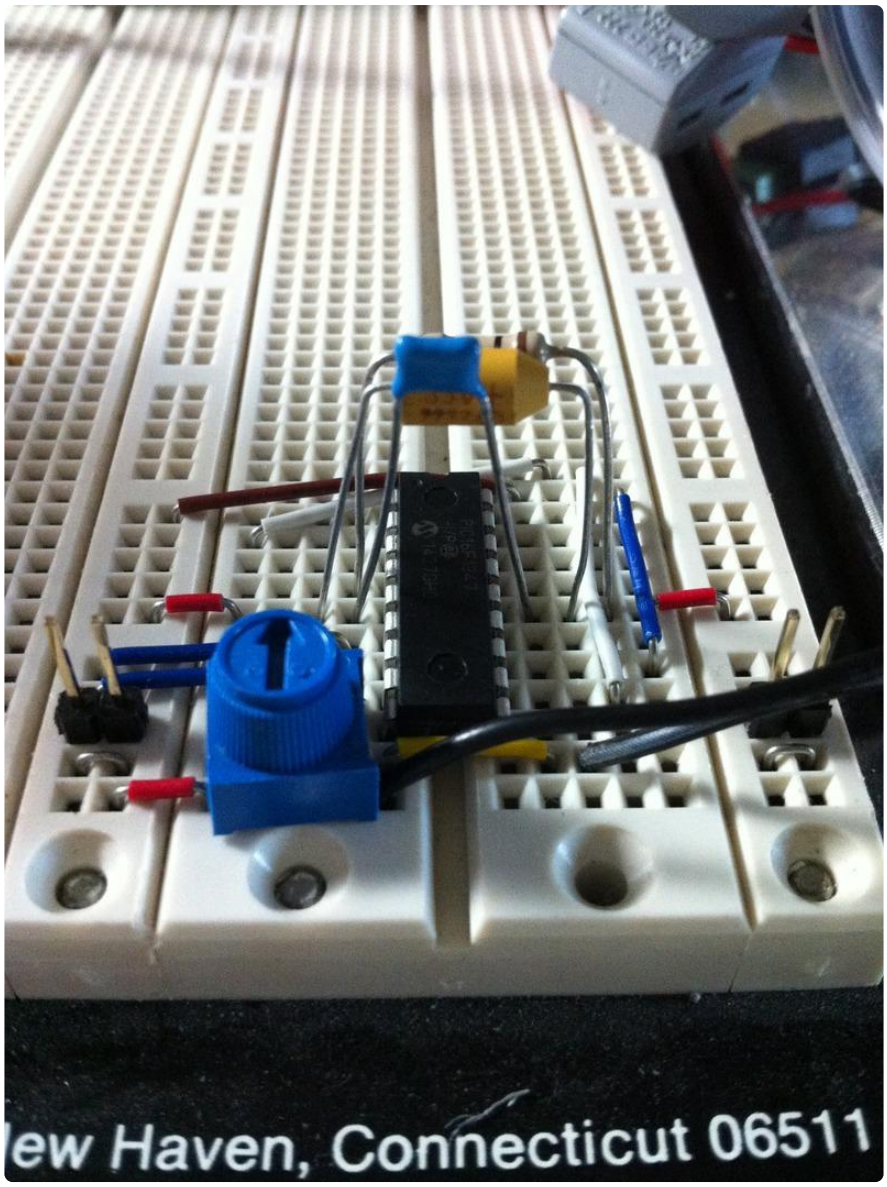
The days grow shorter, the night becomes colder - winter holidays means its time to get your LEDs out! What better time to experiment with NeoPixels during the day, then shine them at night?

This tutorial will cover more advanced uses of the WS2812/NeoPixel LEDs with a PIC. For a different project (which itself is yet another nested sub-project... ;-) I was working on a variation of the mechanism I used to generate the WS2812 bitstream in Madison's NeoClock (). This new approach still uses the PWM and DSM modules as before but drives it with the PIC's EUSART instead of the assembly-coded-bit-banged-PWM-edge-hopping contraption used in the clock. Just as with the clock, there was a bit-ordering "design point" that needed some test code to verify I had it right. This () c ame blinking out of that effort.

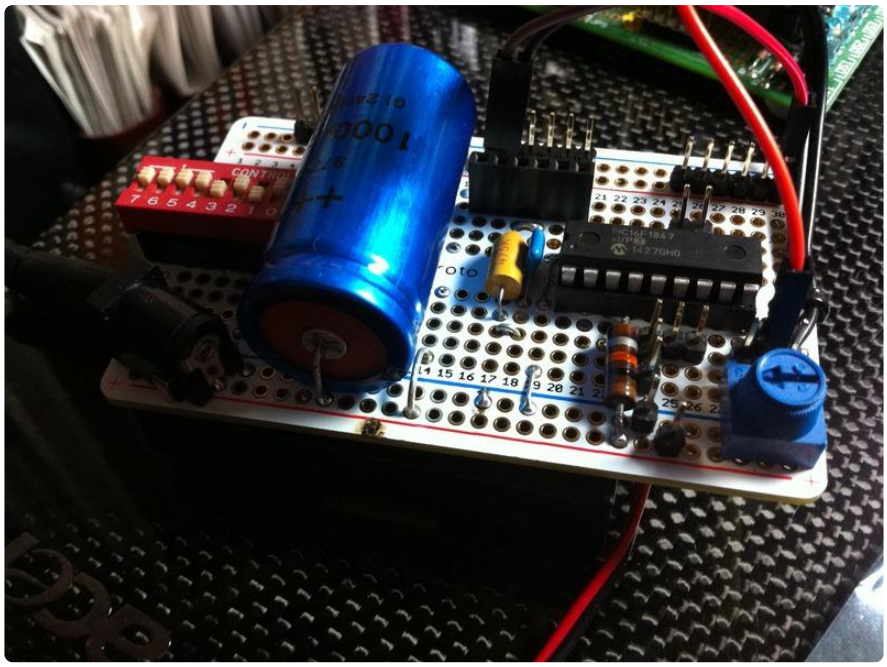
Well, that bit of test code met the immediate need and looked pretty but tended to become a little monotonous after a while. What if I coded this or maybe that instead...and the slippery slope suddenly became very slopey. And away we went...

So, jus' thinkin': it might be nicer if there was a variety of display styles it could generate. And of course it must have a wide palette of colors. And brightness variations. And it would be cool if it could somehow surprise you once in a while. How do I wedge all of that into my little PIC? Grab an idea, grease it up, and get a good running start...

You can follow along with your own PIC16F series with this simple design layout:



I thought it might be generally helpful if I included an inrush capacitor along with the ability to select (from a set of possiblities) how many NeoPixels it drives (another feature!). This became this year's Pumpkin Illuminator:



It seems quite happy to run an 8x8 NeoPixel panel for hours on 3 AA's:

And so it begins with a few basic ideas...

Algorithms, meet Random. Let's Play!

In simplest terms my tools were 24 bits of RGB in the NeoPixel itself, and a little bit of time between each redraw of the pixel colors. I wanted something that would be interesting to watch and, while operating within a necessarily fixed set of capabilities, could make you wonder if you really had seen exactly the same thing a bit ago...

There are 7 basic styles of display in the PICsellator:

- Alternating Foreground/Background Floods
- "Fib Rand Sine"
- Strobies in a Flood Field
- Intensity Ramp
- Sine Springboard
- Color Component Leach
- Strobies Leaving a Random Terminal Color

Each style might alone show something interesting but it became much more appealing when they were folded and blended together randomly! We mix in some random determination of colors, sine function parameters, Fibonacci number selection, delays, and deciding if some styles start afresh with a new (random) color

blend or use what was left from the previously run style. Oh, and some styles randomly determine at which end of the NeoPixel sequence they start their work.

Here are the basic descriptions of each of the PICsellator's display styles and some notes on where some decisions dip into the randomness bucket:

Alternating Foregound/Background Floods

A PICsellator "flood" is the filling of a selected number (the "stride") of NeoPixels with a color followed by a single instance of a (possibly different) color. This style randomly chooses foreground color, stride, and background color values for two separate floods then alternates between the two floods for a random amount of time with randomly chosen hold times during both the first and second floods.

"Fib Rand Sine"

This display style fills the selected number of pixels with the results of parameterized sine function for each of the red, green, and blue color components. Each of the RGB value functions is parameterized with an amplitude offset, a max amplitude, a phase shift, and a sine function argument multiplicand and divisor. Presently the following parameters are randomly chosen: each color channel's max amplitude and the sine function argument's multiplicand and divisor. The sine function argument's multiplicand is selected from a set of compiled-in values (thank you, math.h!) while the divisor is selected from a set of the first few Fibonacci numbers.

Strobies in a Flood Field

PICsellator "strobies" are short-duration bursts of a relatively high-intensity random color. This display style pops off a random number of strobies in either a randomly selected flood or what was left from a previous style. The pixel's color after the flash is randomly chosen to be either the original pre-flash color or dark.

Intensity Ramp

This style begins with a color created from the random presence or absence of each of red, green, and blue (i.e., a "binary RGB" color). It randomly chooses to "start low" or "start high" and then increases or decreases the intensity of the color at a random rate until reaching a randomly chosen stopping point.

Sine Springboard

This display style is similar to Fib Rand Sine. However, instead of Red, Green, and Blue operating independently the Green and Blue max amplitudes are based on Red's

with an additional random adjustment. Also, the sine function's multiplicand is fixed at pi rather than being chosen from a set of numbers. Subtle, but weird - always a good plan! ;-)

Color Component Leach

This display style is somewhat similar to the Intensity Ramp but with some interesting variations. First, it chooses a blend of red, green, and blue and determines if it's to accrete toward, or leach away from, that color. It then does so by adding/subtracting the red, green, and blue components at a random rate until it reaches the target color (either the originally chosen blend or all-off, respectively).

Strobies Leaving a Random Terminal Color

Candy! This style is similar to the previously mentioned Strobies style but adds the selection of a random color to be left at a pixel following the strobie flash.

Whew!

How was all this wedged into the tiny little PIC? "Very carefully" and "over many iterations" are probably the best answers as the code sloshed back and forth from "takes too much RAM" to "not enough Program Memory" (and sometimes both)...

Software: set some bits, clear some bits: stuff happens

Anytime I immerse myself in a project like this I come away with a newfound respect for the sheer genius of the Apollo-era NASA engineers. To the moon - and back! Having far, far less with which to work!

Much of the code below had been pounded heavily to fit in the PIC's RAM. A few things have been re-re-re-re-written to use less program memory but I think one of the biggest challenges came from holding firm to the requirement to support up to 256 NeoPixels. Clinging (sometimes desperately!) to that capability colored how most of the algorithms were finally implemented.

Alternating Foreground/Background Floods

```
{   /* style 1: alternating foreground/background floods */
  FloodParams_t*  pFlood1;
```

```
  FloodParams_t*  pFlood2;


  pFlood1 = (SCALED_RAND( 2 ) == 0) ? (FloodParams_t*) &amp;FloodAllOff :
&amp;Flood1;


  /* If I'm to use them, fill in the first set of flood parameters. */

  if( pFlood1 != &amp;FloodAllOff )
    {

      SetFloodParams( pFlood1 /* pFloodParams */,
                      0 /* FirstPixel */,
                      NumPixelsToDrive - 1 /* LastPixel */,
                      SCALED_RAND( 19 ) /* ForegroundRed */,
                      SCALED_RAND( 19 ) /* ForegroundGreen */,
                      SCALED_RAND( 19 ) /* ForegroundBlue */,
                      RANGED_RAND( 3, 17 ) /* Stride */,
                      SCALED_RAND( 3 ) /* BackgroundRed */,
                      SCALED_RAND( 3 ) /* BackgroundGreen */,
                      SCALED_RAND( 3 ) /* BackgroundBlue */ );


      pFlood2 = (SCALED_RAND( 2 ) == 0) ? (FloodParams_t*) &amp;FloodAllOff :
&amp;Flood2;


    }
  else   /* pFlood1 calls for "all off"... */
    {

      pFlood2 = &amp;Flood2;


    }   /* close if( pFlood1 ... */


  /* If I'm to use them, fill in the second set of flood parameters. */

  if( pFlood2 != &amp;FloodAllOff )
    {

      SetFloodParams( pFlood2 /* pFloodParams */,
                      1 /* FirstPixel */,
                      NumPixelsToDrive - 1 /* LastPixel */,
                      SCALED_RAND( 2 ) /* ForegroundRed */,
                      SCALED_RAND( 2 ) /* ForegroundGreen */,
                      SCALED_RAND( 2 ) /* ForegroundBlue */,
                      RANGED_RAND( 3, 17 ) /* Stride */,
                      SCALED_RAND( 8 ) /* BackgroundRed */,
                      SCALED_RAND( 8 ) /* BackgroundGreen */,
                      SCALED_RAND( 8 ) /* BackgroundBlue */ );


    }   /* close if( pFlood2 ... */


  Alternate( pFlood1 /* pForeFlood */,
             RANGED_RAND( 17, 377 ) /* ForeHold_mSec */,
             pFlood2 /* pBackFlood */,
             RANGED_RAND( 40, 730 ) /* BackHold_mSec */,
             RANGED_RAND( 793, 4076 ) /* Duration_mSec */ );
```

```
}
```

"Fib Rand Sine"

```
{   /* style 2: Fib Rand Sine (with hat tip to Auld Lang Syne) */

  SetFloodParams( &amp;Flood1 /* pFloodParams */,
                  0 /* FirstPixel */,
                  NumPixelsToDrive - 1 /* LastPixel */,
                  0 /* ForegroundRed */,
                  0 /* ForegroundGreen */,
                  0 /* ForegroundBlue */,
                  0 /* Stride */,
                  0 /* BackgroundRed */,
                  0 /* BackgroundGreen */,
                  0 /* BackgroundBlue */ );


  Red_AmplitudeOffset = 0;

  Red_MaxAmplitude = RANGED_RAND( 1, 12 );

  Red_PhaseShift = 0.0;

  Red_ArgMultiplicand = Factors[ SCALED_RAND( NUMELEMS( Factors ) ) ];

  Red_ArgDivisor = Fib[ RANGED_RAND( 6, NUMELEMS( Fib ) ) ];


  Green_AmplitudeOffset = 0;

  Green_MaxAmplitude = RANGED_RAND( 1, 12 );

  Green_PhaseShift = 0.0;

  Green_ArgMultiplicand = Factors[ SCALED_RAND( NUMELEMS( Factors ) ) ];

  Green_ArgDivisor = Fib[ RANGED_RAND( 6, NUMELEMS( Fib ) ) ];


  Blue_AmplitudeOffset = 0;

  Blue_MaxAmplitude = RANGED_RAND( 1, 12 );

  Blue_PhaseShift = 0.0;

  Blue_ArgMultiplicand = Factors[ SCALED_RAND( NUMELEMS( Factors ) ) ];

  Blue_ArgDivisor = Fib[ RANGED_RAND( 6, NUMELEMS( Fib ) ) ];


  FloodRgbFunc( &amp;Flood1 /* pFloodParams */ );

}
```

Strobies in a Flood Field

```
{   /* style 3: strobies in a flood field (or whatever is lingering). */

  SetFloodParams( &amp;amp;Flood1 /* pFloodParams */,
                  0 /* FirstPixel */,
                  NumPixelsToDrive - 1 /* LastPixel */,
                  RANGED_RAND( 0, 3 ) /* ForegroundRed */,
                  RANGED_RAND( 1, 3 ) /* ForegroundGreen */,
                  RANGED_RAND( 0, 3 ) /* ForegroundBlue */,
                  RANGED_RAND( 50, 200 ) /* Stride/NumStrobies */,
                  RANGED_RAND( 70, 137 ) /* BackgroundRed */,
                  RANGED_RAND( 70, 137 ) /* BackgroundGreen */,
                  RANGED_RAND( 70, 137 ) /* BackgroundBlue */ );


  Strobies( &amp;amp;Flood1 );


  Delay_mSec( RANGED_RAND( 400, 1200 ) /* Milliseconds */ );

}
```

Intensity Ramp

```
{   /* style 4: intensity ramp */

  /* Select a color and ramp it up or down. */

  Red = RANGED_RAND( 0, 1 );

  Green = RANGED_RAND( 0, 1 );

  Blue = RANGED_RAND( 0, 1 );


  /* Disallow "all-off": bail and just make it BlueGreen. */

  if( (Red == 0) &amp;amp;&amp;amp;
      (Green == 0) &amp;amp;&amp;amp;
      (Blue == 0) )
    {

      Green = 1;

      Blue = 1;


    }   /* close if( (Red ... */


  RampUp = (SCALED_RAND( 2 ) == 0) ? TRUE : FALSE;


  NumSteps = RANGED_RAND( 10, 27 );
```

```
  NumSteps += 1;   /* Ensure the limit (either the maximum intensity or 0) is
rendered. */


  RampDelay_mSec = RANGED_RAND( 30, 170 );


  Intensity = RampUp ? 0 : NumSteps;   /* Set the intensity starting point (high or
low). */


  for( Step = 0; Step &lt; NumSteps; Step++ )
    {

      SetFloodParams( &amp;Flood1 /* pFloodParams */,
                      0 /* FirstPixel */,
                      NumPixelsToDrive - 1 /* LastPixel */,
                      Red * Intensity /* ForegroundRed */,
                      Green * Intensity /* ForegroundGreen */,
                      Blue * Intensity /* ForegroundBlue */,
                      0 /* Stride/NumStrobies */,
                      0 /* BackgroundRed */,
                      0 /* BackgroundGreen */,
                      0 /* BackgroundBlue */ );


      Flood( &amp;Flood1 );


      Delay_mSec( RampDelay_mSec /* Milliseconds */ );


      /* Adjust the intensity according to the ramp direction. */

      Intensity = RampUp ? (Intensity + 1) : (Intensity - 1);


    }   /* close for( Step ... */


  Delay_mSec( 150 );

}
```

Sine Springboard

```
{   /* style 5: sine springboard */

  SetFloodParams( &amp;Flood1 /* pFloodParams */,
                  0 /* FirstPixel */,
                  NumPixelsToDrive - 1 /* LastPixel */,
                  0 /* ForegroundRed */,
                  0 /* ForegroundGreen */,
                  0 /* ForegroundBlue */,
                  0 /* Stride */,
                  0 /* BackgroundRed */,
                  0 /* BackgroundGreen */,
```

```
                    0 /* BackgroundBlue */ );

  Red_AmplitudeOffset = 0;
  Red_MaxAmplitude = RANGED_RAND( 1, 7 );
  Red_PhaseShift = 0.0;
  Red_ArgMultiplicand = M_PI;
  Red_ArgDivisor = Fib[ RANGED_RAND( 6, NUMELEMS( Fib ) ) ];

  Green_AmplitudeOffset = 0;
  Green_MaxAmplitude = Red_MaxAmplitude + RANGED_RAND( 0, 4 );
  Green_PhaseShift = 0.0;
  Green_ArgMultiplicand = M_PI;
  Green_ArgDivisor = Fib[ RANGED_RAND( 6, NUMELEMS( Fib ) ) ];

  Blue_AmplitudeOffset = 0;
  Blue_MaxAmplitude = Red_MaxAmplitude + RANGED_RAND( 0, 7 );
  Blue_PhaseShift = 0.0;
  Blue_ArgMultiplicand = M_PI;
  Blue_ArgDivisor = Fib[ RANGED_RAND( 6, NUMELEMS( Fib ) ) ];

  FloodRgbFunc( &amp;Flood1 /* pFloodParams */ );

}
```

Color Component Leach

```
{   /* style 6: color component leach */
  uint8_t  RedCeiling;
  uint8_t  GreenCeiling;
  uint8_t  BlueCeiling;

  /* Select a color and accrete up or leach down. */
  RedCeiling = RANGED_RAND( 0, 21 );
  GreenCeiling = RANGED_RAND( 0, 21 );
  BlueCeiling = RANGED_RAND( 0, 21 );
```

```
  /* Disallow "all-off": bail and just make it some nice violet. */

  if( (Red == 0) &amp;amp;&amp;amp;
      (Green == 0) &amp;amp;&amp;amp;
      (Blue == 0) )
    {

      RedCeiling = RANGED_RAND( 1, 21 );

      BlueCeiling = RANGED_RAND( 1, 21 );


    }   /* close if( (Red ... */


  RampUp = (SCALED_RAND( 2 ) == 0) ? TRUE : FALSE;



  NumSteps = max( RedCeiling, GreenCeiling );

  NumSteps = max( NumSteps, BlueCeiling );



  NumSteps += 1;   /* Ensure the limit (either the maximum ceiling or 0) is
rendered. */



  RampDelay_mSec = RANGED_RAND( 30, 170 );



  /* Set the starting points based on if I'm accreting up or leaching down. */

  if( RampUp )
    {

      Red = 0;

      Green = 0;

      Blue = 0;


    }
  else   /* going down... */
    {

      Red = RedCeiling;

      Green = GreenCeiling;

      Blue = BlueCeiling;


    }   /* close if( RampUp ... else */



  for( Step = 0; Step &amp;lt; NumSteps; Step++ )
    {

      SetFloodParams( &amp;amp;Flood1 /* pFloodParams */,
                      0 /* FirstPixel */,
```

```
                      NumPixelsToDrive - 1 /* LastPixel */,
                      Red /* ForegroundRed */,
                      Green /* ForegroundGreen */,
                      Blue /* ForegroundBlue */,
                      0 /* Stride/NumStrobies */,
                      0 /* BackgroundRed */,
                      0 /* BackgroundGreen */,
                      0 /* BackgroundBlue */ );


      Flood( &amp;Flood1 );


      Delay_mSec( RampDelay_mSec /* Milliseconds */ );


      /* Adjust the colors according to the ramp direction. */

      if( RampUp )
        {

          Red = (Red &lt; RedCeiling) ? Red + 1 : Red;

          Green = (Green &lt; GreenCeiling) ? Green + 1 : Green;

          Blue = (Blue &lt; BlueCeiling) ? Blue + 1 : Blue;


        }
      else   /* going down... */
        {

          Red = (Red &gt; 0) ? Red - 1 : 0;

          Green = (Green &gt; 0) ? Green - 1 : 0;

          Blue = (Blue &gt; 0) ? Blue - 1 : 0;


        }   /* close if( RampUp ... else */


    }   /* close for( Step ... */


  Delay_mSec( 376 /* Milliseconds */ );


}
```

Strobies Leaving a Random Terminal Color

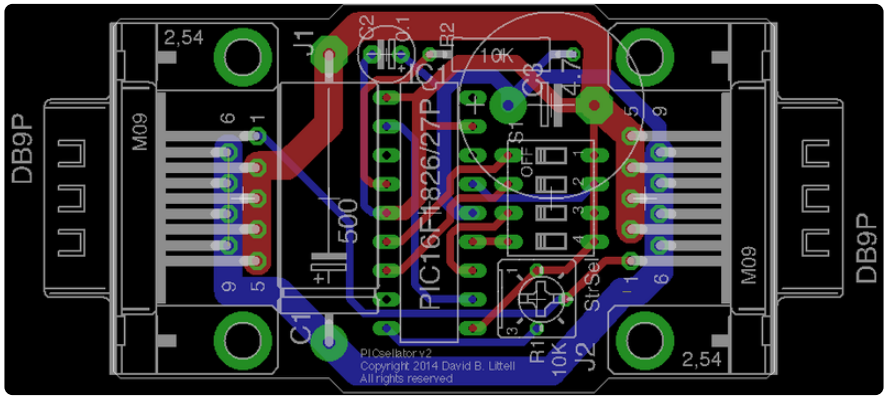
```
{   /* style 7: strobies in a flood field (or whatever is lingering) leaving a
random terminal color -
     * candy!.
     *
     ***/

  SetFloodParams( &amp;Flood1 /* pFloodParams */,
                  0 /* FirstPixel */,
                  NumPixelsToDrive - 1 /* LastPixel */,
                  0 /* ForegroundRed */,
```

```
                    0 /* ForegroundGreen */,
                    0 /* ForegroundBlue */,
                    RANGED_RAND( 100, 250 ) /* Stride/NumStrobies */,
                    RANGED_RAND( 70, 137 ) /* BackgroundRed */,
                    RANGED_RAND( 70, 137 ) /* BackgroundGreen */,
                    RANGED_RAND( 70, 137 ) /* BackgroundBlue */ );


  Strobies( &amp;Flood1 );


  Delay_mSec( RANGED_RAND( 700, 1400 ) /* Milliseconds */ );

}
```

Hardware Design

Since it looked I could wring a decent amount of cool out of the PIC I then wondered what the hardware might look like if I tried to package it. Because the PIC needs hardly any popcorn (i.e., external resistors & capacitors) it seemed like the "biggest" sizing challenges would be connectors able to safely carry the current needed for long-ish strands and the obligatory inrush capacitor. I also wanted to be able to easily change the number of NeoPixels for which it would generate the bitstream, but without having to re-flash the firmware. That translated into some kind of data-entry mechanism. And it has to have a "heat knob thingy"...to dial back the "whiteout" effect yet still be able to get decent leading edges on the WS2812 bitstream pulses across a range of V+ values.

Below is the board in its present form. I'm sure there're lots of improvements that can be made but this is my "first shot" using EAGLE...and I really am just a software guy... ;-)

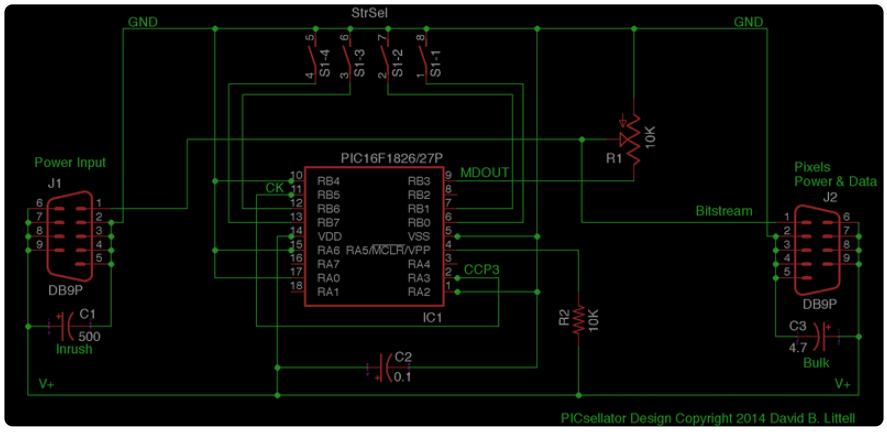


KiCAD versus EAGLE

KiCAD worked very well for designing Madison's clock () but I remembered that hunting down (or creating) device packages for layout was much more of a TimeSponge than I would have preferred. I remembered that EAGLE supposedly had tons of packages, so I thought I could quickly/easily put together something reasonable. I momentarily forgot a crucial lesson: shortcuts aren't. To me, KiCAD seems to have an design flow that wants you to first create a schematic and only associate packages to schematic entities when the doing board layout. Seems quite reasonable, "feels" more natural, (and pushes the icky package-hunting TimeSponge downstream - a good thing, I think) but my first dip into the EAGLE pool appeared to require that flow to be completely flipped.

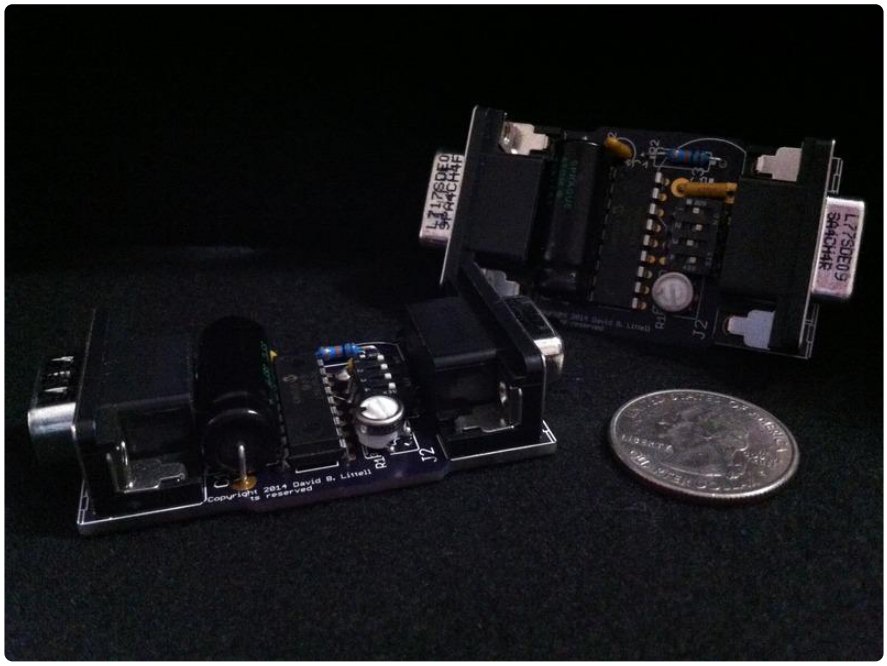
It's quite possible my own inexperience with these tools tricks me (I really am just a software guy...promise!) but EAGLE seemed to require that I select a package before I could even drop a part on the schematic. Hm. Okay, I guess. But is this painful enough right off the bat to not even try EAGLE and stick with KiCAD? No, it just slapped me with the package-hunting TimeSponge much earlier than I expected.

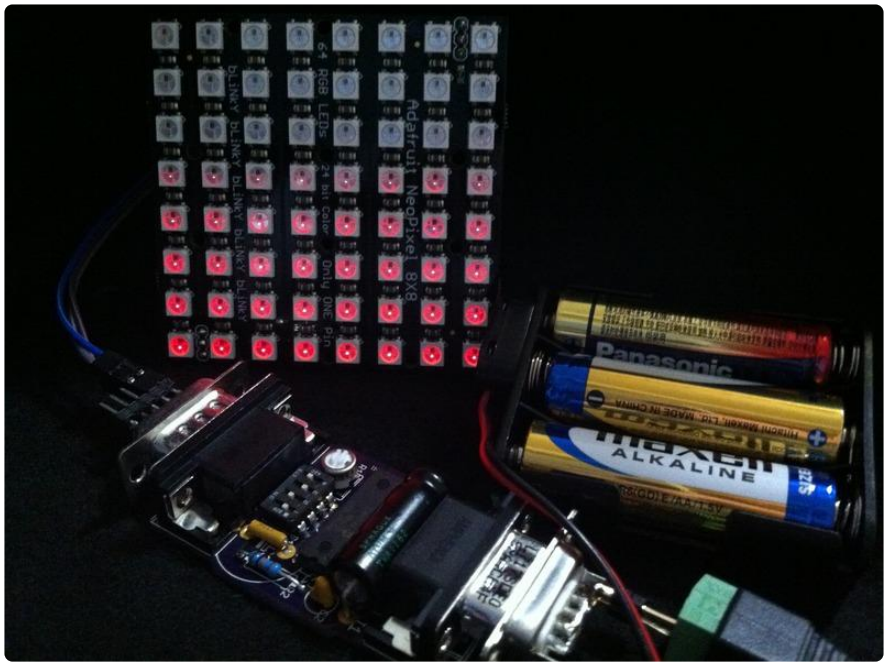
The lesson there was to be careful for what you wish - EAGLE has a unbelievable number and variety of packages for a vast number of devices. But the "challenge" becomes sorting through all the heavily coded and yet vanishingly miniscule information (EAGLE library uber-arcane "geek-speak") to find the right package. That, in itself, became the EAGLE TimeSponge. And I got to wade through that swamp before I could even do a schematic.



Please note that the PIC used is actually a PIC16F1847. EAGLE didn't have a package for an '1847 but thankfully it did for the pin-compatible '1827. Otherwise, I'd probably s till be trying to figure out how to bumble through generating (or even just copy-n-edit) an '1847-specific package...

Update 20141106: I received the first boards from OSH Park and they seem to work! Here are some glamour and action shots:





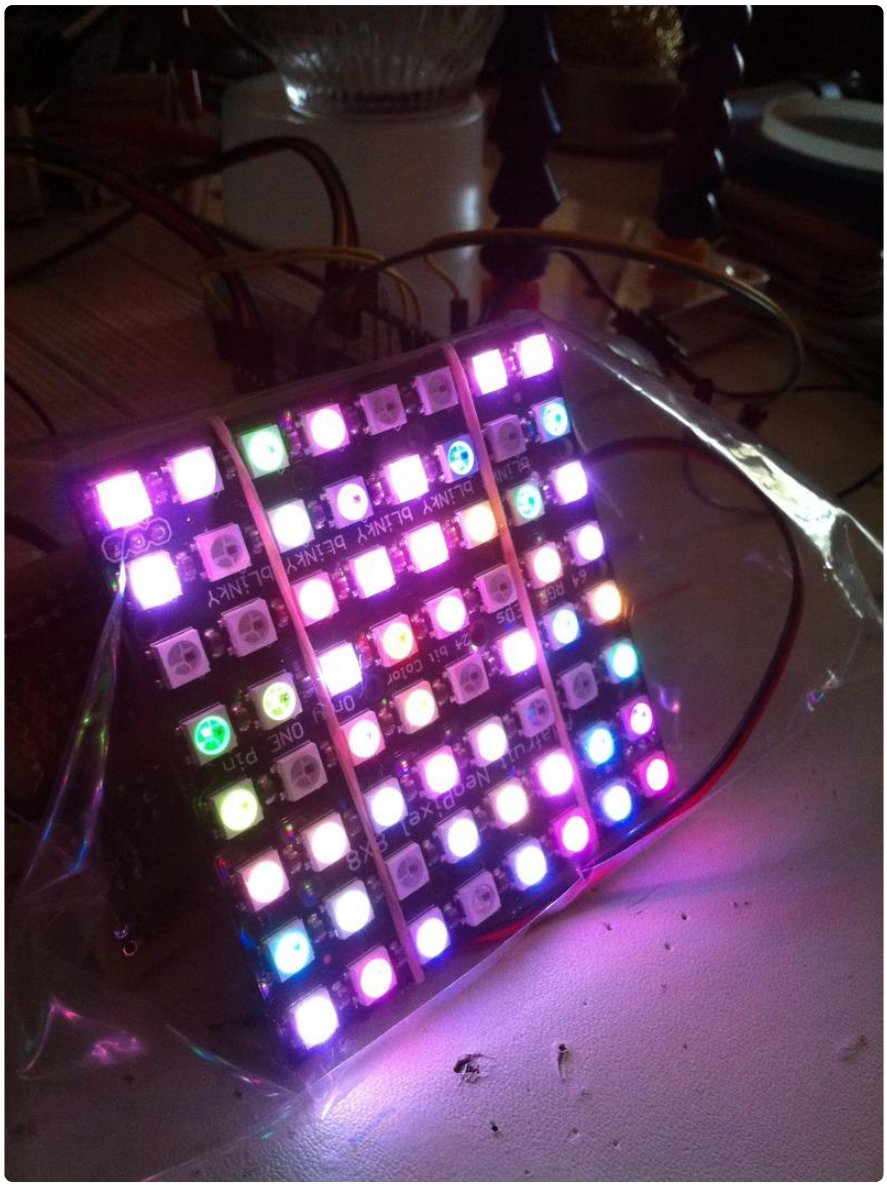
Lightcasting

One idea using a PICsellator, a 144-pixel strand, a glass bowl, and some parchment paper, reminds me of a candy dish:

...and if I flip the bowl and use a 24- and 12-pixel ring I see:

This year's pumpkin illuminator, using a prototype PICsellator, 3 AA batteries, an 8x8 NeoPixel panel, and a sandwich bag:


Continental Specialties Corporation, 44 Kendall St., New Haven, Conn



The NeoLavaPixelLamp, made with a 144-pixel strand, a couple of Crackled Glass Votives, and some parchment paper:

What will it do next?

Here's a seasonal idea: a nice poofy wig, a few strands of NeoPixels wound in and a PICsellator for each strand? NeoFalls! ()