Ohrenschmaus

Der Arduino erzeugt dank eines Lautsprechers Töne und spielt Sounds.

Die bisherigen Projekte setzten vornehmlich auf visuelle Ausgaben, also auf leuchtende LEDs und rotierende Motoren. Genauso wichtig sind aber auch Audio-Signale, die von einfachen Pieptönen bis hin zur Wiedergabe aufgezeichneter menschlicher Sprache reichen können.

Der geringe Speicher des Arduino setzt der Verarbeitung aufgezeichneter Klänge enge Grenzen. Trotzdem lassen sich dem Board ohne großen Aufwand interessante und nützliche akustische Effekte entlocken.

Make Some Noise!

Schall ist eine Schwingung, die durch die Veränderung des Luftdrucks entsteht. Eine solche Schwingung lässt sich mit der Membran auch eines sehr kleinen Lautsprechers oder eines Piezo-Summers erzeugen. Auf natürliche Weise erzeugte Schallwellen, wie zum Beispiel menschliche Sprache oder Musik, sind recht komplex und folgen keinen einfachen Mustern. Schall lässt sich aber auch leicht synthetisch erzeugen.

Für erste Experimente reicht ein Mini-Lautsprecher, der direkt mit dem Arduino verbunden wird. Bei vielen Lautsprechern müssen zuvor noch zwei Drähte angelötet werden. Es gibt sie aber auch schon fertig verkabelt, bei vielen Anbietern ist leider nicht ersichtlich, wie das Bauteil geliefert wird.

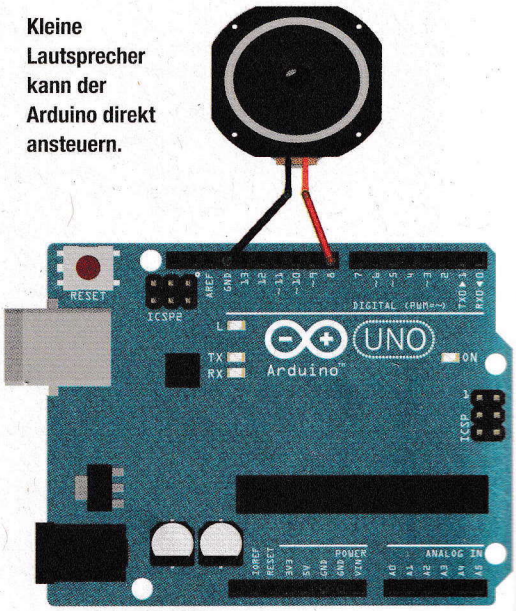
Mit nur zwölf Zeilen Code im Listing Saegezahn lässt sich eine so genannte Sägezahn-Kurve über den Lautsprecher ausgeben. Woher der Name Sägezahn stammt, wird beim Betrachten der Kurve schnell klar. Das Audio-Signal steigt von 0 steil bis zum Maximum an und fällt dann schlagartig wieder auf 0 zurück. Danach beginnt das Spiel von vorn, so dass die Kurve an ein Sägeblatt erinnert.

Der Verlauf der Kurve spiegelt sich weitestgehend auch im Code wider. Nachdem die Konstante `LAUTSPRECHER_PIN` initialisiert und in den Ausgabe-Modus versetzt wurde, kümmert sich die `loop`-Funktion um die Erzeugung der Töne. Dazu dient eine `for`-Schleife, die den Ausgabewert des Lautsprechers alle zehn Millisekunden um eins erhöht. Wenn die Schleife beendet ist, wird sie durch die `loop`-Funktion erneut aufgerufen.

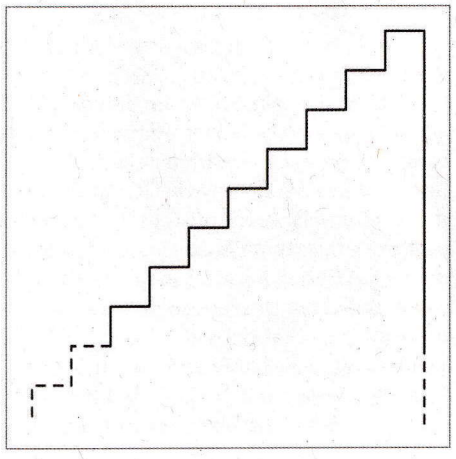
Der erzeugte Ton ist recht markant und die zugrunde liegende Schwingung wieder-

Saegezahn

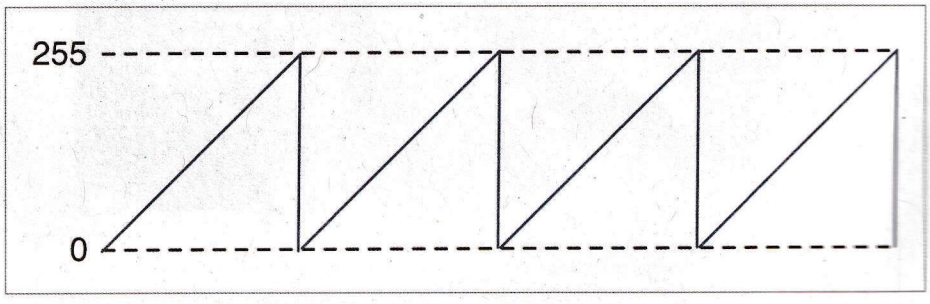
```
1  const unsigned char LAUTSPRECHER_PIN = 9;
2
3  void setup() {
4    pinMode(LAUTSPRECHER_PIN, OUTPUT);
5  }
6
7  void loop() {
8    for (unsigned char i = 0; i < 255; i++) {
9      analogWrite(LAUTSPRECHER_PIN, i);
10     delay(10);
11   }
12 }
```



Kleine Lautsprecher kann der Arduino direkt ansteuern.



Das Sägezahnsignal ist im Detail eigentlich trepppenartig.



Ein Sägezahnsignal am Ausgang des Arduino

holt sich alle 256 * 10 = 2560 Millisekunden. Das Endergebnis ist allerdings keine saubere Sägezahnkurve. Vielmehr wird der ansteigende Teil der Kurve über kleine Treppenstufen angenähert.

Smoooooth

Der erzeugte Ton klingt schon ganz gut, ist aber noch ein wenig eckig. Das ist ein häufiges Problem bei synthetischen Klängen. Insbesondere, wenn sie per Pulsweitenmodulation erzeugt wurden. Bei runden Kurven, wie zum Beispiel Sinusschwingungen, fällt das nicht so sehr ins Gewicht, aber Sägezahn- und Rechteckkurven klingen schon sehr künstlich. Der Grund dafür sind ungewollte Obertöne, die in Kombination mit dem eigentlichen Grundton etwas blechern klingen.

Ein passender Filter kann die unerwünschten Obertöne eliminieren. Einen Filter, der hohe Frequenzbereiche abschneidet und nur die niedrigeren passieren lässt, heißt Tiefpassfilter (englisch: low-pass filter). Der kann sowohl als Software als auch als Hardware umgesetzt werden. Bei der Hardware-Lösung reichen bereits ein Keramik-Kondensator (0,1uF) und ein Widerstand (1kOhm), um den Ton zu glätten. Die Schaltung implementiert einen Tiefpassfilter und entfernt die Störenfriede. Wie Widerstände haben auch Keramik-Kondensatoren keine Polung und können in beliebiger Richtung in die Schaltung eingesetzt werden. Bei Elektrolyt-Kondensatoren ist das nicht der Fall.

Mit dem Tiefpassfilter klingt der Ton runder, er ist aber auch deutlich leiser. Hier hilft bei Bedarf nur ein externer Verstärker.

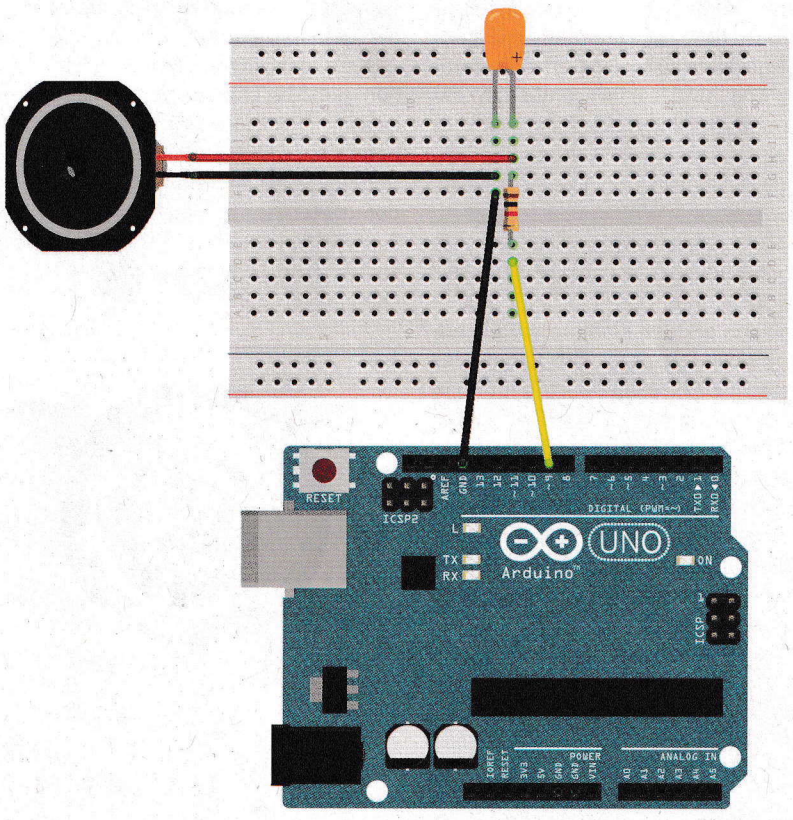
Mehr Struktur

Das erste Beispiel-Programm ist mehr oder weniger chaotisch und dient lediglich als Machbarkeitsstudie. Prinzipiell lassen sich also Töne mit einem Arduino und einem Lautsprecher erzeugen. Im Folgenden geht es darum, den Prozess in geordnete Bahnen zu lenken und nicht nur irgendwelche Töne zu erzeugen, sondern vorgegebene Noten mit festen Frequenzen wiederzugeben.

Das Programm Melodie ist eine abgewandelte Version des Codes, der sich in der Arduino-IDE unter Beispiele > 02.Digital > toneMelody findet. Es spielt eine kurze Melodie, die aus acht Noten besteht, und nutzt dazu die tone-Funktion. Zuerst bindet es die Datei Tonhoehe.h ein, die im selben Verzeichnis liegt, wie das Programm. Diese Datei definiert eine Enumeration (Aufzählung) namens Tonhoehe mit dem Schlüsselwort enum. Sie sieht wie folgt aus:

Melodie

```
#include "Tonhoehe.h"

const unsigned char LAUTSPRECHER_PIN = 9;
const unsigned int GANZE_NOTE = 1000;

struct Note {
  Tonhoehe frequenz;
  unsigned char laenge; // 4 = Viertelnote, 8 = Achtelnote etc.
};

Note melodie[] = {
  { NOTE_C4, 4 }, { NOTE_G3, 8 },
  { NOTE_G3, 8 }, { NOTE_A3, 4 },
  { NOTE_G3, 4 }, { PAUSE,   4 },
  { NOTE_B3, 4 }, { NOTE_C4, 4 }
};

void setup() { }

void loop() {
  unsigned int anzahl_noten = sizeof(melodie) / sizeof(melodie[0]);
  for (unsigned int note = 0; note < anzahl_noten; note++) {
    unsigned int notenLaenge = GANZE_NOTE / melodie[note].laenge;
    tone(LAUTSPRECHER_PIN, melodie[note].frequenz, notenLaenge);
    unsigned int pauseZwischenNoten = notenLaenge * 1.30;
    delay(pauseZwischenNoten);
  }
  delay(3000);
}
```



Der Widerstand und der Kondensator bilden einen Tiefpassfilter.

```
enum Tonhoehe {
    PAUSE = 0,
    NOTE_B0  = 31,
    NOTE_C1  = 33,
    NOTE_CS1 = 35,
    NOTE_D1  = 37,
    NOTE_DS1 = 39,
    ...
    NOTE_CS8 = 4435,
    NOTE_D8  = 4699,
    NOTE_DS8 = 4978,
};
```

Enumerationen sind ein probates Mittel, um lange Listen von Konstanten zusammenzufassen und mit einem Typen zu versehen. Sie eignen sich zum Beispiel, um einen eigenen Datentypen für alle möglichen Wochentage zu definieren. Aber auch für die Definition von Noten in verschiedenen Oktaven sind sie nützlich. `Tonhoehe` weist den einzelnen Mitgliedern der Enumeration die zum jeweiligen Ton gehörende Frequenz zu. Der Wert `PAUSE` bekommt die Frequenz 0 und `NOTE_A4` repräsentiert den Standard-Kammerton mit der Frequenz 440 Hertz.

Im Programm geht es mit der Definition von zwei Konstanten weiter. `LAUTSPRECHER_PIN` legt fest, mit welchem Pin der Lautsprecher verbunden ist. `GANZE_NOTE` definiert die Länge einer ganzen Note gemessen in Millisekunden.

Es folgt die Deklaration einer Struktur (`struct`) namens `Note`, die alle wichtigen Eigenschaften einer Note zu einem Ganzen zusammenfasst. Das sind im Wesentlichen die Frequenz und die Länge. Die Länge wird in diesem Fall als Teiler einer ganzen Note angegeben, also steht 4 für eine Viertelnote, 8 für eine Achtelnote und so weiter.

Als Nächstes wird die eigentliche Melodie beschrieben und zwar in Form eines Feldes von Note-Strukturen. Das Feld heißt `melodie` und die beiden eckigen Klammern ([]) kennzeichnen `melodie` als Feld. Die Länge des Feldes berechnet die Arduino-Umgebung in diesem Fall automatisch. Jeder Eintrag des Felds ist eine Note-Struktur und die Werte ihrer Eigenschaften (`frequenz` und `laenge`) werden in geschweifte Klammern geschrieben und mit einem Komma separiert. Insgesamt besteht die Melodie aus acht Noten, wobei die erste die Viertelnote C4 und die zweite die Achtelnote G3 ist.

In der `setup`-Funktion ist gar nichts zu tun, weil die im Folgenden verwendete `tone`-Funktion sich um die Initialisierung des Lautsprecher-Pins kümmert.

Die `loop`-Funktion berechnet zuerst die Anzahl der Noten im Feld `melodie`. Dazu verwendet sie einen gängigen Trick und den `sizeof`-Operator. Der liefert die Anzahl der Bytes, die eine Variable im Speicher belegt, zurück. Die genaue Anzahl der Bytes spielt in diesem Fall keine wichtige Rolle, denn es geht nur darum, die Anzahl der Elemente in `melodie` zu ermitteln. Diese ergibt sich, wenn die Größe des gesamten Felds durch die Größe eines einzelnen Elements dividiert wird, weil alle Elemente des Felds dieselbe Größe haben.

Anschließend schnappt sich eine `for`-Schleife jede einzelne Note und berechnet deren Dauer, indem die Dauer einer ganzen Note durch die Dauer der aktuellen Note dividiert wird. Für die erste Note ergibt sich zum Beispiel ein Wert von 1000 / 4 = 250 Millisekunden.

Mit der `tone`-Funktion wird die aktuelle Note dann über den Lautsprecher ausgegeben. Sie erwartet die Nummer des Lautsprecher-Pins, die Frequenz des Tons und die Dauer des Tons. Wird die Dauer nicht angegeben, gibt `tone` den Ton so lange aus, bis die Funktion `noTone` aufgerufen wurde.

Die restlichen Anweisungen in der `for`-Schleife sorgen für eine kurze Pause, die zwischen den einzelnen Noten eingelegt wird.

Wird das Programm auf den Arduino geladen, spielt er alle drei Sekunden eine kurze Melodie, die ein wenig an die Spielautomaten in Imbiss-Buden der Achtziger Jahre erinnert. Es lohnt sich, ein wenig mit dem Wert der Konstanten `GANZE_NOTE` zu spielen, denn damit lässt sich die Abspielgeschwindigkeit der Melodie steuern.

Kern des letzten Beispiels ist die `tone`-Funktion. Sie erzeugt per Pulsweitenmodulation ein Rechteck-Signal mit einer vorgegebenen Frequenz und Dauer. Das unterscheidet sich nicht von den Signalen, die zuvor genutzt wurden, um Motoren zu steuern. Die Signale werden jetzt nur nicht mehr von einem Motor, sondern von einem Lautsprecher interpretiert.

Fortgeschrittene Projekte

Die Erzeugung von Schallwellen ist mit dem Arduino erst einmal kein Problem. Komplizierter ist es, gezielt wirklich gute Klänge zu erzeugen. Dazu bedarf es recht komplizierter Software, bei der es auf das richtige Timing bei der Erzeugung von Frequenzen ankommt. Doch auch dazu gibt es jede Menge hilfreicher und nützlicher Bibliotheken.

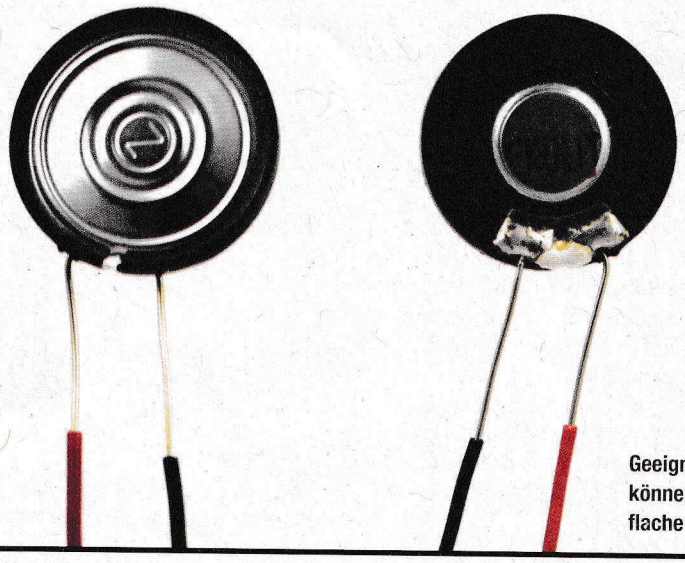
Eines der umfangreichsten Projekte ist Mozzi (http://sensorium.github.io/Mozzi/). Es bietet technikbegeisterten Musikern alles, was das Herz begehrt. Beispielsweise gibt es Funktionen zur Erzeugung synthetischer Klänge aller Art und auch die Wiedergabe von Samples ist problemlos möglich.

Darüber hinaus bietet die Bibliothek eine große Anzahl von Sound-Filtern und verwandelt den Arduino bei Bedarf sogar in ein MIDI-Gerät. Der Preis für die Funktionsvielfalt ist eine recht hohe Komplexität und eine teilweise steile Lernkurve.

Dafür ist das Projekt aber üppig dokumentiert und beinhaltet eine Vielzahl an Beispiel-Projekten. Diese finden sich nach der Installation der Zip-Datei über den Library-Manager im Menü Datei > Beispiele. Um die meisten der Beispiele auszuprobieren, reicht es völlig aus, den Arduino direkt mit einem Mini-Lautsprecher zu verbinden. In der Regel verwenden die Beispiele den digitalen Pin 9.

Fazit

Wie in vielen anderen Bereichen, weiß der Arduino auch bei der Erzeugung von Klängen zu überraschen. Es ist erstaunlich, was ein wenig kreative Software an Sounds aus dem Winzling rausholt. Darüber hinaus katapultiert der Einsatz spezieller Hardware, wie etwa diverse Sound-Shields, den Arduino schnell in eine ganz andere Liga. —*dab*



Geeignete Lautsprecher können auch eine sehr flache Bauform haben.