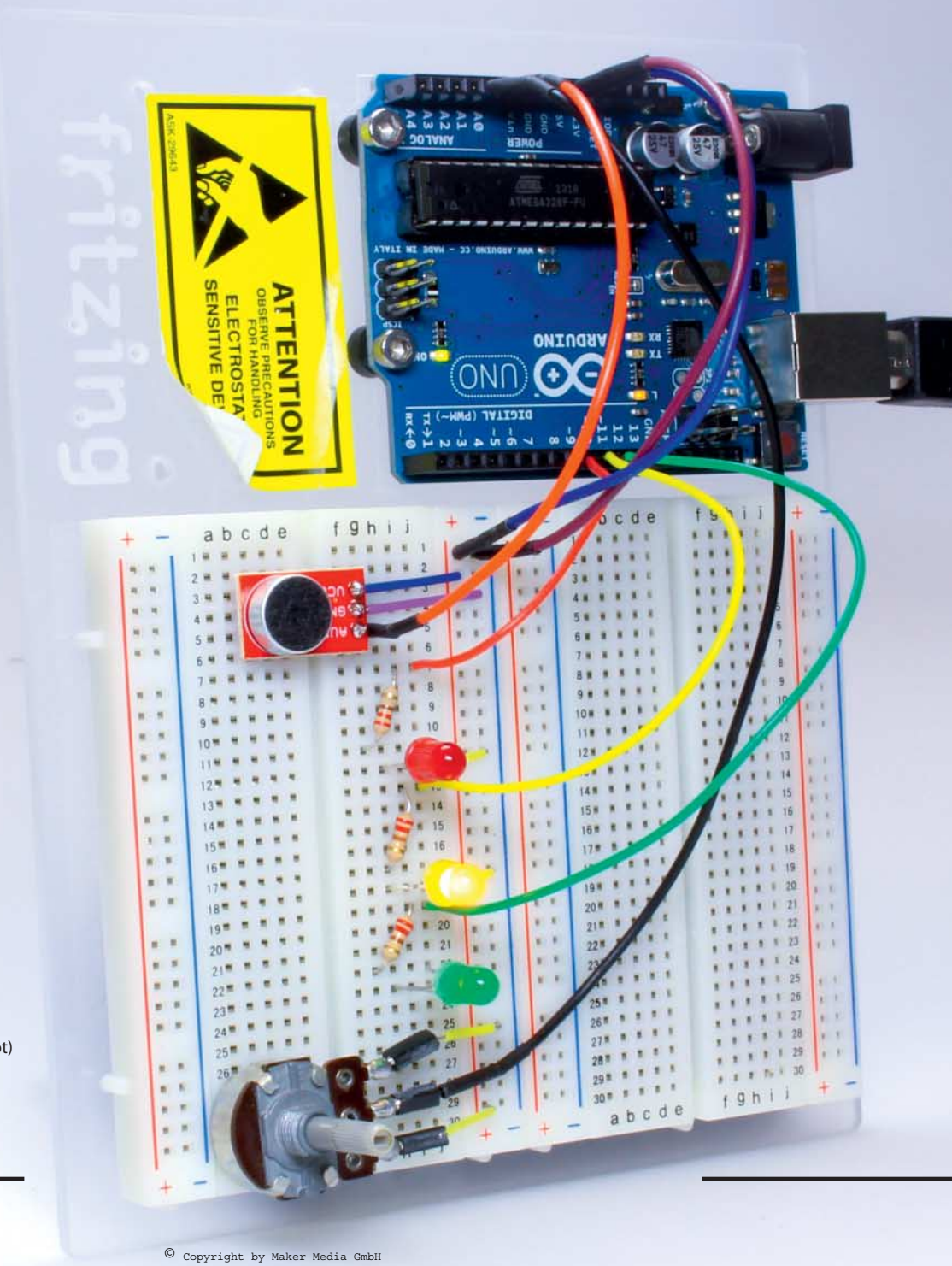
Lärmampel fürs Klassenzimmer

Bunte Warnleuchten gegen Lärm und der Einstieg in die Arduinoprogrammierung – die Lärmampel ist ein Projekt für die Fortbildung von Lehrkräften. Schnell und günstig zusammengesteckt ist sie aber auch ein schönes Projekt für Großraumbüros oder Werkstätten.

von Jens Braband und Dirk Peter



Kurzinfo

Zeitaufwand:
2 Stunden

Kosten:
50 Euro

Elektronik:
Einsteigerprojekt

Schwierigkeitsgrad

leicht — schwer

Material

- **Arduino Uno**
- **Mikrofon Breakout-Board, z. B. SparkFun** (BOB-09964)
- **Breadboard**
- **3 Leuchtdioden** (grün, gelb, rot)
- **lineares Potentiometer** (10 kOhm)
- **3 Widerstände** (220 Ohm)
- **Jumper-Kabel**

Anfang des Jahres traten die Organisatoren von „Schule MIT Wissenschaft“ (www.schule-mit-wissenschaft.de) an uns heran mit der Bitte, für ihre Tagung in Braunschweig einen Workshop zu organisieren. Da wir bereits Veranstaltungen für Schüler wie „Ein Herz für Elektronik“ ausgerichtet haben, begannen wir, einen Einführungsworkshop für Lehrkräfte zu entwickeln. Damit wollen wir die Begeisterung fürs Programmieren und Experimentieren in Schulen fördern.

Workshopentwicklung

Das Projekt sollte in zwei Stunden einen Einblick in die Arbeit mit Mikrocontrollern geben und auch mit wenig Vorkenntnissen realisierbar sein. Außerdem sollte es möglichst direkt in Schulen einsetzbar sein. Unsere Grundidee war ein kleines Arduino-Experiment, zum Beispiel ein kleines Messexperiment oder ein Regelkreis. Die Wahl fiel auf Lärmampeln, die in einigen Schulen eingesetzt werden, um den Geräuschpegel im Klassenzimmer sichtbar zu machen. Kommerzielle Lärmampeln sind ab 100 Euro erhältlich, während die Kosten für den Eigenbau bei 10 Euro ohne Arduino liegen.

Wir haben das Projekt in drei Schritte unterteilt, sodass sinnvolle Zwischenergebnisse erreicht werden können, ohne das Projekt in einem Workshop abschließen zu müssen. Die Lärmampel soll per Mikrofon den Schallpegel messen und über LEDs den Pegel in drei Stufen anzeigen. Als Erweiterung sollen die Pegelniveaus individuell nach dem eigenen Lärmempfinden oder der Umgebung eingestellt werden können.

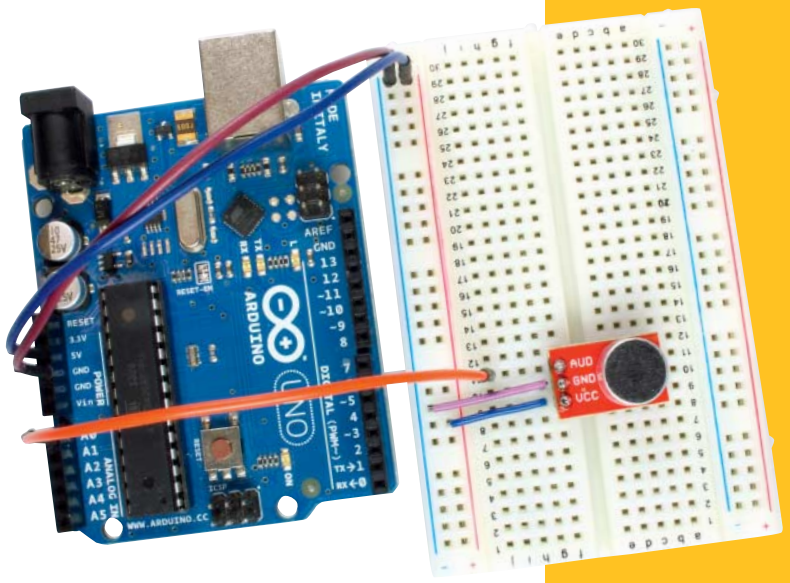
Im Workshop stellen wir zunächst sicher, dass alle Teilnehmer ein Notebook mit der Arduino-Programmierumgebung zur Verfügung haben. Die notwendige Hardware will die Firma Watterott allen Teilnehmern als Sponsor bereitstellen, für die Vertiefung legt der dpunkt-Verlag noch ein Arduino-Einsteigerbuch drauf. Oft lohnt es sich, Firmen in der Umgebung als Sponsoren anzufragen.

Als Einstieg erläutern wir dann kurz den Arduino, das Breadboard sowie die wichtigsten elektrotechnischen Grundlagen wie Stromkreis, Widerstand, analoge Signale und LED. Zur Überprüfung des Arduino laden wir das Beispielprogramm Blink, das in der Programmierumgebung mitgeliefert wird. Es lässt die auf dem Arduino verbaute LED leuchten, die wir im ersten Versuch nutzen.

Schallpegelmessung

Mit einem Mikrofon können wir den Schallpegel in der Umgebung bestimmen. Um es uns einfach zu machen, verwenden wir ein fertiges Breakout-Board. Damit können wir das Mikrofon wie einen beliebigen analogen Sensor einlesen. Das Board hat drei Anschlüsse, zwei für die Stromversorgung und einen für das Signal. Wir haben es mit einem Steckverbinder unverlötet in das Breadboard gesetzt. Das Löten ist aufwändiger, empfiehlt sich aber, da manche Verbindungen sonst wacklig sind und sich das Ergebnis nicht so einstellt, wie es sollte.

Wir verbinden erst die 5V-Spannungsversorgung des Arduino mit dem Breadboard, dann das Mikrofon mit dem Breadboard und schließlich den analogen Mikro-Ausgang AUD mit dem analogen Eingang A0 des Arduino. Im Programm (siehe Listing) deklarieren wir A0 als Eingang und definieren einen Schwellwert zwischen 0 und 1023. Mit `analogRead()` lesen wir den Eingang ein und steuern die interne LED auf Pin 13 des Arduino an, wenn der Schwellen-



Die Kabel zur Spannungsversorgung können direkt an das Mikrofon angeschlossen werden. Wir nehmen den Umweg über die Spannungsversorgung am Breadboard, um das Projekt ohne Umstecken erweitern zu können.

SELBERMACHEN

Dieser Artikel steht auch kostenlos im Internet zur Verfügung (siehe Downloadlink).

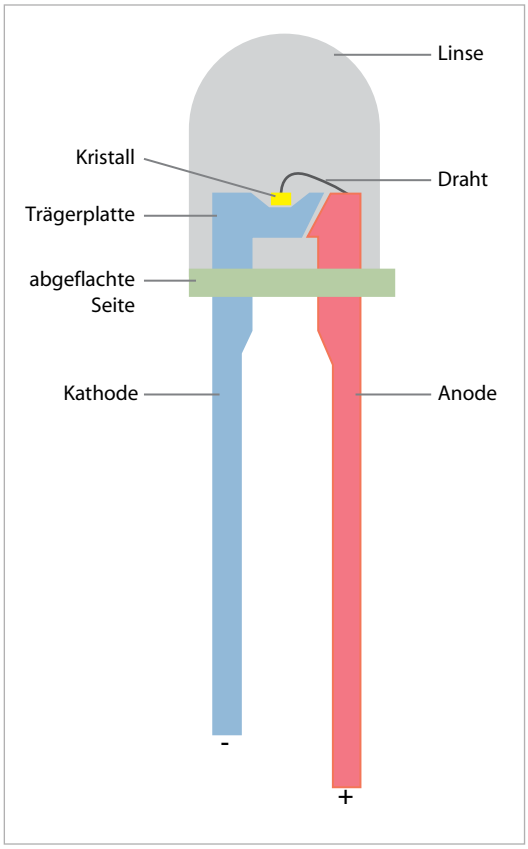
MIKRO

Das Mikrofon-Breakout-Board beruht auf einem sogenannten Kondensatormikrofon, dessen interne Kapazität sich durch Bewegung der Membran durch Schallwellen ändert. Die dabei entstehende minimale Spannungsänderung kann man mit einer elektronischen Schaltung weiter verarbeiten. Auf dem Breakout-Board verstärkt ein Operationsverstärker das Signal hundertfach, das man nun mit dem Arduino messen kann.

SMW Schule MIT Wissenschaft

Seit 2014 bringt die jährliche Fortbildung Schule und Forschung zusammen. Referentinnen und Referenten aus Naturwissenschaften und Technik, darunter Nobelpreisträger, halten Workshops und Vorträge, um aktuelle Entwicklungen und Ideen in den Unterricht zu tragen. Vorbild ist die Lehrerfortbildung der US-amerikanischen Elite-Universität MIT, dem Massachusetts Institute of Technology. Veranstaltet werden die Tagungen vom Verein „MIT Club of Germany“.

Beim Stecken der LEDs auf die Ampelreihenfolge achten: rot, gelb, grün



1023

Wir arbeiten mit den analogen Pins der Arduino, deren Signale von einem 10-bit-Konverter digitalisiert werden. Die eingehende Spannung von 0 bis 5 Volt wird also durch Werte von 0 bis 1023 abgebildet. Es gilt der Dreisatz:

$$\frac{1023}{5\,\text{Volt}} = \frac{x}{\text{gemessene Spannung}}$$

Mehr über den A/D-Wandler erfahren Sie ab Seite 92.

wert überschritten wird. Zusätzlich können wir die gelesenen Werte mit Serial.Println in der Arduino-Programmierumgebung anschauen.

Wir stellen schnell fest, dass der Schallpegel relativ stark schwankt und die LED häufig flackert, was dem Charakter einer Ampel nicht gerecht wird. Deswegen bauen wir mit delay(dauer) eine Verzögerung ein. Wenn ein erhöhter Pegel festgestellt wird, wird die LED für die mit dauer festgelegte Zeit angeschaltet und in dieser Zeit keine Messung durchgeführt. Das Programm finden Sie zum Herunterladen unter dem Downloadlink am Ende des Artikels.

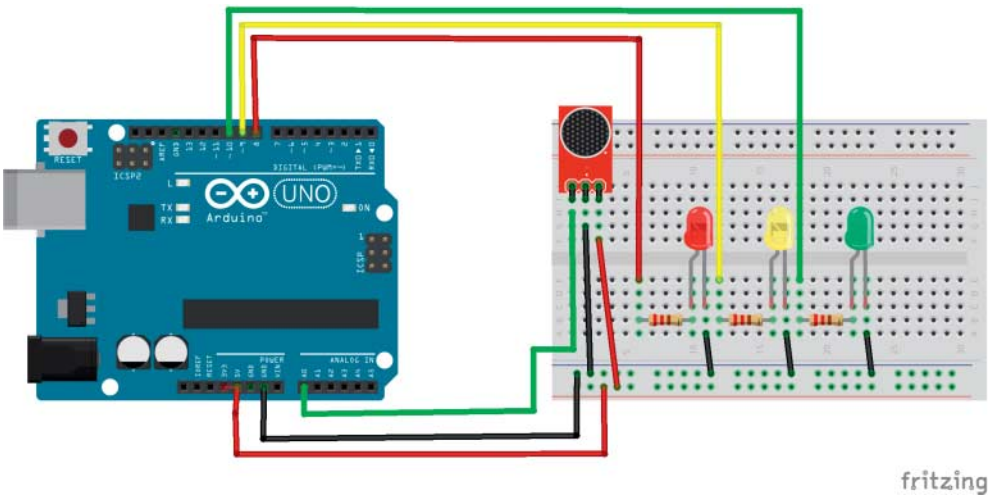
```
Schallpegelmessung

const int mikrofon = A0;
const int grenze = 600;
const int led = 13;
const int dauer = 500;

void setup() {
Serial.begin(9600);
pinMode(led, OUTPUT); }

void loop() {
int lautstaerke = analogRead(mikrofon);
int wert = lautstaerke;
Serial.println(wert);
if (wert>grenze) {
 Serial.println("Laut!");
 digitalWrite(led, HIGH);
 delay(dauer);
}
else
 digitalWrite(led, LOW);
}
```

Die einfache Lärmampel

Im nächsten Schritt wollen wir den Schallpegel in drei verschiedenen Niveaus durch farbige LEDs darstellen. Wir müssen beim Anschließen der LEDs daran denken, jeweils einen Widerstand dazwischen zu schalten, sonst könnten die LEDs durchbrennen. Außerdem müssen wir die LEDs richtig herum einsetzen. Denn die LEDs sind gepolt und leuchten nur, wenn der Strom in Durchlassrichtung fließt. Er muss an der Anode (positiv) eingeleitet werden und zur Kathode (negativ) fließen können. Wenn man versucht, zu viel Strom in der falschen Richtung durchzuleiten, kann die LED zerstört werden. Um die beiden Seiten zu unterscheiden, ist das Bein der Anode etwas länger als das der Kathode und der Rand des LED-Gehäuses ist an der Kathode abgeflacht.

Wir schließen die LEDs jeweils an einen digitalen Pin des Arduino an sowie an die Erdung des Breadboards. Im Programm müssen wir jetzt die entspre-

Damit die LED besser im Breadboard hält, kann das Bein der Anode etwas zur Seite gebogen werden, bis beide Beine gleich lang sind.



chenden Digitalschnittstellen (zum Beispiel `const int led_rot = 8;`) als Ausgabe deklarieren (zum Beispiel `pinMode (led_rot, OUTPUT);`). Wir müssen auch die Schwellwerte für die verschiedenen Schallpegel definieren und entsprechende Abfragen in das Programm einbauen (siehe Listing, das vollständige Programm finden Sie unter dem Downloadlink).

Hier können wir optimieren. Um zum Beispiel das Leuchten robuster zu machen, schalten wir jedes Mal alle LEDs aus, bevor wir eine neue anschalten. Geht man etwas nachlässiger vor, könnte es vorkommen, dass mehrere LEDs gleichzeitig leuchten, wenn ein länger anhaltendes Geräusch erst eine mittlere Lautstärke erreicht (gelbe LED) und dann lauter wird (rote LED).

Die komfortable Lärmampel

Da Menschen unterschiedlich lärmempfindlich sind und Lärm in verschiedenen Situationen anders empfunden wird, optimieren wir das Projekt weiter. Bei der einfachen Lärmampel müsste man immer wieder in das Programm eingreifen, um die Sensitivität der Lärmampel zu ändern. Für die komfortable Regelung nehmen wir ein Potentiometer – einen regelbaren Widerstand in Form eines Drehschalters. Eine andere Möglichkeit sind Schalter für verschiedene Einstellungen der Empfindlichkeit.

Für die Verwendung des Potentiometers gibt es zwei Möglichkeiten: eine Software- und eine Hardwarelösung. Für eine Softwarelösung schließen wir das Potentiometer wie einen Sensor an den Arduino an. Wir lesen sein analoges Signal, das Werte zwischen 0 und 1023 annimmt, und multiplizieren den gemessenen Schallpegel einfach im Programm mit x/1023. Bei höchster Stellung des Potentiometers bleibt der Schallpegel unverändert, sonst wird er anteilig vermindert.

Bei der Hardwarelösung nutzen wir das Potentiometer als Widerstand und schalten ihn direkt zwischen den Ausgang des Mikrofons und den Eingang des Arduino. Der Vorteil ist, dass die Software unverändert genutzt werden kann. Der Nachteil ist, dass wir unsere Breadboard-Schaltung eventuell umstecken müssen und diese unübersichtlich wird.

Wir haben uns für den Workshop für die Software-Variante entschieden, auch da wir das Programmieren mit dem Arduino üben wollten. Hierbei müssen wir noch ein Problem mit der Arithmetik des Arduino berücksichtigen: Lesen wir den Potentiometerwert x als Integer ein, so gibt x/1023 auch wieder einen ganzzahligen Wert zurück, also nur Null oder Eins. Um Abhilfe zu schaffen, lesen wir x gleich als Datentyp Float ein oder dividieren durch 1023.0. Durch die Nachkomma-Null können wir vom Arduino die Ausgabe als Float erzwingen. Diesen Sketch finden Sie ebenfalls im Downloadlink. Zu guter Letzt können wir mit den Grenzwerten und dem Potentiometer experimentieren, um individuelle Vorgabewerte ins Programm einzustellen. *—hch*

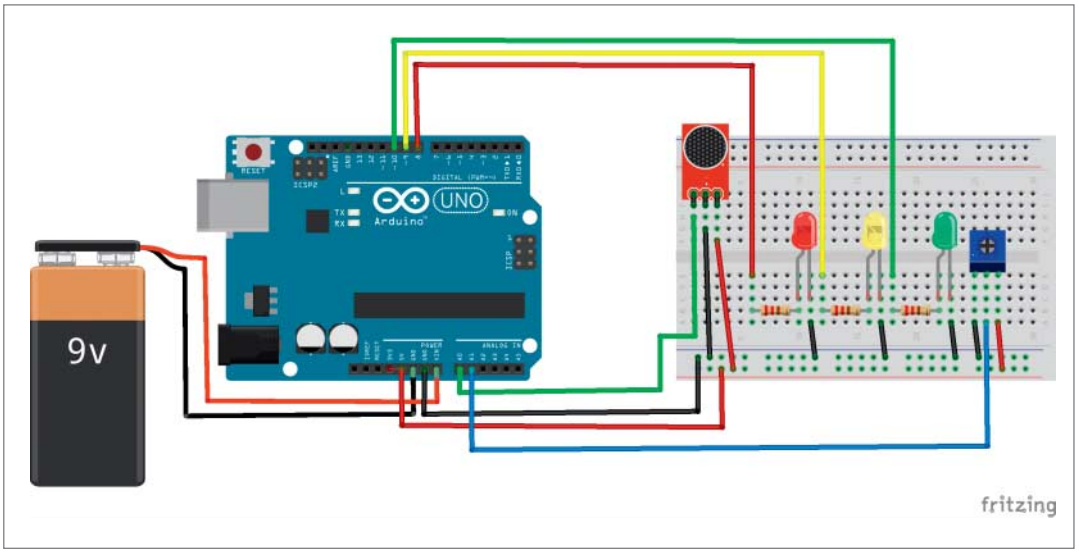
Farbige LEDs ansteuern

```
if (wert>grenze_rot) {
Serial.println("Rot!");
 digitalWrite(led_gruen, LOW);
 digitalWrite(led_gelb, LOW);
 digitalWrite(led_rot, HIGH);
 delay(dauer);
}
else if (wert>grenze_gelb){
 Serial.println("Gelb!");
 digitalWrite(led_gruen, LOW);
 digitalWrite(led_rot, LOW);
 digitalWrite(led_gelb, HIGH);
 delay(dauer);
}
else {
 digitalWrite(led_rot, LOW);
 digitalWrite(led_gelb, LOW);
 digitalWrite(led_gruen, HIGH);
}
```

LED-AUSWAHL

Die hier verwendeten LEDs sind klein – es gibt auch LEDs mit größerer Linse, die besser zu erkennen sind.

Links und Foren
make-magazin.de/xj6m



Für den praktischen Einsatz ist es sinnvoll, eine 9V-Batterie als externe Stromversorgung zu spendieren, damit man die Schaltung unabhängig von einem Notebook einsetzen kann.

Analog-Digital-Wandler

Arduinos wandeln analoge Signale in digitale Werte zwischen 0 und 1023 um. Doch wie passiert das eigentlich genau? Die Arbeit übernimmt ein Analog-Digital-Wandler, genauer gesagt ein Zehn-Bit-Wandler. Wir erklären, wie er funktioniert.

von Helga Hansen

Der Wandler übersetzt die an einem analogen Pin anliegende Spannung in den binären Wertebereich von 0 bis 2^9. Dazu vergleicht er die eingehende Spannung mit einer eigenen Referenzspannung, die er schrittweise anpasst. Das Verfahren heißt sukzessive Approximation, auf Deutsch: schrittweise Annäherung. Ein Register speichert die anfallenden Daten, eine Zeitsteuerung koordiniert die Messung.

Im ersten Schritt wird eine Referenzspannung angelegt und verglichen, die dem höchsten Bit (siehe Tabelle) entspricht. Bei einem Zehn-Bit-Wandler ist das 512, also 2,5 Volt. Ist diese Spannung niedriger als die zu messende Spannung, wird das Bit im Register auf 1 gesetzt und die Spannung gehalten. Ist sie zu hoch, wird sie zurückgesetzt und das Bit als 0 gespeichert.

Mit jedem folgenden Schritt wird die Referenzspannung entsprechend dem nächsten Bit erhöht, verglichen und das Ergebnis gespeichert. Diese Abfolge wiederholt der Wandler, bis alle Bits ermittelt sind. Das Ergebnis ist eine zehnstellige Zahl aus Nullen und Einsen.

In den meisten Fällen ändert sich das eingehende Signal ständig und stört damit das Messverfahren. Um trotzdem Werte berechnen zu können, wird eine Sample-and-Hold-Schaltung eingesetzt, die die eingehende Spannung für den Zeitraum der Messung konstant hält. Obwohl Arduinos über sechs analoge Pins verfügen, können nicht mehrere Werte gleichzeitig ermittelt werden, da nur ein Analog-Digital-Wandler vorhanden ist.

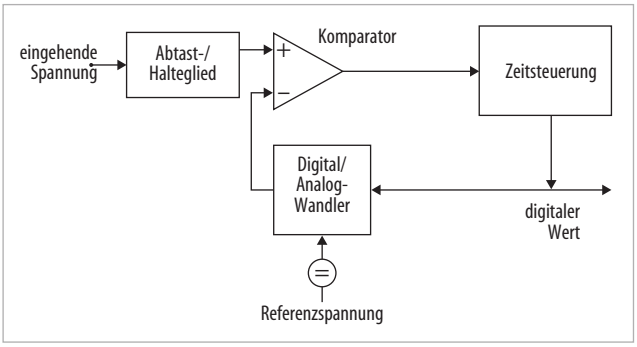
Die Taktrate des Wandlers im Arduino liegt bei 125 KHz (ergibt sich aus dem Quarztakt 16 MHz geteilt durch einen sogenannten Prescaler, der standardmäßig auf 128 gesetzt ist). Vom Beginn einer Messung bis zum Ergebnis vergehen 13 Takte. Das entspricht 104 Millisekunden und einer Abtastrate von 9615 KHz. Das reicht zum Digitalisieren von Sprache aus.

Den aktuell an einem Pin gemessenen Wert können wir uns mit einem Arduinoprogramm ausgeben lassen, etwa das Signal unseres Potenziometers von der Lärmampel. Die Schaltung bleibt gleich. Der aktuelle Wert erscheint im seriellen Monitor, der sich mit einem Klick auf die Lupe, rechts oben in der Arduino-Umgebung, öffnet. —*hch*

Links und Foren
make-magazin.de/xhmn

Bits

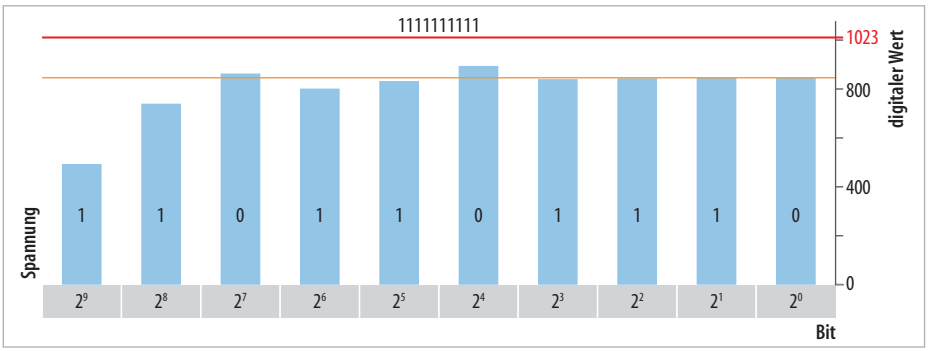
Bit	Wert
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512



Der Komparator vergleicht die eingehende Spannung mit der Referenzspannung und gibt pro Schritt einen aktualisierten digitalen Wert heraus.

Listing

```
void setup() {
  pinMode(A1, INPUT);
  Serial.begin(9600);
}
void loop() {
  int i = analogRead(A1);
  Serial.print("Analoger Wert: ");
  Serial.println(i);
}
```



Pro Schritt wird die Referenzspannung um den Wert des nächsten Bits erhöht. Liegt sie über dem zu messenden orangen Wert (878), wird der Schritt als null gewertet.