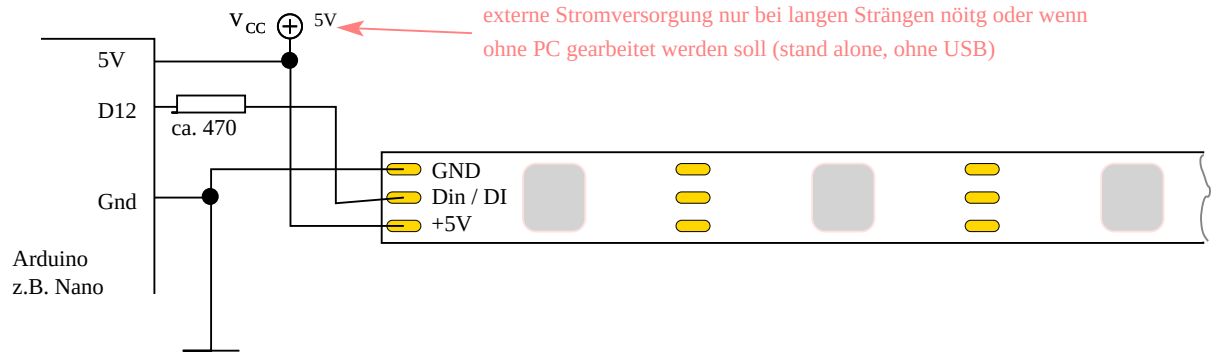
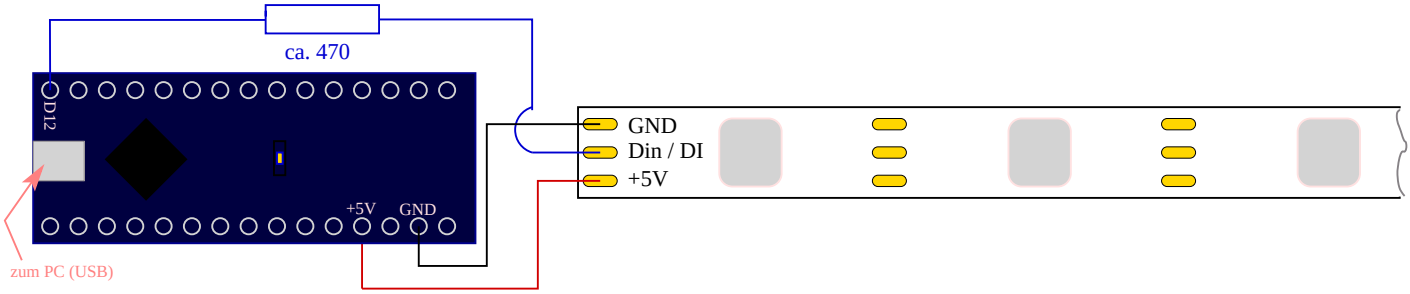
Arduino zum Ansteuern der WS2812B-Stränge

Dies ist eine etwas weniger kostengünstige Variante im Vergleich zum Einsatz eines puren AVR-Mikrocontrollers. Kauft man Arduinos als Originalteile, wird es sogar richtig teuer. Ein Arduino Nano, der gut geeignet ist für Experimente mit Steckbrettern (also auch für den eifrigen Bastelbetrieb!) kostet derzeit ca. 40 Euro. Ein China-Nachbau kostet aber inklusive Versandkosten unter 3 Euro. Ob man den Schmuh mit fehlender Rechnung / unterschlagener Mehrwertsteuer etc. mitmachen will, ist natürlich eine andere Frage. Mit einem China-Arduino ist ein typisches Kleingerät (Elektronischer Würfel, Tic Tac Toe o.Ä.) für unter 10 Euro zu realisieren (inkl. Taster, Batterie etc.).

Kurze Streifen können direkt am Arduino an dessen Stromversorgungsausgang mitbetrieben werden, längere benötigen ein eigenes Netzteil. Ein beliebiger Ausgangspin des Arduinos kann zur Ansteuerung gewählt werden. Hier ein Beispiel mit einem Arduino Nano, dessen Pin "D12" als Anschlusspin verwendet wird:



oder anders dargestellt:



Zur Ansteuerung wird die heute bereitgestellte Bibliothek WS2812b verwendet. Für sie dient die Bibliothek light-ws2812 von https://github.com/cpldcpu/light_ws2812 als Grundlage. Verwendet wird deren Assembler-Inline-Routine, die die Farbdaten eines Arrays zum Strang zu überträgt. Bei WS2812b sind die Übertragungskommandos und die Datenstrukturen aber so angepasst, dass sie sich meiner Meinung nach besser für den Bastelbetrieb eignen.

Anwendung:

1. WS2812b.h und WS2812b.cpp werden in ein Verzeichnis ~/sketchbook/libraries/WS2812b kopiert. Damit ist die Bibliothek ab sofort für alle Projekte verfügbar.[1]
2. Die IDE "arduino" wird gestartet. Beim Speichern des Projekts werden Dateiname und Verzeichnisname (unterhalb von ~/sketchbook) gleich gewählt. Strg-T hilft beim Formatieren des eingegebenen Quelltextes.
3. Die Datei wird mit Strg-U oder Klicken auf das Pfeilsymbol übersetzt und, falls fehlerfrei, übertragen. Evtl. müssen vorher Einstellungen für den Typ des Arduino-Boards und für die Schnittstelle, an der es hängt, gemacht werden (→ausprobieren oder fragen...).

Hier ein Beispielprogramm:

```
#include <ws2812b.h>
WS2812b leds(1,3,1,2); // Breite=1, Höhe=Länge=3,Typ=1(keine Matrix),Daten-Pin="2"
void setup() {
  //Serial.begin(9600); // falls man noch mehr vom Arduino haben möchte...
  //Serial.println("Hallo Welt!"); // dito
  leds.setR(0,0,0xFF); // Pixel 0 rot
  leds.setG(1,0,0xFF); // Pixel 1 grün
  leds.setB(2,0,0xFF); // Pixel 2 blau
  led.setMaxBrightness(5); // auf 5 herunterdimmen = niedrige Helligkeit
  led.showLEDs(); // gesetzte Werte anzeigen
}
void loop() {
}
```

Eine Übersicht aller verfügbaren Kommandos findet sich auf der Rückseite.

[1] Derzeit funktioniert sie aber nur an Pins, die am Port D des Controllers hängen, das sind am Nano z.B. die Pins 0 bis 7. Passt das nicht, muss die WS2812b.h editiert werden, um einen andern Port auszuwählen.

Verfügbare Kommandos:

`WS2812b(`*`Breite`*`,`*`Höhe`*`,`*`Typ`*`,`*`Datenpin`*`)` initialisiert einen angeschlossenen LED-Strang. Bei linearen Strängen (keine Matrix) ist Breite gleich 1 und Höhe ist gleich der Länge des Strangs. Der Typ richtet sich nach dem Verdrahtungsschema (siehe unten). Der Datenpin wird mit der Nummer angegeben, die auf dem Arduino aufgedruckt ist.

`setR(`*`Zeile`*`,`*`Spalte`*`,`*`Farbwert`*`)` setzt das Pixel bei Zeile *zeile* und Spalte *spalte* auf einen Rotwert zwischen 0 (aus) und 255 (volle Helligkeit).

`setG(`*`Zeile`*`,`*`Spalte`*`,`*`Farbwert`*`)` dito für Grün

`setB(`*`Zeile`*`,`*`Spalte`*`,`*`Farbwert`*`)` dito für Blau

`setRGB(`*`Zeile`*`,`*`Spalte`*`,`*`Farbwert`*`)` setzt das Pixel bei Zeile *zeile* und Spalte *spalte* auf einen RGB-Farbwert zwischen `0x00000` und `0xFFFFFF`, dabei gilt die Form "`0xRR`*`GG`*`BB`", also zuerst der Rotanteil **RR**, dann der Grünanteil *GG*, dann der Blauanteil **BB**. Der Vorsatz "`0x`" erlaubt hierbei die Eingabe hexadezimaler Farbwerte. Ohne ihn muss ein (umgerechneter) Dezimalwert angegeben werden. Es gehen auch Binär-Farbwerte, z.B. in der Form "`0b 00000000 11111111 00000000`".

`showLEDs()` zeigt die aktuelle Belegung an — wird aufgerufen, nachdem man alle Farbwerte zuvor gesetzt resp. aktualisiert hat.

`clearLEDs()` macht alle LEDs dunkel und löscht zudem das Array mit den Farbwerten.

`getR(`*`Zeile`*`,`*`Spalte`*`)` liefert den Rotwert (zwischen 0 und 255) des Pixels in Zeile *zeile* und Spalte *spalte*.

`getG(`*`Zeile`*`,`*`Spalte`*`)` dito für Grün

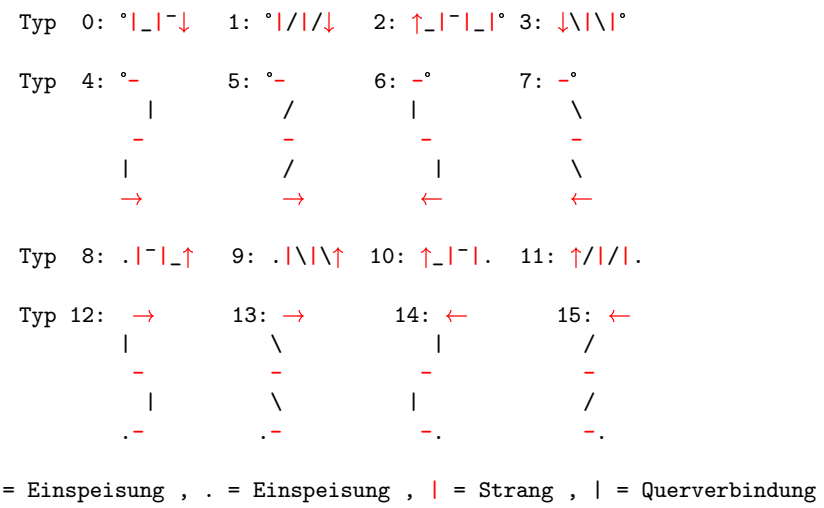
`getG(`*`Zeile`*`,`*`Spalte`*`)` dito für Blau

`getRGB(`*`Zeile`*`,`*`Spalte`*`)` liefert als Dezimalwert den RGB-Farbwert des Pixels bei Zeile *zeile* und Spalte *spalte*.

`setMaxBrightness(`*`maximalerFarbwert`*`)` bestimmt, welcher Farbwert maximal ausgegeben werden darf — kann zum Dimmen der LEDs verwendet werden; mögliche Maximalwerte liegen zwischen 0 (nicht sinnvoll, alles bleibt dunkel) und 255 (keine Dimmung). Für Experimente mit schwacher Spannungsversorgung empfiehlt sich z.B., hier einen niedrigen Wert zu setzen, welcher später erhöht wird, wenn man eine bessere Spannungsversorgung zur Verfügung hat.

Neben der Verwendung der obigen Befehle zum Setzen und Lesen der Pixel-Farbwerte kann auch direkt auf das Array `pixel` zugegriffen werden. Allerdings fehlt durch dessen lineare Struktur die einfache Zuordnung zu den Pixelkoordinaten, wie sie die Befehle je nach angegebenem Verdrahtungsschema auch bei Matrix-Anordnung der LEDs liefern.

Der Typ des Verdrahtungsschemas richtet sich nach der gewählten Anordnung der LEDs (Verdrahtungsschema):



° = Einspeisung , . = Einspeisung , | = Strang , | = Querverbindung

---

Hier noch ein (platzsparend formatiertes ;-) Blinklicht für eine LED:

```
#include <WS2812b.h>
WS2812b leds(1,1,1,12); // Matrix 1x1 Typ 1 = 1 LED, an Pin D12
void setup() {}
void loop() {
  setRGB(0,0,0x030201);showLEDs();delay(100);clearLEDs();delay(500);
}
```