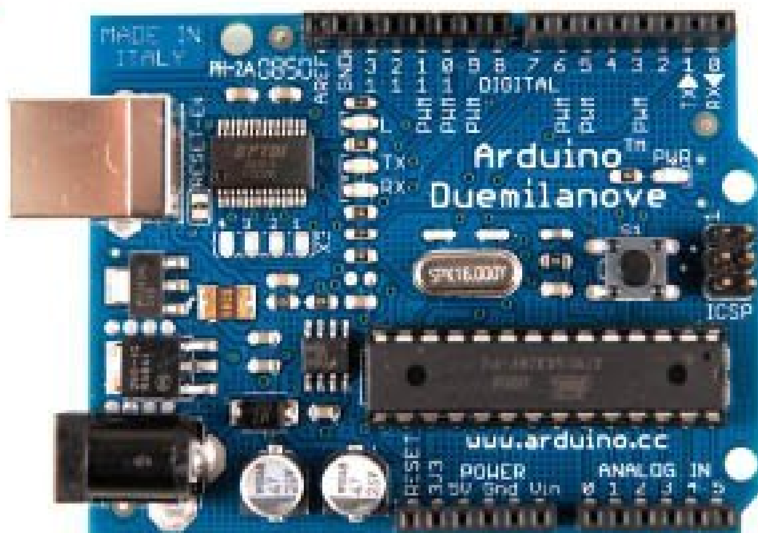
o'reillys basics

Arduino

Physical Computing für Bastler, Designer & Geeks

- Microcontroller-Programmierung für alle
- Prototype Your Life
- Mit kompletter Programmiersprachen-referenz



Manuel Odendahl,
Julian Finn & Alex Wenger

O'REILLY

Inhalt

Einleitung . . . IX

1 Von Königen und Kondensatoren . . . **1**
Die Geschichte des Arduino-Projekts . . . 1
Der Arduino, das unbekannte Gerät . . . 3
Arduino-Projekte: eine kleine Vorstellung . . . 8
Hardware . . . 17
Die Arduino-Entwicklungsumgebung . . . 22

2 Physical Computing, elektrische Grundlagen und der Sprung ins kalte Wasser . . . **31**
Elektrische Grundlagen . . . 35
Schaltungen, Bauteile und Schaltbilder . . . 43
Löten . . . 64
Fehlersuche in elektronischen Schaltungen . . . 73

3 Workshop LED-Licht . . . **83**
Erste Schritte . . . 83
Eine blinkende LED – das »Hello World« des Physical Computing . . . 84

4 LEDs für Fortgeschrittene . . . **109**
LED-Matrix . . . 109
Animationen . . . 113
Interrupts . . . 115
Tamagotchi . . . 118
Brainwave und Biofeedback . . . 120

5 Sprich mit mir, Arduino! . . . 129
Nach Hause telefonieren mit der seriellen Konsole . . . 131
Automatisierung mit Gobetwino . . . 137
Processing . . . 142

6 Arduino im Netz . . . 153
Hello World – ein Mini-Webserver . . . 157
Sag's der Welt mit Twitter . . . 160
Fang die Bytes – Datalogger . . . 165

7 Sensoren . . . 171
Sensoren . . . 171
Aktoren . . . 189
Elektronischer Würfel . . . 193

8 Ein kleiner Arduino-Roboter . . . 199
Flieg, Arduino, flieg! . . . 202
Arduino und der Asuro-Robot . . . 205

9 Musik-Controller mit Arduino . . . 211
Musik steuern mit dem Arduino . . . 211
Das MIDI-Protokoll . . . 217
Die MidiDuino-Bibliothek . . . 223
Zweiter Sketch: Mikroskopischer Controller . . . 225
Dritter Sketch: Miniatur-Controller . . . 226
Vierter Sketch: Ein MIDI-Zauberstab . . . 229
Fünfter Sketch: MIDI-Input . . . 232

10 Musik mit Arduino . . . 235
Töne aus dem Arduino . . . 235
Erster Sketch: Töne mit langsamer PWM . . . 239
Zweiter Sketch: Angenehme Klänge mit schneller PWM . . . 240
Dritter Sketch: Steuerung von Klängen . . . 242
Vierter Sketch: Berechnungen in einer Interrupt-Routine . . . 244
Fünfter Sketch: Musikalische Noten . . . 247

A Arduino-Boards und Shields . . . 253
Arduino-Boards . . . 253
Arduino-Shields . . . 257

B Arduino-Bibliotheken **263**
EEPROM-Bibliothek: Werte langfristig speichern 264
Ethernet-Bibliothek: mit dem Internet kommunizieren 265
Firmata-Bibliothek 270
LiquidCrystal-Bibliothek 270
Servo-Bibliothek 272
Debounce-Bibliothek 274
Wire-Bibliothek 275
capSense-Bibliothek 277

C Sprachreferenz **281**
Übersicht: Programmiersprachen 281
Struktur, Werte und Funktionen 282
Syntax 283
Programmwerte (Variablen, Datentypen und Konstanten) 287
Ausdrücke und Anweisungen 299
Ausdrücke 300
Kontrollstrukturen 318
Funktionen 330
Sketch-Struktur 340
Funktionsreferenz 343

D Händlerliste **361**

Index **363**

KAPITEL 3
Workshop LED-Licht

In diesem Kapitel:

- Erste Schritte
- Eine blinkende LED – das »Hello World« des Physical Computing

Nun soll also der erste Workshop beginnen, dessen Ziel es ist, eine in allen Farben des Regenbogens leuchtende LED-Lampe zu bauen. Natürlich geht es nicht darum festzulegen, welche Form oder Farbe sie bekommen soll. Vielmehr wird dieses Kapitel hoffentlich genügend Anleitung geben, um im Anschluss ein eigenes Licht programmieren zu können. Dieses kann aus einer oder mehreren Lichtquellen bestehen, die sich auch in ihrer Helligkeit verändern lassen, entweder durch Programmierung oder durch ein Steuerungselement wie einen Schalter oder Drehknopf. Zudem wird erklärt, wie man aus Rot, Grün und Blau Farben mischen kann, um den Raum auch mehrfarbig zu erhellen. Am Ende des Kapitels werden zusätzlich weitere Projekte beschrieben, die mit Anleitungen aus dem Internet nachgebaut werden können. Auf dem Weg durch das Kapitel werden die Grundlagen erläutert, die zum Programmieren eines Arduino nötig sind. Dieser Workshop richtet sich also auch an Leute, die soeben zum ersten Mal ein Arduino-Board angeschlossen haben.

Erste Schritte

Um dieses Kapitel durchzuarbeiten, sind zum ersten Mal in diesem Buch einige Bauteile nötig, nämlich

- vier LEDs (drei davon in Rot, Grün und Blau) und passende 100-Ohm-Widerstände,
- ein Taster,
- ein Schalter und
- ein Drehknopf.

ino - Physical Computing für Bastler, Designer und Geeks", ISBN 978-3-89721-893-2
/
:ht. © O'Reilly Verlag 2009

Ein einfaches Arduino-Programm: Übersicht

Ein Arduino-Programm kann aus vielen miteinander verbundenen Dateien bestehen, hat jedoch mindestens zwei Teile: das Setup und die Hauptfunktion.

Setup

Der Setup-Teil des Programms ist gekennzeichnet durch die Funktion `setup()`:

```
void setup()
{
}
```

Zwischen die geschweiften Klammern werden nun alle Befehle gesetzt, die vor dem Start des Hauptprogramms zum Einrichten des Arduino benötigt werden, etwa die Festlegung einzelner Pins als Ein- oder Ausgang.

Die Setup-Routine wird nur ein einziges Mal ausgeführt, wenn das Board neu an eine Stromquelle (oder per USB an den Rechner) angeschlossen oder neuer Code hochgeladen wird.

Loop

Die Funktion `loop()` wird auch als *Hauptfunktion* bezeichnet. Von hier aus werden andere Bestandteile des Programms aufgerufen oder Befehle abgearbeitet. Wie der Name schon verrät, läuft `loop()` in einer Schleife, das heißt sie beginnt immer wieder von vorn, sobald sie durchlaufen wurde. Ganz analog zur `setup()`-Funktion sieht auch `loop` so aus:

```
void loop()
{
}
```

Eine blinkende LED – das »Hello World« des Physical Computing

Beim Erlernen von Programmiersprachen ist es üblich, zuerst einen einfachen Text auf dem Bildschirm auszugeben, um über die Sprache einen Gruß an die Welt hinaus zu senden. Auf dem Arduino wird dieser Gruß mit Licht übermittelt, indem eine angeschlossene LED zum Blinken gebracht wird. LEDs sind Licht aussendende Dioden (light emitting diodes). Eine Diode ist ein Bauteil, das

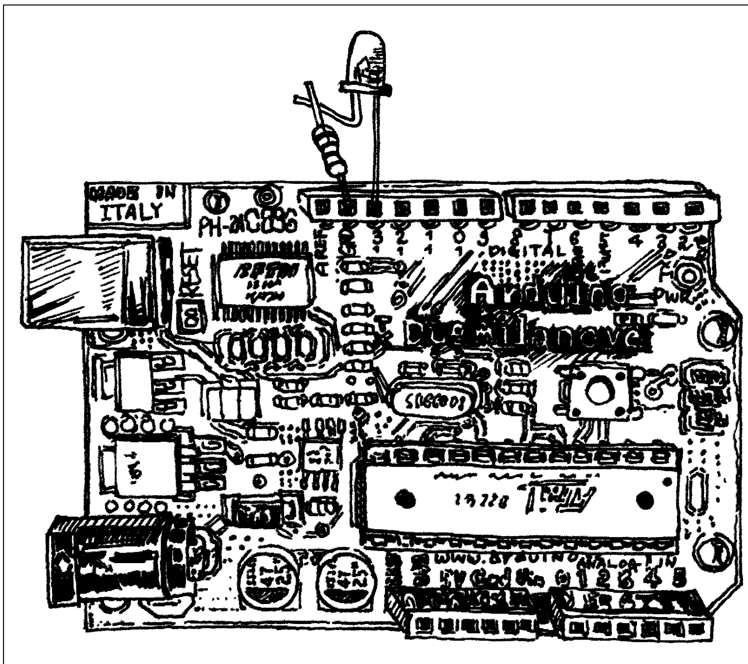
Dies ist ein Auszug aus dem Buch "Arduino - Physical Computing für Bastler, Designer und Geeks", ISBN 978-3-89721-893-2
http://www.oreilly.de/catalog/micprogger/
Dieser Auszug unterliegt dem Urheberrecht. © O'Reilly Verlag 2009

Strom nur in eine Richtung durchlässt, bei einer LED wird dabei sichtbares oder unsichtbares Licht erzeugt.

Die Lichtfarbe einer LED hängt von ihrem Aufbau und den verwendeten Materialien ab. Lange Zeit war es nicht möglich, blaue und weiße LEDs herzustellen. In vielen älteren Geräten gibt es daher nur rote, grüne und gelbe LEDs. Durch neuere Technologien sind mittlerweile fast alle Farben herstellbar. Trotzdem hat sich eine Handvoll Farben etabliert. Mehr Informationen zu genauen Daten verschiedener LEDs finden Sie im Anhang.

Hardware

LEDs besitzen ein langes und eines kurzes Beinchen, die Pins: Anode und Kathode (kleine Merkhilfe: *k*urz = *K*athode). Sind die Pins bereits abgeschnitten worden, etwa um sie irgendwo zu verlöten, findet man die Kathode an der Seite, an der die LED leicht abgeflacht ist. Die LED wird mit der Anode, also der positiven Seite, mit dem 100-Ohm-Widerstand verbunden und dieser am Arduino-Pin 13 angebracht, die negative Seite wird an der Masse (GND, zu englisch »ground«) angeschlossen.



◀ **Abbildung 3-1**
Arduino-Board

Dies ist ein Auszug aus dem Buch "Arduino - Physical Computing für Bastler, Designer und Geeks", ISBN 978-3-89721-893-2
http://www.oreilly.de/catalog/micprogger/
Dieser Auszug unterliegt dem Urheberrecht. © O'Reilly Verlag 2009

Die LED beginnt zu leuchten, wenn durch sie ein Strom von ca. 2 bis 20 mA fließt. Zu viel Strom ist schädlich für die LED und kann sie im schlimmsten Falle sofort zerstören. LEDs dürfen immer nur mit einem Vorwiderstand oder einer speziellen Schaltung betrieben werden, damit der Strom nicht zu groß wird.

- U = Spannung der Batterie/des Arduino-Boards (5 Volt)
- Uled = Flussspannung der LED (siehe Datenblatt, 3 Volt ist für die meisten LEDs ein guter Richtwert)
- I = der gewünschte Strom durch die LED (ein sicherer Wert ist 10 mA)
- R = der zu berechnende Widerstand
- R = (U-Uled) / I

Bei 5 Volt und einer LED mit einer Flussspannung von 3 Volt ergibt sich für 20 mA ein Vorwiderstand von 100 Ohm.

Programmierung

Zunächst wird Pin 13 mit einer Variablen verbunden. Das geschieht, um im weiteren Verlauf des Programms auf *ledPin1* zurückgreifen zu können. So kann schnell der Pin geändert werden, ohne dass das ganze Programm durchgearbeitet werden muss. Zudem steht so keine 13 im Programmverlauf, sondern eine verständliche Bezeichnung, die beim Lesen oder nachträglichen Bearbeiten des Codes behilflich ist.

Variablen können verschiedene Datentypen besitzen, je nachdem, welche Inhalte in ihnen gespeichert werden. In diesem Fall wird eine ganze Zahl (»Integer«) gespeichert, dem Variablennamen wird bei der Initialisierung also der Typ *int* vorangestellt.

Im Setup wird dieser Pin als Ausgang definiert. Im Hauptteil wird schließlich ein digitales Signal auf diesen Ausgang geschrieben. `HIGH` lässt dabei Strom durch den Ausgang fließen, während `LOW` diesen Stromfluss unterbricht. So leuchtet die LED oder erlischt. Dazwischen wird der Programmablauf mit dem Befehl `delay` für 1. 000 Millisekunden, also eine Sekunde, unterbrochen – es wird gewartet.

Hinweis An jeder Stelle des Programms können Kommentare verfasst werden. Sie werden mit // oder # markiert und beim Übersetzen des Programms nicht berücksichtigt. Das erhöht die Verständlichkeit des Codes.

Dies ist ein Auszug aus dem Buch "Arduino - Physical Computing für Bastler, Designer und Geeks", ISBN 978-3-89721-893-2
http://www.oreilly.de/catalog/micprogger/
Dieser Auszug unterliegt dem Urheberrecht. © O'Reilly Verlag 2009

```
int ledPin1 = 13;         // LED an digitalen Pin 13 angeschlossen

void setup()
{
  pinMode(ledPin1, OUTPUT);  // setze digitalen Pin als Output
}

void loop()
{
  digitalWrite(ledPin1, HIGH);   // schalte LED ein
  delay(1000);                  // warte eine Sekunde
  digitalWrite(ledPin1, LOW);    // schalte LED aus
  delay(1000);                  // warte eine Sekunde
}
```

Aufgabe:

- Eine zweite LED anschließen und beide wechselnd blinken lassen.

LEDs über Schalter/Taster steuern

Die digitalen Pins können auch als Eingabepins definiert werden. In diesem Beispiel wird ein Schalter benutzt, um zwischen den beiden LEDs hin- und herzuschalten. Dabei wird die *if-else-Abfrage* eingeführt.

Taster und Schalter finden sich in jedem Haushalt: Entweder direkt in der Wand, um das Licht einzuschalten, oder an Geräten, um deren Funktionen einzustellen, zu starten oder zu stoppen.

Ein Schalter hat die zwei Stellungen Ein und Aus und verbleibt in der zuletzt vom Benutzer gewählten Stellung.

Im Unterschied dazu springt ein Taster nach dem Loslassen sofort wieder in den ungedrückten Zustand zurück. Er eignet sich also nicht, um eine Lampe dauerhaft einzuschalten. Für die Verwendung bei einer Klingel dagegen ist er ideal: Nach dem Betätigen soll sie wieder aufhören zu schellen, der Taster springt zurück.

Schalter und Taster besitzen mindestens zwei Anschlüsse. Es gibt sie aber in beliebig vielen Varianten:

- Mit drei Anschlüssen, als Umschalter (der mittlere Anschluss wird wahlweise mit dem einen oder dem anderen Pin verbunden)
- Mit mehreren Schaltern

Dies ist ein Auszug aus dem Buch "Arduino - Physical Computing für Bastler, Designer und Geeks", ISBN 978-3-89721-893-2
http://www.oreilly.de/catalog/micprogger/
Dieser Auszug unterliegt dem Urheberrecht. © O'Reilly Verlag 2009

- Mit mehr als zwei Schaltstellungen (damit kann man verschiedene Aktionen mit einem einzigen Schalter auswählen)

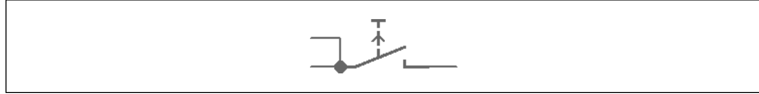


Abbildung 3-2 ▶ Das Schaltzeichen für einen Taster

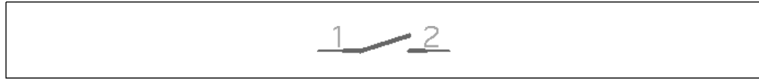


Abbildung 3-3 ▶ Das Schaltzeichen für einen Schalter

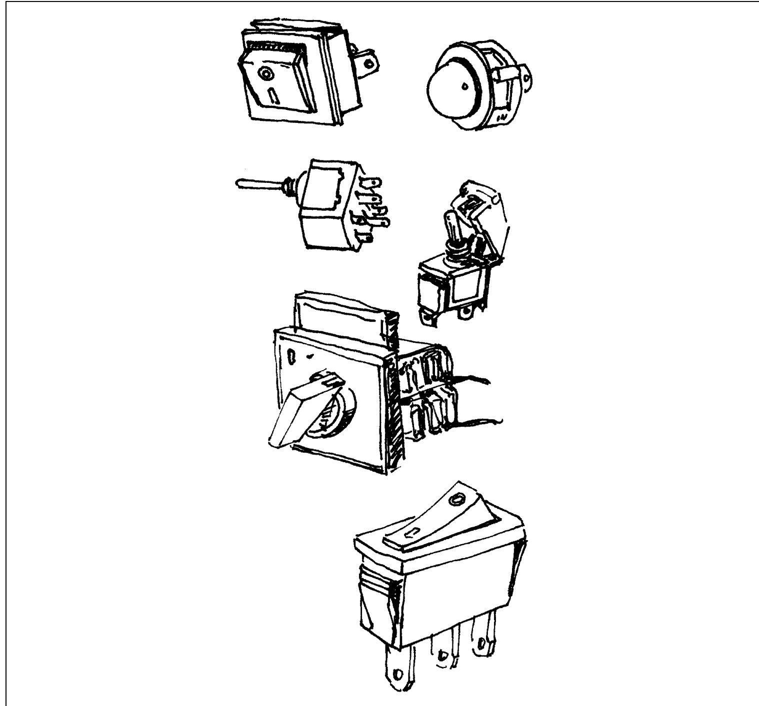


Abbildung 3-4 ▶ Verschiedene Schalter/Taster

Dies ist ein Auszug aus dem Buch "Arduino - Physical Computing für Bastler, Designer und Geeks", ISBN 978-3-89721-893-2
http://www.oreilly.de/catalog/micprogger/
Dieser Auszug unterliegt dem Urheberrecht. © O'Reilly Verlag 2009

Hardware

Um einen Schalter oder Taster mit dem Arduino abzufragen, wird die eine Seite des Bauteils mit einem IO-Pin (in diesem Fall Pin 2) und die andere mit GND verbunden.

Der Arduino erkennt an seinen IO-Pins hohe Spannungen (2–5 Volt) als eine 1 und kleinere Spannungen als eine 0. Ist der Taster/Schalter gedrückt, wird der PIN des Arduino mit GND verbunden, die gemessene Spannung ist also null.

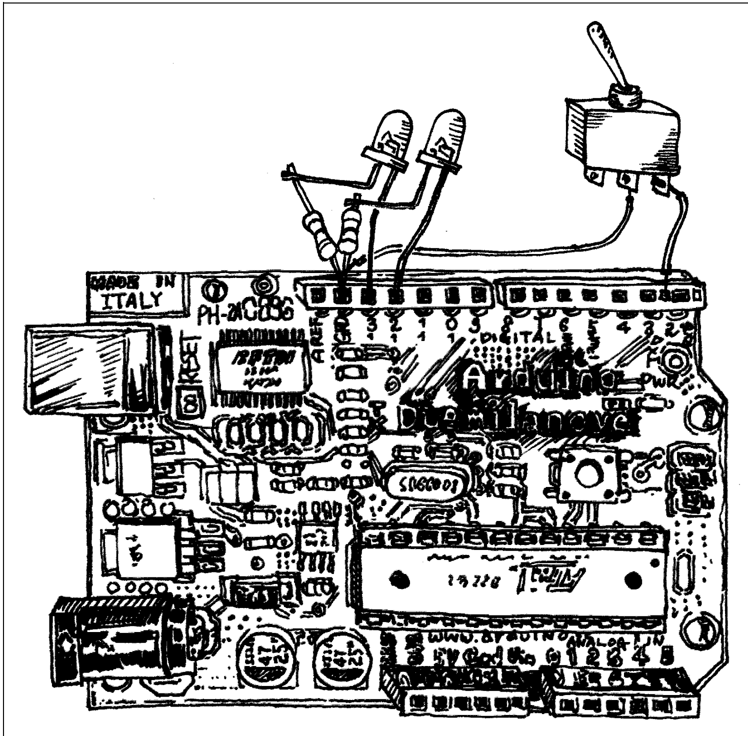


Abbildung 3-5
Schalteraufbau

Dies ist ein Auszug aus dem Buch "Arduino - Physical Computing für Bastler, Designer und Geeks", ISBN 978-3-89721-893-2
http://www.oreilly.de/catalog/micprogger/
Dieser Auszug unterliegt dem Urheberrecht. © O'Reilly Verlag 2009

Programmierung

Der Schalter wird, wie auch die LEDs, auf eine Variable gelegt. Im Setup wird er als Eingang definiert:

```
pinMode(schalter, INPUT);    // setze den digitalen Pin auf Input
```

Ist der Taster/Schalter nicht gedrückt, ist der Stromkreis unterbrochen. Der IO-Pin hat dann keine Verbindung, weder mit GND (0 Volt) noch mit VDD (5 Volt) – die vom Arduino gemessene Spannung ist dann von vielen Umweltfaktoren abhängig und schwankt zufällig hin und her. Um auch hier eine definierte Spannung an diesem PIN zu haben, wird ein sogenannter *Pull-up-Widerstand* eingesetzt, der den Spannungspegel nach oben zieht. Dieser kann entweder als richtiger Widerstand mit einem Wert von 1.000–100.000 verwendet werden, oder man verwendet im Arduino bereits integrierte Pull-up-Widerstände. Je größer der Widerstand desto stromsparender, aber auch umso mehr anfälliger für elektromagnetische Störungen wird unsere Schaltung.

Die internen Pull-up-Widerstände werden mit

```
digitalWrite(schalter, HIGH);
```

eingeschaltet.

Im Hauptteil wird durch `digitalRead` die Eingabe gelesen. Nun wird eine if-else-Abfrage benutzt. Sie steht für eine Verzweigung im Programmablauf: Wenn der Strom durch den Schalter fließt, ist das Eingangssignal mit GND verbunden, also `LOW`, und der zweite Programmblock wird ausgeführt, der durch die geschweiften Klammern begrenzt ist. Ansonsten liegt am Eingang durch den Pull-up ein `HIGH`-Signal vor, also wird der erste Programmblock ausgeführt.

Vergleiche in if-Abfragen werden dabei immer mit zwei Gleichheitszeichen vorgenommen. Wird das Gegenteil benötigt, steht `!=` für »ungleich«.

```
int val = digitalRead(schalter);     //lies Input vom Schalter
int ledPin1 = 13;     // LED an digitalen Pin 13 angeschlossen
int ledPin2 = 12;     // LED an digitalen Pin 12 angeschlossen
int schalter = 2;     // Schalter an digitalen Pin 2 angeschlossen
void setup()
{
  pinMode(ledPin1, OUTPUT);  // setze digitalen Pin als Output
  pinMode(schalter, INPUT);    // setze digitalen Pin auf Input
  digitalWrite(schalter, HIGH);
}
void loop() {
  int val = digitalRead(schalter); //lies Input vom Schalter
  if (val == HIGH) {      // wenn der Wert von val gleich HIGH ist
    digitalWrite(ledPin1, HIGH);   // schaltet LED1 ein
    digitalWrite(ledPin2, LOW);    // schaltet LED2 aus
  }
  else {
    digitalWrite(ledPin1, LOW);
    digitalWrite(ledPin2, HIGH);
  }
}
```

Vier LEDs nacheinander blinken lassen

Schließt man mehr als zwei LEDs an den Arduino an, wird es schnell unübersichtlich: An jeder Stelle muss mehrfach der gleiche Code geschrieben werden. Muss dieser Code nachträglich verändert werden, treten schnell Probleme auf: Versteckt sich in einer Zeile ein Fehler, müssen gleich mehrere ähnliche Zeilen angepasst werden, um ihn zu korrigieren. Schnell wird dadurch auch eine der duplizierten Zeilen übersehen. Generell gilt: Je kleiner der Pro-

Dies ist ein Auszug aus dem Buch "Arduino - Physical Computing für Bastler, Designer und Geeks", ISBN 978-3-89721-893-2
http://www.oreilly.de/catalog/micprogger/
Dieser Auszug unterliegt dem Urheberrecht. © O'Reilly Verlag 2009

grammcode, desto einfacher ist er zu lesen und zu modifizieren, und umso geringer ist auch die Gefahr, dass sich Fehler einschleichen.

Im weiteren Verlauf dieses Beispiels wird erklärt, wie for-Schleifen benutzt werden.

Setup

Vier LEDs werden an die digitalen Pins 10, 11, 12, 13 angeschlossen, und ein Taster an Pin 2 (so wie der Schalter aus dem vorigen Beispiel).

Programmierung

Ein Array (also eine Tabelle) fasst mehrere Variablen in einer zusammen. Über sogenannte Indizes können die einzelnen Werte dann als Elemente angesprochen werden.

Ein Array wird ähnlich wie eine Variable initialisiert. Mit

```
int led[4] = { 10,11,12,13};
```

werden die vier Werte 10, 11, 12 und 13 auf ein Array mit vier Elementen gelegt. Da die Indizes bei 0 beginnen, ist der Wert von `led[0]` nun also 10, der von `led[1]` 11 und so weiter.

Nun können diese Werte mit einer Schleife durchlaufen (»iteriert«) werden. In diesem Fall wird `for()` zusammen mit einer Zählvariable `i` verwendet, die pro Schleifendurchlauf mit `i++` um eins erhöht wird.

Schleifen sind ein weiterer häufig benutzter Bestandteil von Programmiersprachen. Im Arduino-Hauptteil wird eine *loop()-Schleife* verwendet, die unendlich lange weiterläuft (sofern kein Programmabbruch festgelegt wird). Hinzu kommt nun die *for()-Schleife*, der drei Bestandteile hinzugefügt werden: Die Startbedingung (z.B. »beginn bei null«), die Endbedingung (z.B. »zähl bis vier«) und die Zählbedingung (z.B. »zähl immer eins dazu«).

Um Codezeilen zu sparen, wird die Zählvariable direkt in der Schleife initialisiert:

```
for (int i = 0; i<4; i++) {
  pinMode(led[i], OUTPUT);
 }
```

Das bedeutet: Definiere i = 0, solange i kleiner als 4 ist, und erhöhe i um 1 pro Schleifendurchlauf.

Dies ist ein Auszug aus dem Buch "Arduino - Physical Computing für Bastler, Designer und Geeks", ISBN 978-3-89721-893-2
http://www.oreilly.de/catalog/micprogger/
Dieser Auszug unterliegt dem Urheberrecht. © O'Reilly Verlag 2009

Innerhalb der Schleife wird jeder der zuvor eingetragenen Pins als Output festgelegt.

Funktionen

Um eine LED leuchten zu lassen, kann der Code der zweiten Übung benutzt werden. Allerdings wäre dieser nun pro Schritt vier statt bisher zwei Zeilen lang. Bei allen vier LEDs, die nacheinander leuchten sollen, wären das also 16 Zeilen und nicht mehr vier. Zeit für eine Vereinfachung!

Eine Funktion ist eine einzelne Einheit innerhalb eines Programms, die fast überall aufgerufen werden kann. Sie erledigt selbstständig eine Aufgabe und gibt, wenn gewünscht, einen Wert zurück. Je nach Typ dieses Werts werden Funktionen und Variablen, also Typen zugewiesen. Geben sie keinen Wert zurück, ist der Typ *void*. Zudem nimmt eine Funktion Werte an, mit denen sie arbeiten soll – sogenannte »Argumente«

Um eine LED blinken zu lassen und die anderen auszuschalten, muss der Funktion mitgeteilt werden, um welche LEDs es sich handelt. Gleichzeitig ist die Aufgabe der Funktion lediglich die Ausgabe von Signalen. Sie gibt also keinen Wert zurück.

Der Code

```
void setLED(int ledNr) {

}
```

deklariert eine Funktion `setLED`, die keinen Wert zurück gibt (also `void`) und ein Argument vom Typ *int* annimmt. Das bedeutet, dass dieser Funktion ein Wert mit dem Namen `ledNr` mitgegeben wird, den diese Funktion nun verwenden kann. Dabei wird im Speicher eine Kopie der Variablen angelegt und während der Funktion verwendet. So kann die Funktion zwar mit dem Wert arbeiten, verändert ihn aber nicht dauerhaft.

Nun wird jede LED durchlaufen und auf `LOW` gesetzt, sofern sie nicht diejenige ist, die angeschaltet werden soll:

```
for (int i = 0;i<4;i++) {
   if (i == ledNr) {
    digitalWrite(led[i], HIGH);
   }
   else {
     digitalWrite(led[i], LOW);
   }
```

Dies ist ein Auszug aus dem Buch "Arduino - Physical Computing für Bastler, Designer und Geeks", ISBN 978-3-89721-893-2
http://www.oreilly.de/catalog/micprogger/
Dieser Auszug unterliegt dem Urheberrecht. © O'Reilly Verlag 2009

Damit alles nach einem Durchlauf wieder von vorn beginnt, wird nun eine weitere Funktion benötigt, die einen Zähler auf 0 zurücksetzt, sobald sie 4 erreicht. Diese Funktion muss den Zähler, mit dem sie arbeitet, wieder zurückgeben, sie ist also vom Typ *int*.

```
int setCount(int count) {
  if (count == 3) {
    count = 0;
  }
  else {
    count++;
  }
  return count;
}
```

Hinweis Noch einfacher ließe sich das Problem mit dem »Modulo« lösen, dem Rest, der beim Teilen durch eine Zahl entsteht. Erhöht man pro loop()-Durchlauf die Variable count um eins, kann man `setLED` jederzeit mit `setLed(count mod 4)` oder auch `setLed(count % 4)` aufrufen. Es wird immer eine Zahl zwischen 0 und 3 übergeben.

Da der Taster sehr schnell abgefragt wird, soll sein Impuls nur dann zu einem Lichtwechsel führen, wenn er gerade gedrückt wurde. Dazu wird eine Variable `oldVal` eingeführt, auf die pro loop()-Durchlauf der aktuelle Wert des Eingangs geschrieben wird. Nur wenn diese sich im Vergleich zum vorigen Durchlauf ändert, wird die Funktion `setLED` wirklich aufgerufen.

Das sieht nun also wie folgt aus:

```
int led[4] = { 10,11,12,13};
int oldVal = 0;
int counter = 0;
void setup() {
   for (int i = 0; i<4; i++) {
   pinMode(led[i], OUTPUT);
 }

void setLED(int ledNr) {
   for (int i = 0;i<4;i++) {
     if (i == ledNr) {
       digitalWrite(led[i], HIGH);
     }
     else {
       digitalWrite(led[i], LOW);
     }
  }
}
```

Dies ist ein Auszug aus dem Buch "Arduino - Physical Computing für Bastler, Designer und Geeks", ISBN 978-3-89721-893-2
http://www.oreilly.de/catalog/micprogger/
Dieser Auszug unterliegt dem Urheberrecht. © O'Reilly Verlag 2009

```
int setCount(int count) {
  if (count == 3) {
    count = 0;
  }
  else {
    count++;
  }
  return count;
}

void loop()
{
  int val = digitalRead(taster);     //lies Input vom Taster
  if (val != oldVal && val == HIGH) {
    count = setCount();
    setLED(count);
    delay(10);    // warte ein wenig
  }
  oldVal = val;
}
```

Die Debounce-Bibliothek

Möchte man den Taster als Umschalter verwenden, kann es oft zu mehrfachen Betätigungen kommen, selbst wenn man eine Abfrage wie die obige einbaut. Gegen dieses sogenannte »Prellen« hilft die Debounce-Bibliothek, mit der wir jetzt einen kurzen Ausflug in die Welt der Libraries machen.

Viele Anwendungen rund um Arduino sind im Prinzip recht einfach, benötigen aber hohen Programmieraufwand. Zum Beispiel möchte man einen bestimmten Sensor eigentlich einfach nur auslesen oder einen Motor nur mit einer bestimmten Geschwindigkeit betreiben. Die Technik dahinter ist aber recht komplex und benötigt verschiedene Signale und Voreinstellungen vom Arduino.

Bibliotheken fassen diese Funktionen zusammen. Denn wer einmal die Arbeit gemacht hat, kann sie auch anderen zur Verfügung stellen und sie über eine sogenannte API benutzen lassen. Der Begriff API steht für »Application Programming Interface« und ist quasi das Tor zur Bibliothek: ein Satz von Funktionen, mit denen die komplexeren Funktionalitäten der Bibliothek verwendet werden können. Für den Arduino gibt es eine ganze Reihe von Bibliotheken, die zum Teil die Programmierung einfacher machen, so wie die Debounce-Bibliothek, um die es hier geht. Wichtiger sind aber diejenigen Bibliotheken, die Bauteile und Geräte ansteuern können. Auf der Arduino-Webseite finden Sie Listen von Bibliotheken, genauso wie im Verzeichnis von *http://www.freeduino.org*.

Dies ist ein Auszug aus dem Buch "Arduino - Physical Computing für Bastler, Designer und Geeks", ISBN 978-3-89721-893-2
http://www.oreilly.de/catalog/micprogger/
Dieser Auszug unterliegt dem Urheberrecht. © O'Reilly Verlag 2009

Unter Prellen oder »Bouncing« versteht man ein mechanisches Problem, das durch die Eigenschaft von Schaltern und Tastern hervor gerufen wird: Weil in diesen Bauteilen federnde Effekte auftreten, öffnen und schließen sie sich beim Betätigen und Loslassen mehrmals, statt dass sofort ein elektrischer Kontakt zustande kommt. Hiergegen hilft nur, dass nach einem Kontakt zunächst jede weitere Eingabe vom Taster/Schalter für einige Millisekunden gesperrt wird.

Bibliotheken werden mit dem Befehl `include` in den aktuellen Sketch eingebunden. Damit sie beim Kompilieren auch gefunden werden, muss der komplette Ordner mit der Bibliothek ins Arduino-Verzeichnis unter *hardware\libraries* kopiert werden. Nun kann die Bibliothek auch in der Programmierumgebung über SKETCH → IMPORT LIBRARIES eingefügt werden. Hat man die Debounce-Bibliothek von *http://www.arduino.cc/playground/Code/Debounce* heruntergeladen, entpackt und in das entsprechende Verzeichnis kopiert, bindet man sie mit

```
#import <debounce.h>
```

in den Sketch ein. Nun stehen drei Funktionen zur Verfügung, die es bisher in der Arduino-Programmiersprache noch nicht gab: `read()`, `update()` und `write()`. Diese Funktionen gehören zu einem Objekt, das die aktive Bibliothek repräsentiert. Objekte sind Daten- und Funktionssammlungen. Das heißt, sie beherbergen neben den API-Funktionen auch weitere Funktionalitäten und Datenstrukturen. Man kann ein Objekt also mit einer Küche vergleichen, in der Essen vorbereitet wird. Dabei werden Herde und Öfen bedient, Teig vorgehalten und einzelne Gerichte gestapelt, bis eine Bestellung beisammen ist. Über eine oder mehrere Luken, Türen oder Zettelsysteme spricht die Bedienung mit dieser Küche und tauscht Informationen oder Nahrungsmittel aus. Dabei bleibt natürlich auch nicht alles zwischen Küche und Bedienung. Vielmehr gibt es einen Hinterausgang, über den neue Rohstoffe bestellt und geliefert werden.

Ein Objekt ist dabei immer die Instanz einer sogenannten Klasse. Das bedeutet, dass die importierte Bibliothek auch mehrfach verwendet werden kann, beispielsweise wenn man mehrere gleiche Geräte an einen Arduino hängen möchte. Klassen sind die eigentlichen Programmierkonstrukte, die im laufenden Betrieb zu Objekten werden. Sie verfügen zusätzlich über besondere Funktionen. Der Konstruktor wird zum Beispiel aufgerufen, wenn das Objekt

Dies ist ein Auszug aus dem Buch "Arduino - Physical Computing für Bastler, Designer und Geeks", ISBN 978-3-89721-893-2
http://www.oreilly.de/catalog/micprogger/
Dieser Auszug unterliegt dem Urheberrecht. © O'Reilly Verlag 2009

instanziiert, also erzeugt wird. Im Falle des Debouncers wird nun ein Objekt erstellt, dem 20 Millisekunden Debounce-Zeit eingeräumt werden:

```
// verbinde Taster mit Pin 5
int taster = 5;
// erschaffe ein neues Objekt vom Typ "debounce" mit 20
// Millisekunden Debounce-Zeit, verbunden an Pin 5
Debounce debouncer = Debounce( 20 , taster );
```

Im eigentlichen Programm können nun die oben erwähnten Funktionen aufgerufen werden. Die Funktion `update()` prüft nach, ob sich der Status geändert hat, und vermeldet das Ergebnis mit dem Rückgabewert `TRUE` oder `FALSE`. Mit `read()` kann dann der derzeitige Pin-Status gelesen werden, man erhält also `HIGH` oder `LOW` zurück. Um nun also die Informationen des Tasters korrekt zu lesen, benötigt man

```
// Debouncer-Status updaten
debouncer.update();
// Wert über den Debouncer auslesen
int tasterVal = debouncer.read();
```

Die anderen Funktionen in der Biblitohek sind `write()` und `interval()`. `write()` erlaubt das Senden eines digitalen Signals über den Debouncer. Da der Pin schon festgelegt ist, wird als Argument nur das Signal, also `HIGH` oder `LOW`, benötigt. Mit `interval()` lässt sich schließlich das Debounce-Intervall ändern, ohne dass das Objekt neu instanziiert werden muss.

Das gesamte Programm mit einer LED und der Debouncer-Bibliothek sieht also so aus:

```
#import <debounce.h>
// verbinde Taster mit Pin 5
int taster = 5;
// erschaffe ein neues Objekt vom Typ "debounce" mit 20
// Millisekunden Debounce-Zeit, verbunden an Pin 5
Debounce debouncer = Debounce( 20 , taster );
int ledPin1 = 13;          // LED an digitalen Pin 13
                           // angeschlossen
int schalter = 2;          // LED an digitalen Pin 13
                           // angeschlossen

void setup()
{
  pinMode(ledPin1, OUTPUT);  // setze digitalen Pin als Output
  pinMode(schalter, INPUT);    // setze digitalen Pin auf Input
  digitalWrite(schalter, HIGH);
}
```

Dies ist ein Auszug aus dem Buch "Arduino - Physical Computing für Bastler, Designer und Geeks", ISBN 978-3-89721-893-2
http://www.oreilly.de/catalog/micprogger/
Dieser Auszug unterliegt dem Urheberrecht. © O'Reilly Verlag 2009

```
void loop() {

  // Debouncer-Status updaten
  debouncer.update();
  // Wert über den Debouncer auslesen
  int tasterVal = debouncer.read();
  int val = digitalRead(schalter);     //liest Input vom Schalter
  if (val == HIGH) {      // wenn der Wert von val gleich HIGH ist
    digitalWrite(ledPin1, HIGH);   // schaltet LED1 ein
    digitalWrite(ledPin2, LOW);    // schaltet LED2 aus
  }
  else {
    digitalWrite(ledPin1, LOW);
    digitalWrite(ledPin2, HIGH);
  }
}
```

Aufgabe:

- Einen »Funkenhimmel« bauen, der so viele LEDs wie möglich verwendet und sie zufällig blinken lässt.

Hinweis Die Funktion random(13) gibt bei jedem Aufruf eine neue zufällige Zahl zwischen 0 und 12 zurück.

LEDs mit Pulsweitenmodulation verschieden hell leuchten lassen

Bis zu diesem Punkt gab es für die LEDs nur die zwei Zustände `HIGH` und `LOW`, also An und Aus. Das liegt daran, dass über den Arduino keine unterschiedlichen Stromstärken ausgegeben werden können. Das menschliche Auge hingegen lässt sich recht leicht austricksen. Da es nur 25 Bilder pro Sekunde wahrnehmen kann, nimmt es Flackern in Frequenzen, die weit darüber liegen, nicht mehr als solches wahr. Vielmehr erscheint uns das Licht als heller oder dunkler, je nach Häufigkeit des Flackerns. Der Arduino hat diese sogenannte *Pulsweitenmodulation* (PWM) auf sechs Pins direkt eingebaut. Diese lassen sich nicht nur mit `digitalWrite()` ansprechen, sondern auch mit der Funktion `analogWrite()`, die nicht nur die Signale `LOW` und `HIGH` als Argumente annimmt, sondern 8 Bit verarbeiten kann, also 255 unterschiedliche Werte.

Hardware

Eine LED an Pin 10 anschließen.

Dies ist ein Auszug aus dem Buch "Arduino - Physical Computing für Bastler, Designer und Geeks", ISBN 978-3-89721-893-2
http://www.oreilly.de/catalog/micprogger/
Dieser Auszug unterliegt dem Urheberrecht. © O'Reilly Verlag 2009

Software

Die LED wird mit Pin 10 verbunden und als OUTPUT definiert. Das Programm soll nun von 0 bis 255 zählen und pro Schritt ein »stärkeres«, also schneller gepulstes Signal an den Ausgang senden. Anschließend soll der Zähler wieder von 255 bis 0 laufen.

Dafür werden ein Zähler (counter) und ein Vorzeichenmarker (changeMarker) initialisiert:

```
int counter = 0;
int changeMarker = 1;
```

Die Funktion changeCounter addiert nun pro Aufruf einmal changeMarker zum counter. Erreicht der Counter den Wert 255 oder 0, wird der changeMarker auf -1 bzw. 1 gesetzt:

```
int ledPin = 10
void setup() {|
  pinMode(ledPin, OUTPUT);
}
int counter = 0;
int changeMarker = 1;
int changeCounter() {
  if (counter == 255) {
    changeMarker = -1;
  }
  if (counter == 0) {
    changeMarker = 1;
  }
  counter = counter + changeMarker;
  return counter;
}
```

Nun steht einem pulsierenden Licht nichts mehr im Wege:

```
void loop()
{
  counter = changeCounter();
  analogWrite(led, counter);
  delay(10);    // ein bisschen warten ...
}
```

Beim genauen Hinsehen stellt man fest, dass das Licht nicht gleichmäßig pulsiert. Das liegt daran, dass das Auge das Licht nicht linear wahrnimmt, also geradlinig, sondern als logarithmische Funktion. Wenn die LED fast aus ist, sieht man kleinere Veränderungen sehr stark; ist die LED fast ganz eingeschaltet, ändert sich praktisch gar nichts mehr. Um diesem Umstand entgegenzuwirken, kann changeCounter() zu einer aufwendigeren Version ausgebaut werden, sodass der Logarithmus korrekt errechnet wird. Viel einfacher ist

Dies ist ein Auszug aus dem Buch "Arduino - Physical Computing für Bastler, Designer und Geeks", ISBN 978-3-89721-893-2
http://www.oreilly.de/catalog/micprogger/
Dieser Auszug unterliegt dem Urheberrecht. © O'Reilly Verlag 2009

jedoch eine Wertetabelle die uns diese Übersetzung abnimmt, sie gilt für alle Projekte mit gedimmten LEDs. Für diesen Zweck reichen 64 Werte aus, da die PWM mit ihren 256 Stufen keine weitere Verfeinerung erlaubt:

```
int loga[64] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
16, 18, 20, 22, 25, 28, 30, 33, 36, 39, 42, 46, 49, 53, 56, 60,
64, 68, 72, 77, 81, 86, 90, 95, 100, 105, 110, 116, 121, 127, 132,
138, 144, 150, 156, 163, 169, 176, 182, 189, 196, 203, 210, 218,
225, 233, 240, 248, 255};
```

Das Programm muss nun an zwei Stellen geändert werden. In der Hauptfunktion wird

```
analogWrite(led, counter)
```

ersetzt durch

```
  analogWrite(led, loga[counter]);
```

Schließt man vier LEDs an, verändert sich im Prinzip nicht viel. Um die LEDs in einer Welle pulsieren zu lassen, benötigt man vier Counter und vier Marker:

```
int counter[4] = {0,32,64,32};
int changeMarker[4] = {1, 1, -1, -1};
```

Das wird nun auch in die Funktion `changeCounter()` eingebaut, die nun in der Lage ist, einen der vier Counter bzw. Marker zu verändern. Zudem werden die ersten beiden Zeilen geändert, so dass die Funktion nun wie folgt aussieht:

```
void changeCounter(int i) {
  if (counter[i] == 63) {
    changeMarker[i] = -1;
  }
  if (counter[i] == 0) {
    changeMarker[i] = 1;
  }
  counter[i] = counter[i] + changeMarker[i];
}
```

So wird nun auch die Hauptfunktion verändert:

```
void loop()
{
  for (int i = 0;i<4;i++)
  {
    changeCounter(i); // verändere den Counter der LED i
    analogWrite(led[i], loga[counter[i]]);
    //  setze die neue Helligkeit der LED i
  }
  delay(10);    // ein bisschen warten ...
}
```

Dies ist ein Auszug aus dem Buch "Arduino - Physical Computing für Bastler, Designer und Geeks", ISBN 978-3-89721-893-2
http://www.oreilly.de/catalog/micprogger/
Dieser Auszug unterliegt dem Urheberrecht. © O'Reilly Verlag 2009

Aufgabe:

- Den Taster wieder anschließen und die Lichtstärke damit regulieren lassen.

Dies ist ein Auszug aus dem Buch "Arduino - Physical Computing für Bastler, Designer und Geeks", ISBN 978-3-89721-893-2
http://www.oreilly.de/catalog/micprogger/
Dieser Auszug unterliegt dem Urheberrecht. © O'Reilly Verlag 2009

Mach es bunt

Nun kann schon die erste Lampe gebaut werden, die mit Farbverläufen eine gemütliche Atmosphäre im Raum schafft. Dazu werden drei LEDs in den Grundfarben Rot, Grün und Blau benötigt. Mit diesen Farben lassen sich durch Mischung alle weiteren Farben des sichtbaren Spektrums erzeugen, indem man jede LED mit einer entsprechenden Helligkeit pulst. Die folgende Lampe wählt zufällig Farben aus und dimmt sich langsam von der einen zur anderen und weiter zur nächsten.

Hardware

Drei LEDs in den Grundfarben Rot, Grün und Blau werden über einen passenden Widerstand (100 Ohm) mit dem Arduino verbunden. Welcher Widerstand genau benötigt wird und mit welcher Eingangsspannung und Stromstärke die LEDs betrieben werden, hängt von der Bauart ab. Genauere Daten können Sie dem Datenblatt entnehmen. Es gibt auch fertige RGB-LEDs, die mit sechs Beinen ausgestattet sind und somit auf die gleiche Weise angeschlossen werden wie drei einzelne. Je nach Stromstärke müssen die Leuchtdioden an einen anderen Stromkreis angeschlossen werden, besonders wenn eine stärkere Leistung erwartet wird. Am Ende dieses Kapitels wird unter dem Stichwort »Mehr Power« erläutert, wie man das zum Beispiel mit Transistoren machen kann. Das folgende Programm geht davon aus, dass die rote LED an Pin 9, die grüne an 10 und die blaue an 11 angeschlossen ist.

Programmierung

Wie weiter oben beschrieben, werden drei LEDs in einer Tabelle initialisiert und mit den passenden Pins verbunden. Auch die Logarithmustabelle wird benötigt. Zudem braucht man zwei Tabellen, die Quell- und Zielfarbwert speichern.

```
// PINs für die RGB-LED
int ledPin[] = { 9, 10, 11};

// Startfarbe
```

```
int sourceValue[3] = { 0, 0, 0};
// Zielfarbe
int targetValue[3] = { 0, 0, 0};
// Überblendposition
int currentPos = 0;
```

In der Setup-Funktion werden die beiden Farben mit zufälligen Werten initialisiert und die verwendeten PINs als Ausgang geschaltet.

```
void setup()
{
  for (int i = 0; i < 3; i++) {
    // Startfarbe zufällig wählen
    sourceValue[i] = random(63);
    // Zielfarbe zufällig wählen
    targetValue[i] = random(63);
    // LED-Pin = Ausgang
    pinMode(ledPin[i], OUTPUT);
  }
}
```

`currentPos` wird jetzt der Mittelwert nach der Formel *farbe = start * (128-currentPos) + ende * currentPos* auf ihre entsprechende LED geschrieben und die `loop()`-Funktion abgeschlossen:

```
void loop()
{
  // für alle drei Grundfarben Mittelwert an der
  // Stelle currentPos berechnen und ausgeben
  for (int i = 0; i < 3; i++) {
    int helligkeit = (sourceValue[i] * (128 - currentPos) +
                      targetValue[i] * currentPos)/128;
    analogWrite(ledPin[i],helligkeit);
  }
  currentPos++;
  if (currentPos > 128) {
    for (int i = 0; i < 3; i++) {
      sourceValue[i] = targetValue[i];
      targetValue[i] = random(63);
    }
    currentPos = 0;
  }
  // 10ms warten.
  delay(10);
}
```

Und fertig ist die Lampe! Nun kann damit experimentiert werden. Mit Sicherheit finden sich viele Möglichkeiten, dieses Projekt zu erweitern und anzupassen. Zum Beispiel könnte man eine weitere RGB-LED anschließen und die aktuelle Farbe von einer zur anderen wandern lassen.

Dies ist ein Auszug aus dem Buch "Arduino - Physical Computing für Bastler, Designer und Geeks", ISBN 978-3-89721-893-2
http://www.oreilly.de/catalog/micprogger/
Dieser Auszug unterliegt dem Urheberrecht. © O'Reilly Verlag 2009

Helligkeit mit einem Drehknopf steuern

Zum Abschluss dieses Workshops soll nun die Helligkeit einer LED mit einem Drehknopf gesteuert werden. Dazu wird ein sogenanntes Potentiometer verwendet. Es besteht aus einem Metallstift, der auf einer Fläche angebracht ist. Wird diese Fläche durch Drehen entlang des Stiftes bewegt, verändert sich der Widerstand und es fließt weniger Spannung an den Ausgang. Ein Potentiometer ist somit ein sogenannter resistiver Sensor. Mehr Informationen über resistive Sensoren finden Sie auch in Kapitel 7.

Die Messung erfolgt über den analogen Eingang, wobei das Signal niemals genau ist. Abweichungen im Wert müssen bei der Programmierung berücksichtigt werden. Somit können nicht die vollen 1.024 Stufen ausgeschöpft werden, was angesichts von 256 möglichen Helligkeitsstufen bei den pulsweitenmodulierten Digitalausgängen allerdings kein Problem darstellt, zumal ohnehin nur die 64 Stufen aus der Tabelle verwendet werden sollen.

Hardware

An den Arduino wird zunächst eine LED an einen PWM-fähigen Digitalausgang angeschlossen, in diesem Fall Ausgang 9. Zudem wird ein Potentiometer mit dem mittleren Pin an den analogen Eingang 0 angeschlossen, die anderen beiden Pins an GND und den 5-Volt-Anschluss. Dabei kommt es darauf an, wie der Knopf gedreht werden soll: Je nachdem, wie der Strom fließt (von links nach rechts oder umgekehrt), muss auch gedreht werden, um den die Spannung zu erhöhen oder zu senken.

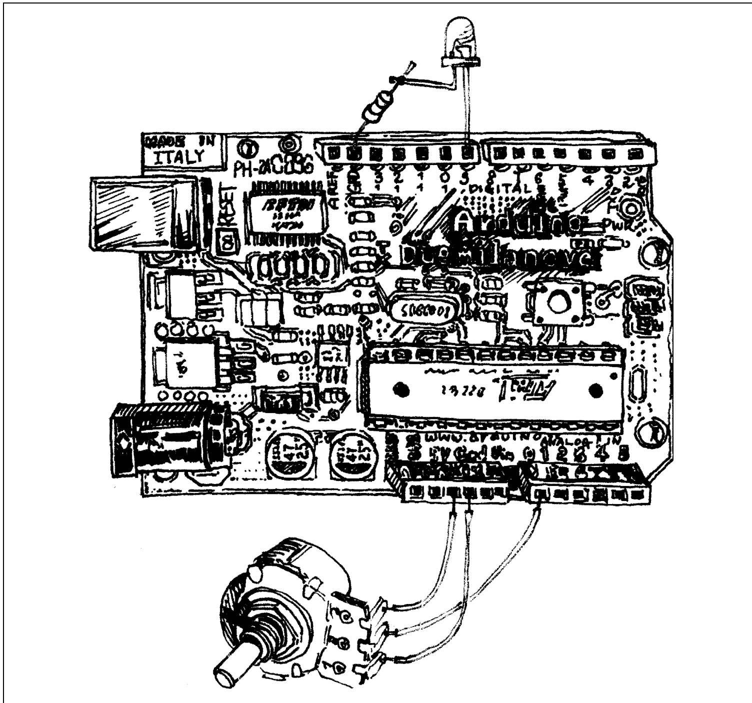
Programmierung

Zunächst werden die beiden Pins festgelegt und im Setup als Ein- bzw. Ausgang definiert. Zudem wird wieder die Tabelle für die Helligkeitsstufen verwendet.

```
int potPin = 0;    // lege potPin als Pin 0 fest
int ledPin = 9;   // lege ledPin als Pin 9 fest
int potiVal = 0;  // eine Variable, um den Input zu speichern
int loga[64] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
16, 18, 20, 22, 25, 28, 30, 33, 36, 39, 42, 46, 49, 53, 56, 60,
64, 68, 72, 77, 81, 86, 90, 95, 100, 105, 110, 116, 121, 127, 132,
138, 144, 150, 156, 163, 169, 176, 182, 189, 196, 203, 210, 218,
225, 233, 240, 248, 255};

void setup() {
  pinMode(ledPin, OUTPUT);  // initialisiere ledPin als Output
}
```

Dies ist ein Auszug aus dem Buch "Arduino - Physical Computing für Bastler, Designer und Geeks", ISBN 978-3-89721-893-2
http://www.oreilly.de/catalog/micprogger/
Dieser Auszug unterliegt dem Urheberrecht. © O'Reilly Verlag 2009



◀ **Abbildung 3-6**
Aufbau für PWM-LE

Dies ist ein Auszug aus dem Buch "Arduino - Physical Computing für Bastler, Designer und Geeks", ISBN 978-3-89721-893-2
http://www.oreilly.de/catalog/micprogger/
Dieser Auszug unterliegt dem Urheberrecht. © O'Reilly Verlag 2009

Nun können der analoge Eingang gelesen und das Signal gespeichert werden. Anschließend werden das Signal durch 16 geteilt und der Ausgangswert anhand des oben erklärten Sinus berechnet. Schließlich wird der entsprechende Wert in der Logarithmustabelle auf den digitalen Ausgang geschrieben, die LED wird nun vom Potentiometer gesteuert.

```
void loop()
{
  int potiVal = analogRead(potPin); // lies analoges
                                    // Eingangssignal
  potiVal = potiVal / 16; // teile den Signalwert durch 16
  analogWrite(led, loga[potiVal]);  // schreibe den Tabellenwert
                                    // auf den Ausgang
  delay(10);
}
```

Flower Power

Nicht nur die Helligkeit kann mit einem Potentiometer geregelt werden, sondern auch die Wunschfarbe passend zum Abendkleid.

Dazu verwenden wir den Quellcode von »Mach es bunt« in diesem Kapitel und verändern nur die Funktion `loop()`, sodass abhängig von der Reglerstellung eine Farbe ausgewählt wird. Nun gibt es keinen einfachen Weg durch alle Farben, es muss also ein bestimmter Verlauf einprogrammiert werden. Die 1.024, die maximal vom Potentiometer zurückgeliefert werden, teilen wir dazu in vier Teile auf. Im ersten Teil dimmen wir von Rot nach Grün, im zweiten Teil von Grün nach Blau, im dritten von Blau nach Rot, und mit dem verbleibenden Teil dimmen wir nach Weiß. Durch die Logarithmustabelle benötigen wir Werte von 0–63 für jede Farbe, wir können den Potentiometerwert also noch durch 4 teilen, um in jedem Teilabschnitt 64 Punkte zu haben.

```
void loop() {
  int potiVal = analogRead(potPin) / 4; // lies analoges
                                        // Eingangssignal

  if (potiVal < 64)
  {                                           // fade Rot zu Grün
    analogWrite(ledPin[0], loga[63-potiVal]); // Rot
    analogWrite(ledPin[1], loga[potiVal]);    // Grün
    analogWrite(ledPin[2], loga[0]);          // Blau
  }
  else if (potiVal < 128)
  {                                           // fade Grün zu Blau
    analogWrite(ledPin[0], loga[0]);          // Rot
    analogWrite(ledPin[1], loga[127 - potiVal)]);// Grün
    analogWrite(ledPin[2], loga[potiVal - 64]);// Blau
  }
  else if (potiVal < 192)
  {                                           // fade Blau zu Rot
    analogWrite(ledPin[0], loga[potiVal - 128]);// Rot
    analogWrite(ledPin[1], loga[0]);          // Grün
    analogWrite(ledPin[2], loga[191 - potiVal]);// Blau
  }
  else
  {                                           // fade Rot zu Weiß
    analogWrite(ledPin[0], loga[63]);         // Rot
    analogWrite(ledPin[1], loga[255-potiVal]); // Grün
    analogWrite(ledPin[2], loga[255-potiVal]); // Blau
  }
}
```

Mehr Power

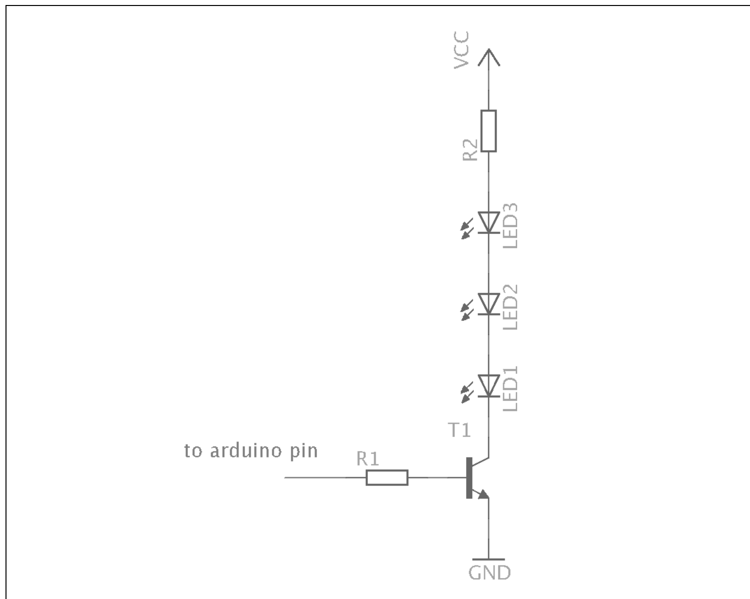
Bis jetzt wurden nur sehr kleine LEDs mit dem Arduino gesteuert, ein Pin kann nur ca. 20 mA Strom liefern. Bei einer 3-Volt-LED entspricht das 0,06 Watt. Damit bekommt man natürlich kein Zimmer erleuchtet.

Dies ist ein Auszug aus dem Buch "Arduino - Physical Computing für Bastler, Designer und Geeks", ISBN 978-3-89721-893-2
http://www.oreilly.de/catalog/micprogger/
Dieser Auszug unterliegt dem Urheberrecht. © O'Reilly Verlag 2009

Für größere LED können Transistoren oder Mosfets (Metalloxid-Halbleiter-Feldeffekttransistoren) verwendet werden, die schnell genug sind, um auch PWM für das Dimmen der LEDs zu ermöglichen. Transistoren sind in der Lage, mit einem kleinen Signal auf der einen Seite einen großen Stromkreis auf der anderen zu schalten.

Transistoren haben ihren Aufschwung in den 1960er Jahren erlebt, als sie zum ersten Mal in Form von Feldeffekttransistoren auf Galliumarsenid praktikabel genug wurden. Vorher hatte man Computer mit Vakuumröhren gebaut, die man günstig herstellen konnte und die das einzige Schaltelement waren, das auch schnell genug für diese Anforderungen war. Ein Transistor besteht aus drei Anschlüssen: Basis, Emitter und Kollektor. Transistoren gibt es in den beiden Polungen NPN und PNP. Ein NPN-Transistor als Schalter ist am besten für die negative Seite der Last geeignet (LED, Lampe, Motor usw.), PNP für die positive. Mehr Informationenen zu Relays und größeren Stromstärken findet man unter Anderem auf der Arduino-Seite unter:*http://www.arduino.cc/playground/uploads/Learning/relays.pdf* sowie bei der New York University unter http://itp.nyu.edu/physcomp/Tutorials/HighCurrentLoads.

Dies ist ein Auszug aus dem Buch "Arduino - Physical Computing für Bastler, Designer und Geeks", ISBN 978-3-89721-893-2
http://www.oreilly.de/catalog/micprogger/
Dieser Auszug unterliegt dem Urheberrecht. © O'Reilly Verlag 2009



◀ **Abbildung 3-7**
Transistorausgangsschaltung

Gesteuert wird der Transistor über den Stromfluss, der in die Basis hineinfließt. Dabei muss der Basisanschluss mindestens 0,7 Volt höher liegen als der Kollektor. Dazu wird die Basis über einen

Widerstand von 1.000–10.000 mit dem Ausgangspin des Arduino angeschlossen. Der Kollektor wird direkt mit GND verbunden, und an den Emitter kommt dann die eine Seite der Last, die andere Seite wird mit 5V verbunden. Das Schöne an dieser Schaltung ist, dass die Spannung auch viel größer sein kann, ohne dem Arduino zu schaden (das Maximum entnehmen Sie bitte im Datenblatt des Transistors). Dadurch kann man viele LEDs in einer Reihe verbinden oder besonders starke LEDs verwenden, um den Raum hell zu erleuchten.

Und nun wird losgebastelt

Nun sollte es möglich sein, eine Lampe zu basteln. In den letzten Jahren sind LEDs immer günstiger und auch heller geworden, sodass inzwischen auch Lampen erschwinglich sind, die einen ganzen Raum erleuchten können. Interessant ist das insbesondere, wenn RGB-LEDs verwendet werden, die mit der oben gezeigten Logarithmustabelle immerhin 262.144 Farben darstellen können. Um noch weichere Farbübergänge zu erzeugen, müsste eine PWM von mehr als 8 Bit verwendet werden. Ein weiterer Vorteil von LEDs ist ihre Größe: Glühbirnen oder Leuchtstofflampen benötigen recht viel Platz, während Leuchtdioden sehr klein sind und in Tischtennisbälle, kleine Kästen, Flaschen und allerhand andere Behältnisse passen. Doch Vorsicht: Auch LEDs können je nach Stärke sehr heiß werden und müssen womöglich auf einen Kühlkörper aufgebracht werden (die gibt es inzwischen auch im Fachhandel, zum Beispiel in Onlineshops, die LEDs verkaufen). Man kann aber auch einen CPU-Kühler aus einem alten PC verbauen. Wichtig ist dabei allerdings, dass der Körper auch ausreichend groß ist und man Wärmeleitpaste verwendet.

Mit den sechs PWM-fähigen Pins des Arduino können zwei RGB-LEDs angeschlossen und entsprechend gepulst werden. Bringt man diese Lichter in entsprechender Stärke an zwei Enden des Raumes an oder lässt sie in entgegengesetzte Richtungen entlang einer Wand leuchten, ergeben sich schöne und stimmungsvolle Effekte. Diese Lampe lässt sich natürlich auch mit weiteren Sensoren ausstatten. Möglich ist zum Beispiel ein Mikrofon, das an einen Tiefpassfilter angeschlossen wird und die Lampe pulsen lässt, wenn ein Bass ertönt. Oder man nimmt einen passenden Gassensor, der das Licht melden lässt, wenn die Konzentration eines bestimmten Stoffes in der Luft zu hoch wird. Eine entsprechende Anleitung finden

Dies ist ein Auszug aus dem Buch "Arduino - Physical Computing für Bastler, Designer und Geeks", ISBN 978-3-89721-893-2
http://www.oreilly.de/catalog/micprogger/
Dieser Auszug unterliegt dem Urheberrecht. © O'Reilly Verlag 2009

Sie z.B. unter *http://www.instructables.com/id/How-To-Smell-Pollutants/?ALLSTEPS*.

Ein anderes mögliches Projekt könnte ein Lichtwecker sein, wie er für viel Geld auch im Handel erhältlich ist. Dabei wird eine Lampe ab einer festgelegten Uhrzeit allmählich immer heller, um Dämmerung und Sonnenaufgang zu simulieren. Einen solchen Wecker hat zum Beispiel Mark Ivey gebaut und unter *http://zovirl.com/2008/12/11/arduino-prototype-for-a-sunrise-alarm/* erläutert.

Ein gelungenes Beispiel ist die Milk Lamp von David Hayward, die aus einer Reihe an der Decke hängender Milchlampen mit weißen LEDs darin besteht. Beim Dimmen werden diese nicht wie gewohnt heller und dunkler geregelt, sondern sie werden der Reihe nach ein- und ausgeschaltet. Eine genaue Beschreibung des Projekts, das mithilfe der Informationen aus diesem Kapitel auch nachprogrammiert werden kann, finden Sie unter *http://functional-autonomy.net/blog/?p=442*.

Dies ist ein Auszug aus dem Buch "Arduino - Physical Computing für Bastler, Designer und Geeks", ISBN 978-3-89721-893-2
http://www.oreilly.de/catalog/micprogger/
Dieser Auszug unterliegt dem Urheberrecht. © O'Reilly Verlag 2009

Dies ist ein Auszug aus dem Buch "Arduino - Physical Computing für Bastler, Designer und Geeks", ISBN 978-3-89721-893-2
http://www.oreilly.de/catalog/micprogger/
Dieser Auszug unterliegt dem Urheberrecht. © O'Reilly Verlag 2009