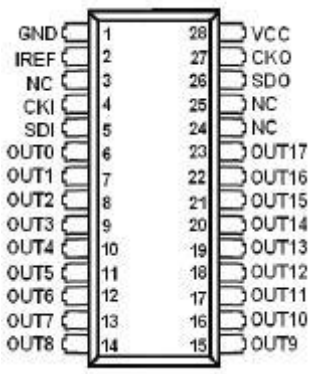
World Semiconductor WS2803 Testing

18 Channel PWM Constant Current LED Driver



Fritzing Part definition for WS2803 in DIP28 package WS2803D.fzpz

Simplistic code to test WS2803 for Arduino, Raspberri Pi, MBED LPC1768.

The WS2803 LED Driver IC from World Semiconductor consists of 18 constant current sink outputs. A resistor sets a maximum current reference which is mirrored to the 18 outputs. Each output includes an eight bit Pulse Width Modulator(PWM) to control the average current actually applied to the LED.

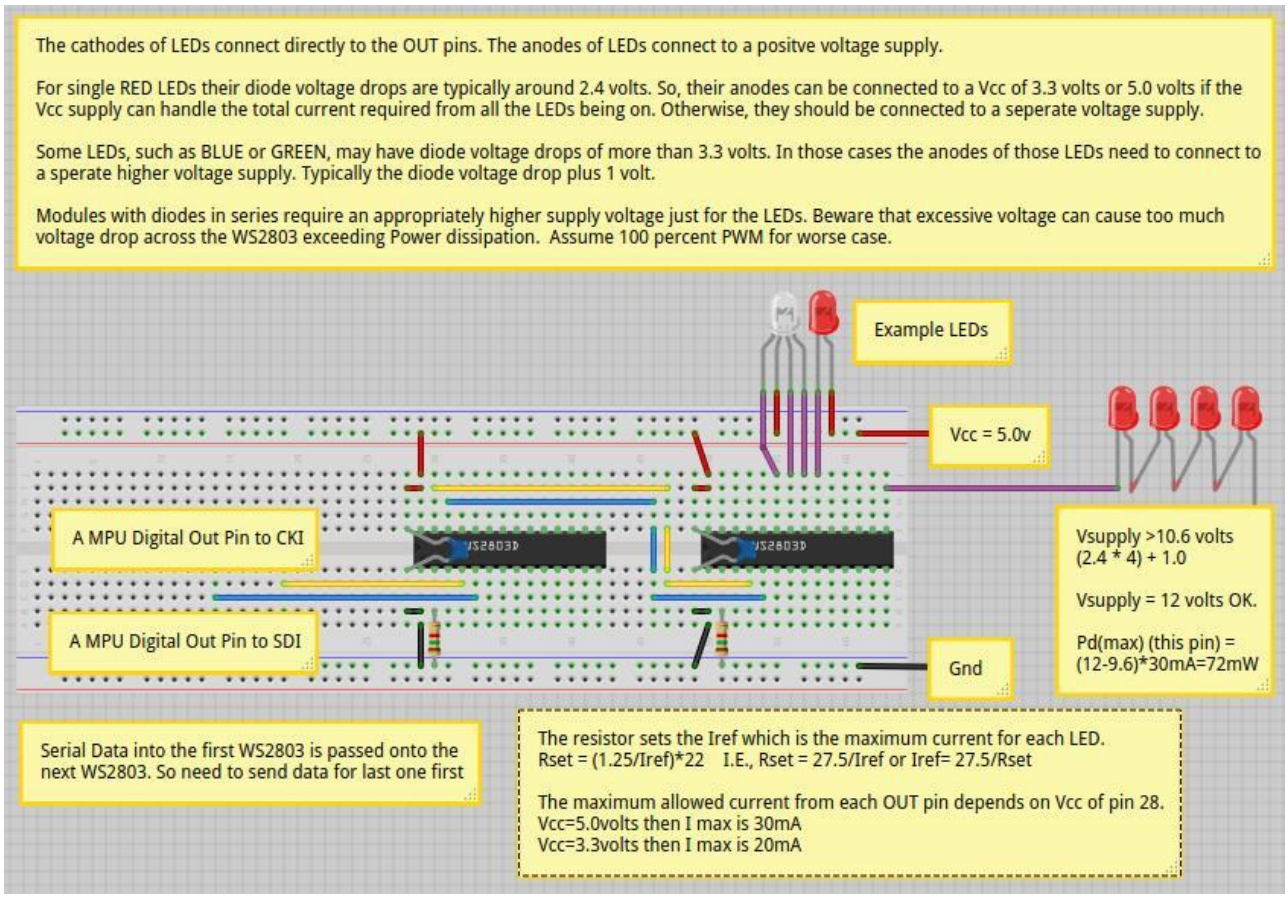
The eight bit PWM values are shifted into the WS2803 in packets of 18 bytes. The first byte goes to the PWM for output 0. The 18th byte goes to the PWM for output 17. The individual bytes are bit shifted in MSB to LSB order. The shifted in data is committed to the PWM's when the clock line is low for more 600 microseconds.

Multiple WS2803 can be daisy chained to drive more LEDs. In this example there are two WS2803 daisy chained. The preliminary specification for the WS2803 implies that the first 18 bytes would be captured by the first WS2803 and the second 18 bytes would be relayed to the second WS2803 and so on. Perhaps it's baby sister, the WS2801 works that way, but the WS2803 does not work that way. The first 18 bytes are clocked right on through as the next 18 bytes are received. If only 18 bytes are clocked into the first WS2803 the second WS2803 will receive the 18 bytes that was previously stored in the first WS2803.

To show this order the WS2803_test1.ino program sends 36 bytes in one packet. The program flashes all LEDs at once and then proceeds to walk one active LED through the array. It is observed that the walking LED appears on the second WS2803 first. Depending on what the purpose is for the LEDs the packet order has to be taken under consideration in a real application.

The Fritzing drawing shows how to daisy chain two WS2803s. All the LEDs are not shown as it should be obvious to the casual observer how to add additional LEDs compared to the examples shown.

The drawing does not show an Arduino connected as this drawing is meant to be more generic. The example code below is a very simple Arduino program using the Arduino shiftOut call, where the WS2803 clock-in (CKI) is connected to Arduino digital pin 7, and WS2803 serial-in (SDI) is connected to Arduino digital pin 8. Any Arduino digital out pins can alternatively be used depending on what an application requires.



Arduino code to test

WS2803 WS2801_test1.ino

```
// WS2803_test
// By Thomas Olson
// teo20130202.01
// WS2803 18 channel LED driver
// Arduino 5V 16Mhz
// WS2803 pin4 CKI - > w s2803_clockPin
// WS2803 pin5 SDI - > w s2803_dataPin
// WS2803 pin2 IREF = Rext to GND.
// WS2803 pin6-23 = OUT0-17

// According to the documentation...
// WS2803 w ill receive 144 bits(18 bytes), latching it to itself
// and then relay further bits to the next WS2803.
//
// But in fact, if you send more than 18 bytes say 36, then the
// first 18 bytes are passed on to the second WS2803. And the next 18 bytes
// are displayed in the first WS2803. And this is the behavior that is indicated
// in timing chart (Fig 4.) of the documentation.
//
// This test code assumes tw o WS2803 in the chained together.
```

On the idea that "ideal" current sources or sinks can be parallelled for additive current supply this test evaluates if the WS2803 can be used in this fashion. In theory all 18 channels of the WS2803 should be able to be tied together, each supplying up to its maximum programmed current to the load. Even with pulse width modulation the average current of the combined channels ought to not cause problems.

Here the chained WS2803's are connected to a Raspberry Pi GPIO SPI pins SCLK(GPIO 10) and MOSI(GPIO 11). The Vcc supply on the WS2803 are connected to 3.3 volts. The anodes of the 18 red LEDs are tied to 5.0 volts. The anode of the 30W RGB LED module is connected to 30 volts.

Linux Code on Raspberri Pi to test

WS2803 WS2803_simple.c

```
                        loadWS2803();
                        clearWS2803();
                }
        }
        sleep(1);

        //parallel mode
        for(iOut = 7; iOut < 64; iOut*=4){
                for(w sOut = 0; w sOut < nLEDs; w sOut++){
                        ledBar[0][w sOut] = ledBar[1][w sOut] = (uint8_t)(0xFF
                        loadWS2803();
                        usleep(1000000);
                }
                usleep(10000);
        }


        }//for(;;)

        close(fd_spi);
```

MBED LPC1768 code to test

WS2803 WS2803_TestMBED.cpp

```
    for(int iOut=7; iOut < 256; iOut*=2){
       for(int w sOut = 0; w sOut < nLEDs; w sOut++){
          ledBar[0][w sOut] = ledBar[1][w sOut] = (uint8_t)(0xFF & iOut);
          loadWS2803();
          clearWS2803();
       }
    }

    w ait(2.0);
    //parallel mode
    for(int iOut = 7; iOut < 64; iOut*=4){
          for(int w sOut = 0; w sOut < nLEDs; w sOut++){
                ledBar[0][w sOut] = ledBar[1][w sOut] = (uint8_t)(0xFF & iOut);
                loadWS2803();
                w ait_us(1000000);
          }
          w ait_us(10000);
    }
  }// w hile(1)
}
```