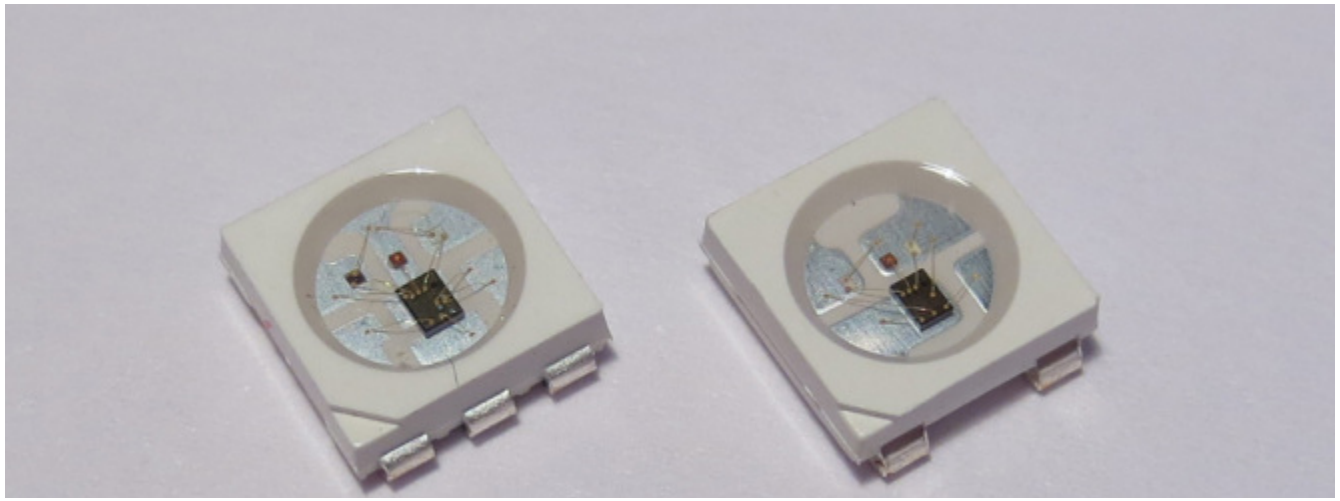
Light_WS2812 library V2.0 – Part I: Understanding the WS2812

JANUARY 14, 2014APRIL 27, 2014 CPLDCPU 34 COMMENTS

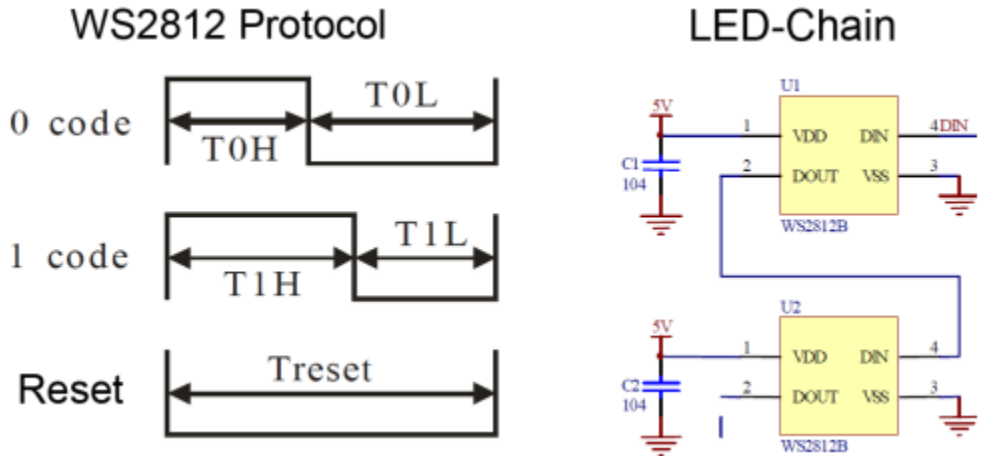
WS2812 LEDs are amazing devices – they combine a programmable constant current controller chip with a RGB LED in a single package. Each LED has one data input and one data output pin. By connecting the data output pin to the data input pin of the next device, it is possible to daisy chain the LEDs to theoretically arbitrary length.

Unfortunately, the single-line serial protocol is not supported by standard microcontroller periphery. It has to be emulated by re-purposing suitable hardware or by software timed I/O toggling, also known as bit-banging. Bit-banging is the preferred approach on 8 bit microcontrollers. However, this is especially challenging with low clock rates due to the relatively high data rate of the protocol. In addition, there are many different revisions of data sheets with conflicting information about the protocol timing. My contribution to this was the light_ws2812 library (https://github.com/cpldcpu/light_ws2812/) V1.0 for AVR and Cortex-M0, which was published a while ago. A V2.0 rewrite of the lib was in order due to various reasons. And, to do it right, I decided to reverse engineer and understand the WS2812 LED protocol to make sure the lib works on all devices.



(https://cpldcpu.files.wordpress.com/2014/01/ws2812_compared.jpg)

As of now, there are two different revisions of the WS2812 on the market: The original 6 pin WS2812(S) and the newer 4 pin WS2812B. The data sheets can be downloaded from the website of world-semi (www.world-semi.com), the original manufacturer, here (http://www.world-semi.com/uploads/soft/130222/1-130222154S6.pdf) and here. (http://www.world-semi.com/uploads/soft/130904/1_1500205981.pdf)



(https://cpldcpu.files.wordpress.com/2014/01/ws2812_protocol.png)

The data transmission protocol itself is relatively simple: a digital “1” is encoded as a long high-pulse, “0” as a short pulse on “Din”. When the data line is held low for more than 50µs, the device is reset. After reset, each device reads the first 24 bit (GRB 8:8:8) of data into an internal buffer. All consecutive bits after the first 24 are forwarded to the next device go through internal data reshaping and are then forwarded via “Dout” to the next device. The internal buffer is written to the PWM controller during the next reset.

WS2812/WS2812S

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	

WS2812B

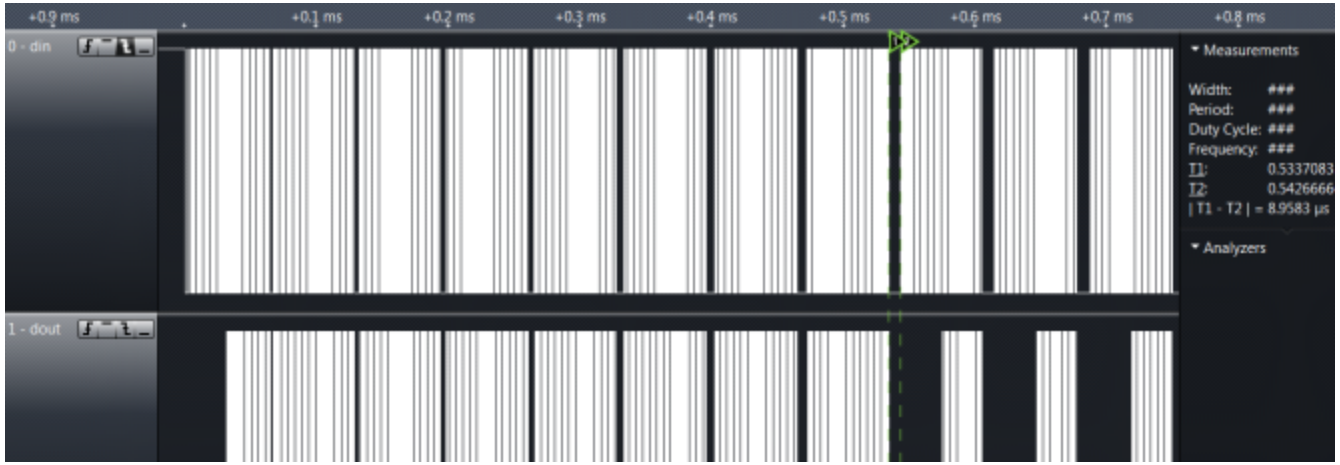
Data transfer time(TH+TL=1.25μs±150ns)

T0H	0 code ,high voltage time	0.35us	±150ns
T1H	1 code ,high voltage time	0.9us	±150ns
T0L	0 code , low voltage time	0.9us	±150ns
T1L	1 code ,low voltage time	0.35us	±150ns
RES	low voltage time	Above 50μs	

(https://cpldcpu.files.wordpress.com/2014/01/ws2812_timing.png)

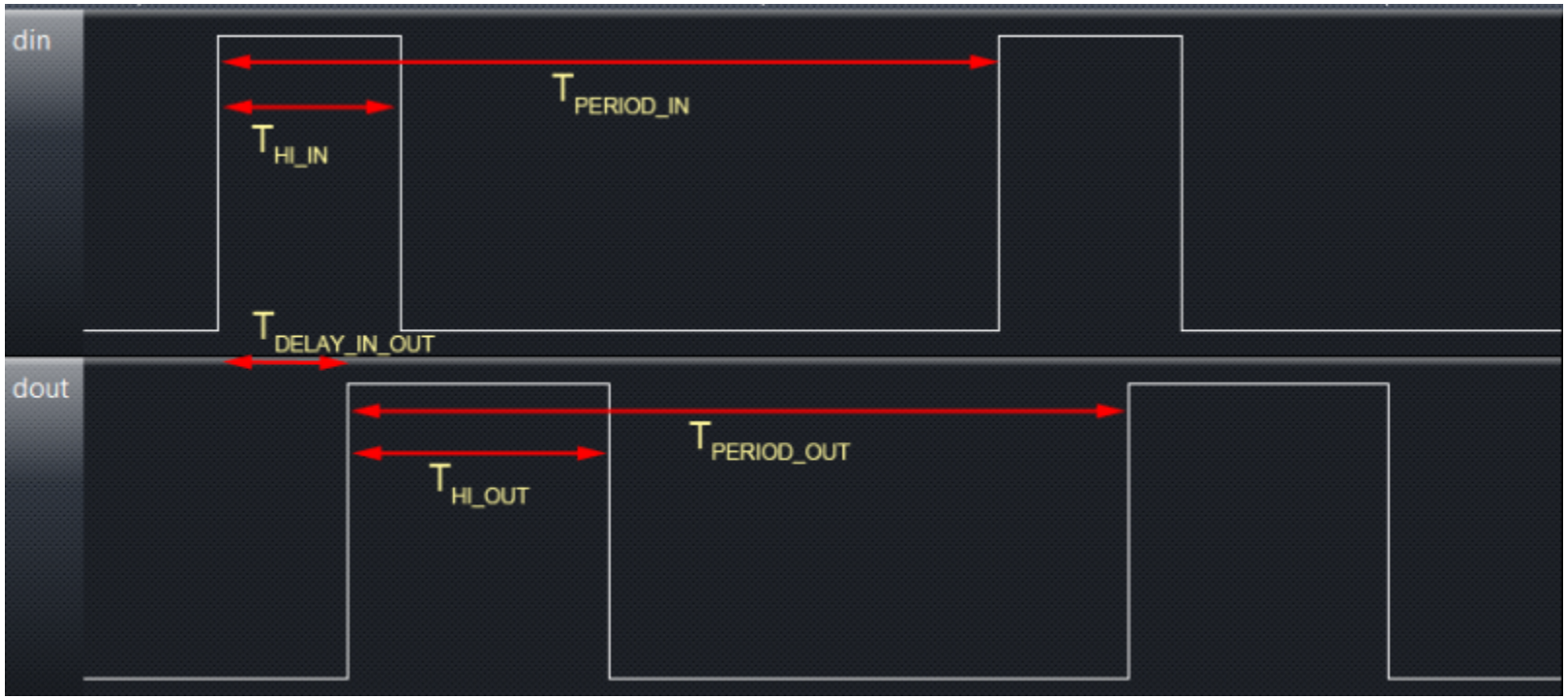
So far so good. This is where things get confusing. I copied the timing specification from both datasheets above. As you can see, both devices have slightly different timing for the encoding of the "1". Furthermore, the tolerances for the "data transfer time" are completely different and are in conflict with the "voltage time". So what are the real tolerances and can we find a set of timing parameters that fits both devices?

Luckily there is a relatively easy way to probe the inner workings of the device: When data is forwarded, it is passed through the internal reshaping mechanism. Therefore we can exercise Din and verify the correct interpretation of the input data by comparing it to Dout. To do this, I hooked a single WS2812 to a ATtiny 85 which took the role of a signal generator. I then monitored both Din and Dout with a Saleae (http://www.saleae.com/) logic analyzer. There are some issues with aliasing, since the maximum sampling speed is only 24 Mhz, but the data seemed still sufficient to understand the WS2812.



(https://cpldcpu.files.wordpress.com/2014/01/ws2812_reset_sweep.png)

In my first experiment I tried to determine the minimum time needed to reset the LED. My program emitted blocks of 48 bits with increasing delay time in between the blocks. As you can see above on the left side, all input data is forwarded to the output if the reset delay is too short. Once a certain delay threshold is reached, a reset is issues and data forwarding will only start after the first 24 bits, as seen on the right side. For the WS2812 under test here, the minimum reset length was 8.95 µs, way below the specifications. The suggested reset time of 50 µs is therefore more than sufficient to reset the LEDs. On the other hand, it means that no more than 9 µs of idle time may occur during data transfer, or a reset may mistakenly be issued.

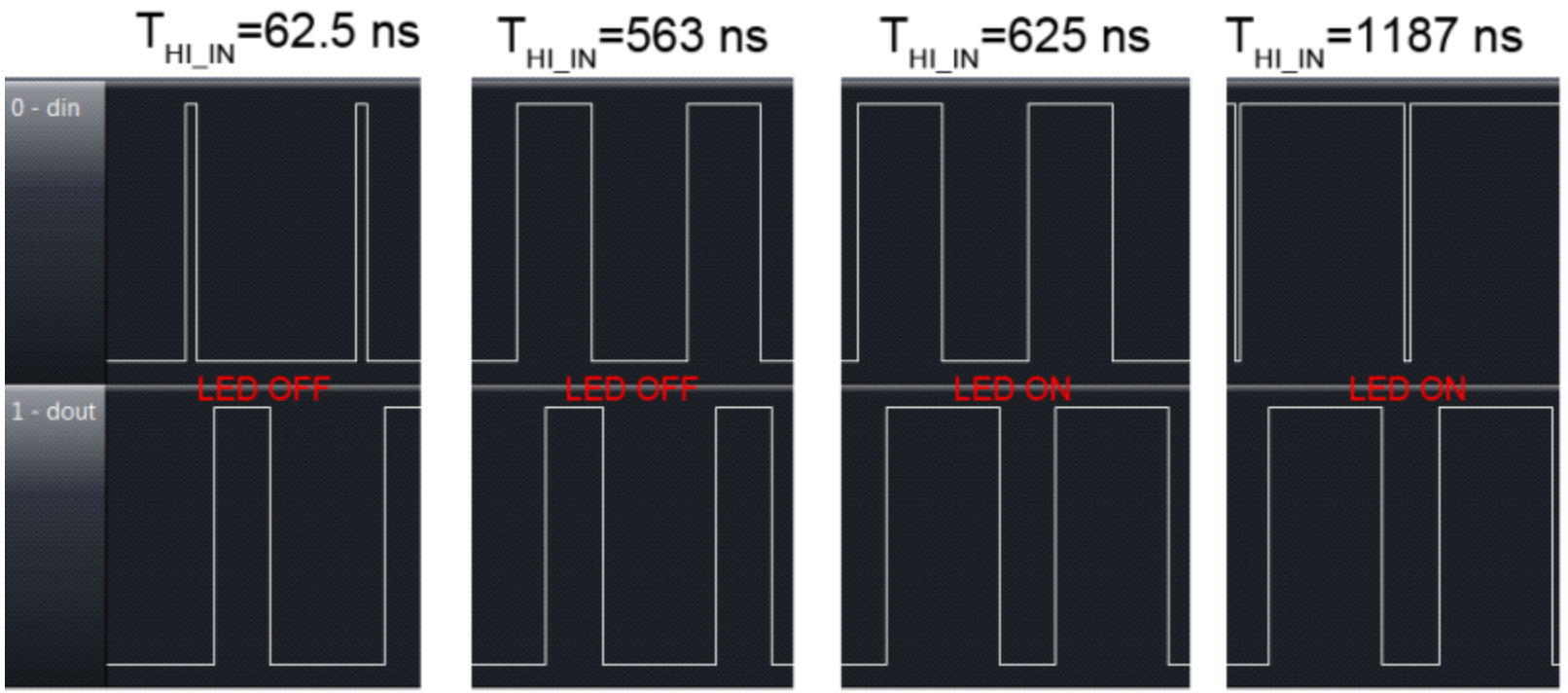


(https://cpldcpu.files.wordpress.com/2014/01/ws2812_timing_definitions1.png)

In the next step I looked at the data timing itself. The image above shows an exemplary measurement of input and reshaped output waveforms. Both waveforms can be described by two parameters each: The duration of the hi pulse and the total period. I programmed the microcontroller to cycle through all possible pulse input combinations between 62.5 ns (1 CPU cycle at 16 MHz) and 4 µs with a granularity of 62.5 ns. You can find the code is here. (https://gist.github.com/cpldcpu/8387639#file-gistfile1-c)

My original intention was to perform an automatic evaluation of the captured data to create a shmoo plot (http://en.wikipedia.org/wiki/Schmoo_plot). However, I quickly noticed that the behavior was quite regular and instead opted to analyze the data manually.

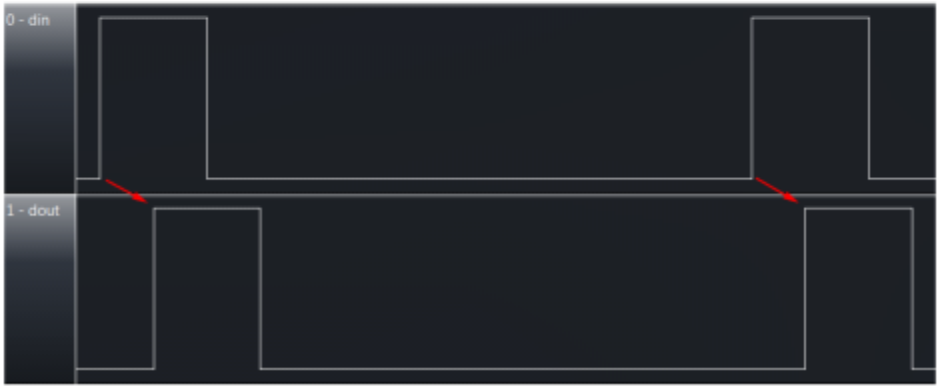
One of the first observations was that the delay between the leading edge of the input pulse and the leading edge of the output pulse, *T_delay_in_out*, was constant regardless of the timing of the input pulse.



(https://cpldcpu.files.wordpress.com/2014/01/ws2812_hi_variation1.png)

The image above shows a variation of *T_hi_in* for a constant *T_period_in*. The period length, called *total data transfer time* in the datasheet was set to the specification value of 1250 ns. As is obvious, there are only two states of the output signal: A short pulse for a "0" and a long pulse for a "1". Even the shortest input pulse (62.5 ns) is identified as "0", while even the longest input pulse (1250-62.5=1187.5 ns) is identified as a "1". The threshold between "0" and "1" is somewhere between 563 and 625 ns. The LED brightness changes accordingly, suggesting that the observations from the output signal are indeed consistent with the internal state of the LED.

T_{PERIOD_IN} = 2560 ns



(https://cpldcpu.files.wordpress.com/2014/01/ws2812_period.png)

Next, I varied *T_period_in*. When the period time of the input signal was much shorter than 1250 ns, the WS2812 started to reject input pulses. As can be seen for 333 ns, only about every fifth input pulse is replicated in the output pulses. The shortest pulse period time where all input pulses appeared on the data output was 1063 ns. Below that the input pulses were partially or fully rejected. Above this threshold all input pulses were interpreted correctly and the period of the output signal reflected the period of the input signal up to 9 µs when the reset condition was met.

This is an interesting observation, because it means that while there is a strict lower limit for the period time of the input signal, there is no real upper limit. For practical purposes, this allows relaxed timing in the software driver.

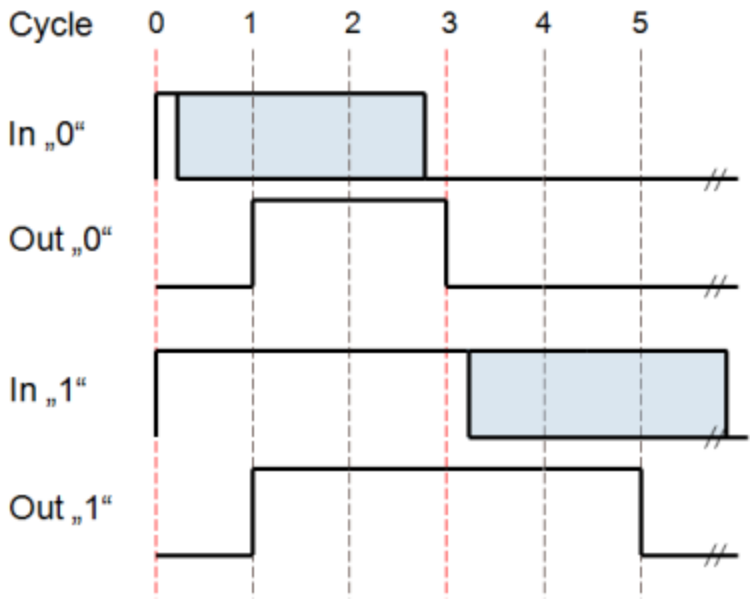
	Timing WS2812B	Timing WS2812	WS2812 Cycles
T_{HI_IN} "0"	62.5 ns - 563 ns	62.5 ns - 500 ns	<3
T_{HI_IN} "1"	≥ 625 ns	≥ 563 ns	>3
T_{PERIOD_IN}	≥ 1063 ns	≥ 875 ns	>5
$T_{DELAY\_IN\_OUT}$	~ 208 ns	~ 166 ns	1
T_{HI_OUT} "0"	~ 416 ns	~ 333 ns	2

T_{HI_OUT} "1"	~ 832 ns	~ 666 ns	4
T_{RESET}	> 9 µs	> 10.8 µs	-

(https://cpldcpu.files.wordpress.com/2014/01/ws2812_timing_table.png)

The table above summarizes my findings from the WS2812 and WS2812B each. It is possible that there are significant differences between production batches of both types, therefore these number can only serve as a rough indication. All timings seem to be a bit shorter on the WS2812. This is consistent with the data-sheet which indicates a longer pulse time for the "1" on the WS2812B.

An interesting observation is that the timing values for both LEDs are multiples of a smaller number, ~208 ns for the WS2812B and ~166 ns of the WS2812. It appears that the internal controller circuit is actually a clocked design – possibly realized by a small state machine.



This becomes much more obvious with the diagram above, which normalizes the timing to "WS2812 cycles". The internal WS2812 state machine only needs to sample the input twice per bit: First, it waits for a rising edge of the input. This will initiate the sequence above. The input is latched again after cycle 2. The voltage of the input pin at this point determines whether a '1' or a '0' is read. Depending on whether the LED already has received 24 bits or not, this value will either be loaded into an internal shift register or decide whether a 2 or 4 cycle 'hi' level signal is emitted. The sequence ends after cycle 5 and repeats again with the next rising edge.

So, what did we learn from this?

- A reset is issued as early as at 9 µs, contrary to the 50 µs mentioned in the data sheet. Longer delays between transmissions should be

avoided.
- The cycle time of a bit should be at least 1.25 µs, the value given in the data sheet, and at most ~9 µs, the shortest time for a reset.
- A "0" can be encoded with a pulse as short as 62.5 ns, but should not be longer than ~500 ns (maximum on WS2812).
- A "1" can be encoded with pulses almost as long as the total cycle time, but it should not be shorter than ~625 ns (minimum on WS2812B).

Next part: Optimized "Bit-Banging" (http://wp.me/p3pzLu-2r)

HARDWARE, LED, REVERSE ENGINEERING BITBANGING, LED, LIGHT EMITTING DIODE, REVERSE ENGINEERING, WS2812

34 thoughts on "Light_WS2812 library V2.0 – Part I: Understanding the WS2812"

1. **Rewriting WS2812 Driver Libraries for Optimization** says:
   February 2, 2014 at 9:01 am
   […] resources the library used to simply output the required timing signal for these LED modules. When he set out to build version 2.0, he dug much deeper than just optimizing his own […]

2. REPLY
   tz says:
   February 2, 2014 at 3:02 pm
   The only thing I'm curious or surprised about is that you can't pack the short bits closer together than the long bits, that the period has to be >1250 for both. I.e. that the low time has to be the opposite length.

   REPLY
   cpldcpu says:
   February 2, 2014 at 3:18 pm
   Yes, that surprised me too. But on the other hand it provides some level of robustness towards glitches. I don't think this behavior really adds complexity to the state machine.

3. REPLY
   RobG says:

February 2, 2014 at 6:14 pm
My guess is that there are few (4 or 5) simple RC pulse generators that take care of the timing.

REPLY

cpldcpu says:
February 2, 2014 at 7:13 pm
From my observiations I believe it is a clocked design. That makes a lot of sense, since RC oscillators take up a lot of area on the chip.

4. REPLY

Controlling RGB LED strip | Alan C. Assis says:
February 3, 2014 at 12:31 pm
[…] Update: This is a nice post about WS2812 explaining more details about how it works: https://cpldcpu.wordpress.com/2014/01/14/light_ws2812-library-v2-0-part-i-understanding-the-ws2812/ […]

5. REPLY

Mark says:
February 23, 2014 at 7:32 pm
Thanks for this investigation cpldcpu. The WS28xx datasheets are quite poor, so it was nice to find some more in-depth testing.

I have a fairly dumb question, but one I haven't seen answered anywhere. I'd like to design a bicycle-wheel POV thing with some of these LEDs, and I haven't been able to tell what happens when the string is reset. Do all of the LEDs go dark, and then turn back on, one at a time, as the new data is clocked in? Or does each LED retain its previous state until the new data reaches it? As you may imagine, the distinction is make-or-break for a POV project.

REPLY

cpldcpu says:
February 23, 2014 at 9:21 pm
The LEDs are updated almost instantly after a reset without a blanking period.

However, you should note that the WS2812 is PWM based with a frequency of roughly 400 Hz. There is visible flicker when moving the LEDs quickly. In a POV application you are probably limited to maximum and minimum brightness to avoid the "dotted line" problem with PWM.

REPLY

Mark says:
February 27, 2014 at 5:08 pm

Oh wow, I can't believe I didn't come across that little detail (PWM) in the datasheet or anywhere else online! That completely rules it out for my application. Guess I'll have to keep looking.

6.

Compact says:
March 17, 2014 at 11:10 pm
A very nice post.
Want to see over 3000 WS2812s working with a single micro and without any bit bashing ?
Take a look at http://www.youtube.com/watch?v=6sJH7wv0Lg8
and http://www.youtube.com/watch?v=8dYcjvWnKQo

REPLY

cpldcpu says:
March 18, 2014 at 7:39 am
Nice! For some reason Cypress really seems to like the WS2812. They also had some on display on their booth at EW2014.

No bits are bashed anywhere, though!

7. REPLY

Iman Davoudi says:
June 2, 2014 at 7:44 am
Big thanks for this useful post.
I'm working on ws2811 using this post and I want to drive it. I would like to know is there any problem if I use read my RGB colors data from a MMC. I want to store my lighting effects to a MMC and read them by AVR and send data to a ribbon of 5050 leds are driven by ws2811.
I know how to read MMC by AVR and how to send data to ws2811 but I want to know if it is fast enough and is it possible at all?
Please help me to do this.

REPLY

cpldcpu says:
June 2, 2014 at 4:55 pm
You have to store the data in the SRAM intermediately, you cannot read from the MMC and write to the WS2811 at the same time, as the MMC my cause delays that are too long for the WS2811.

REPLY

Iman Davoudi says:
June 2, 2014 at 6:15 pm

thanks for your reply.
This process, reading data from MMC and write to SRAM and then send to ws2811 as data takes time. Dose not this time (delay) cause problem? I have a ribbon including 700 leds and for each color one byte is needed so I must read 700*3=2100 bytes. Reading this 2100 bytes takes time.

- **cpldcpu** says:
  June 2, 2014 at 6:23 pm
  yes, it will take some time. 700 LEDs is a lot. You may have to consider a soluation with a more powerful cpu then AVR.

8. **Timing of WS2812 clones – PD9823 | Tim's Blog** says:
   June 16, 2014 at 9:22 pm
   […] other device. So are these really compatible to the WS2812? Only one way to find out: I used the same setup to extract the timing as described earlier for the WS2812. You can find the results […]

9. REPLY

   NeoPixels Revealed: How to (not need to) generate precisely timed signals | josh.com says:
   July 8, 2014 at 5:59 pm
   […] Thanks to Hack-a-day, I found Tim's similar work on WS2812Bs. He was motivated to optimize for speed and efficiency, but ends up with code that is also simple […]

10. REPLY

    First Adafruit NeoPixel Blinks with the FRDM Board | MCU on Eclipse says:
    July 13, 2014 at 3:56 pm
    […] Tim's blog about the WS2812: my reference guide for anything about the LED timing. He explains the differences between the 6-pin WS2812(S) and the new 4-pin WS2812B. […]

11. REPLY

    Chris Lomont says:
    July 30, 2014 at 4:33 pm
    Nice job

    We also needed details on how these modules worked, and ended up with similar work and results. Here is a post about it.

    http://hypnocube.com/2013/12/design-and-implementation-of-serial-led-gadgets/

    Chris Lomont

REPLY

cpldcpu says:
August 21, 2014 at 6:15 pm
Nice! You basically ended up with the same model.

12. REPLY

Malcolm says:
August 21, 2014 at 12:31 pm
"So, what did we learn from this?"

…That you're a legend, Tim, for having researched and published this information and those vital four points.
I had been frustrating over these devices for too many hours but thanks to you, I got my little critter to work.

Note for http://www.world-semi.com/en/ : publish a spec' sheet which is truly useful, not just a 'guide'.

Cheers and grateful thanks…

REPLY

cpldcpu says:
August 21, 2014 at 6:52 pm
You are welcome!

Yeah, World-Semi is puzzling me. They are messing up the launch of their successor device (WS2821) by not publishing anything about it. But on the other hand, the "maker" business is probably only a small fraction of their revenue, so I can't blame them…

13. REPLY

APA102 aka "Superled" | Tim's Blog says:
August 27, 2014 at 8:20 pm
[…] clones and variations of the venerable WS2812, there finally seems to be a new RGB-LED with integrated controller that actually improves on […]

14. REPLY

driving WS2812 programmable RGB LEDs using hardware SPI | Productize says:
October 7, 2014 at 7:14 am
[…] https://cpldcpu.wordpress.com/2014/01/14/light_ws2812-library-v2-0-part-i-understanding-the-ws2812/ […]

REPLY

Dennis D says:
November 20, 2014 at 3:23 pm
Hi Tim!
Love your work and how you explain how things works, and your blog is actually where i learn how to drive a 12 led ws2812B ring. What i wanted to do is to find a way to drive this ring with less hardware as possible. I meant if you just want to make a Christmas tree star, there's no need to hook up your arduino just for that purpose. What i found out was that you can drive you pixels at a VERY low bit rate. I could send one bits every 50 ms(!) without a reset condition. Thats really amazing because if you dont have a long strand and a huge amount of led to drive, then you can drive your leds with ease, no assembler (which is really good to learn) is required.

16. REPLY

Dennis D says:
November 20, 2014 at 3:30 pm
It works by stressing the logic 1 bits high time as long as you need to get the next bit (bits) and then pull the data line low for a new bit to start again. Not a elegant solution, but i just want to show that its possible to prevent a reset to occur if you need to wait (lets say more than 10us) for new data.

REPLY

cpldcpu says:
November 21, 2014 at 7:35 am
Hi,

yes, this makes a lot of sense and agrees fully with the behavior of the LEDS as described above. If you want really small code to drive the WS2812 you can use my lib: https://github.com/cpldcpu/light_ws2812

17. REPLY

Tom Dowad says:
December 22, 2014 at 5:53 am
Thank you so much for this spot on analysis. I like that you reverse engineered the state machine. I'm imagining the internal oscillator could change frequency with heat, or with manufacturing variations, which could change the timing figures.

The fatal flaw in the WS2812 design, from my perspective, is that the reset timeout is so short. It is shorter than a typical timer interrupt period, which makes it difficult to service these LEDs in a system with concurrent processes.

The chip I'm using doesn't have DMA, but it has a 16 bit data register on the SPI. So what I reckon is, set the SPI bit period to 500 ns (2MHz clock), so it takes 8us to shift out the 16 bits. If you add the 9 us before reset, that makes a minimum period of 17 us. Too short for a timer interrupt I would say, the MCU would be bogged down by interrupt overhead.

So…write 16 bits to the SPI at the start of the timer interrupt, do everything else that needs to be done, then at the end of the interrupt write another 16 bits. That will require some tweeking, as the interrupt handler execution period must be 8 us. And if the other processes could take longer than 8 us, polling of the SPI ready status would need to be interleaved. Assuming this concept works, the timer interrupt period would be 25us. That is feasible, in fact that is what I am using at the moment.

- REPLY

cpldcpu says:
December 22, 2014 at 10:11 am
Yes, in many cases using the periphery will not relax the CPU time requirements by much. As unelegant "Bitbanging" may look, it is often the lesser of two evils…

18. REPLY

Craig says:
January 11, 2015 at 5:56 am
FANTASTIC post. Thank you!

19. REPLY

Tor S says:
March 17, 2015 at 8:23 pm
Hi – very nice work! I am just playing around with your arduino code and some 5 meter of 2812b strip with 60 LEDs/meter. This gives me a total of 5*60 = 300 LEDs on one strip. However, I am only able to power 255 LED's along the strip using your git code for "fade_rgb". When I try compile and upload with 256 LEDs in total, the Arduino doesn't turn ANY LEDs on at all. Going back to 255 in the code again works just fine… In an attempt to troubleshoot it, I tried to change the "#define LEDCount 256" to "int LEDCount = 256;", since the int type should be 16-bit according to the references, but it still doesn't work

Do you have any idea why I can't power on more than 255 LEDs using your code? I am on Arduino UNO.

- REPLY

Tor S says:
March 17, 2015 at 8:25 pm
(5V DC is supplied to both ends of the strip using a 10A adapter, and nothing seems to be wrong with the brightness of the individual LEDs, so I'm quite sure the circuit is not missing juice.)

- REPLY

cpldcpu says:
March 18, 2015 at 4:17 am

This looks like a bug in line 37 of the example. I changed it on github. Can you test it? Thanks,Tim.

20. REPLY

Tor S says:
March 19, 2015 at 9:32 am
Works like a charm!

By the way, in your code, for this “fade_rgb” example, the green and red properties are switched around. This means that the following code sets the intensity for the green colour and not the red. And vice versa. Blue is fine
value.b = 0;
value.g = 0;
value.r = intensity; // RGB Value -> Red Only

I am currently switching between your code and the FastLED library, and it seems like the emitted colours are not flickering as much with your code. Have you compared this aspect as well? I am going to do some more comprehensive testing when it gets dark (it’s morning now:)

REPLY