Madison's NeoClock - A PIC + KiCAD + NeoPixel adventure

Created by David Littell

https://learn.adafruit.com/madisons-neoclock-a-pic-kicad-neopixel-adventure

Last updated on 2023-08-29 02:36:30 PM EDT

Table of Contents

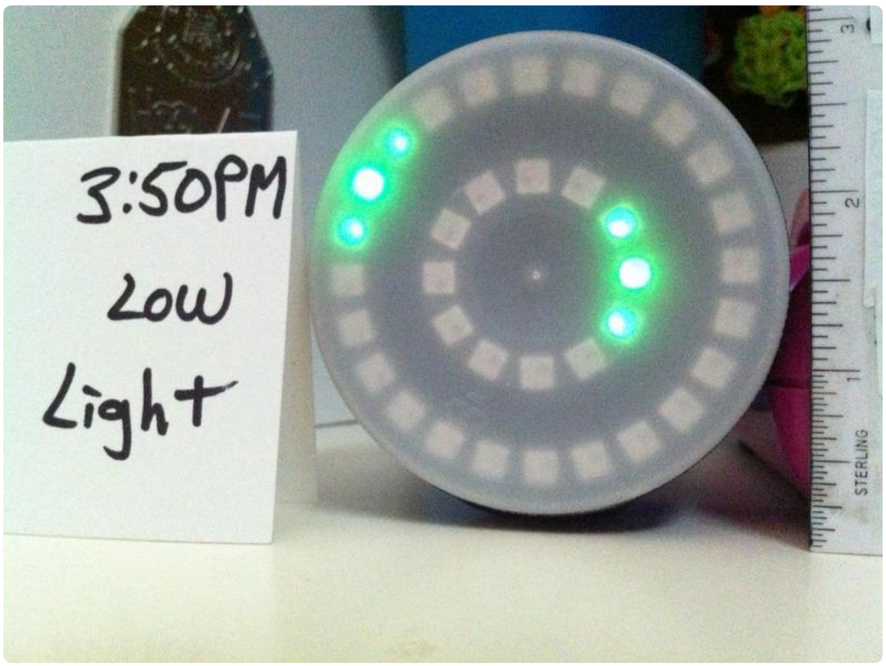
The Backstory

My name is David Littell. I'm an embedded systems engineer with about 35 years experience with microprocessors, computer hardware, software, and systems of all sizes. I was one of "those" kids that was constantly taking apart (and sometimes even fixing!) all things electronic. I grew up spending my allowance at Radio Shack for kits, books, and parts and my nights with fingers glued to shortwave radios. At age 15 I built a Processor Technology Sol-20 from the kit - one of the only kids in my home state to have a computer (in 1978). It worked at first power-up. (I still have it and it (probably) still works - resurrection is yet another item on the bucket list.)

Fast-forward a few decades and I now have a daughter that loves when Daddy builds things for her. I had noticed the NeoPixels at Adafruit and thought they'd be interesting to work with, so I ordered a NeoPixel Ring-12 and started studying the Data Sheet. Slippery slope, that...

One idea led to another led to an avalanche of "Hey! This might actually work! What about this..." and the next thing I knew I was learning KiCad to design my first board to drive NeoPixel rings to make a desk clock for my daughter. Enter Madison's NeoClock:

Well, these are the famed NeoPixels so I wanted to use color in a sane but interesting way.

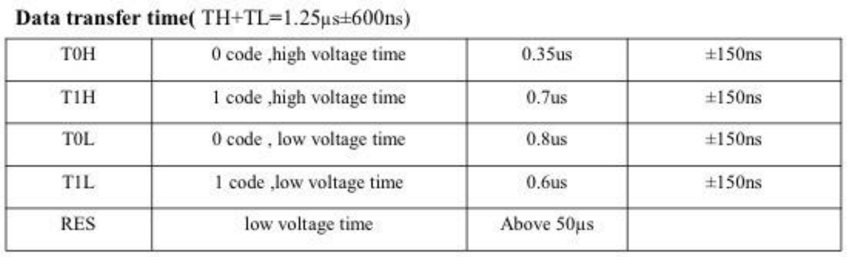


And wow will they light up a room at night! Sleeping in sunglasses isn't all that comfortable, so what can be done about that? Should it throttle back based on only time or can it somehow know the brightness around itself?

Where to begin? Where the rubber meets the road, I think: creating a bitstream for the NeoPixels. There are many options that work well but I already had a few Microchip PIC's that I'd been using in other projects. Can this do the job? Time to visit those WS2812 and PIC Data Sheets...

Rubber, meet Road

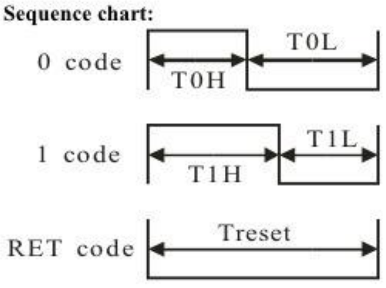
The tough part about using NeoPixels is the 1-wire control protocol. Frankly, looking at the WS2812 Data Sheet for the first time gave me the creeps!



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

T0H	0 code ,high voltage time	0.35us	±150ns
T1H	1 code ,high voltage time	0.7us	±150ns
T0L	0 code , low voltage time	0.8us	±150ns
T1L	1 code ,low voltage time	0.6us	±150ns
RES	low voltage time	Above 50µs	

Sequence chart:



I like my protocols & interfacing simple. This ain't quite what I had in mind. So what now?

OK, if you stand right here and squint like this and the light is just right...there it is! There is a way to push a little here, pull a little there, and get the "0-code" and "1-code" overall durations to match while still having reasonable T0H and T1H times. That's what I needed because I'm using one of the PIC CCP (Capture Compare PWM) modules in PWM mode to create 0's and an additional CCP in PWM mode to create 1's. The CCP's in PWM mode have some constraints and this (from the PIC's Data Sheet) is pretty much all I know:

FIGURE 24-3: CCP PWM OUTPUT SIGNAL

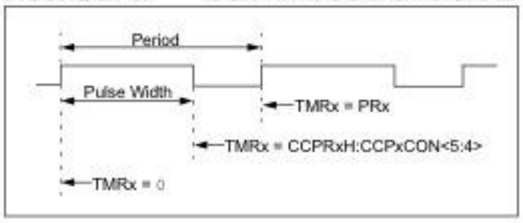
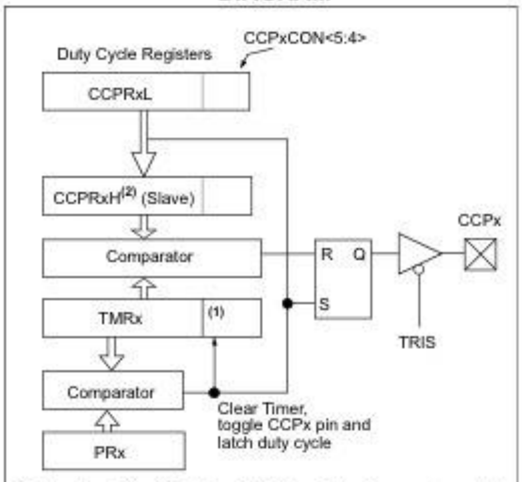


FIGURE 24-4: SIMPLIFIED PWM BLOCK DIAGRAM

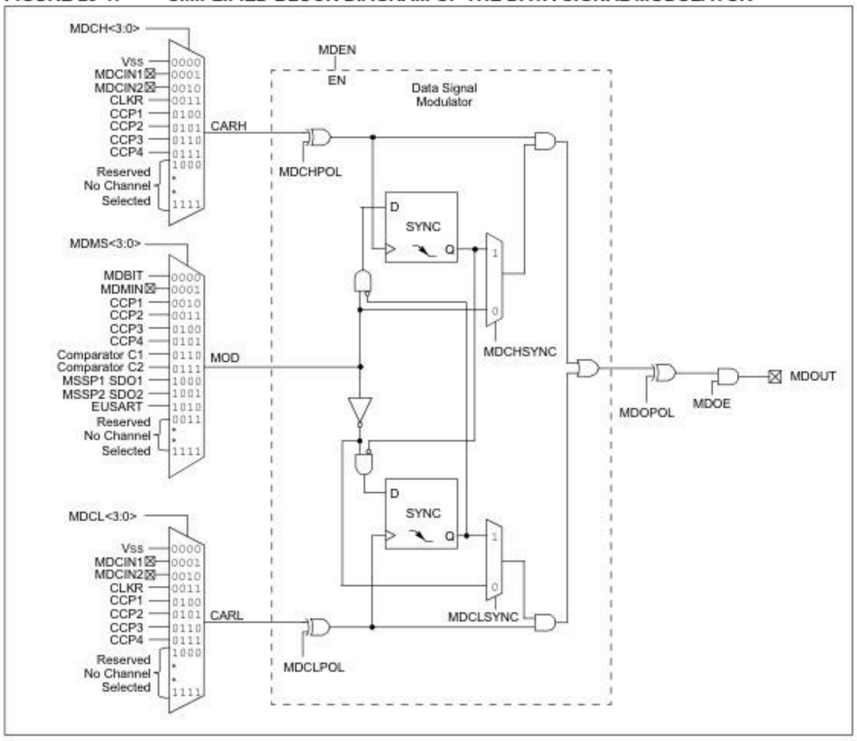


Note 1: The 8-bit timer TMRx register is concatenated with the 2-bit internal system clock (FOSC), or two bits of the prescaler, to create the 10-bit time base.

2: In PWM mode, CCPRxH is a read-only register.

Ha! Now I have a constant stream of 0's and a constant stream of 1's. Great. Full on or full off - take your pick. However, I want this clock to be colorful which means I need 0's and 1's intermixed. To do that, there's a wicked-cool little spoonfull of voodoo in the PIC's bag of tricks - the Data Signal Modulator (DSM):

FIGURE 23-1: SIMPLIFIED BLOCK DIAGRAM OF THE DATA SIGNAL MODULATOR



Now this is cool: one "carrier" is the PWM stream of 0 bits and the other "carrier" is the PWM stream of 1 bits. The "modulated data" will be the output of one of the MSSP's in SPI mode. This is coming together!

Then Reality comes stomping in and kicks your nice little Lego house all over the room: the MSSP in SPI mode simply will not be convinced to output bytes back-to-back. There's no way around this little 1 clock lapse between bytes transmitted by the SPI hardware. Well, that just throws a big ol' monkey wrench into my plan...

So instead, I'll try something else - pushing data through bit-by-bit via the handy MDBIT input to the DSM.

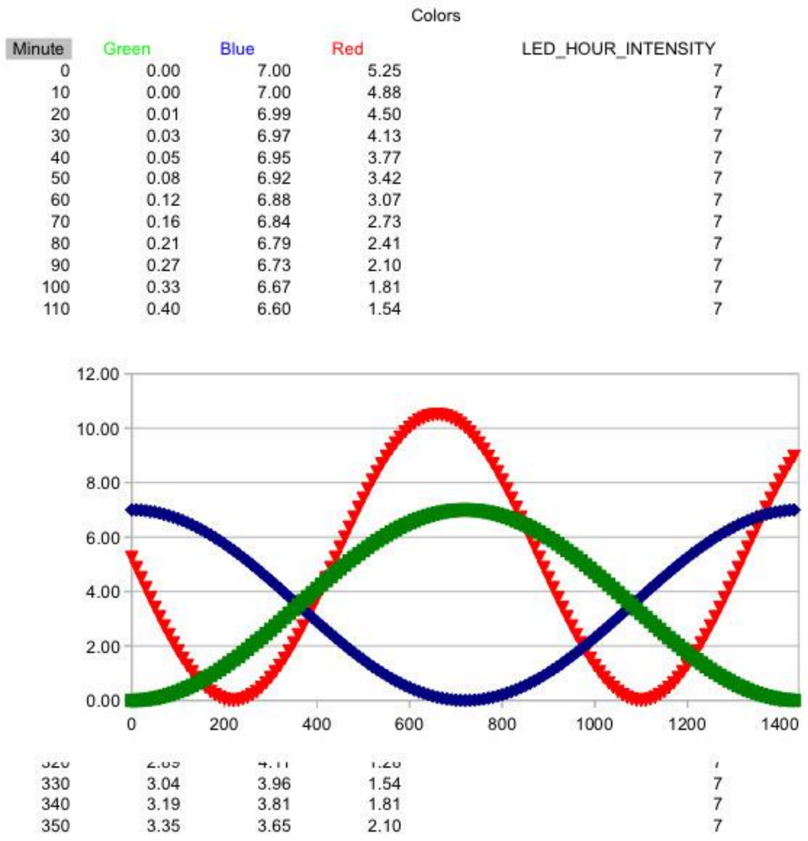
That's where this portion of the adventure ended up - with a little help from the interrupt-on-change inputs so the code can know exactly when to flip MDBIT to a 0 or a 1 to keep the output bitstream coherent. It's not as fire-and-forget as SPI but I did indeed learn a lot while writing (and rewriting and rewriting...) those 400 lines of PIC assembly. WooHoo FUN!

So now I can finally talk to this thing - what do I want to tell it?

Eschew Pallor

NeoPixels are all about color - beautiful, vibrant, in some cases retina-scorching, color. But this project is supposed to be a clock, not a beacon for alien spacecraft entering orbit. I want a color palette that's interesting over a 24-hour period but not extreme or overwhelming.

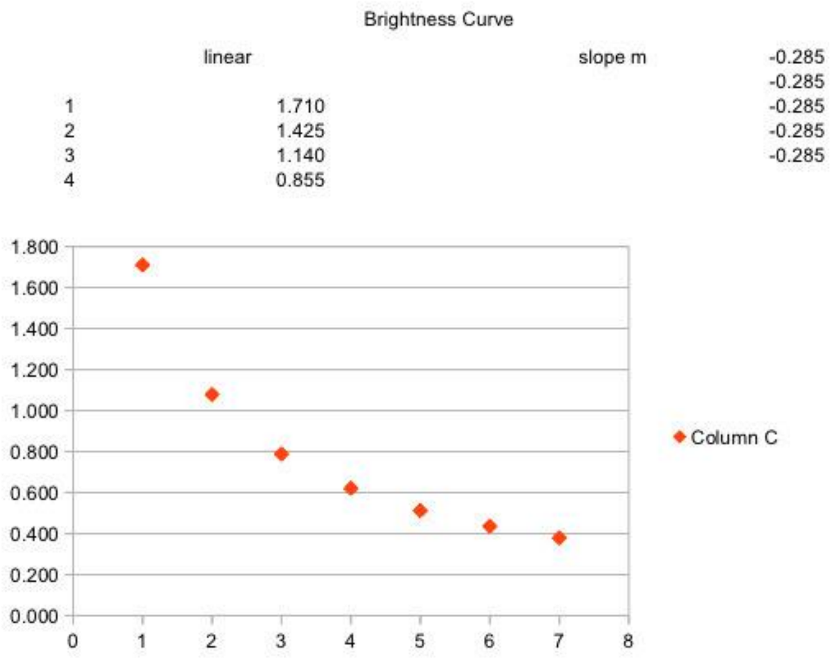
It's like they always say: "There's a sinusoid for that!" (Well, maybe not always...) After way too much geekiness with a spreadsheet I settled on something like this:



Colors

Minute	Green	Blue	Red	LED_HOUR_INTENSITY
0	0.00	7.00	5.25	7
10	0.00	7.00	4.88	7
20	0.01	6.99	4.50	7
30	0.03	6.97	4.13	7
40	0.05	6.95	3.77	7
50	0.08	6.92	3.42	7
60	0.12	6.88	3.07	7
70	0.16	6.84	2.73	7
80	0.21	6.79	2.41	7
90	0.27	6.73	2.10	7
100	0.33	6.67	1.81	7
110	0.40	6.60	1.54	7
330	3.04	3.96	1.54	7
340	3.19	3.81	1.81	7
350	3.35	3.65	2.10	7

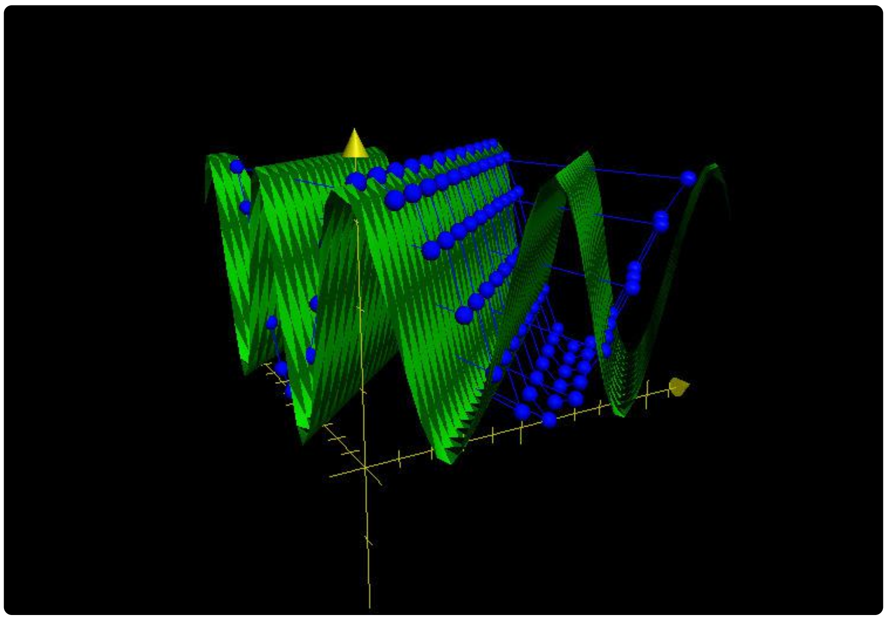
One "special effect" I was interested in implementing was a tapering before and after the brightest point (which indicates the actual time on the clock). A number of false starts and bad ideas later I ended up with something like this:



Brightness Curve

	linear	slope m	-0.285
			-0.285
1	1.710		-0.285
2	1.425		-0.285
3	1.140		-0.285
4	0.855		

But not before having too much fun with the Mac's Grapher (which I now know I am nowhere near smart enough to actually use...):



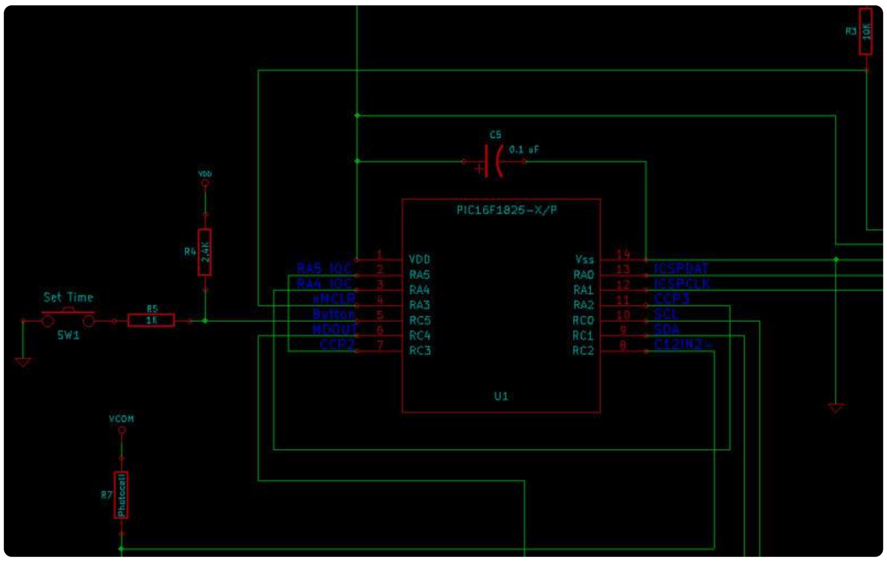
Using the NeoPixel ring-12 to represent hours and the ring-24 to represent minutes was OK, but I wanted a little something extra (feature creep alert!) for seconds. It had to be different from both the hours and the minutes representation and still be (relatively) obvious. Enter the white chasing taper. A white dot makes the round of the ring-12 each second and is "chased" by a few more tapering-intensity white dots. All that doesn't disturb (too much) the ring-12's hour display and its own tapering.

Connect the Dots

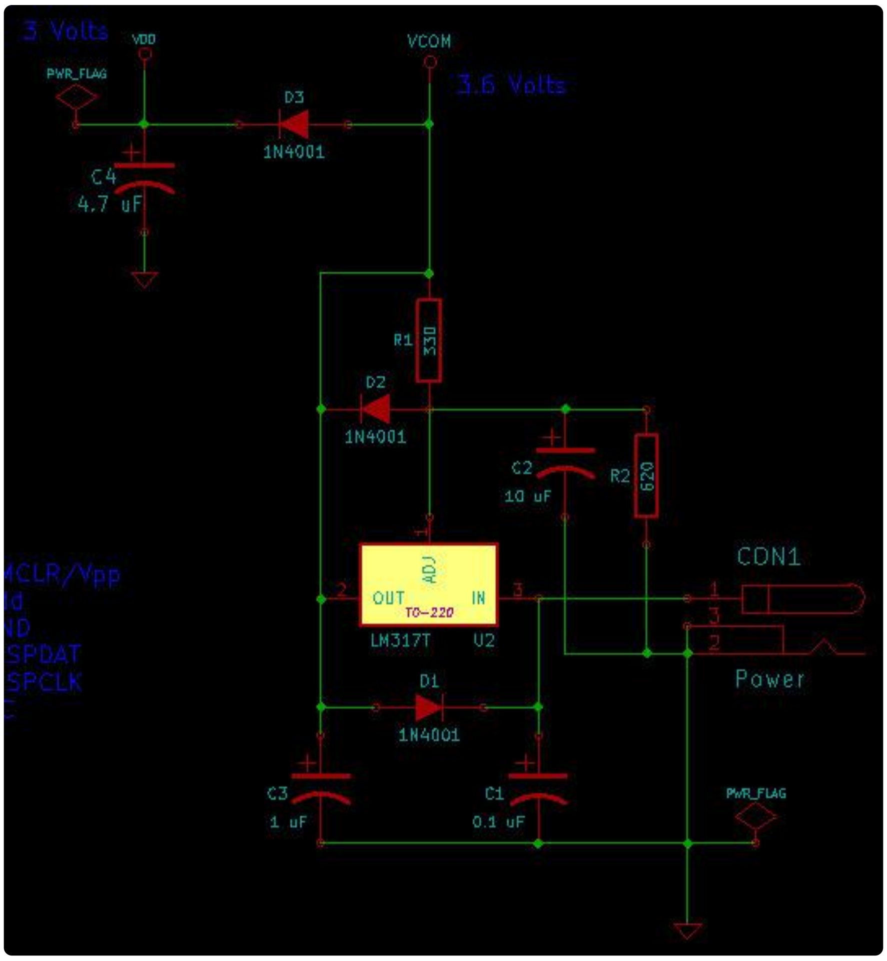
After adding the ability to sense ambient light so it can calm itself when the light is dim I finally realized that the clock's timing was drifting all over the place. The ultimate answer, of course, was more cowbell! Juggling pins on the PIC freed up the I2C signals for the newly added ChronoDot and Frankenstein's monster was (nearly) complete.

Sub-adventure 37a in this saga required learning KiCad to get the schematic onto something other than taped-together scribble sheets, a visit to Dave's Hack-a-matic Board Layout Shack, and my first shot at a getting multiple boards to fit together.

The overall circuitry is straightforward as most of the magic is in the software and the stitching together of the various PIC modules:

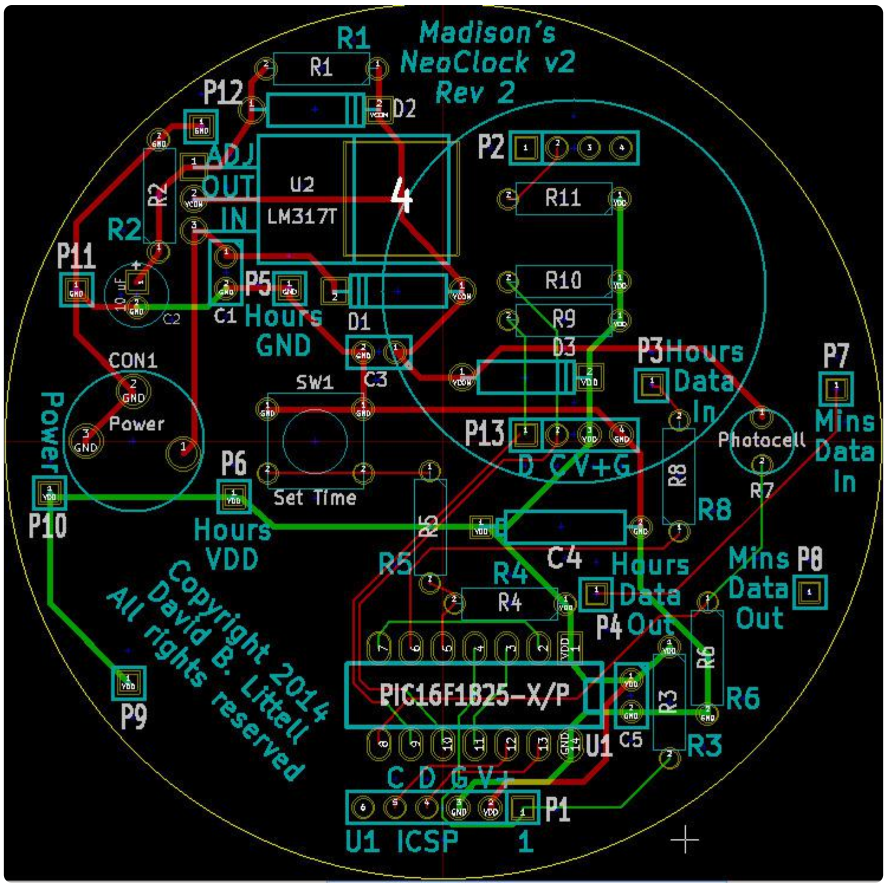


The power section was interesting as I tried to incorporate the recommendations from the ST LM317 Data Sheet and lessons learned from working with some really good hardware designers. Mistakes are mine alone - but it hasn't smoked yet!



Hard Times

After all that (and then some) I still needed to make a physical printed circuit board. That meant more fun with KiCad, which I do like pretty well. I spent a lot of time hunting down component parts and footprints and such - definitely on my faves list. But I somehow managed to produce something that OSH Park could manufacture:



I'm sure some layout guys are busily sharpening their knives and I know there's much, much, much I don't know. My defense:

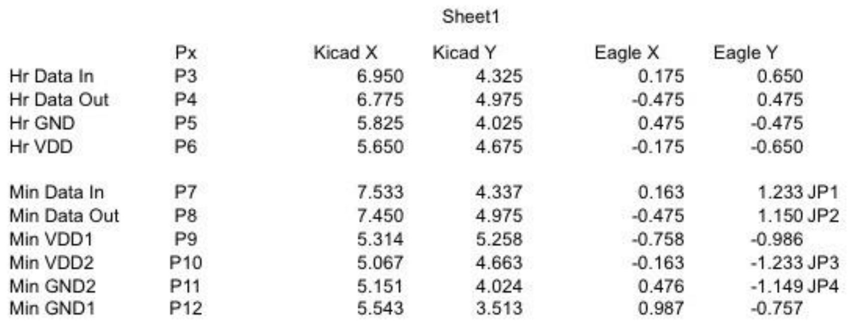
I came,

I saw,

I hacked,

Deal. ;-)

One particular challenge was placement of the connectors to which the NeoPixel rings attach. I had to bumble around in Eagle enough to understand how the power and data connectors were placed in its coordinate frame and then translate that into KiCad's. I still have bruises on my scalp from standing on my head (and in the end I had to pretty much just guess on the additional ring-24 power and grounds). I whipped up yet another spreadsheet to help me work out the translation:



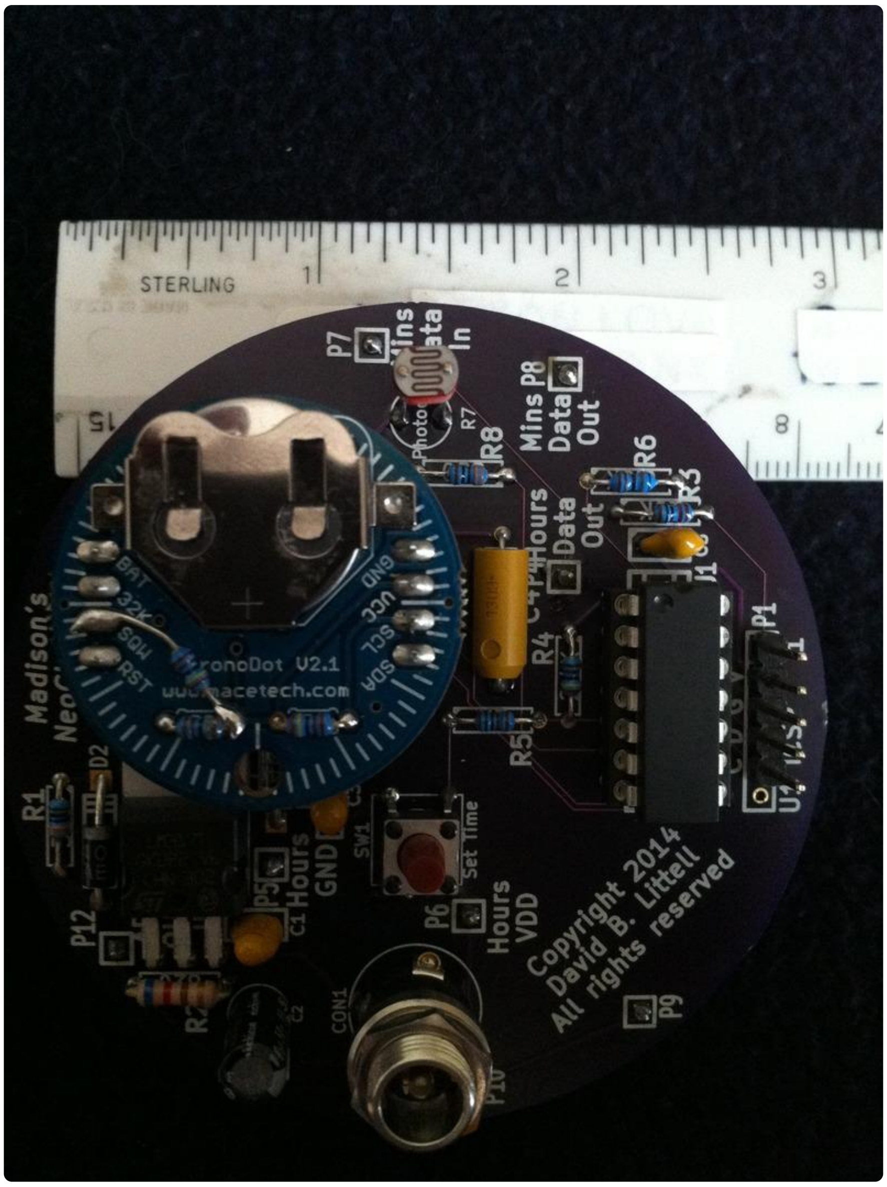
Sheet1

	Px	Kicad X	Kicad Y	Eagle X	Eagle Y	
Hr Data In	P3	6.950	4.325	0.175	0.650	
Hr Data Out	P4	6.775	4.975	-0.475	0.475	
Hr GND	P5	5.825	4.025	0.475	-0.475	
Hr VDD	P6	5.650	4.675	-0.175	-0.650	
Min Data In	P7	7.533	4.337	0.163	1.233	JP1
Min Data Out	P8	7.450	4.975	-0.475	1.150	JP2
Min VDD1	P9	5.314	5.258	-0.758	-0.986	
Min VDD2	P10	5.067	4.663	-0.163	-1.233	JP3
Min GND2	P11	5.151	4.024	0.476	-1.149	JP4
Min GND1	P12	5.543	3.513	0.987	-0.757	

In the end, I think it came together very well. Madison loves it and keeps it on her desk. I had a blast, learned a lot along the way, and stretched myself more than a little. I lost count early on of how many hours I put into this but it's been like many other projects over the years - some little nugget I learned here is guaranteed to help in another project later on.

Parting Shots

Here, then, are the guts of the finished article:



Artistic lighting? Totally unintentional - I'm just as much of a hack with photography...

Madison's
NeoClock v2
Power
P10
P11
P12
R1
R2
SQW
BAT
RST
GND
VCC
SCL
SDA

Copyright 2014
David B. Littell
All rights reserved
U1 ICSP
Power
P10
P11
P9
P6
Hours
VDD
Set Time
Hours
GND
P13
R5
R6
R3
Mins
Data
In
Mins
Data
Out