Tetris-Klon mit LED-Streifen

Bei herkömmlichen RGB-LED-Streifen kann man zwar die Farbe variieren, aber nur für alle LEDs zusammen. Bei neueren Streifen sitzt in jeder LED ein eigener Controller, mit dem sich die Farbe jeder LED einzeln programmieren lässt – und das ohne großen Verkabelungsaufwand. Wir haben damit einen Tetris-Klone realisiert: Make:Block.

von Ulrich Schmerold

Kurzinfo

Zeitaufwand:
1 Wochenende

Kosten:
100 Euro

Programmieren:
Arduino

Löten:
leichte Lötarbeiten

Holzarbeiten:
Bohren, Senken

Schwierigkeitsgrad

leicht — schwer

Einkaufsliste

- 3 Meter **LED-Stripe mit WS2812B**
- **Arduino Nano/Uno**
- **Gehäuse**
- **4 Taster**
- **Stromversorgung,** 5 V, 1 A
- **Ikea-Rahmen, Ribba**
- **MDF-Platte,** 8 mm Dicke
- **Folie, Papier, Moosgummi**
- **Kabel, Draht**
- **10-mm-Bohrer, Senker**
- **Widerstand,** 220 Ω

In diesem Projekt zeigen wir Ihnen, wie Sie mit LED-Stripes ein individuelles Display herstellen können. Die Stripes beruhen auf dem Chip WS2812B, der im Grunde aus einem Schiebregister mit 24 Bit Breite besteht, wobei jeweils 8 Bit die Helligkeit einer roten, grünen und blauen LED bestimmen. Ein Arduino erzeugt den Takt und die Signale für die Schieberegister. Eine detaillierte Erklärung zum Chip finden Sie auf Seite 17.

Als Beispielanwendung für das RGB haben wir das nach wie vor sehr beliebte Retro-Spiel Tetris ausgewählt. Als Display-Größe wählen wir 15 × 10 Pixel. Prinzipiell ist es problemlos möglich, mehr als 1500 Chips mit einem Arduino zu steuern. Um das Budget nicht unnötig zu belasten, wählten wir als Bildschirmgehäuse einen Ikea-Bilderrahmen der Ribba-Serie mit den Maßen 20 × 30 cm. Bei der Verwendung von LED-Stripes mit 60 Chips pro Meter bietet dieser Platz für das gewählte Raster von 10 × 15 Chips. Bei eBay werden Stripes mit dem WS2812B in den Längen 1 m (17 Euro), 2 m (34 Euro) und 4 m (65 Euro) angeboten. Ein 1-Meter- und ein 2-Meter-Streifen reichen für unser Projekt aus.

Display

Nun beginnt man, aus den LED-Streifen ein Display zu fertigen. Prinzipiell kann man 10 Streifen mit je 15 LEDs zurechtschneiden, alternativ 15 Streifen mit je 10 LEDs. Wir entschieden uns für letztere Variante, da wir dann weniger Verbindungsstellen und somit weniger Lötaufwand hatten. Zudem passt das besser zum 1-Meter-Streifen: bei insgesamt 60 LEDs lassen sich 4 Streifen mit je 15 LEDs ohne Rest herausschneiden.

Die LEDs haben auf unserem Streifen einen Abstand von exakt 16,7 Millimeter. Also müssen auch die Reihen diesen Abstand einhalten. Am besten zeichnet man die Reihen vorher genau auf das Einlegebrett des Ribba-Rahmens und orierentiert sich beim Aufkleben daran. Beim Befestigen der Stränge muss man unbedingt auf die richtige Richtung achten: Der Pfeil muss bei unserer Variante immer nach rechts zeigen.

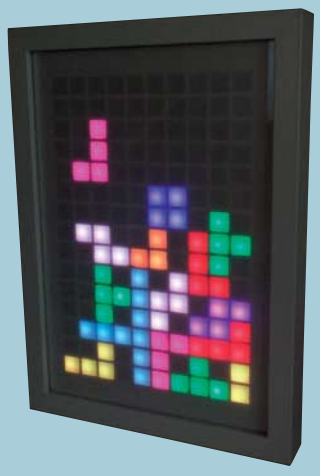
Am Anfang und am Ende jedes Streifens bohrt man je zwei Löcher (2 Millimeter) in die MDF-Platte, um die Spannungsversorgung und die Datenleitung D_{IN} auf der linken Seite, sowie GND und D_{OUT} auf der rechten Seite nach hinten führen zu können.

Auf der Rückseite führt man die Datenleitungen zu den Streifen und verbindet dort alle GND- und 5V-Anschlüsse. Bei den Datenleitungen beginnt man mit der Dateneinspeisung links unten. Den jeweiligen Ausgang D_{OUT} eines Streifens verbindet man mit D_{IN} des darüber liegenden Streifens.

Blendwerk

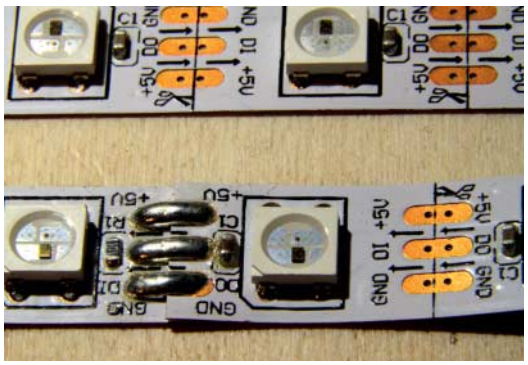
Da der WS2812B einen sehr breiten Abstrahlwinkel hat, kann das Material für unsere Frontblende relativ dünn ausfallen. Schon bei 4 mm Abstand zwischen LED und Mattscheibe (Diffusor) wird eine durchaus

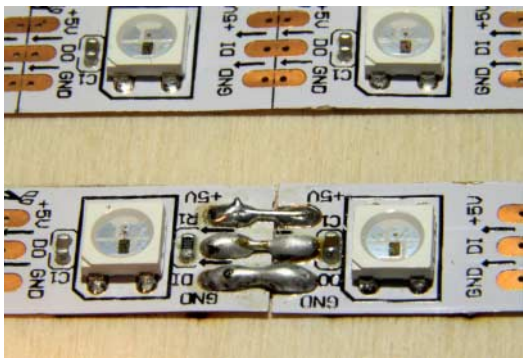
Je nach gewünschter Höhe und Breite des Displays muss man sich die Streifen zurechtschneiden.



VERSUCH MACHT KLUG

Als Wareneingangskontrolle unserer eBay-Bestellung (und weil wir es kaum erwarten konnten, die LED leuchten zu sehen) schlossen wir jeden Stripe provisorisch am Arduino an und ließen einen Testsketch ablaufen. Bei dem 1 Meter kurzen Stück waren wir begeistert, dass er sofort problemlos und störungsfrei funktionierte. Beim 2 Meter langen Stück zeigten sich hingegen Fehlfunktionen: Die LEDs ab etwa der Mitte des Stranges leuchteten unkontrolliert in allen möglichen Farben. Nach intensivem Googeln tauschten wir unsere ältere Arduino-Bibliothek FastSPI_LED gegen die neue FastLED von GitHub aus. Zudem folgten wir nun auch der Empfehlung, zwischen Arduino und LED-Strip einen Widerstand von 200 bis 300 Ohm in die Datenleitung einzufügen. Für das erste RGB-LED-Segment in unserem Strang kam diese Erkenntnis jedoch zu spät. Wir schnitten es einfach ab, löteten die Anschlüsse neu an und siehe da, alles funktionierte wie erhofft.





Links ist es falsch, rechts hingegen richtig repariert.

REPARIERTE STRIPES

Unsere Stripes wurden anscheinend bereits vom Hersteller oder Vertreiber repariert geliefert. Die Verbindungsstellen waren überlappend zusammengeklebt und mit drei Lötpunkten verbunden. Damit verändert sich jedoch das Abstandsmaß, was später zu einem ungleichmäßigen Bild oder sogar zu einer Kollision mit unserer Blende führen kann. Derartige Stellen haben wir wieder aufgetrennt und im richtigen Abstand mit drei Stückchen Kupferdraht angelötet.

brauchbare Ausleuchtung erzielt. Bei einem 8-Millimeter dicken MDF-Brett ist die Ausleuchtung dann absolut gleichmäßig.

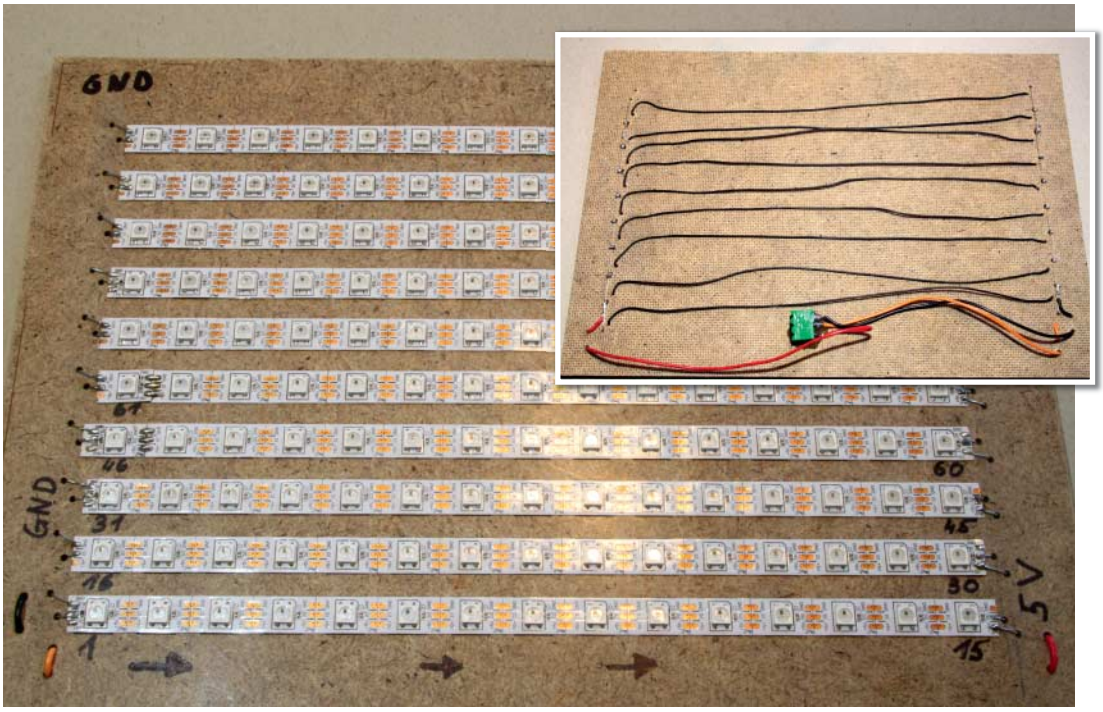
Für ein schönes Display ist der gleichmäßige Abstand der Bohrungen von entscheidender Bedeutung. Da Bohrer naturgemäß immer wieder verlaufen, haben wir uns die Mühe gemacht, die 15 mm Lochkreise anzuzeichnen. So können wir beim Bohren im Bedarfsfall gegensteuern.

Man bohrt zunächst Löcher mit einem 10-Millimeter-Bohrer und senkt die Löcher anschließend mit einem 15-mm-Kegelsenker auf. Um noch schärfere Kreise zu erhalten, druckt man eine entsprechende Schablone auf Folie aus und legten sie zwischen MDF-Blende und Mattscheibe. Die Vorlage zum Ausdrucken finden Sie unter dem Link am Ende zum Download. Damit erhält man ein sehr exaktes Ergebnis. Ein einfaches weißes Blatt Papier genügt übrigens durchaus als Mattscheibe.

Controller

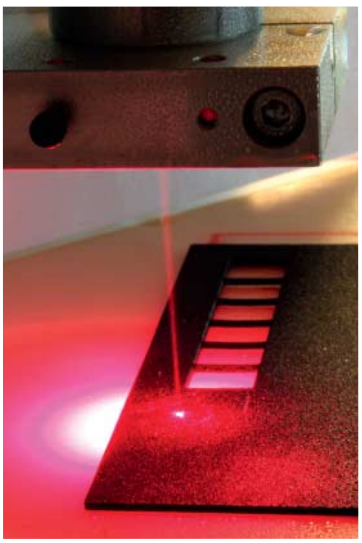
Statt eines Arduino UNO kann man auch einen Arduino Nano zur Steuerung der LED-Streifen nehmen. Der passt zusammen mit den 4 Steuertasten auch besser in ein Handgehäuse. Die Tasten sind direkt mit den Digitalports 8, 9, 10, 11 sowie GND verbunden. Eine dreiadrige Leitung versorgt den Arduino Nano mit Strom (5 V an Pin 27, GND an Pin 29, Bild siehe Link) und das Display über Pin 12 (beim Arduino Uno Pin 13) mit den Daten. Zum Flashen der Software haben wir eine Öffnung für den Mini-USB-Stecker in das Gehäuse gebohrt.

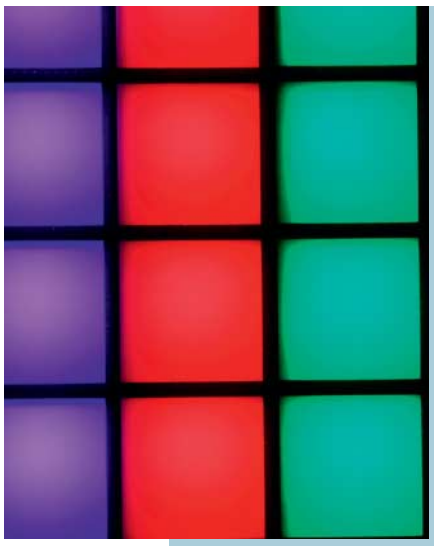
Theoretisch könnten, wenn alle LEDs bei maximaler Helligkeit eingeschaltet werden, 9 A Strom fließen (150 × 60 mA). Da wir jedoch die Helligkeit im Arduino-Sketch nur auf circa 32 Prozent einstellen (LEDS.setBrightness(80)) und nie alle LEDs weiß leuchten, genügt als Stromversorgung ein 5V-Ste-



Rückseite: Links und rechts sind die „Anschlussschienen“ für die Stromversorgung der einzelnen Stripes. Die schwarzen Kabel verbinden jeweils D_{IN} mit D_{OUT}.

Großes Bild: In Reih und Glied: So sollte das Display nach dem Verkleben der Stripes aussehen.





Ein CNC-gesteuerter Laser schneidet aus Moosgummi die Blende.

ckernetzteil mit 1 A. Unsere Messungen zeigten bei diesen Bedingungen maximal 0,9 A an. Betreibt man das Spiel jedoch mit höheren Helligkeitswerten, so muss man unbedingt ein stärkeres Netzteil verwenden.

Den Arduino-Sketch für das Spiel stellen wir Ihnen am Ende des Artikels zum Download bereit. Darin enthalten sind auch Routinen, um Texte darzustellen (Lauftext, Text blenden …) und ein fast kompletter ASCII-Zeichensatz. Zum Übersetzen des Sketches ist die Bibliothek FastLED notwendig (http://fastled.io). Aus dem heruntergeladenen Bibliotheks-Archiv entpacken Sie den Ordner FastLED-3.0.x in das Library-Verzeichnis ihrer Arduino-Installation. Dann laden Sie den Sketch MAKE-BLOCK_WS2812B.ino in die IDE, übersetzen ihn und laden ihn auf den Arduino. Der sollte anschließend automatisch Make:Block starten.

Den Highscore des Spieles speichert der Sketch im EEPROM des Arduino ab, so dass er auch nach dem Ausschalten erhalten bleibt. Wer das Spiel mit einem kleineren oder größeren Display bauen

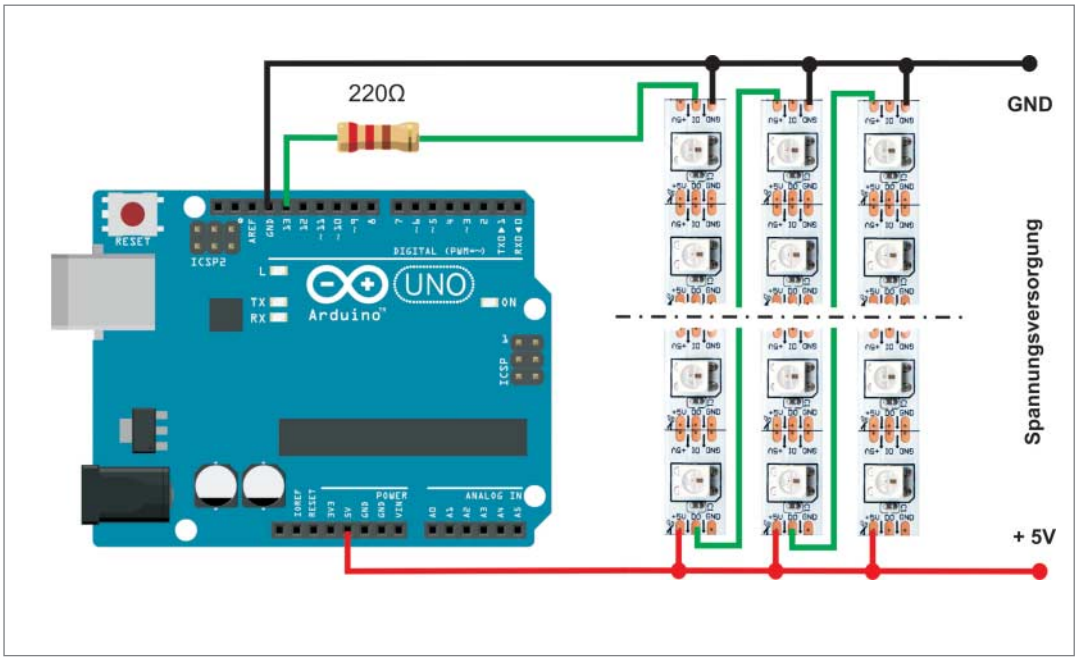
MOOSGUMMI-BLENDE

Wer stolzer Besitzer einer (c't-Hacks-)CNC-Fräsmaschine ist, kann die Blende auch damit anfertigen. So lassen sich passend für einen Tetris-Klon anstelle von runden Löchern auch quadratische Durchbrüche fräsen. Einige Blenden stellten wir mit unserer c't-Hacks-CNC-Fräsmaschine in Verbindung mit einem Eigenbau-Abflussrohr-Laser her. Als Material dienten dann verschiedene Moosgummiplatten, die sich mit dem Laser leicht bearbeiten ließen. Das damit erzielte Ergebnis war äußerst überzeugend.

Erst bohren, dann senken: Für eine schöne Ausleuchtung ist eine genau gefertigte Maske erforderlich.

Die Schablone druckt man sich mit einem Drucker selbst aus.

Zum Schutz der LEDs kommt in die Datenleitung zum Arduino ein Widerstand.

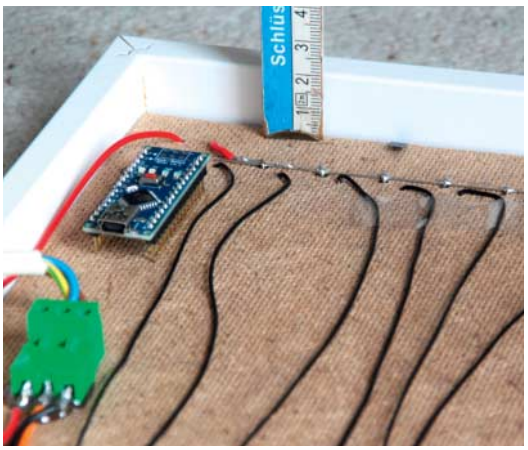


BEDIENUNG

Steuerungstaste vor dem Spiel:
Wird die *Taste Drehen* beim Start des Arduino gedrückt gehalten, wird der Vorspann übersprungen.

Steuerungstasten während des Spiels:
Taste Drehen: Der Stein wird jeweils um 90 Grad gedreht. *Taste Links:* Der Stein wird nach links verschoben. *Taste Rechts:* Der Stein wird nach rechts verschoben. *Taste Runter:* Der Stein wird schnell nach unten bewegt. *Taste Drehen* und *Runter* gleichzeitig: Das Spiel wird angehalten – fortsetzen mit *Taste Runter*. Alle 4 Tasten gleichzeitig: Das Spiel wird sofort neu gestartet.

Steuerungstasten nach dem Spiel:
Taste Rechts: Der Highscore wird angezeigt. *Taste Links:* Die Punkte des letzten Spiels werden angezeigt. *Taste Runter:* Ein neues Spiel wird gestartet.

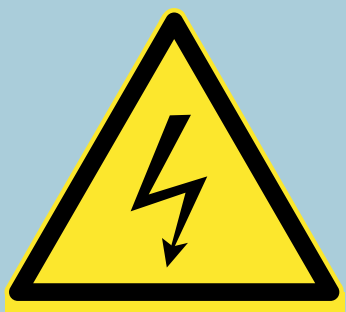


Für stationäre Anwendungen ohne Bedienung kann man den Nano hinter dem Rahmen verstecken.

möchte, muss nur die Werte für field_width und field_height am Beginn des Sketches abändern.

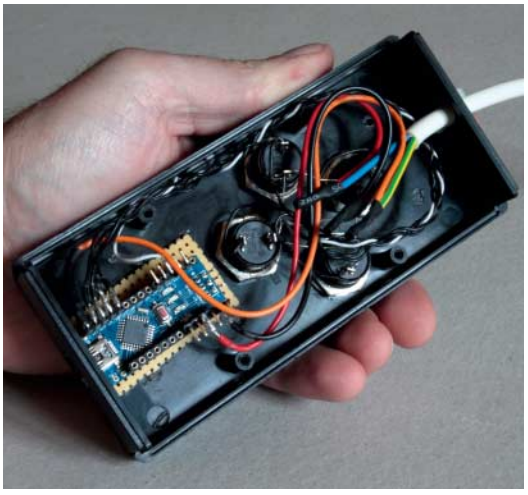
Weitere Spiele wie das beliebte Pong oder ähnliche Retrospiele lassen sich nach dem gleichen Prinzip sicher leicht selber programmieren. Posten Sie Ihren Sketch doch im Forum zu diesem Artikel (siehe Link). Hängt das Display später im Wohnzimmer, so kann es – wenn gerade nicht gespielt wird – auch die Uhrzeit, schönes Ambiente-Licht oder Ähnliches anzeigen.

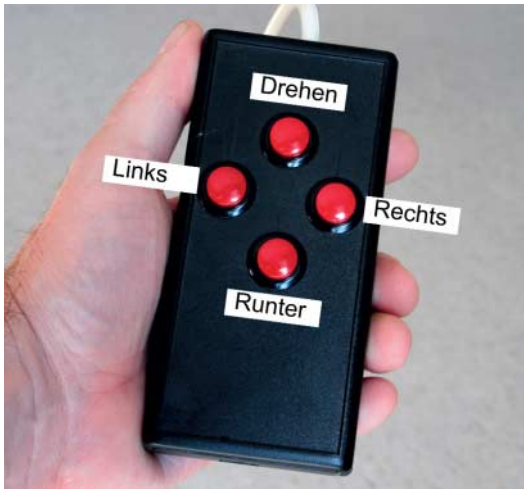
Hinter dem Panel bleibt auch noch genügend Raum, um dort direkt einen Arduino zu verbauen. Nach diesem Konzept ist es leicht, weitere Designobjekte zu entwerfen und zu bauen. Enthusiasten der Arduino-Group aus Hannover haben beispielsweise eine Videowand mit 4128 LEDs gebaut. Der Hackerspace Bremen realisierte ein Jump 'n' Run Game auf einem Display mit 1800 Pixeln.



VORSICHT:

Betreiben Sie das Display nicht über die Stromversorgung des Arduino. Er ist für diese Leistung nicht vorgesehen, sodass der Spannungsregler dabei durchbrennen würde.





Das Gehäuse nimmt den Arduino und die Steuertasten auf.

So funktionierts

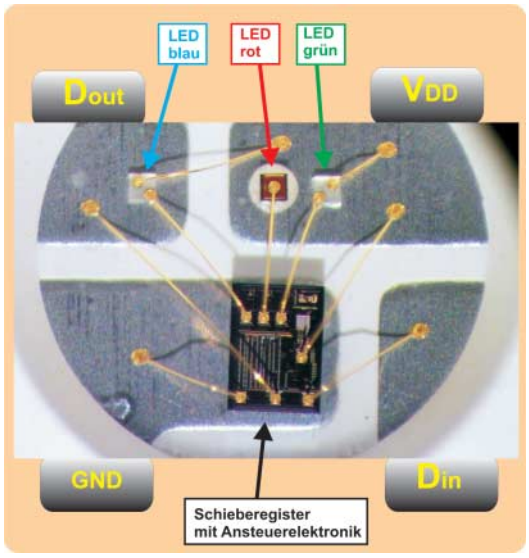
Grundsätzlich besteht der WS2812B aus einer RGB-LED (5050-RGB) kombiniert mit einem 3×8-Bit-Schieberegister-IC mit Pulsbreitenmodulation. Mit der Lupe lässt sich dieses winzige IC noch gut erkennen. Die Bezeichnung 5050 bezieht sich übrigens auf die Größe des RGB-LED-Gehäuses, das genau 5,0 mm × 5,0 mm misst.

Die Helligkeit jeder einzelnen Farbe kann in 255 Stufen geregelt werden. Damit lassen sich theoretisch 16 777 216 Farben mischen. Möchte man jedoch auch die Helligkeit der Gesamtkomposition steuern, so ergeben sich dann daraus weit weniger Farben (aber immer noch mehr als genug). Das Schieberegister des WS2812B nimmt pro LED jeweils 8 Bit (1 Byte) entgegen, also insgesamt 24 Bit. Weitere Bits werden an den D_{OUT}-Pin beziehungsweise an den nächsten Chip weitergeleitet. Damit lassen sich leicht beliebig lange Schieberegister-Ketten zusammenschließen. Solche Ketten werden als „Daisy Chain" (engl.: Gänseblümchenkette) bezeichnet.

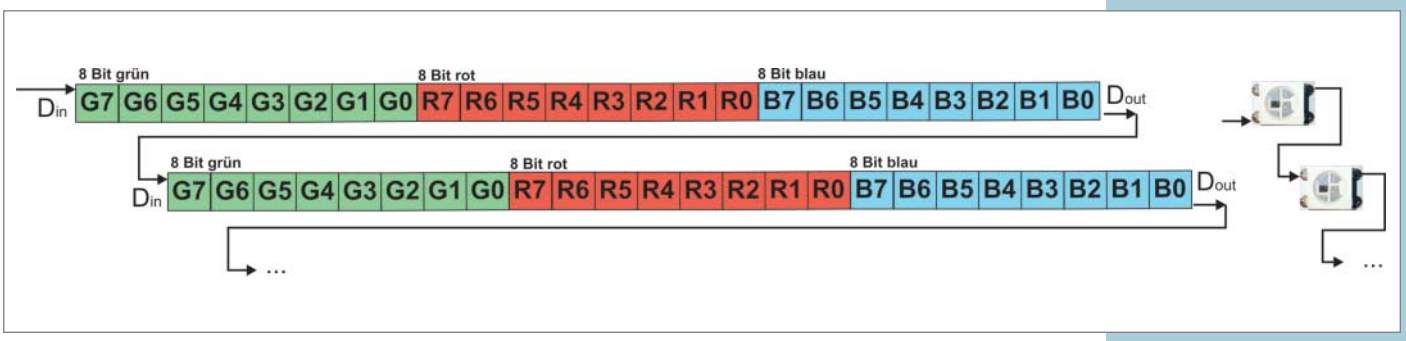
Die Datenübertragungsrate beträgt bei diesem LED-Streifen 800 kbps und kann nicht verändert werden. Anders als die meisten Schieberegister besitzt der WS2812B nur eine Datenleitung. Ein Anschluss für die Taktung ist nicht vorhanden. Ein Blick in das Datenblatt verrät, dass der WS2812B die Daten mit einer vorgegebenen „Data Transfer Time" erwartet. Dieses Timing muss exakt eingehalten werden. Zu hohe Abweichungen im Datenstrom – auch Jitter genannt – führen unweigerlich zu Fehlfunktionen, etwa zu wirren Irrlichtern der LEDs. Die maximale Data-Transfer-Toleranz gibt der Hersteller mit 0,15 µs an.

Aus diesem Grund muss man alles verhindern, was den Datenstrom unterbrechen würde. Insbesondere müssen mögliche Interrupts vor der Datenübertragung deaktiviert werden. Danach kann man sie jedoch wieder zulassen, da sich die Schieberegister ihren Wert solange merken, bis sie einen neuen Wert erhalten. Reihen aus WS2812-LEDs gibt es als flexible LED-Streifen mit rückseitig aufgebrachter Klebeschicht im Internet zu kaufen. Dabei werden Streifen mit 30 LED/Meter, 60 LED/Meter und 144 LED/Meter angeboten. Sie können nach jeder LED problemlos aufgetrennt oder auch wieder zusammengelötet werden. Die Stripes gibt es mit weißem oder schwarzem Basismaterial und sogar wasserdicht vergossen.

Der WS2812B ist für eine Spannung zwischen 3,5 V und 5,3 V ausgelegt. Dabei spielt es keine Rolle, von welcher Seite des Streifens man sie einspeist. Der Strombedarf einer einzelnen LED liegt bei etwa 20 mA (pro Farbe!). Daraus ergeben sich bei einem ein Meter langen 60 LED/Meter-Strip eine maximale Leistung von 3,6 A (gemessen haben wir 3,3 A). Bei einem 4 Meter langen Stripe können dann bereits mehr als 14 A über die dünnen Leiterbahnen fließen. Bei unserem 2-Meter-LED-Streifen mit insgesamt

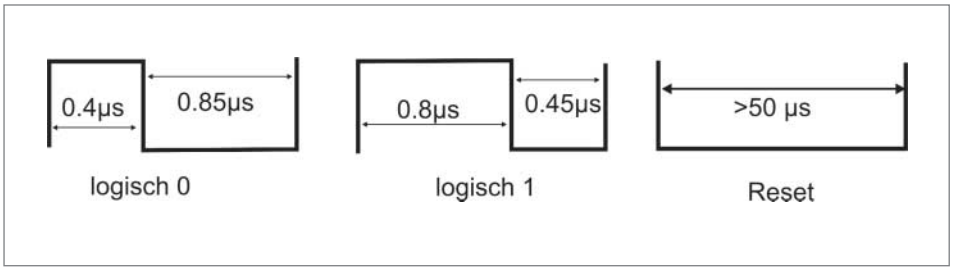


Die RGB-LEDs im Detail: Drei verschiedenfarbige LEDs und ein Schieberegister sitzen im Gehäuse.



Jedes Schieberegister ist 24 Bit breit. Bei unseren 150 Pixeln muss der Arduino insgesamt 3600 Bits generieren.

Der Arduino muss das Timing des Datensignals sehr genau einhalten.



120 WS2812B betrug der Spannungsabfall am Ende des Strips bereits 0.8 V. Bei einem 4-Meter-Strip kommen an der letzten LED dann nur noch 3,8 V an. Deshalb sollte man spätestens ab 4 Meter Länge beziehungsweise ab 240 LEDs die Versorgungsspannung erneut einspeisen. Wir empfehlen dies jedoch schon nach etwa 180 LED.

Da jeder WS2812B-Chip das Datensignal für den nächsten Chip ausgibt, gibt es prinzipiell keinen Verlust auf der Datenleitung. Somit ist die maximale Anzahl der LED, die an einem Strang betrieben werden können, praktisch unbegrenzt.

Immer wieder liest man von „Qualitätsproblemen“ des WS2812B (siehe Link). Unseren Recherchen zufolge ist dies jedoch meist auf sogenannte Overshots zurückzuführen, die die erste LED im Strang beschädigen. Auch bei unseren Versuchen starb einmal die erste LED im Strang. Dies passierte, als wir die 5V abklemmten, GND und die Datenleitung aber noch am Arduino hingen. Die LEDs glimmten dann immer noch vor sich hin. Die Versorgungsspannung floss offensichtlich über die Datenleitung in den Strang. Abhilfe schafft hier ein 220-Ohm-Widerstand, der zwischen Controller und LED-Strip in die Datenleitung gelötet wird.

Befindet sich dennoch einmal eine defekte LED im Strang, so schneidet man mit einer Schere den betreffenden Teil heraus und lötet ein neues Stück in den Strang. Dabei darf der kleine SMD-Kondensator in dem Stück aber nicht beschädigt werden. Auch bei neuen Streifen findet man oft mehrere Stellen, an denen die Streifen bereits neu verbunden wurden.

Programmierung

Das oben beschriebene Timing programmiert man am sinnvollsten direkt in Assembler. Damit müssen sich Maker aber nicht unbedingt selbst aufhalten, da es für den Arduino schon zahlreiche Bibliotheken zum Download gibt. Bei unseren Tests funktionierten aber nicht alle Bibliotheken gleichermaßen gut. Vor allem ältere Codes bildeten das Timing nicht immer exakt genug ab, um den WS2812B fehlerfrei zu betreiben. Gerade bei längeren Streifen zeigten sich mit älteren Bibliotheken Fehlfunktionen. Bei GitHub gibt es die Bibliothek FastLED zum Download, die bei unseren Versuchen am besten funktionierte (https://github.com/FastLED/FastLED). Sie ist abwärtskompatibel und arbeitet somit auch zum Beispiel mit WS2810, WS2811, Neopixel und vielen anderen LED-Chips zusammen.

Die Programmierung mit Arduino ist dann fast schon trivial:

```
#include "FastLED.h" //Library einbinden
#define NUM_LEDS 120 // Anzahl der LEDs
#define DATA_PIN 13 // Dateneingang am Arduino für den
WS2812B-Din
CRGB leds[NUM_LEDS]; // Array der LED'S mit je 1 Byte
pro Farbe
void setup()
{
  FastLED.addLeds<WS2812B, DATA_PIN, RGB>
(leds, NUM_LEDS); //LED-Array initialisieren
  LEDS.setBrightness(100); //Helligkeit auf 100 %
setzen
}
void loop()
{
  for (int i = 0 ; i < NUM_LEDS ; i++)
  {
     leds[ i ] = CRGB(0 , 255 , 0); // Rote LED setzen
(green, red, blue)
     // leds[ i ] = CRGB::Red; // Macht das gleiche wie
die vorhergehende Zeile
     LEDS.show(); // Jetzt werden die Daten in den
Strip übertragen
     delay(300); // Kurz warten, wir wollen ja etwas
sehen
  }
  LEDS.clear(); // Alle LED wieder ausschalten
  for (int i = 0 ; i < NUM_LEDS ; i++)
  {
     leds[ i ] = CRGB::Blue; // Blaue LED setzen
     LEDS.show(); // Jetzt werden die Daten in den
Strip übertragen
     leds[ i ] = CRGB::Black; // Für den nächsten
Durchlauf wieder ausschalten
     delay(300); // Kurz warten, wir wollen ja etwas
sehen
  }
}
```

An der zweiten for-Schleife erkennt man deutlich, dass der Befehl `leds[ i ] = CRGB::Blue;` noch nichts bewirkt. Erst durch den Befehl `LEDS.show();` werden die Zustände aller LEDs an den Stripe übertragen. Deshalb kann für den nächsten Schleifendurchlauf sofort die LED wieder „ausgeschaltet“ beziehungsweise der Zustand Schwarz in das Array gespeichert werden, ohne dass sie gleich erlischt. *—dab*

Links und Foren
make-magazin.de/xe36