Teil 2

Lektion 2

RGB LEDS

Übersicht

RGB LEDs sind ein einfacher Weg, um Farbe in Ihre Projekte zu bringen. Da sich in einer RGB-LED nichts anderes als drei reguläre LEDs in einem befinden, ist die Handhabung dieser ähnlich.

Es gibt sie in zwei Versionen: Mit gemeinsamer Anode und mit gemeinsamer Kathode.
Bei gemeinsamer Anode liegt an dem gemeinsam genutzten Pin 5V an, bei gemeinsamer Kathode wir der gemeinsame Pin mit GND (Ground bzw Erdung) verbunden.

Wie bei jeder LED müssen einige Widerstände vorgeschaltet werden (3 insgesamt), damit der fließende Strom begrenzt wird.

In unserem Sketch lassen wir die LED zuerst in rot leuchten, lassen sie dann nach grün übergehen, danach nach blau und schließlich zurück zur roten Farbe. Dabei werden wir durch den Übergang zwischen den Farben durch einen Großteil aller möglichen Farben schalten.

Benötigte Bauteile:

(1) x Elegoo Uno R3

(1) x 830 Punkte Breadboard

(4) x M-M Kabel (Männlich zu Männlich DuPont Jumper Kabel)

(1) x RGB LED

(3) x 220 Ohm Widerstände

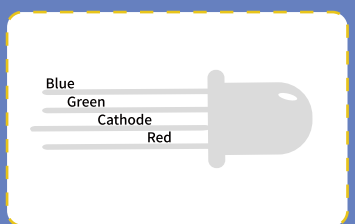
Component Introduction

RGB:

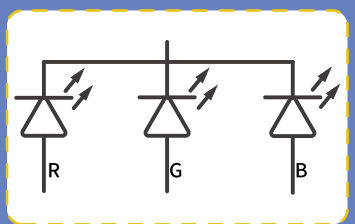
Auf den ersten Blick sehen RGB (Rot, Grün, Blau) LEDs wie normale LEDs aus. Innerhalb der RGB-LED befinden sich jedoch drei eigentliche LEDs: eine rote, eine grüne und eine blaue. Durch Anpassen der Helligkeit jeder der drei Farben und zusammenmischen dieser, lassen sich so gut wie alle Farben produzieren.

Die Farben werden dabei ähnlich wie Farbe auf einer Farbpalette gemischt – durch Anpassen der Menge der Grundfarben bzw der drei LEDs. Der schwerste Weg dies zu erreichen wäre durch das Benutzen verschiedener Widerstände oder regelbarer Widerstände, wie wir es bereits in Lektion 2 getan haben, aber das bringt einen großen Aufwand mit sich. Glücklicherweise hat das UNO R3 Board eine analogWrite-Funktion, die einem erlaubt die Spannung an den analogen Pins des Boards (mit a~ markiert) anzupassen.

Die RGB-LED hat vier Kontakte. Es gibt drei Kontakte, die jeweils zum positiven Ende der drei Farb-LEDs gehen und ein Kontakt, der gemeinsam von allen Farben als negativer Anschluss genutzt wird (gemeinsame Kathode).



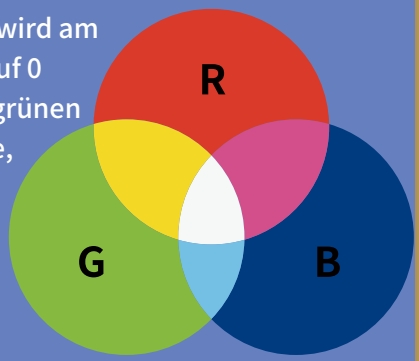
Hier auf den Bildern sehen Sie eine 4-Elektroden-LED. Die positiven Anschlüsse für Grün, Blau und Rot werden Anoden genannt. An diese muss immer der +-Pol angeschlossen werden. Die Kathode wird dagegen immer mit GND (= Ground = Erdung = Minus-Pol) verbunden. Wenn Sie die LED andersherum anschließen, wird sie nicht aufleuchten.



Normalerweise ist der gemeinsame negative Anschluss (gemeinsame Kathode) der zweite Pin von der flachen Seite aus. Es ist außerdem der größte aller vier Pins und wird mit GND verbunden. An jedem der drei positiven Anschlüsse muss ein 220Ω Widerstand vorgeschaltet werden, um den Strom der LED zu begrenzen.Die anderen Enden der Widerstände müssen mit dem UNO Board verbunden werden, sodass die LEDs mit dem Board verbunden sind.

Farbe:

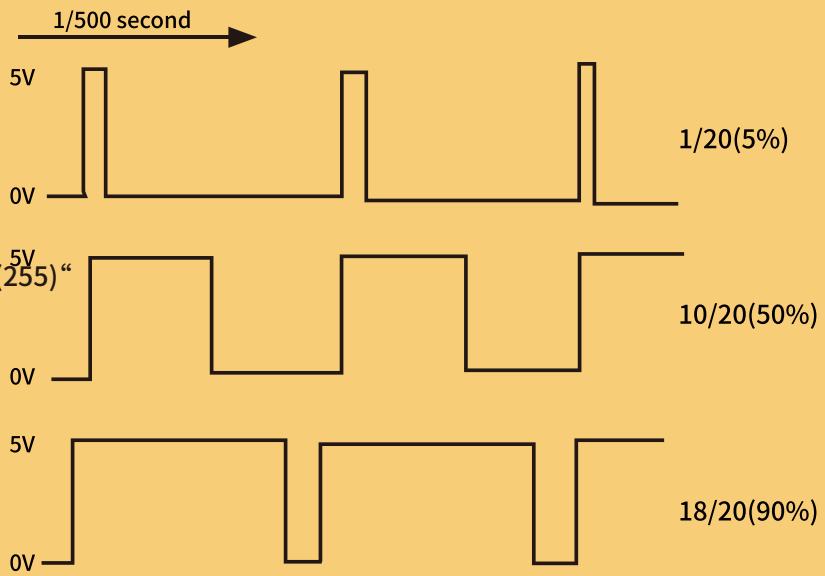
- **The** Der Grund, warum man durch Anpassen der Helligkeit von rot, grün und blau jede beliebige Farbe mischen kann, ist, dass das menschliche Auge drei verschiedene Lichtrezeptoren (rot, grün und blau) hat. Das Gehirn verarbeitet die Menge von rot, grün und blau und mischt sie zu einer Farbe aus dem Farbspektrum zusammen.
- **In** Wir nutzen diesen Trick also aus, um künstlich verschiedene Farben durch Anpassen der rot- grün- und blau-Werte zu erzeugen. Das RGB-Farbmodell wird beispielsweise außerdem in Fernsehern benutzt, bei denen jeder Pixel auf dem LCD-Panel aus drei Farben (rot, grün und blau) besteht.
- **If** Wenn wir die Helligkeit aller drei LEDs auf den gleichen Wert einstellen, wird am Ende weiß herauskommen. Wenn wir dann die Helligkeit der blauen LED auf 0 schalten, wird ein gelbes Licht leuchten.Wir können die Helligkeit der rot, grünen und blauen Teile der LED separat einstellen, was uns ermöglicht jede Farbe, die wir möchten, zusammenzumischen.
- **Bei** Schwarz handelt es sich dagegen um keine eigentliche Farbe, da schwarz im Prinzip nur das Nichtvorhandensein von jeglichem Licht darstellt. Daher können wir mit unserer RGB-LED schwarz nur darstellen, indem wir alle drei Farben ausschalten bzw auf 0 setzen.



PWM-Technik

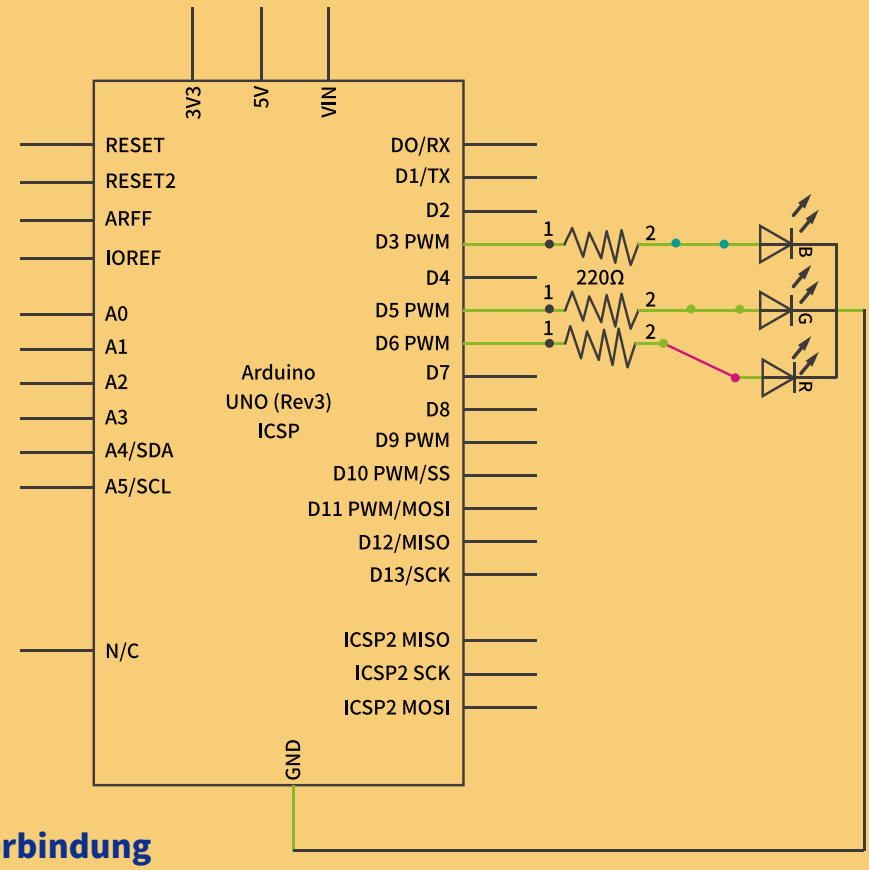
- Die Pulsweitenmodulation (PWM) ist eine Technik, um die elektrische Leistung zu kontrollieren bzw zu begrenzen. Wir benutzen diese Technik auch, um die Helligkeit der einzelnen LEDs zu steuern.
  Das Diagramm unten zeigt ein Beispiel eines PWM-Signals aus einem PWM Pin des UNO Boards.

- Ungefähr jedes 1/500 einer Sekunde gibt der PWM Ausgang einen Puls aus. Die Länge dieses Pulses wird durch die analogWrite Funktion bestimmt. So wird bei„analogWrite(0)“ nicht einmal ein Puls ausgegeben,während„analogWrite(255)“ einen Puls ausgibt, der die ganze Zeit dauert, bis der nächste Puls ansteht, sodass der Ausgang eigentlich die ganze Zeit eingeschaltet ist.

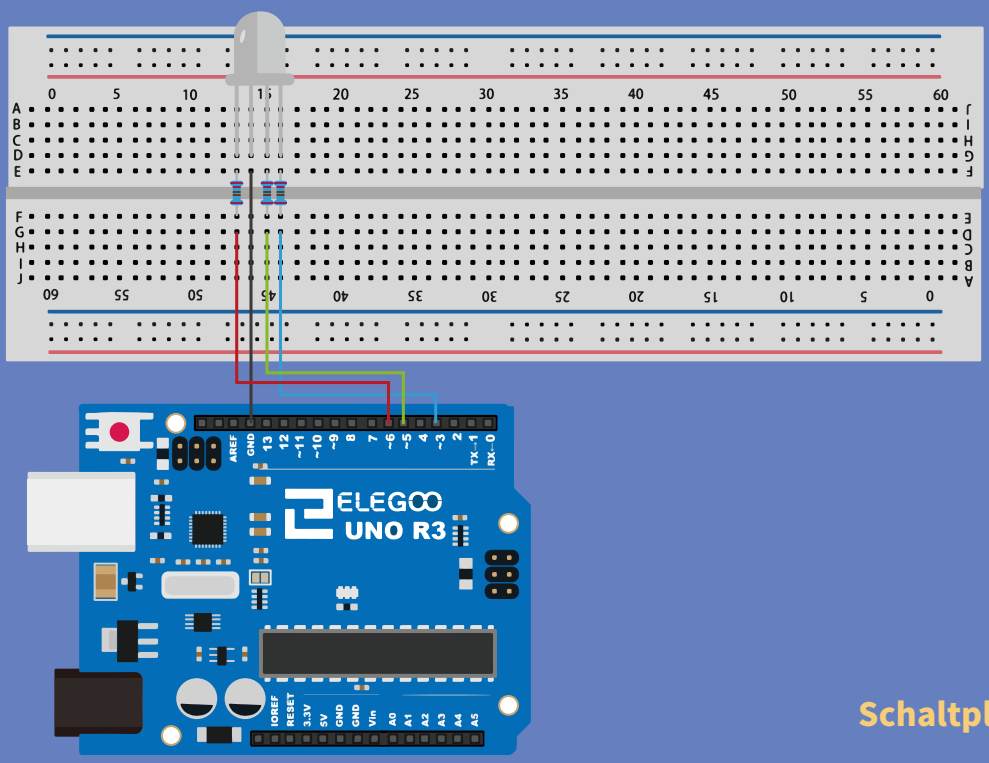


- Wenn wir einen Wert irgendwo zwischen 0 und 255 für die analogWrite-Funktion festlegen, wird ein Puls produziert. Wenn die Länge des Pulses nur 5% der Zeit beträgt, bekommt das angeschlossene Gerät auch nur 5% der vollen Leistung.

- Wenn der Ausgang bei 5V zu 90% der Zeit eingeschaltet ist, wird das Gerät 90% der vollen Leistung erhalten. Trotz des Wechsels zwischen An und Aus, können wir dieses Ein- und Ausschalten nicht an den LEDs erkennen, da unser Auge die schnell wechselnden Zustände zusammenmischt, sodass es für uns aussieht, als würde sich die Helligkeit ändern.



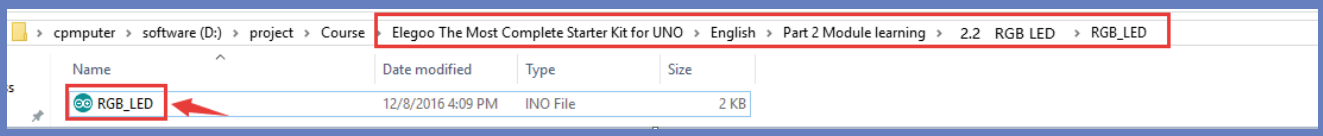
Schema der Verbindung



Schaltplan

Code

- Öffnen Sie nach dem Verdrahten den Pfad zum Sketch im Ordner: **\Elegoo The Most Complete Starter Kit for UNO\English\Part 2 Module Learning\2.2 RGB LED** und klicken Sie auf HOCHLADEN, um das Programm auf den Arduino zu übertragen.



- Weitere Informationen zum Hochladen von Sketchen finden Sie in Lektion 5 in Teil 1, sofern Fehler auftreten sollten.
- Der Sketch beginnt mit der Festlegung, welcher PIN für jede der Farben verwendet werden soll:

```
// Define Pins
#define BLUE 3 // The compiler will replace any mention of ledPin with the value 3 at compile time.
#define GREEN 5
#define RED 6
```

#define constantName value

ist eine nützliche C++ - Komponente, mit welcher der Programmierer einen konstanten Wert einem Namen zuordnen kann, und zwar bevor der Sketch kompiliert wird. Definierte Konstanten belegen im Arduino keinen Seicherplatz auf dem Chip. Der Compiler ersetzt Verweise auf diese Konstanten beim Kompilieren durch den definierten Wert.

Dies kann jedoch einige unerwünschte Nebenwirkungen haben, wenn beispielsweise ein Konstantenname, der mit #define festgelegt wurde, in einer anderen Konstanten- oder Variablennamen enthalten ist. In diesem Fall würde der Text durch die #define Anweisung ersetzt.

Parameters

constantName: Der Name der zu definierenden Konstante.

value: Der zugewiesene Wert der Konstanten.

Notes and Warnings

- Nach der Anweisung #define steht kein Semikolon! Wenn Sie eines hinzufügen, wirft der Compiler kryptische Fehler innerhalb der Arduino IDE aus.

#define ledPin 3; // führt zu einem Fehler

- Ein Gleichheitszeichen führt ebenfalls zu einem Compiler-Fehler:

#define ledPin = 3 // führt zu einem Fehler

Der nächste Schritt ist das Schreiben der 'Setup'-Funktion. Wie wir in früheren Lektionen gelernt haben, wird die Setup-Funktion nur einmal ausgeführt. In unserem Beispiel definieren wir die drei verwendeten PINs als Ausgänge.

pinMode(pin, mode) konfiguriert den angegebenen PIN so, dass er sich entweder als Eingang oder als Ausgang verhält.

Ab Arduino 1.0.1 können die internen Pullup-Widerstände mit Hilfe von INPUT_PULLUP aktiviert werden. Darüber hinaus deaktiviert der INPUT-Modus die internen Pullups explizit.

```
void setup()
{
pinMode(RED,OUTPUT);
pinMode(GREEN,OUTPUT);
pinMode(BLUE, OUTPUT);
digitalWrite(RED,HIGH);
digitalWrite(GREEN,LOW);
digitalWrite(BLUE, LOW);
```

Parameters

pin: Die Arduino-Pin-Nummer zum Einstellen des Modus.

mode: INPUT, OUTPUT oder INPUT_PULLUP. Auf der Seite Digital Pins finden Sie eine ausführlichere Beschreibung der Funktionen.

Int: ist ein Integer-Datentyp. Integers sind Ihr primärer Datentyp für die Speicherung von Ganz-Zahlen.

```
// define variables
int redValue;
int greenValue;
int blueValue;
```

Auf dem Arduino Uno (und anderen ATmega-basierten Devices) speichert ein Integer einen 16-Bit-Wert (2 Byte). Dies ergibt einen Bereich von -32.768 bis 32.767 (Minimalwert von -2 ^ 15 und Maximalwert von (2 ^ 15) - 1). Auf den Arduino Due- und SAMD-basierten Karten (wie MKR1000 und Zero) speichert ein int einen 32-Bit-Wert (4 Byte). Dies ergibt einen Bereich von -2.147.483.648 bis 2.147.483.647 (Minimalwert von -2 ^ 31 und Maximalwert von (2 ^ 31) - 1).

Integer speichert negative Zahlen mit einer Technik namens (2's Complement Math). Das höchste Bit, das manchmal als "Vorzeichen" -Bit bezeichnet wird, kennzeichnet die Zahl als negative Zahl. Der Rest der Bits wird invertiert und 1 wird hinzugefügt.

Syntax

```
int var = val;
```

Parameters

var: Variablenname.

val: Der Wert, den Sie dieser Variablen zuweisen.

- Der Sketch fängt damit an, dass erstmal bestimmt wird, welche Pins für die verschiedenen Farben genutzt werden.

Die Variablen definieren

```
redValue = 255; // choose a value between 1 and 255 to change the color.
greenValue = 0;
blueValue = 0;
```

- Diese Funktion hat drei Argumente, eins für die Helligkeit der roten, eins für die Helligkeit der grünen und eins für die Helligkeit der blauen LED. In jedem möglichen Fall werden die Werte Zahlen im Bereich von 0 bis 255 sein, wobei 0 ausgeschaltet und 255 maximale Helligkeit bedeutet. Die Funktion ruft dann analogWrite auf, um die Helligkeitswerte anschließed auch zu setzen.

```
for (int i = 0; i < 255; i += 1) // fades out red bring green full when i=255
{
redValue -= 1;
greenValue += 1;
// The following was reversed, counting in the wrong directions
// analogWrite(RED, 255 - redValue);
// analogWrite(GREEN, 255 - greenValue);
analogWrite(RED, redValue);
analogWrite(GREEN, greenValue);
delay(delayTime);
}
```

for

[Control Structure]

Description

Die for-Anweisung wird verwendet, um einen Anweisungsblock in geschweiften Klammern zu wiederholen. Ein Inkrementzähler wird normalerweise verwendet, um die Schleife hochzuzählen und damit zu beenden. Die for-Anweisung ist für jede sich wiederholende Operation nützlich und wird häufig in Kombination mit Arrays verwendet, um Sammlungen von Daten / PINs zu bearbeiten.

Syntax

```
for ( initialization; condition; increment ) {
  // statement(s);
}
```

Parameters

initialization: geschieht zu Beginn und zwar ein mal.
condition: Jedes Mal, wenn die Schleife durchlaufen wird, wird auf die Erfüllung der Bedingung hin getestet. Wenn dies der Fall ist, wird der Anweisungsblock ausgeführt und die Bedingung erneut getestet. Wenn die Bedingung falsch ist, endet die Schleife.
increment: Wird jedes Mal durch die Schleife ausgeführt, wenn die Bedingung erfüllt ist.

=

[Arithmetic Operators]

Description

Das einzelne Gleichheitszeichen = in der Programmiersprache C++ wird als Zuweisungsoperator bezeichnet. Es hat eine andere Bedeutung als in Algebra, wo es eine Gleichung oder Gleichheit anzeigt. Der Zuweisungsoperator weist den Mikrocontroller an, den Wert/Ausdruck auf der rechten Seite des Gleichheitszeichens auszuwerten und in der Variablen links vom Gleichheitszeichen zu speichern.

Notes and Warnings

Die Variable auf der linken Seite des Zuweisungsoperators (=) muss den darin gespeicherten Wert aufnehmen können. Wenn der gewählte Datentyp nicht groß genug ist, um einen Wert aufzunehmen, ist der zugeordnete Wert der Variablen falsch.

Verwechseln Sie den Zuweisungsoperator (=) (ein einzelnes Gleichheitszeichen) nicht mit dem Vergleichsoperator [==] (ein doppeltes Gleichheitszeichen), welcher bewertet, ob zwei Ausdrücke gleich sind.

+= / -=

[Compound Operators]

Description

Dies ist eine praktische Abkürzung, um eine Addition / Subtraktion für eine Variable mit einer anderen Konstanten oder Variablen durchzuführen:

Syntax

```
x += y; //  äquivalent zum Ausdruck x = x + y;
x -= y; //  äquivalent zum Ausdruck x = x -y;
```

Parameters

x: variabel. Zulässige Datentypen: int, float, double, byte, short, long.
y: variabel oder konstant. Zulässige Datentypen: int, float, double, byte, short, long.