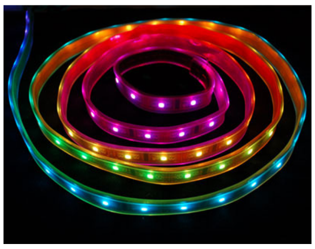
RGB strips



There is now a big choice of strips of colored leds. They are supported by libraries for Arduino, Raspberry and ESP8266.

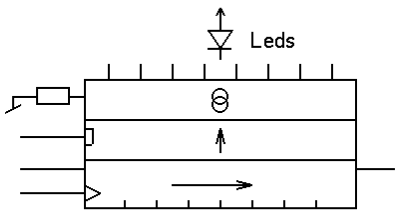
We are interested here to understand about the functionality of chips from different makers and how to write the required low level control routines.

The leds of a strip are serially connected and use shift registers. As shown later, shift registers are controlled with a clock and a data line. A third information is required to know when the data is complete. SPI add a third line. RGB strips count the clocks or no clock for e.g. 10ms means the end of transmission. RGB smart leds with 2 control lines accept a clock frequency up to 20 MHz and include the early WS2801 and by 2017 the one-chip APA102C, APA102-2020 and Sk9822 listed in appendix.
Clock and data can be encoded. The saving of one line makes the decoding tricky and interrupt sensitive. Frequency is fixed, 800 or 580 kHz. Existing chips are the WS2812b, APA104, SK6812. More listed in appendix. The problem of all these chinese chips is the specifications are not clear. Translation and extracts by distributors make some important data to desappear.
Strips of different density are "supported" by numerous web pages which does not explain what follows.

Refresh about serial transfers

Shift registers are used to convert a stream of bits synchronized by a clock into parallel data.
A parallel register keeps the previous data during the shift and is updated at the end of the transfer. This is the principle of the SPI transfer. Registers can be cascaded.

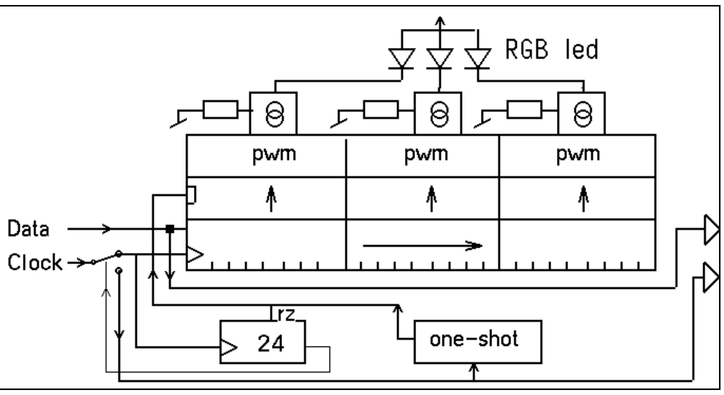
The frequent application is to control LEDs. Several manufacturers propose shift registers with constant current sources at the outputs. A single resistor sets the current, same for all LEDs. PWM can be done on every LED. See .www.didel.com/diduino/CommandeLeds.pdf
for an elegant solution updating PWM while shifting.



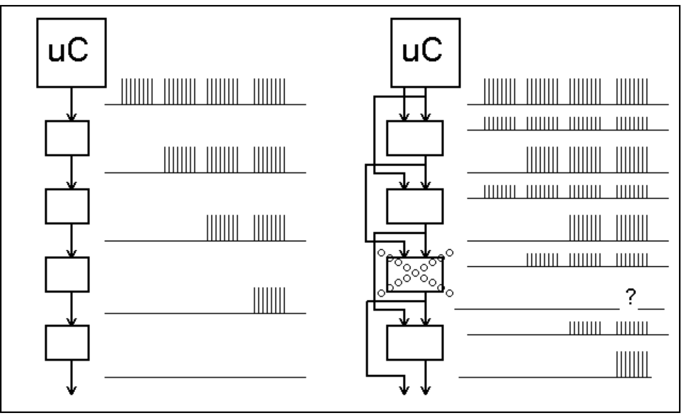
WS2801 as example of the inside logic

The WS2801 includes a 24 bit shift register with three 8-bit PWM controllers. Three LEDs are controlled by the circuit, usually a RGB SMD LED in a 5050 package. Intensity of LEDs is controlled by a resistor, constant current circuitry makes the intensity independant of the voltage (with 3-5V range).

The block diagram show something new. Data is not shifted through the complete serially connected circuits. The first 24 bits stay in the first circuit. When 24 clocks are counted, a switch transfers the clock to the output. Since there is no load line to transfer the shifted data to the PWM circuit, a one-shot circuit is activated when there is no clock for more than 1 ms. The signals are regenerated, allowing long strips, but adding a delay.

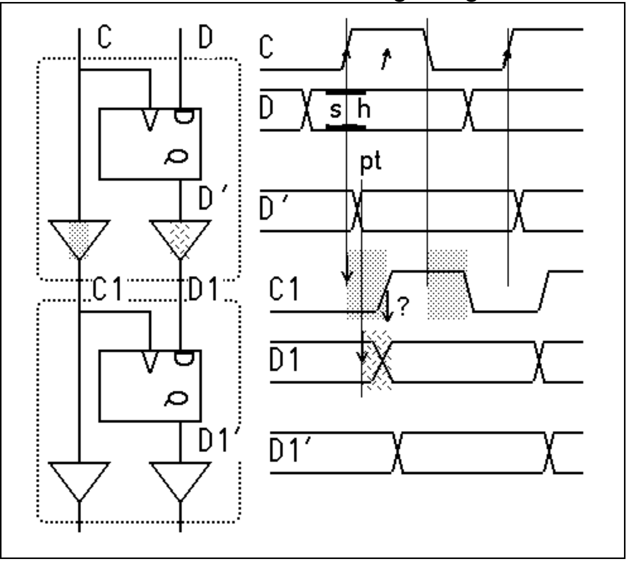


As said before, each circuit takes the first set of clocks and data he receives for himself, not transmitting that information further. The next clocks and data are amplified and transmitted. The delay introduced between each LED may be noticed over very long strips.
The APA102C implement an additional trick to make the transmission reliable. It results in a strange formula, incorrect and not explained for the terminating frame. Not easy to well understand if you have not been educated with integrated circuit logic. Let us try!



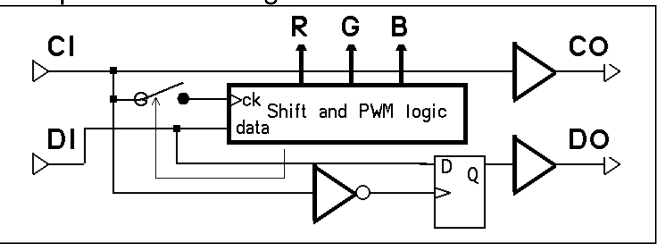
Before that, the WS2813 which is one-wire and does not need to use next trick, solves partly the reliability problem.
An additional input on every LED takes the clock of the previous circuit. If the normal signal is not active when it should, the signal is taken from the previous LED and the LED can light if good.

For the correct operation of a shift register, the flipflops must have the data stable when the clock edge occurs. set-up and hold time is 5 to 20 ns depending on the technology.The output D' changes after e.g. 20ns, and this must match the set-up time of the next flip-flop. One needs to add output buffers that introduce their own delay, plus the delay of the line. Depending on the difference between the clock and data propagation time, the transfer may not work correctly.
One solution is to increase the delay on the data line. This was probably done on the WS2801. It is more reliable to add a flip-flop on the data output, that add a delay of half a clock. This was implemented in the 1970' CD4094 8-bit shift register and is used on the APA102 and SK9822.



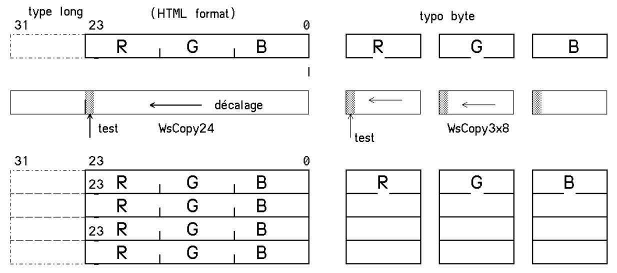
The main advantage is the very high transfer speed it permits with long wires.

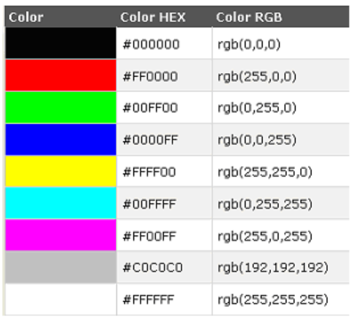
Due to the mechanism that allows to access the pixels in the order they have been sent, and not as if it was a simple shift register, data is delayed by half a clock for each consecutive pixel. The last bit sent for the last pixel must be pushed half a clock at a time till its destination.



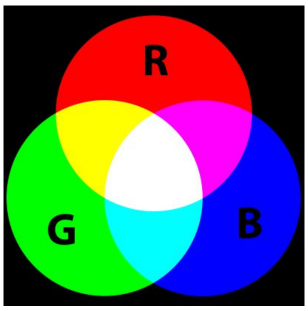
Color control

In order to refresh the colors of a strip, data must be snt in sequence. Data can generated on the spot, the frequent case for test programs. A copy of the RGB data of all LEDs is usually built in memory, as one RGB table, or as three tables for the colors.
The second picture documents the HTML colors

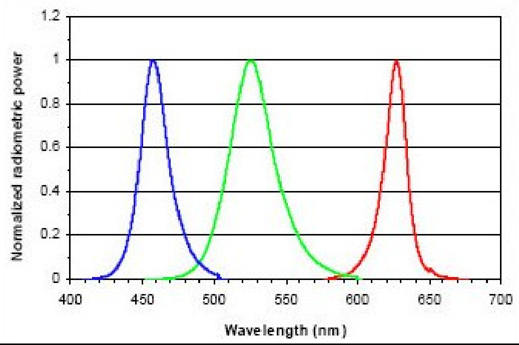




Color	Color HEX	Color RGB
⬛	#000000	rgb(0,0,0)
🟥	#FF0000	rgb(255,0,0)
🟩	#00FF00	rgb(0,255,0)
🟦	#0000FF	rgb(0,0,255)
🟨	#FFFF00	rgb(255,255,0)
■	#00FFFF	rgb(0,255,255)
🟪	#FF00FF	rgb(255,0,255)
■	#C0C0C0	rgb(192,192,192)
⬜	#FFFFFF	rgb(255,255,255)



Note the color order on the shift register is not always RGB. It is not a reason to use a different order at the high level.



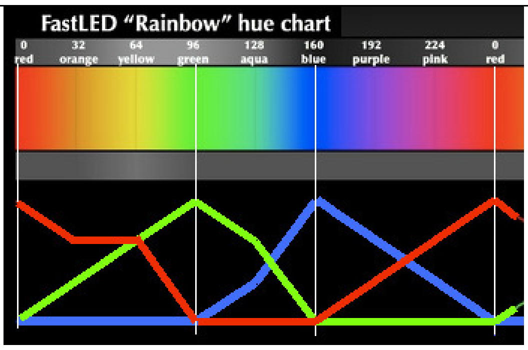
Discrete LED chips are soldered inside 5050 packages. It is not well documented what are the specifications and how the difference of LED intensity is compensated. In the best case, the information is.

Red	620-625nm	390-420mcd	2.0-2.2V
Green	522-525nm	660-720mcd	3.0-3.4V
Blue	465-467nm	180-200mcd	3.0-3.4V

Current for each color is not documented.
Color frequency influences the next section.

RGB vs HSV

RGB is not adequate for mixing colors. Using a "rainbow" variable (called hue) that goes through the rainbow colors is obtained from mixing RGB, as with the rather simple algorithm next from
https://github.com/FastLED/FastLED/wiki/Pixel-reference
we have reprogrammed in C (see appendix).
The **hue H** has a usually a 0-360 value due to its cylindrical representation. 0-255 is more convenient.
The **saturation S** says how the image is "rich" vs "pale" and will be kept at its maximum of 255.
The **value V** is the brightness 0-255.



Encoding colors is a rich and complex field. It is well described on
http://www.slideshare.net/michelalves/about-perception-and-hue-histograms-in-hsv-space

Note on luminosity

PWM is linear, 0 to 255. Our eye is not linear. 30% of PWM already feels as full light.
Not convinced? Program intensity 0,16, 32… with a change every 0.2 seconds. The first steps are visible, not the last 5.
This means 16 or 32 intensity values are enough if converted to exponential values.

```
  uint8_t talum[16] = {  0,5,11,18, 26,35,44,54, 64,76,90,110, 135,170,210,255};
```

Programming existing strips

We are not concerned here about commercial products and their associated libraries.
We document how to program from the low level up, using simple portable C. Understanding how to initialize a port is the only hardware requirement. Arduino digitalWrite is slow and should not be used, but update speed is not so critical for short strips.

WS2801/APA102C software

These curcuits have a SPI-like transfer over 2-control lines. The max clock speed for the WS2801 and APA102C is 20 MHz and data set-up time is 30 ns. The color order for the WS2801 depends on the wiring between the 14-pin circuit and the RGB LED.
The WS2801 restarts when there is no clock for 1ms. It is not documented if the transfer to the output latches is made when the register is filled or when the 1ms timeout occurs.

The APA102C and SK9822 need a start frame of 32 clocks with null data. Then, for every pixel, 4 bytes in the order - intensity, blue, green, red. Finally, additional clocks are given as explained before. Some old APA102 strips have a different color order.
No clock for 1ms transfers the data to the LED register, hence minimal clock rate is 1kHz, interrupts must be less than 1ms.
The first function one need, Send8 (byte); transfers 8 bits. AVR SPI can be used, with Ck on pin 13 and Data on pin 11. SS pin is not used, but that pin must be initialised as an output otherwise transmission does not happen. Byte transfer is 2us.
Shifting data out by software is slower (7us per byte) but any pin can be used and speed is adequate. Using the ShiftOut() Arduino function is no sense (slow and not easier).

See https://github.com/nicoud/RgbLeds for details and downloading the code.

??? You can find useful information on the APA102C on https://www.pololu.com/product/2554 with comments on APA102C vs SK6812.

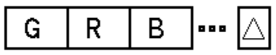
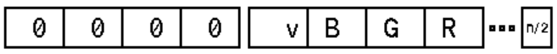
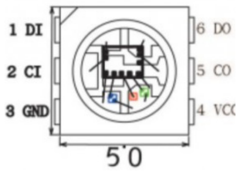
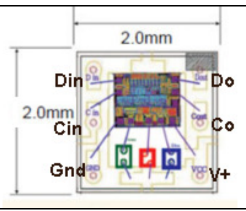
1-wire smart LEDs

Saving one wire is not a real advantage, since it is slower and less reliable both on hardware and software side. These circuits have critical timing need need bit-banging assembler code or macros. Interrupts must be disables during transfers.

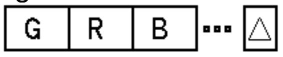
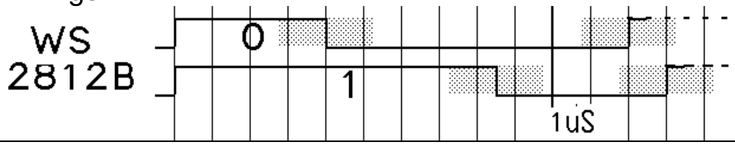
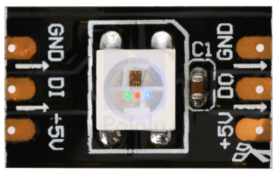
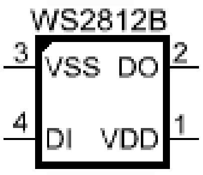
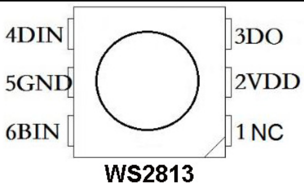
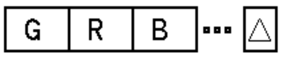
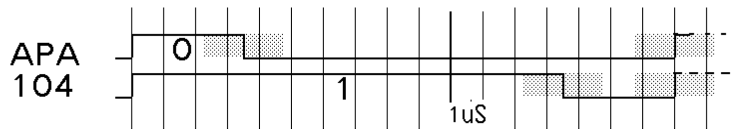
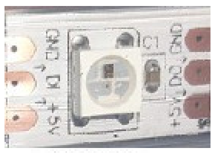
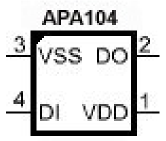
Appendix – RGB available smart LEDs

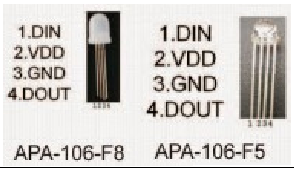
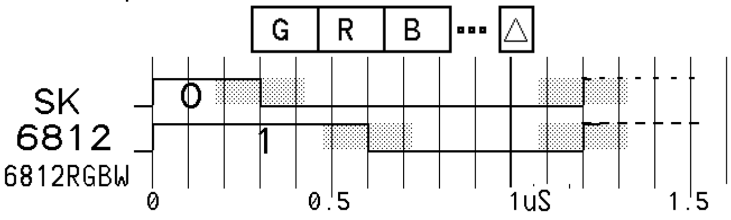
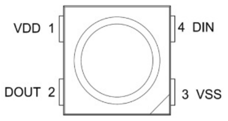
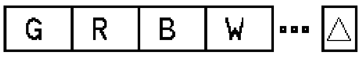
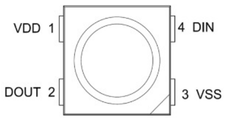
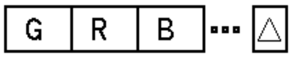
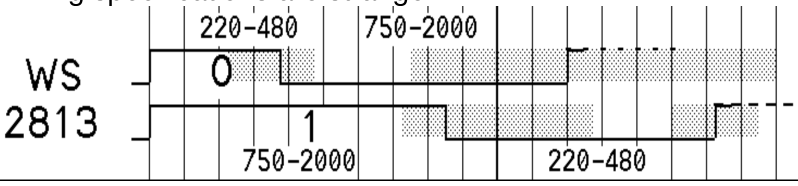
Separate leds can be bought from several vendors. It is of course more easy to buy strips, usually with a silicon protection. The strips can usually be cut to any length with solder pads.

2 -wires-control 0 – 20 MHz

WS2801 This 14-pin circuit includes three 8-bit PWM . Easy to program, as shown later. Can be used with any leds and any resistor value. Ck/Data serial transfer.  The **LPD8806/LPD6806** is similar but desappeared from the market. Transfer was three 5-bit PWM (16-bit words).	**WS2801**
The **APA102C** and **Sk9822** have SPI-like transfer, no timing constraint. New - miniature **APA102C-2020** The APA102C is an additional control register v that sets the brightness of every pixel, independently of its color. APA102 has 20 kHz PWM, SK9822 1.2 kHz  Some old APA102 strips use a G B R order	**APA102C SK9822** 
The new **APA102C-2020** is very compact. I does not have a glass window that diffuse the light. I may be discontinued	

1-wire control   800 or 580 kHz

The **WS2812b** is controlled by one wire with a critical timing that needs bit-banging routines on 16MHz AVR.  The **WS2812** (not shown) was the first RGB 5050 package with the microcontroller inside. 6-pin package replaced by WS2812b. Timings:  	**WS2812b**  
The WS2813	
The **APA104** has a slightly different timing than the WS2812B, but can be compatible. A 50us delay terminates the shift-in. LED PWM is ??   	**APA104**  

The **APA106** is the through-hole LED version of the APA104. Timing is the same. Pololu proposes similar LEDs (obsolete?) that include a WS2811 driver.	**APA106**  1.DIN 2.VDD 3.GND 4.DOUT APA-106-F8 APA-106-F5
The **SK6812** receives 24 bit words. Transfer speed is 800 kHz. LED PWM is ??  G R B ··· SK 6812 6812RGBW 0 1 0 0.5 1uS 1.5	**SK6812** 
The recent (2016) **SK6812RGBW** adds a white LED. Transfer speed is 800 kHz. LED PWM is ??  Same timing as SK6812	**SK6812RGBW** 
The recent (2016) **WS2813** has one more input connected to one chip before, in case the previous chip is bad. Transfer speed is 800 kHz, min 500 kHz  Timing specifications are strange.  220-480 750-2000 WS 2813 0 1 750-2000 220-480	**WS2813**