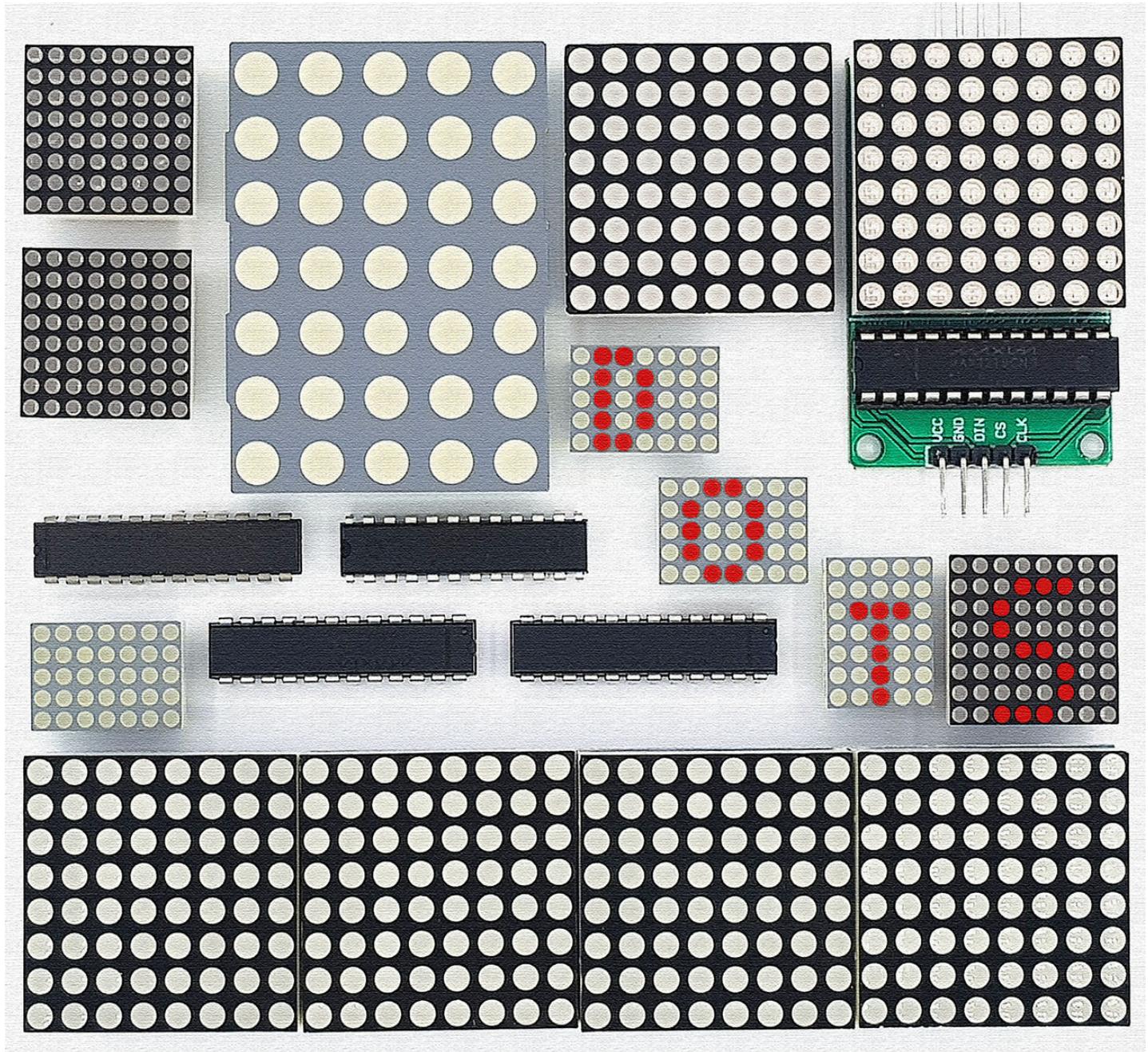


# LED Matrix Display ansteuern • Wolles Elektronikkiste



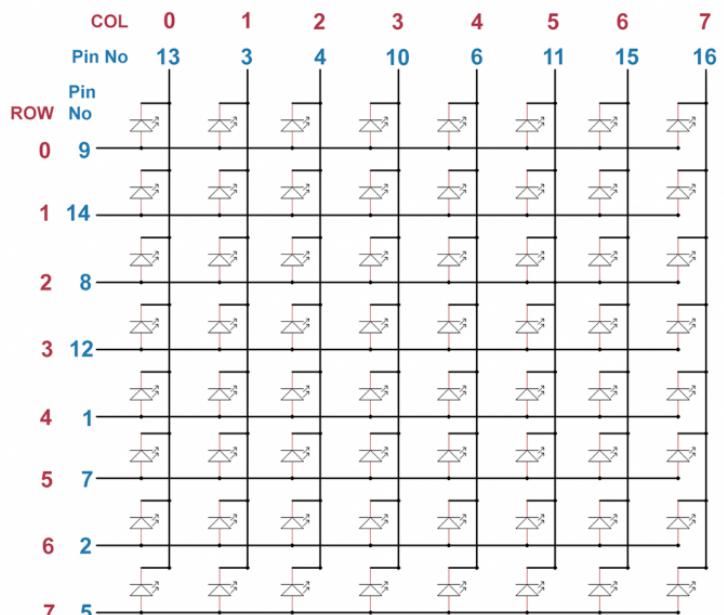
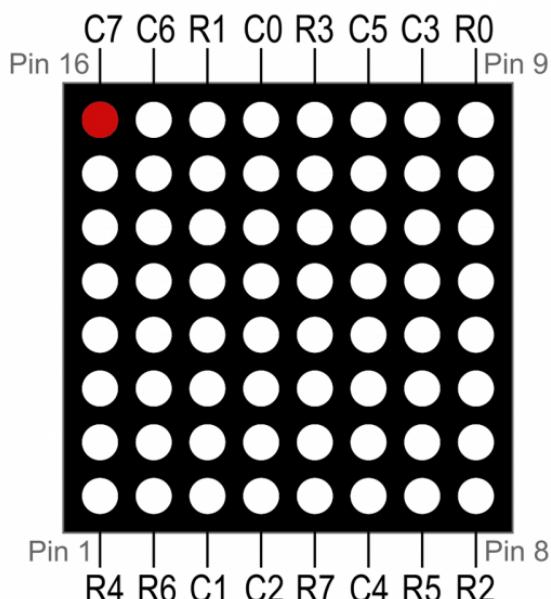
## Über den Beitrag

In diesem Beitrag möchte ich euch Methoden vorstellen, wie ihr ein LED Matrix Display (engl.: Dot Matrix Display) ansteuern könnt. Das ist natürlich nicht der erste Beitrag zu diesem Thema im Netz, aber vielleicht gibt es doch noch das eine oder andere Neue für euch zu entdecken. Im Einzelnen behandle ich die folgenden Themen:

- Was ist ein LED Matrix Display und wie funktioniert es?
- Direkte Ansteuerung mit einem Arduino
- Ansteuerung über den MAX7219 / MAX7221 (mit Bibliothek)
- Verwendung von Displays mit integriertem MAX7219
- Programmierung einer Laufschrift
- Ansteuerung über den MAX7219 / MAX7229 ohne Bibliothek
- Anhang: Ansteuerung eines zweifarbigen, TA6932-basierten Matrix Displays

Als Appetizer hier schon mal ein kleines Video:

## Was ist ein LED Matrix Display und wie funktioniert es?



8x8 LED Matrix Display mit Reihen-Eingang und Spalten-Ausgang, Rot: LED (0/0)

Ein LED Matrix Display ist ein LED Dioden Array. Jeweils ein Eingangs- bzw. Ausgangspin verbindet die LEDs einer Reihe (Row) bzw. einer Spalte (Column). Ob die Reihen oder die Spalten die Ein- oder Ausgänge sind, variiert von Display zu Display.

Durch Anlegen einer Spannung an einen Eingangspin und durch Verbinden eines Ausgangspins mit Ground bringt ihr genau eine LED am Schnittpunkt zum Leuchten.

Wenn ihr keinen Ansteuerungs-IC wie den MAX7219 oder MAX7221 verwendet, dann müsst ihr Vorwiderstände einsetzen. Die benötigte Größe hängt von den technischen Daten eurer Displays ab. Meistens liegt ihr aber bei 5 Volt mit 330 Ohm schon richtig.

Noch eine Anmerkung zum Schema oben: normalerweise findet man in Datenblättern die Reihen- und Spaltennummern von 1 bis 8. Gewöhnt euch lieber die Nummerierung 0 bis 7 an. Denn dann denkt ihr gleich „richtig“ in Bits und Bytes und das ist bei der Ansteuerung hilfreich.

Und noch etwas: wenn ich eine LED Position (einen Dot) in diesem Beitrag als Wertepaar (a/b) angebe, dann ist a die Reihe und b die Spalte. Also nicht wie x und y im Koordinatensystem.

## Woher bekommt Ihr LED Matrix Displays?

LED Matrix Displays gibt es in vielen verschiedenen Ausfertigungen. Dabei variiert die Anzahl der LEDs (Dots), die Größe der Displays und die Farbe der LEDs. Darüber hinaus gibt es sie mit integriertem Ansteuerungs-IC und teilweise auch im Verbund, z.B. als 8 x 8 x 4. Sucht in dem Online-Shop eures Vertrauens nach „LED Matrix Display“ oder „Dot Matrix“. Die Preise variieren recht stark mit der Herkunft und der Menge.

## Direkte Ansteuerung per Microcontroller oder Arduino

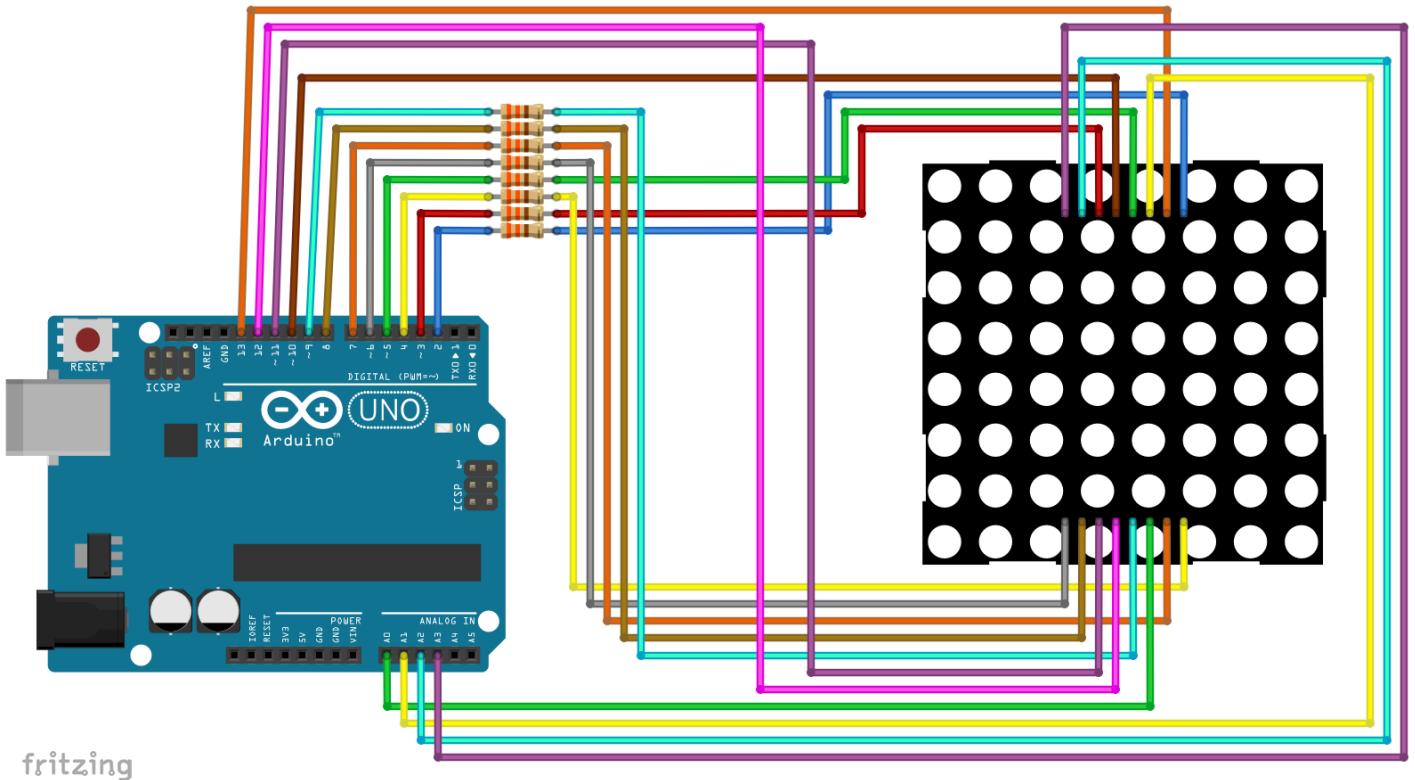
Ich verwende als Beispiel das oben schematisch abgebildete 8x8 LED Matrix Display mit Reihen-Eingang und Spalten-Ausgang. Vorab: die Verdrahtung macht keinen Spaß, da acht Eingänge, acht Ausgänge und acht Widerstände verbunden werden müssen. Außerdem sind die Reihen und Spalten den Pins ohne erkennbares System zugeordnet. Da müsst ihr ziemlich viel suchen. Teilweise ist auch nicht offensichtlich welches der Pin 1 ist. In diesem Fall müsst ihr erstmal ein wenig ausprobieren.

Der ganze erste Abschnitt dient denn auch mehr der Didaktik als der Praxis. Ansteuerungs-ICs wie der MAX7219 machen das Leben leichter. Und noch komfortabler sind die Displays mit integrierter Ansteuerung. Da komme ich dann später zu. Trotzdem sollte man erstmal das Prinzip verstanden haben.

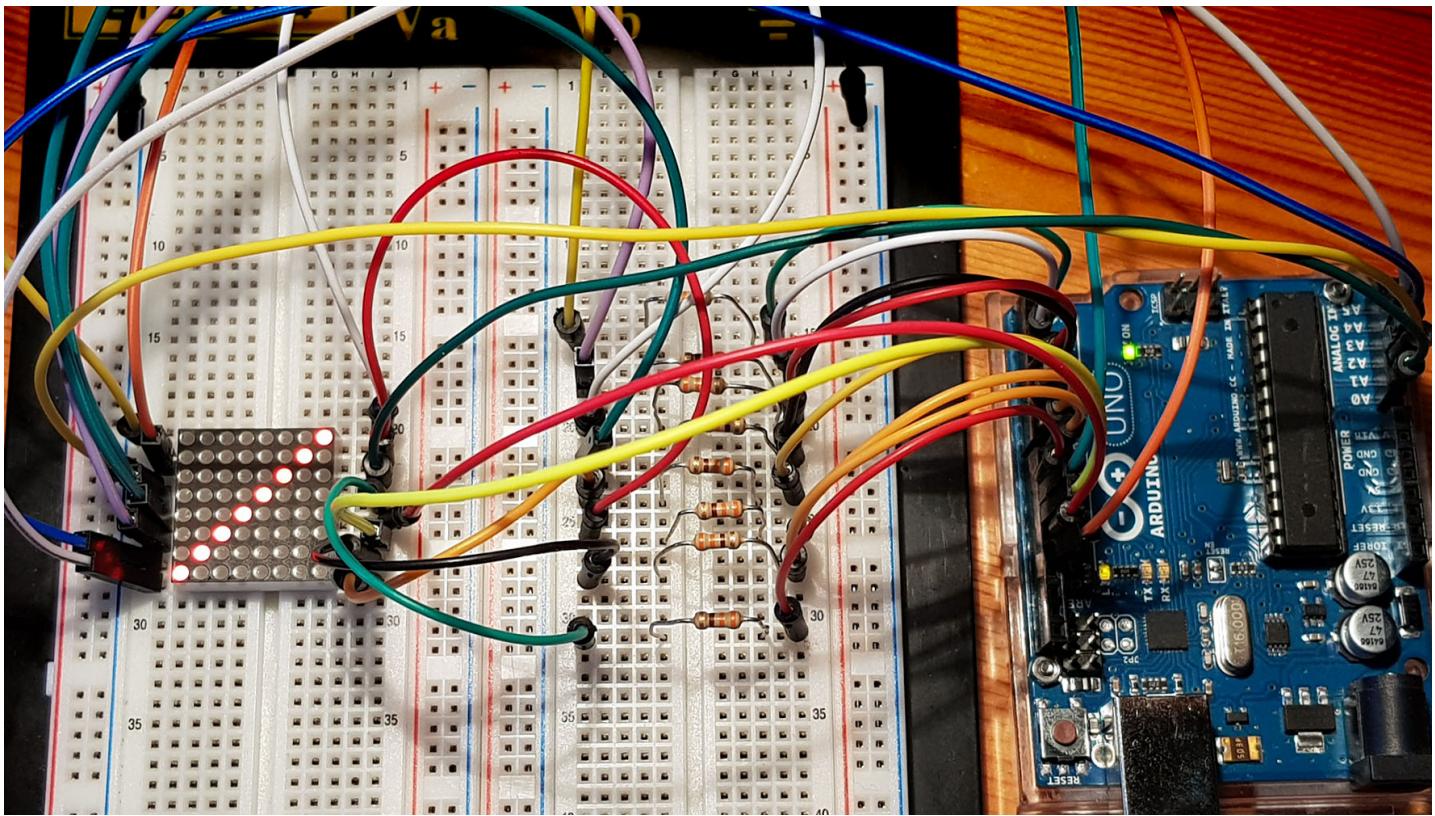
# Beschaltung

Bei 16 benötigten Ein- und Ausgängen müssen auf dem Arduino auch die analogen Pins mit einbezogen werden. Oder ihr verwendet eine Portweiterung wie den [MCP23017](#).

So sieht die Beschaltung schematisch und praktisch aus:



Beschaltung für ein 8x8 LED Matrix Display



Verkabelung in der Praxis - eine ziemliche "Wurschtelei"

## Ein kleiner Beispielsketch

Im folgenden Beispielsketch werden zunächst die Arduino Pins für die Reihen und Spalten definiert. Die Eingänge (Reihen) werden auf LOW gesetzt, die Ausgänge (Spalten) auf HIGH. Damit sind alle LEDs aus. Eine LED an der Stelle „r/c“ wird angeschaltet, wenn die Reihe „r“ auf HIGH gesetzt wird und die Spalte „c“ auf LOW.

Als erste kleine Spielerei gibt es ein Lauflicht. Dabei geht jede LED einmal kurz an. Dann wird die Diagonale von (0/0) bis (7/7) zum Leuchten gebracht.

## Unerwünschte Querbeeinflussung

Ersetzt im letzten Sketch einmal die Sketch Hauptschleife durch die folgende:

```
switchLED(row[0],column[0],1);  
switchLED(row[0],column[1],1);
```

```
switchLED(row[1],column[0],1);

void loop() { switchLED(row[0],column[0],1); delay(1000);
switchLED(row[0],column[1],1); delay(1000); switchLED(row[1],column[0],1);
delay(1000); clearDisplay(); }
```

Eigentlich soll der Sketch die Dots (0/0), (0/1) und (1/0) anschalten. Das tut er zwar auch, aber mit dem Dot (1/0) wird auch (1/1) angeschaltet. Falls euch nicht klar sein sollte, warum das so ist, schaut dazu noch einmal in das Schema des 8x8 Displays. Dann sollte die Problematik offensichtlich sein.

## Die Lösung: Multiplexing

Ersetzt nun noch einmal die Hauptschleife durch die folgende:

loop\_with\_several\_dots\_multiplexed

```
switchLED(row[0],column[0],1);
```

```
switchLED(row[0],column[0],0);
```

```
switchLED(row[0],column[1],1);
```

```
switchLED(row[0],column[1],0);
```

```
switchLED(row[1],column[0],1);
```

```
switchLED(row[1],column[0],0);
```

```
void loop() { switchLED(row[0],column[0],1); delay(1);
```

```
switchLED(row[0],column[0],0); switchLED(row[0],column[1],1); delay(1);
```

```
switchLED(row[0],column[1],0); switchLED(row[1],column[0],1); delay(1);
```

```
switchLED(row[1],column[0],0); }
```

In diesem Fall leuchten tatsächlich nur die Dots (0/0), (0/1) und (1/0), da jeder Dot separat für kurze Zeit angeschaltet wird. Ein Flackern sieht man nicht, dafür ist der Wechsel zu schnell.

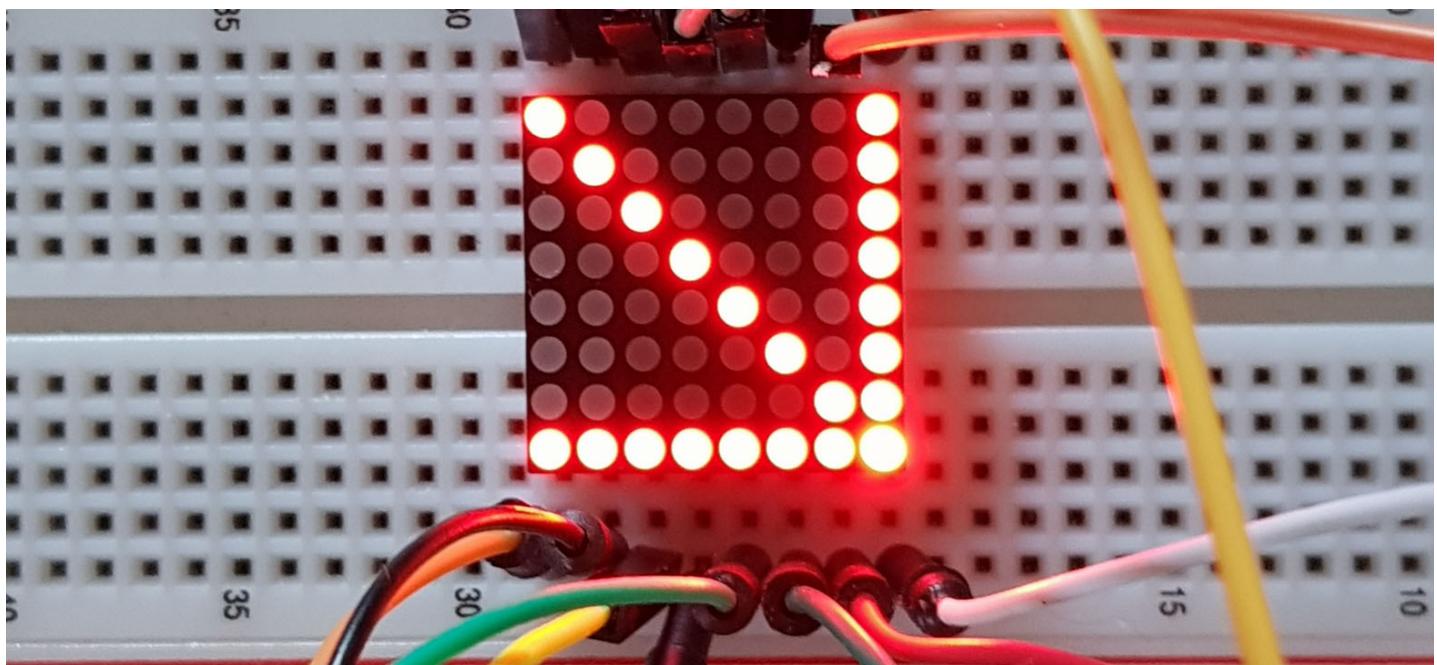
## Die Grenzen des einfachen Multiplexings

Zunehmendes Multiplexing in dieser Form führt zu einer Abnahme der Intensität. Im folgenden Beispiel ist der Dot (7,7) immer an, der Rest der Diagonale ist „gemultiplext“. Dass der Rest von Reihe und Spalte 7 auch leuchten, ist ein Nebeneffekt.

Zu beachten: Die Sketchzeilen 1 und 2 gehören noch ins Setup.

```
switchLED(row[7],column[7], 1);  
switchLED(row[i],column[i],1);  
switchLED(row[i],column[i],0);  
switchLED(row[7],column[7], 1); } void loop() { for(int i=0; i<=6; i++){  
switchLED(row[i],column[i],1); delay(1); switchLED(row[i],column[i],0); } }
```

Auf dem Foto erkennt man die Unterschiede nicht so gut wie in der Realität. LEDs sind nicht einfach zu fotografieren. Dennoch könnt ihr vielleicht erahnen, dass der Dot (7,7) heller ist.



"Immer an" vs. "gemultiplext"

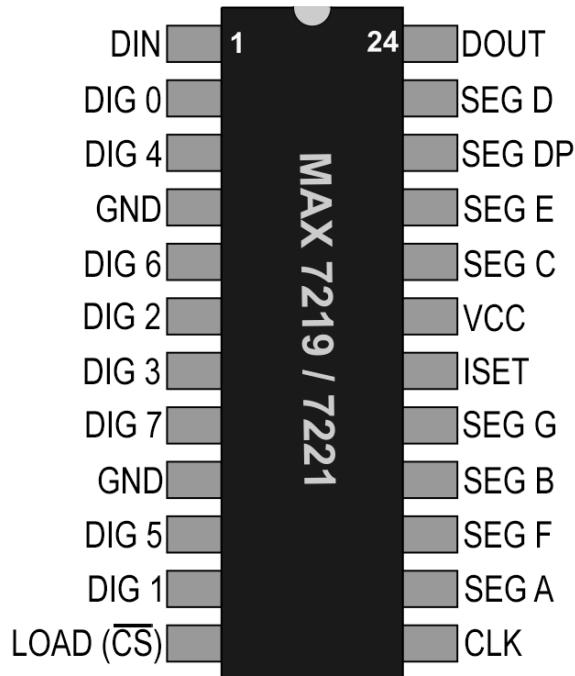
Auch für dieses Problem gibt es eine Lösung. Über Pulsbreitenmodulation (PWM, siehe mein Beitrag über [Timer](#)) lässt sich die Intensität je nach Anzahl der „gemultiplexten“ LEDs gewichten. Darauf gehe ich aber jetzt nicht näher ein und möchte die bequemere und allgemein übliche Lösung vorstellen.

## Ansteuerung mit dem MAX7219 / MAX7221

Alle eben genannten Probleme lassen sich mit den Ansteuerungs-ICs MAX7219 oder MAX7221 lösen. Da beide fast identisch sind, nenne ich ab jetzt nur den MAX7219, meine aber beide.

Primär ist der MAX7219 für die Ansteuerung von 7-Segment Displays konzipiert. Er lässt sich aber auch für die Ansteuerung von LED Matrix Displays oder losen LED Arrangements verwenden. Das ist auch nicht weiter verwunderlich, da die Ansteuerung mehrerer 7-Segment Anzeigen im Grunde dieselbe Herausforderung wie die Ansteuerung eines LED Matrix Displays darstellt.

## Pinout und technische Eigenschaften des MAX7219



Pinout des MAX7219 / MAX7221; CS gilt nur für den MAX7221

Der MAX7219 besitzt 24 Pins:

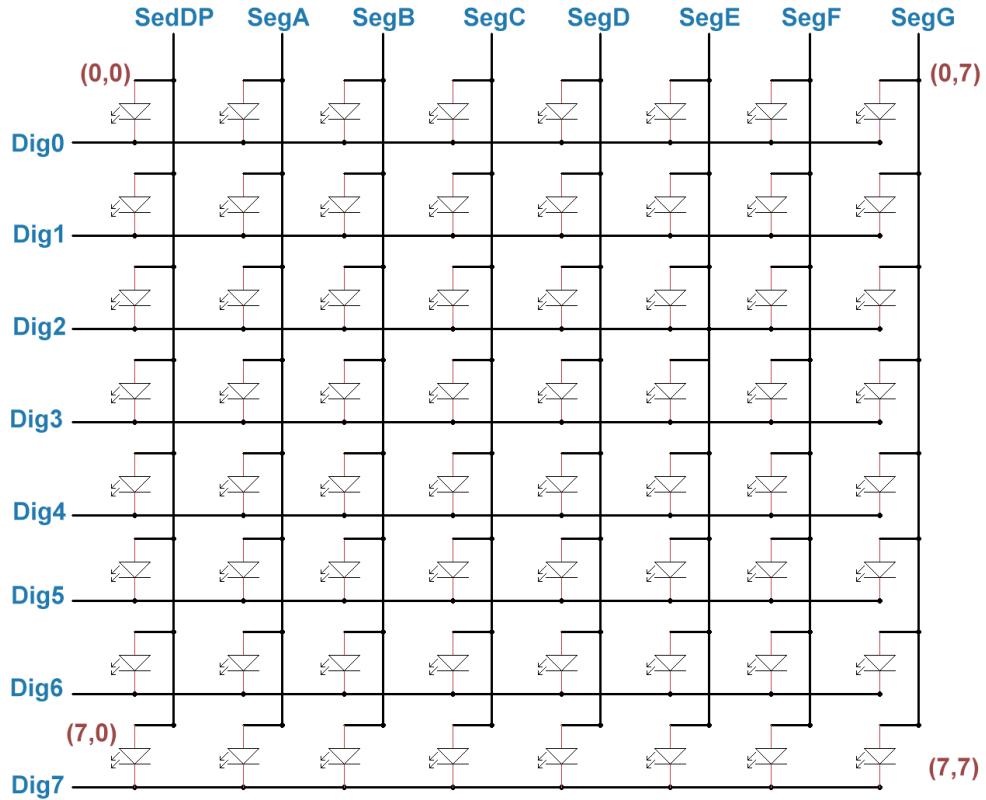
- Ob diese mit den Reihen- oder Spaltenpins verbunden werden, hängt davon ab, wie herum die LEDs im Display verbaut sind.
- Die Pins SEG X liefern die positive Spannungsversorgung
- Ob diese mit den Reihen- oder Spaltenpins verbunden werden, hängt davon ab, wie herum die LEDs im Display verbaut sind.
- Die Pins DIG X schalten nach GND.
- DIN ist der Data Input, also MOSI (Master Out, Slave In).
- Ein MISO (Master In, Slave Out) gibt es nicht; es ist also eine One-Way Kommunikation.
- CLK: Clock Pin.
- LOAD / CS: Chip Select Pin.
- Der MAX7219 wird per SPI angesteuert:

- DIN ist der Data Input, also MOSI (Master Out, Slave In).
- Ein MISO (Master In, Slave Out) gibt es nicht; es ist also eine One-Way Kommunikation.
- CLK: Clock Pin.
- LOAD / CS: Chip Select Pin.
- Wenn ihr mehrere Displays verwendet, wird DOUT mit dem DIN des nächsten Displays verbunden.
- An VCC dürfen 4 bis 5.5 Volt verwendet werden
- An ISET wird die Höhe des Stroms für die LEDs eingestellt. Ich habe 10 kOhm verwendet. Im [Datenblatt](#) des MAX7219 gibt es eine grafische Darstellung, welcher Strom mit welchem Widerstand bei welcher Spannung bereitgestellt wird.

## Verwendung der Bibliothek LedControl

Zur Ansteuerung des MAX7219 verwende ich die Bibliothek LedControl von [Eberhard Fahle](#), die ihr von [Github](#) oder direkt über die Bibliotheksverwaltung der Arduino IDE installieren könnt. Eine sehr gute Einführung gibt es [hier](#).

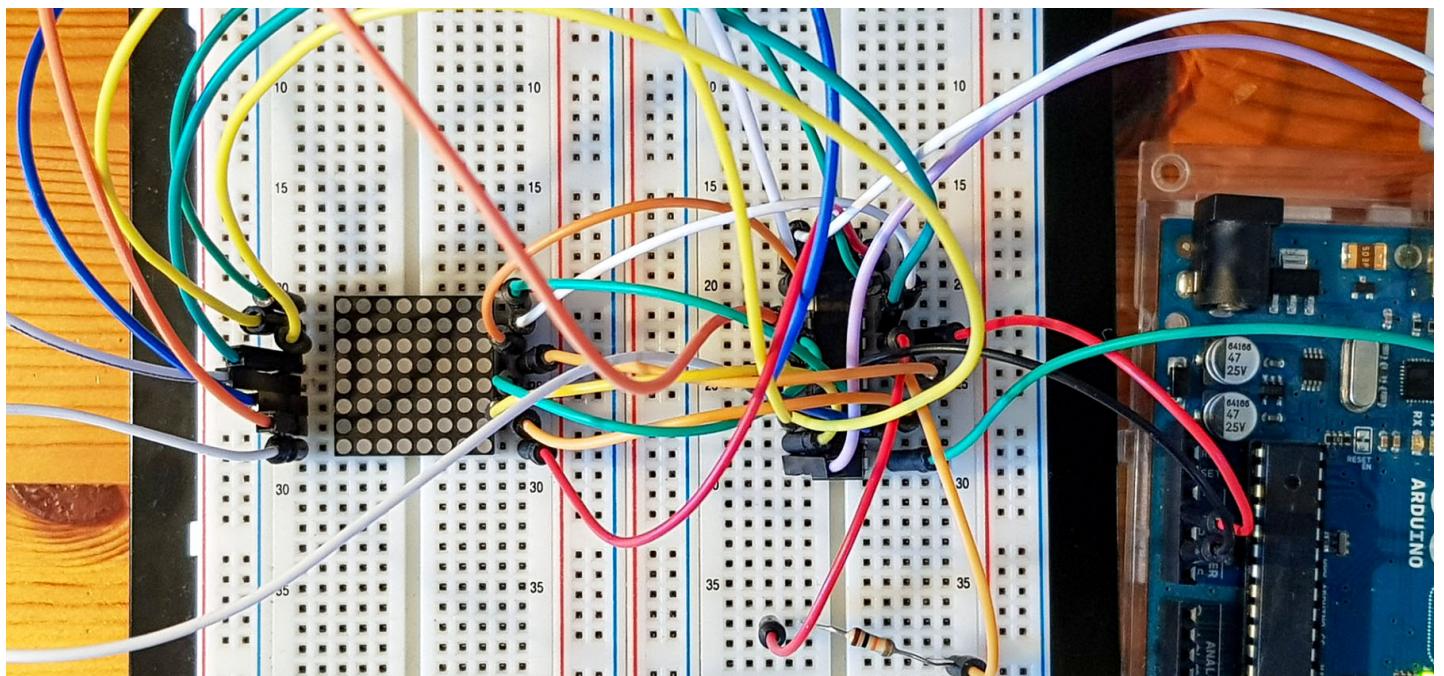
Die Bibliothek erwartet eigentlich den Spannungseingang an den Spalten und den Spannungsausgang an den Reihen. Das ist bei dem von mir verwendeten Modell anders herum. Man kann einfach entsprechend andersherum verkabeln (Segx an die Reihenpins, Digx an die Spaltenpins), muss dann aber nachher noch etwas umdenken.



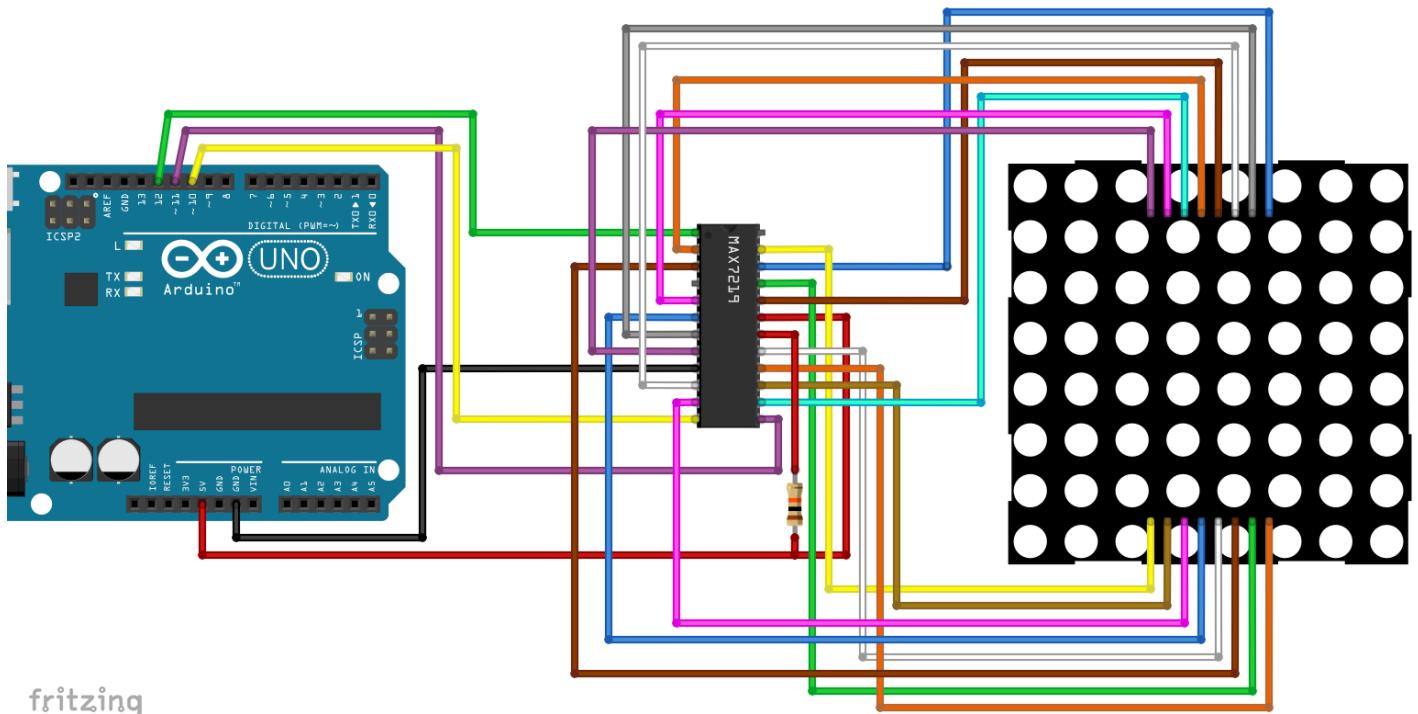
Diese LED Matrix Display Struktur erwartet die Bibliothek LedControl

Der MAX7219 besitzt nicht sonderlich viele Register. Deshalb lässt er sich auch ohne all zu viel Aufwand ohne Bibliothek ansteuern. Wie das geht, zeige ich am Ende dieses Beitrages.

Was die Komplexität der Verkabelung des LED Matrix Displays angeht, bringt der MAX7219 noch keinen großen Vorteil, aber immerhin braucht ihr die Vorwiderstände nicht:



## Verkabelung eines 8x8 LED Matrix Displays mit dem MAX7219



8x8 LED Matrix Display mit MAX7219 am Arduino - Fritzing Schema

Zu beachten ist, dass die oben abgebildete Verkabelung spezifisch für das von mir verwendete LED Matrix Display ist. Vielleicht sind die Anschlüsse eures Displays gleich, vielleicht aber auch nicht.

## Ein kleiner Beispielsketch

LedControl\_Max7219\_test.ino

```
//12= data pin(DIn), 11= CLK Pin, 10= Load/CS Pin, 1 = num of devices
```

```
LedControl lc88=LedControl(12,11,10,1);
```

```
lc88.shutdown(0,false); // Wake up! 0= index of first device;
```

```
for(int row=0; row<=7; row++){
```

```
lc88.setLed(0,row,0,true);
```

```
for(int col=0; col<=7; col++){
```

```
lc88.setLed(0,0,col,true);
```

```
#include <LedControl.h> //12= data pin(DIn), 11= CLK Pin, 10= Load/CS Pin, 1 = num of devices
LedControl lc88=LedControl(12,11,10,1);
void setup(){
lc88.shutdown(0,false); // Wake up! 0= index of first device;
lc88.setIntensity(0,2); lc88.clearDisplay(0); delay(500); }
void loop(){ for(int row=0; row<=7; row++){ lc88.setLed(0,row,0,true); delay(250); } for(int col=0; col<=7; col++){ lc88.setLed(0,0,col,true); delay(250); } delay(500);
lc88.clearDisplay(0); delay(2000); }
```

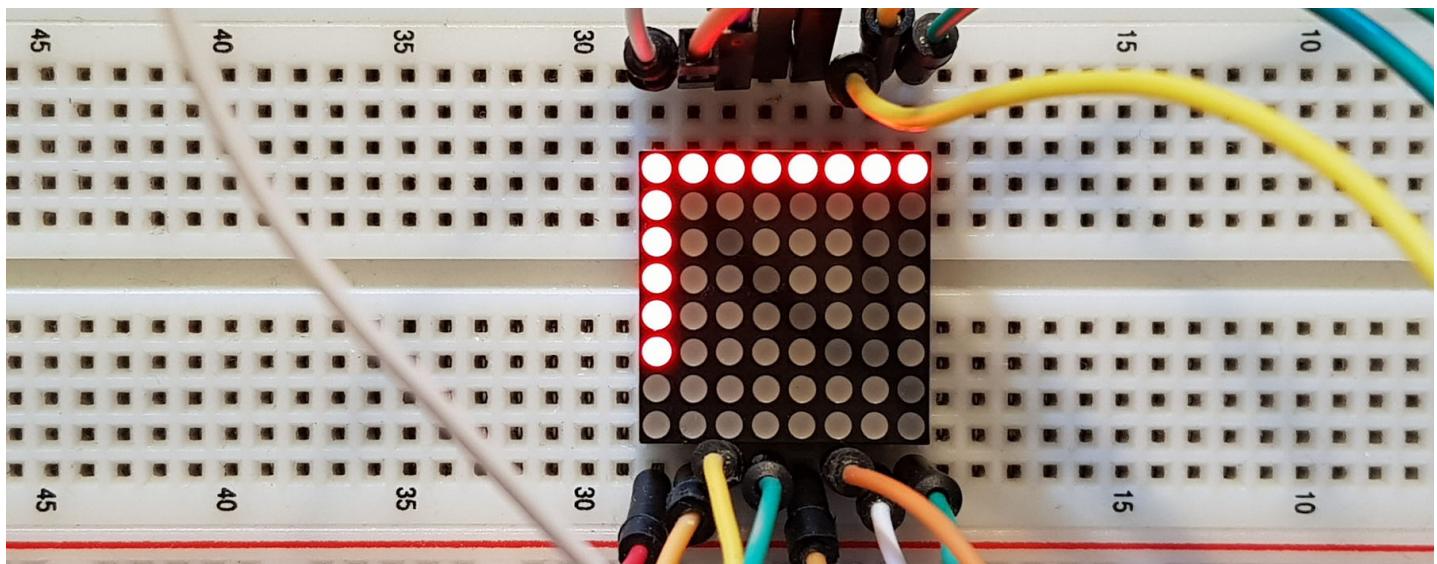
An diesem Beispiel sollten die meisten Funktionen der Bibliothek schon klar werden:

- zunächst wird ein Display Objekt erzeugt und dabei die Anschluss Pins und die Zahl der Einzeldisplays festgelegt
- shutdown(display, false) weckt das Display; mit true geht es schlafen
- setIntensity(display, value) legt die Helligkeit der Dots fest mit  $0 \leq \text{value} < 16$
- clearDisplay(display) schaltet alle Dots eines Displays aus
- setLed(display, row, col, true/false) schaltet genau einen Dot in Reihe row und Spalte col an oder aus.

Weitere nützliche Funktionen, die ihr selber ausprobieren könnt, sind:

- setRow(display, row, val) – die Reihe row eines Displays wird mit dem Wert val belegt, z.B. 0b11110000 -> die ersten vier LEDs sind an, die letzten vier aus
- diese Funktion ist erheblich langsamer als die setRow-Funktion, da sie die setLed-Funktion (achtmal pro setColumn) verwendet.
- setColumn(display, col, val) – wie setRow, nur eben zum Setzen von Spalten
- diese Funktion ist erheblich langsamer als die setRow-Funktion, da sie die setLed-Funktion (achtmal pro setColumn) verwendet.

Da ich bei meinem LED Matrix Display die SEG und die DIG Anschlüsse vertauschen musste, sind auch Reihen und Spalten vertauscht. Eigentlich sollte der Sketch erst die Spalte 0 und dann die Reihe 0 füllen. Offensichtlich ist es anders herum:



LedControl\_Max7219\_test.ino: Spalten und Zeilen sind an diesem Display vertauscht

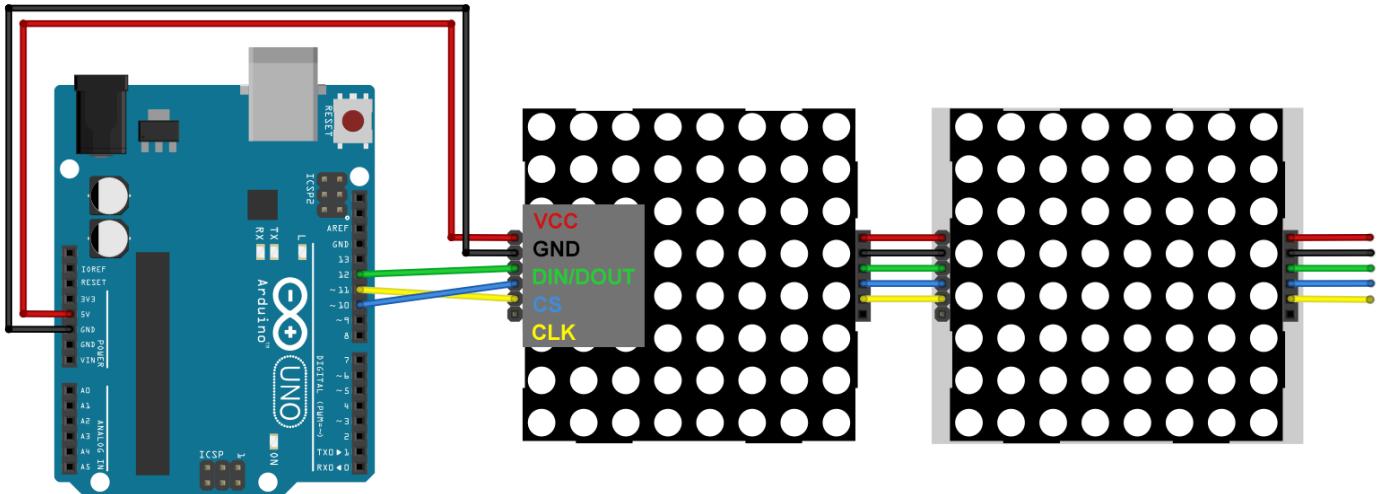
Auf dem Bild oben seht ihr, dass die obere Reihe schon komplett leuchtet und nun die linke Spalte vervollständigt wird.

Eine Lösung lautet, das Display einfach um 90° gegen den Uhrzeigersinn zu drehen. Dann ist der Dot (0/0) unten links (wie in einem Koordinatenkreuz) und nicht oben links. Ihr müsst dann bei der Zeilenummerierung umdenken.

## Verwendung von Displays mit integriertem MAX7219

### 8x8x1 LED Matrix Display

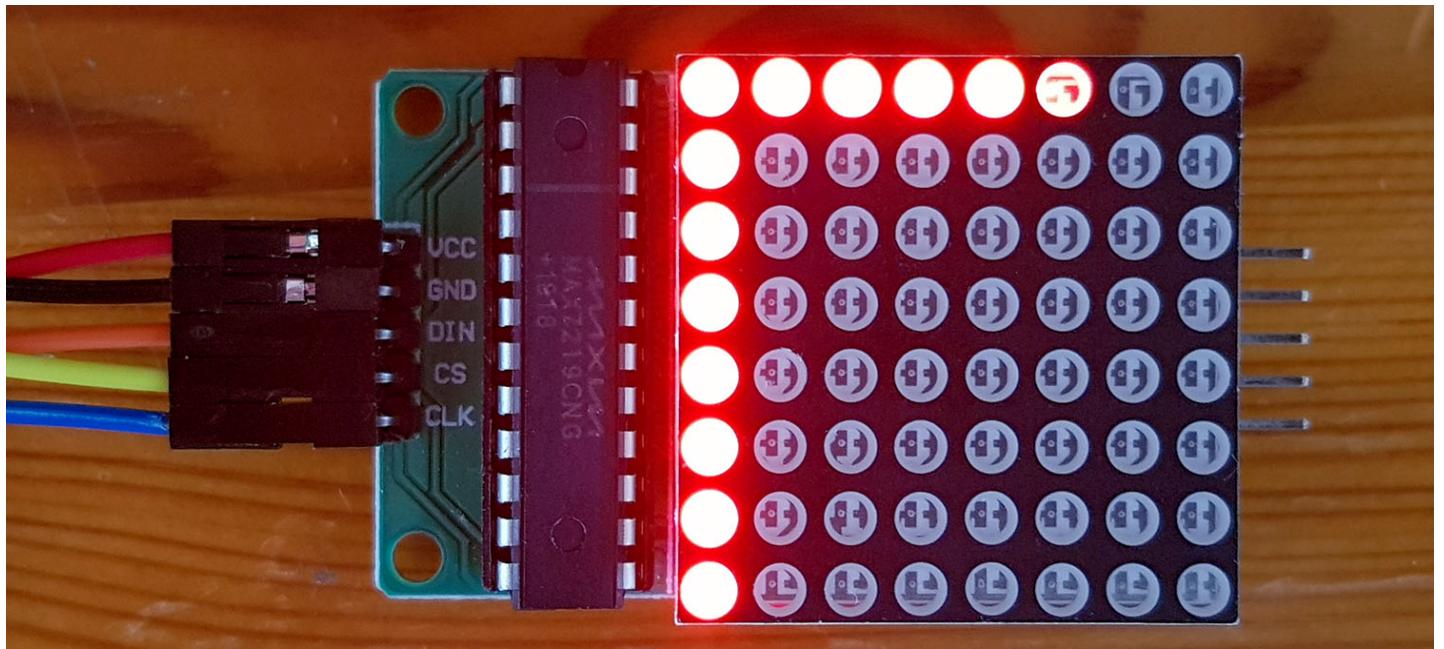
Wer es bequem haben möchte, dem rate ich zur Verwendung von Displays mit integriertem MAX7219. Die Verkabelung solcher Teile sieht schon deutlich angenehmer aus:



fritzing

Verkabelung von Displays mit integriertem MAX7219 – auch das Kaskadieren ist einfach

Wo der Nullpunkt (0/0) liegt, kann unter Umständen überraschend sein. Probiert es aus. Dieses LED Matrix Display macht es wie erwartet:



8x8 LED Matrix Display mit integriertem MAX7219

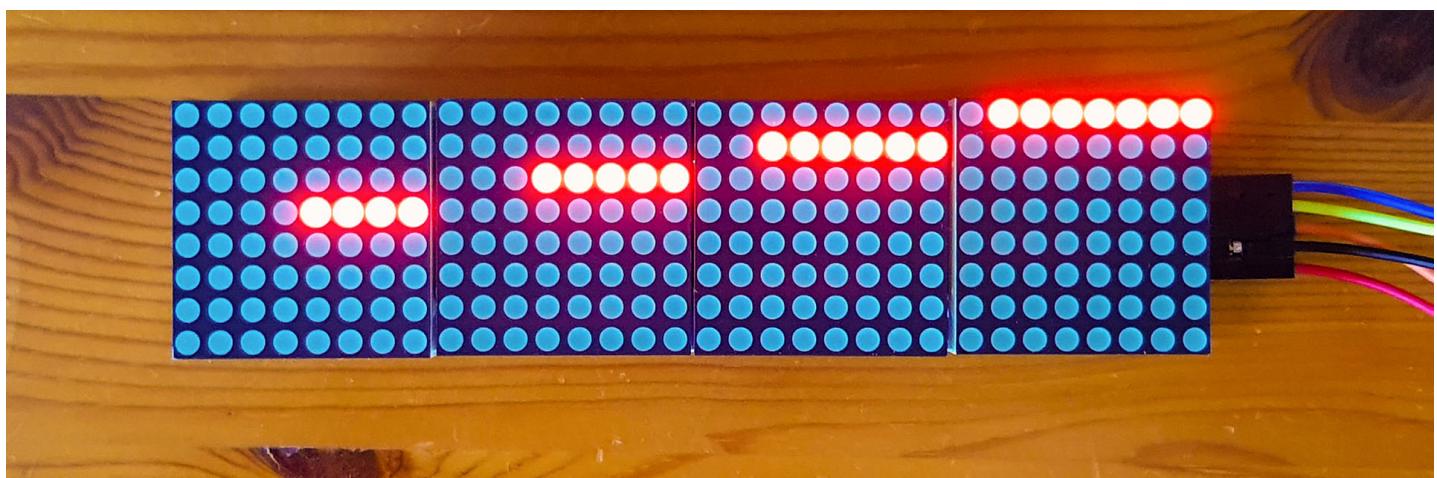
Hinweis: wenn ihr alle LEDs eines 8x8 Matrix Displays gleichzeitig leuchten lasst, dann kann der Strombedarf mehrere 100 mA pro Display betragen. Bei Verwendung mehrerer Displays könnt ihr dann sehr locker die Limits des Arduinos überschreiten. Mehr als 500 mA solltet ihr einem Arduino UNO bzw. eurem USB Anschluss nicht zumuten. Setzt ggf. eine separate Stromquelle ein.

## 8x8x4 LED Matrix Display

Wollt ihr mehrere Displays aneinanderhängen, kommt für euch vielleicht ein Verbunddisplay infrage. Vor allem sind 8x8x4 Displays weit verbreitet. Auch hier seid ihr unter Umständen überrascht, wo sich der Dot (0,0) befindet. Zum Testen habe ich den folgenden kurzen Sketch verwendet:

```
LedControl lc884=LedControl(12,11,10,4);  
unsigned long delaytime=3000;  
  
lc884.setRow(0,0,B01111111);  
lc884.setRow(1,1,B00111111);  
lc884.setRow(2,2,B00011111);  
lc884.setRow(3,3,B00001111);  
  
#include "LedControl.h" LedControl lc884=LedControl(12,11,10,4); unsigned long  
delaytime=3000; void setup() { for(int i=0;i<4;i++){ lc884.shutdown(i,false);  
lc884.setIntensity(i,8); lc884.clearDisplay(i); } } void loop() {  
lc884.setRow(0,0,B01111111); lc884.setRow(1,1,B00111111);  
lc884.setRow(2,2,B00011111); lc884.setRow(3,3,B00001111); }
```

Eigentlich hätte ich gerne das Display 0 links und das Display 4 rechts. Außerdem hätte ich gerne den Nullpunkt oben links und eine Reihenabbildung, die der Reihenfolge der Bits entspricht. D.h. ein 0b00001111 soll vier Dots einer Reihe rechts leuchten lassen und nicht links. Ich habe zwei Displays aus unterschiedlichen Quellen getestet und mit beiden konnte ich nicht alle Wünsche erfüllen. Mit einer umgekehrten Displayreihenfolge konnte ich am ehesten leben:



8x8x4 LED Matrix Display: meine Ausrichtung

Display 0 ist hier rechts, dafür ist aber der Rest so wie ich es will. **Diese Anordnung verwende ich im Folgenden für die Entwicklung der Laufschrift.** Wenn euer Display die Dinge anders abbildet, dann müsst ihr die Sketche entsprechend anpassen.

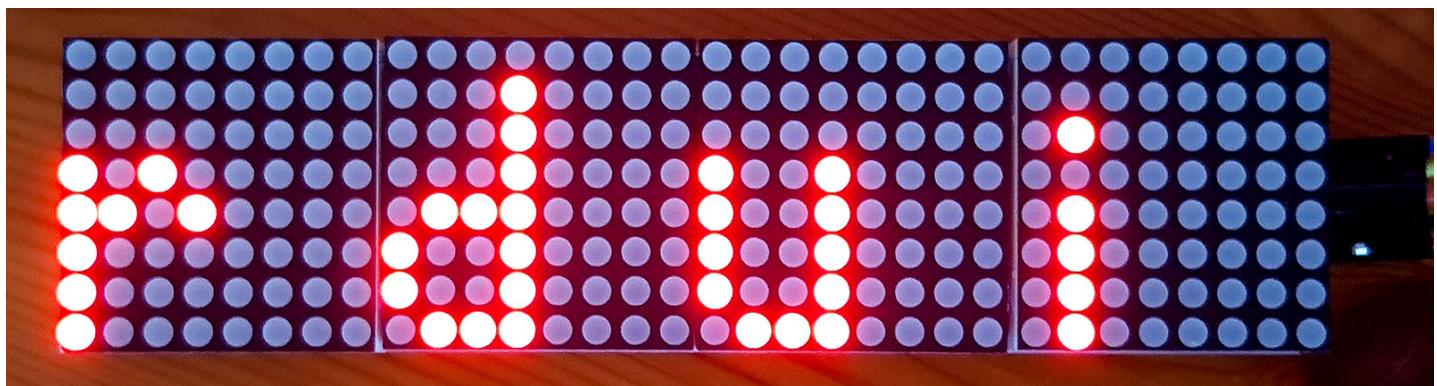
## Programmierung einer Laufschrift (Banner)

### Vorbereitung: Kreieren und Darstellen von Buchstaben

Buchstaben, Zahlen und Zeichen sind auf einem Blatt kariertem Papier schnell entwickelt. Dann schreibt ihr das Ergebnis zeilenweise im Byteformat in Arrays. Im folgenden Sketch seht ihr Beispiele für A,r,d,u,i,n und o. Ich habe die Buchstaben recht schmal gemacht, da ich sie später noch „zusammenschieben“ möchte, sodass der ganze Arduino Schriftzug auf einmal auf das Display passt.

Die Buchstaben werden zunächst nacheinander auf dem Display 0 dargestellt. Dann machen wir den ersten Schritt in Richtung Laufschrift, indem wir die Buchstaben Display-weise von rechts nach links durchlaufen lassen. Mein Bezugsdisplay ist dabei das nicht sichtbare, virtuelle Display -1.

Wie das im Ergebnis aussieht, könnt ihr in dem Video zu Beginn des Beitrages sehen. Hier ein Ausschnitt:



Ausschnitt der Laufschrift

### Ein Zwischenschritt: statische Darstellung als unsigned long Array

Das Durchlaufen „Display für Display“ war einfach. Aber es ist auch noch nicht besonders schick. Schöner wäre ein Verschieben „Dot für Dot“. Zugegebenermaßen habe ich mich damit schwerer getan als ich dachte. Ich zeige ein paar Lösungen. Vielleicht denke ich auch zu kompliziert – wenn ihr einfachere Methoden habt, immer her damit!

Für meinen Lösungsweg Nr. 1 kommt hier erst einmal ein Zwischenschritt. Zunächst habe ich die Buchstaben so zusammengefasst, dass sie auf das Viererdisplay passen. Dazu habe ich die Zwischenräume auf ein Mindestmaß reduziert und alle 0ten, 1sten, 2ten....7ten Reihen in jeweils einem unsigned long Wert zusammengefasst. Da eine unsigned long Variable 32 bits hat, entspricht das der Breite des 4er Displays. Bei der späteren Darstellung auf dem Display muss man den Wert dann allerdings wieder in bytes „aufdröseln“. Das macht die Funktion displayBanner().

### 884\_display\_static\_short.ino

```
LedControl lc884=LedControl(12,11,10,4);

unsigned long banner[8]={0b00000000000000000000000000000000,
0b01100000000010000000000000000000,
0b10010000000010000001000000000000,
0b10010101000001010010001010001100,
0b100101101001110100101011010010010,
0b111101000010010010101001010010010,
0b100101000010010010101001010010010,
0b10010100000111001110101001001100};

displayBanner(); delay(500);

currentMatrix[i] = (((banner[i])>>(j*8))&0b1111111);

displayMatrix(currentMatrix,j);

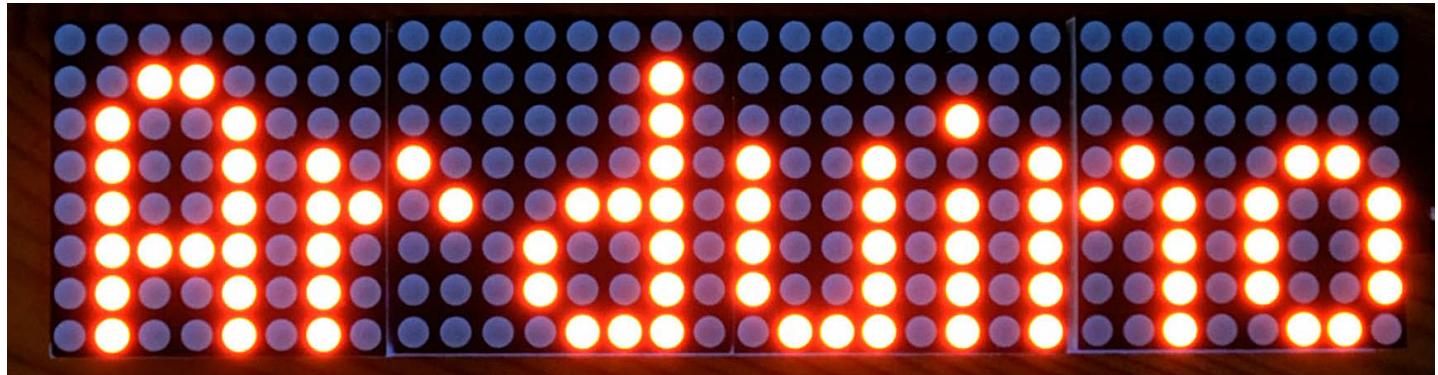
void displayMatrix(byte *matrix, int matrixNumber){

lc884.setRow(matrixNumber,i,matrix[i]);

#include "LedControl.h" LedControl lc884=LedControl(12,11,10,4); long
delayTime=500; unsigned long
banner[8]={0b00000000000000000000000000000000,
0b01100000000010000000000000000000,
0b10010000000010000001000000000000,
0b10010101000001010010001010001100,
0b100101101001110100101011010010010,
0b111101000010010010101001010010010,
0b100101000010010010101001010010010,
```

```
0b10010100000111001110101001001100}; void setup() { for(int i=0;i<4;i++){  
  lc884.shutdown(i,false); lc884.setIntensity(i,8); lc884.clearDisplay(i); } } void  
loop() { displayBanner(); delay(500); } void displayBanner(){ byte currentMatrix[8];  
for(int j=0; j<4; j++){ for(int i=0; i<8; i++){ currentMatrix[i] = (((banner[i]>>(j*8)) &  
0b11111111); } displayMatrix(currentMatrix,j); } } void displayMatrix(byte *matrix,  
int matrixNumber){ for(int i=0; i<=7;i++){ lc884.setRow(matrixNumber,i,matrix[i]);  
} }
```

Und so sieht das Ergebnis aus:



Der "kondensierte" Arduino Schriftzug

## Das unsigned long Array dotweise laufen lassen

Der Vorteil der Darstellung als unsigned long array ist, dass die Verschiebung wesentlich leichter über [Binäroperationen](#) zu programmieren ist als Byte für Byte. Das Banner wird schrittweise nach links verschoben und auf die Einzeldisplays aufgeteilt. Beim Nachvollziehen des Sketches vergesst nicht, dass sich das Display 0 rechts befindet.

Als kleines Feature habe ich noch eingebaut, dass das Banner kurz stoppt, wenn es vollständig auf dem Display ist. Es wird kurz heller, dann wieder dunkler und läuft dann aus dem Display. Das Ergebnis seht ihr im Video zu Beginn des Beitrages.

## Gößere Banner laufen lassen

Nun passt das, was ihr vielleicht als Banner über das Display laufen lassen wollt, nicht unbedingt auf das Display bzw. in ein unsigned long Array. Aber das geht auch. Als Beispiel nehme ich den ungekürzten Arduino Schriftzug, der sich über sieben Einzeldisplays erstreckt. Diesen verteile ich in ein zweidimensionales unsigned long Array „bannerPart[2][8]“. Dabei verschwende ich natürlich ein wenig Speicherplatz, da ich das Array nicht ganz nutze.

Die Übernahme der einzelnen Buchstaben in das Array bannerPart habe ich diesmal automatisiert. Das geschieht im Setup in den ersten beiden for-Schleifen. In calcCurrentBanner() finden sich drei for-Schleifen, die für drei Phasen stehen. In der ersten Phase wird das erste Bannerteil hineingeschoben. Während der zweiten Phase geht das erste Teil hinaus und die frei werdenden Bereiche werden mit dem zweiten Bannerteil aufgefüllt. In der dritten Phase wird das zweite Bannerteil hinausgeschoben.

Das Ergebnis findet ihr wieder im Video.

Wenn ihr eigne Banner mit einer anderen Breite kreiert, dann müsst ihr einige Werte im Sketch anpassen. Ich hatte offengestanden keine Lust mehr den Sketch zu verallgemeinern.

## Weitere Methoden

Das Verschieben mittels Byte Arrays ist etwas komplizierter als man zunächst meinen sollte. Zumindest habe ich mich damit ein wenig schwergetan. Ihr findet den Sketch am Schluss des Beitrages.

Die mit Abstand einfachste Methode wäre ein bool Array, in dem jeder Dot eine separate Variable ist. In dieses Array könnte man auch die vier leeren Displays zu Beginn und die vier leeren Displays am Ende integrieren. Macht  $4 + 7$  (für das Arduino Banner)  $+ 4 = 15$  Einzeldisplays. Daraus würde ein  $120 \times 8$  Array mit 960 bool Werten entstehen. Das Durchlaufen des Banners wäre so sehr einfach zu programmieren. Aber leider belegt jede bool Variable ein ganzes Byte. Diese Verschwendungen von Speicherplatz ist mir dann doch zu groß.

## LED Matrix Display Ansteuerung ohne MAX7219 Bibliothek

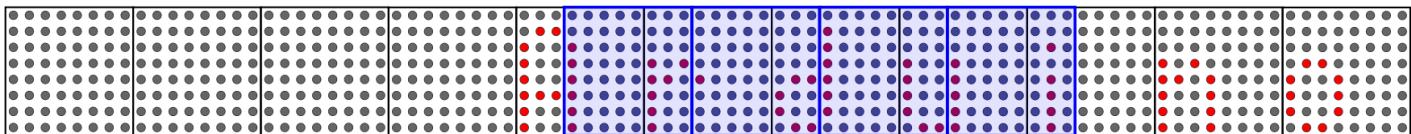
Wie schon erwähnt, hat der MAX7219 nicht übermäßig viele Register und kann deswegen recht leicht auch ohne Bibliothek angesteuert werden. Im nächsten Sketch könnt ihr sehen wie das geht.

Ich möchte den Sketch aber auch nicht im Detail erläutern, da der Beitrag sowieso schon so lang ist. Wenn ihr euch das Datenblatt danebenlegt, sollte er nicht so schwer zu verstehen sein. Wenn ihr trotzdem Fragen habt, fragt! Nur ein paar Anmerkungen:

- Der Sketch verwendet die Standard SPI Anschlüsse für MOSI und CLK (beim UNO Pin 11 bzw. 13). Ihr müsst also ein wenig umstöpseln.
- MISO gibt es nicht, der MAX7219 ist nicht gesprächig.
- Da es kein MISO gibt, lässt sich der Zustand der Dots nicht abfragen.
- Da man bei der setLed Funktion nur einzelne Dots ändern möchte, aber nur ganze Reihen Bytes schreiben kann, muss der Sketch sozusagen Buch führen. Das macht er über das Array dots.

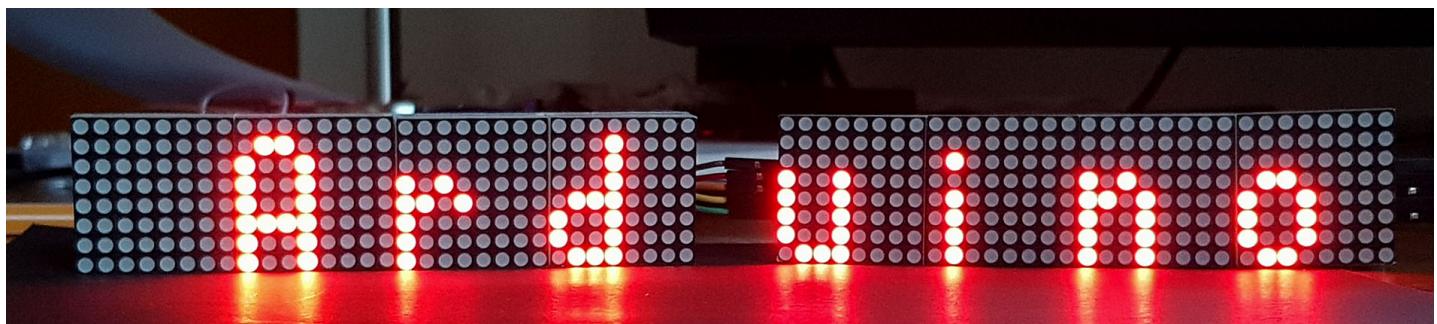
## Zu guter Letzt...

...noch der Banner Sketch auf Basis der Byte Arrays. Hier habe ich als Denkmodell ein virtuelles Display verwendet, welches das physikalische Display quasi von rechts nach links durchwandert (siehe Schema unten). Ein bisschen wie ein Filmstreifen, der im Projektor hinter dem Objektiv entlang läuft.



Grau: Virtuelles Display, Blau: physikalisch vorhandenes Display

Ich habe den Sketch so verfasst, dass ihr ihn leicht auf andere Displaygrößen anpassen könnt. Ich habe ihn z.B. auf zwei hintereinandergeschalteten 8x8x4 Matrix Display laufen lassen (dazu musste lediglich die numberOfRowsPhysicalDisplays von vier auf acht geändert werden):

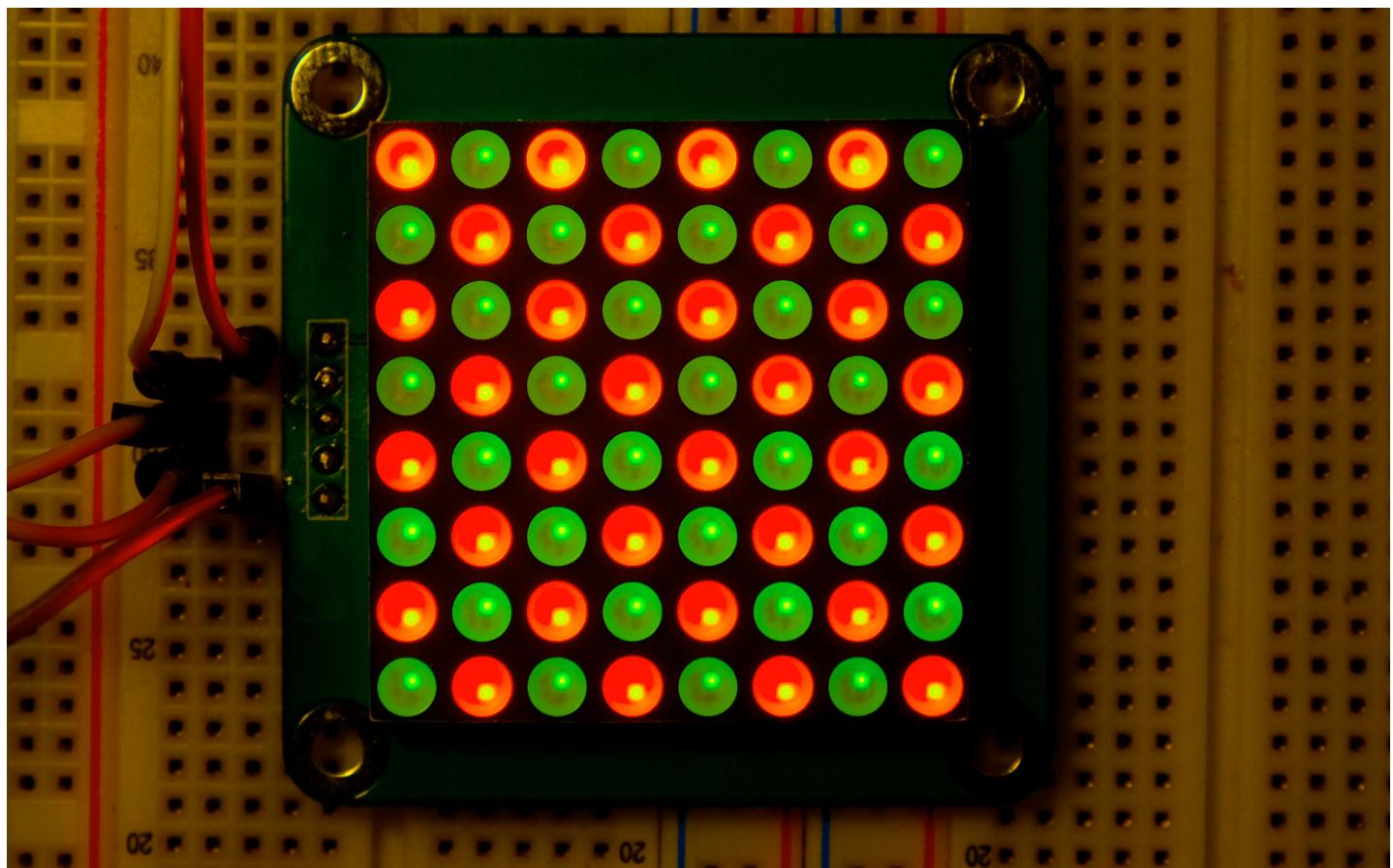


Anwendung auf ein 2x8x8x4 Display

Der Sketch ist recht kompakt, aber einigermaßen schwer „verdaulich“. Ich habe versucht, ihn durch Kommentare halbwegs verständlich zu gestalten. Viel Spaß beim Nachvollziehen meiner Logik!

## Anhang: Ansteuerung eines zweifarbigen TA6932 basierten Displays

Es gibt auch zweifarbige Displays mit Dots, die rot oder grün leuchten können. Diese werden intern über einen TA6932 Chip gesteuert. Eine gleichnamige Bibliothek gibt es [hier](#) auf GitHub oder ihr installiert sie über die Arduino Bibliotheksverwaltung.



Mit der Bibliothek ist die Ansteuerung sehr einfach. Mit `displayCache(row) = value` legt ihr fest, welche Dots einer Reihe (row) leuchten sollen. Die roten Dots sind als Reihen 0 bis 7 definiert. Die grünen Dots sprechen ihr als Reihen 8 bis 15 an. Das sollte durch den folgenden Sketch klarer werden.