Erzeugung zeitkritischer Frequenzsignale mit dem Arduino

Verbindung von Physik und Informatik im Schülerexperiment zum KUNDT'schen Rohr aus dem 3D-Drucker mit Arduino-Betriebsgerät

NILS HAVERKAMP – ALEXANDER PUSCH – PAUL SCHLUMMER – MALTE UBBEN

In diesem Beitrag geht es darum, wie man den Arduino für zeitkritische Anwendungen, wie die Erzeugung eines Frequenzsignals, verwenden kann. Obwohl Rechtecksignale vergleichsweise einfach zu erzeugen sind, zeigen Analysen mittels Oszilloskops aber, dass einige Anpassungen am Programmcode notwendig sind, um einen präzisen Frequenzgenerator zu programmieren. Hierdurch lassen sich Grundlagen zu zeitkritischen Programmabläufen und -funktionen an einem experimentellen Anwendungsbeispiel für die Informatik und die Physik erlernen.
Als Anwendungsbeispiel für den Frequenzgenerator im Physikunterricht verwenden wir eine günstige Variante des KUNDT'schen Rohres mit 3D-gedruckten Bauteilen und Ultraschallsendern und Arduino-Betriebsgerät.

1 Erzeugung eines einfachen Rechtecksignals

Um ein Rechtecksignal erzeugen zu können, muss lediglich einer der Ausgänge des Arduinos im regelmäßigen Abstand an- und wieder ausgeschaltet werden. Im Folgenden wird ein einfacher Programmcode Schritt für Schritt erweitert und die resultierenden Ergebnisse jeweils mit einem Oszilloskop gemessen und erläutert. Hierbei programmieren wir zunächst ein Signal von 40 Hz, um dabei die grundlegende Herangehensweise und dabei auftretende Schwierigkeiten zu zeigen. In späteren Schritten werden dann Befehle für zeitkritische Programme am Beispiel der Erzeugung eines exakten Signals von 40.000 Hz (40 kHz) vorgestellt, welches u.a. für den Betrieb des KUNDT'schen Rohres aus dem 3D-Drucker geeignet ist.

Mit Programmcode 1 soll zunächst ein Rechtecksignal mit 40 Hz erzeugt werden. Die Periodendauer und damit auch die Frequenz kann über die Variable *T* angepasst werden. Um darauf aufbauend später eine Frequenz von 40.000 Hz zu erzeugen, müsste die Variable *T* theoretisch lediglich auf den Wert 0,025 gesetzt werden (Spoiler: dies liefert nicht das gewünschte Ergebnis).

Eine Messung mit dem Oszilloskop zeigt allerdings, dass der Programmcode bereits bei 40 Hz zeitlich ungenau wird. Die ausgegebene Frequenz beträgt etwa 41,7 Hz anstatt der programmierten 40 Hz (Abb. 1). Der Grund dafür liegt im *delay()*-Befehl. Schaut man in die Referenz zum *delay()*-Befehl, findet man heraus, dass dieser nur mit dem Variablentyp *unsigned*

```
int T = 25; //Periodendauer in Millisekunden

void setup(){
  pinMode (A0, OUTPUT);              // Pinmode auf Output festlegen
}
void loop(){
  digitalWrite(A0,HIGH);             // Pin einschalten
  delay(T/2);                        // Halbe Periodendauer warten
  digitalWrite(A0,LOW);              // Pin ausschalten
  delay(T/2);                        // Halbe Periodendauer warten
}
```

Programmcode 1. Einfacher Programmcode zum Erzeugen eines 40 Hz Rechtecksignals

long funktioniert. Die Variable *T*/2 muss also ganzzahlig sein, da Im Variablentyp *unsigned long* nur positive Ganzzahlen gespeichert werden können. Außerdem gibt es durch den vorgesehenen Speicherplatz eine Größenbegrenzung, die bei 2^{32}-1 = 4 294 967 295 liegt. Es wird bei der Verarbeitung des Befehls also 12,5 ms (d.h. *T*/2) auf 12 ms gerundet.

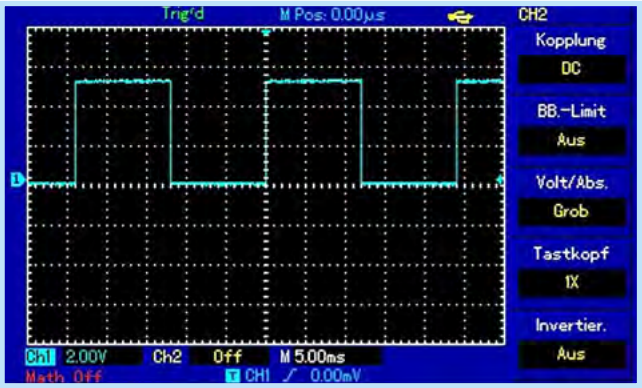

Abb. 1. Messung des Signals von Programmcode 1 am Oszilloskop. A0 gegen GND, 5 ms Einteilung der Zeitachse.

Um eine bessere zeitliche Auflösung zu erreichen, muss mit einer Alternative zum *delay()*-Befehl gearbeitet werden (für weitere Informationen siehe Referenz zum *delay()*-Befehl). Die naheliegendste Variante ist der Befehl *delayMicroseconds()*, in dem die Verzögerungsdauer in Mikrosekunden statt in Millisekunden angeben wird. Auf diese Weise wird das Programm wie folgt angepasst, um direkt eine Frequenz von 40.000 Hz (40 kHz) auszugeben (Programmcode 2).

Weil die Funktion *delayMicroseconds()* aber auch nur den Variablentyp *unsigned long* als Eingabe akzeptiert, würden wir analog zum vorherigen Programmcode erwarten, dass die halbe Periodendauer auf 12 µs gerundet wird, sich also eine gesamte Periodendauer von 24 µs ergibt. Tatsächlich zeigt die Messung mit dem Oszilloskop jedoch etwas anderes: die halbe Periodendauer beträgt sogar fast 14 µs anstatt der erwarteten 12 µs (Abb. 2). Die Frequenz ist also etwa 35,7 kHz statt 40 kHz.

Dies liegt daran, dass ein Mikrocontroller auch eine gewisse Zeit benötigt, um Befehle wie z.B. den *digitalWrite()*-Befehl auszuführen. Dass Befehle eine gewisse Zeit zum Ausführen benötigen, fällt allerdings erst auf, wenn man mit Zeitdauern in einer ähnlich geringen Größenordnung arbeitet. Anscheinend benötigt unser Arduino Uno knapp 2 µs, um den Befehl auszuführen. Eine Möglichkeit, dieser Problematik zu begegnen, wäre den *delay()*-Befehl entsprechend anzupassen. Auf ähnliche Weise ließe sich auch das Rundungsproblem angehen, indem einer der beiden Delays um eine Mikrosekunde verlängert wird. So ergibt sich zumindest näherungsweise eine korrekte Frequenz aber kein perfektes Rechtecksignal (Programmcode 3). Zudem wirkt diese Variante sehr „unelegant“ sowie umständlich, falls weitere Frequenzen dargestellt werden sollen.

```
int T = 25; //Periodendauer in Mikrosekunden

void setup(){
  pinMode (A0, OUTPUT);              // Pinmode auf Output festlegen
}
void loop(){
  digitalWrite(A0,HIGH);             // Pin einschalten
  delayMicroseconds(T/2);            // Halbe Periodendauer warten
  digitalWrite(A0,LOW);              // Pin ausschalten
  delayMicroseconds(T/2);            // Halbe Periodendauer warten
}
```

Programmcode 2. einfacher Programmcode zum Erzeugen eines 40 kHz Rechtecksignals

```
int T = 25; //Periodendauer in Mikrosekunden

void setup(){
  pinMode (A0, OUTPUT);         // Pinmode auf Output festlegen
}
void loop(){
  digitalWrite(A0,HIGH);         // Pin einschalten
  delayMicroseconds((T/2)+1-2); // Halbe Periodendauer warten
  digitalWrite(A0,LOW);          // Pin ausschalten
  delayMicroseconds(T/2-2);      // Halbe Periodendauer warten
}
```

Programmcode 3. einfacher Programmcode zum Erzeugen eines 40 kHz Rechtecksignals

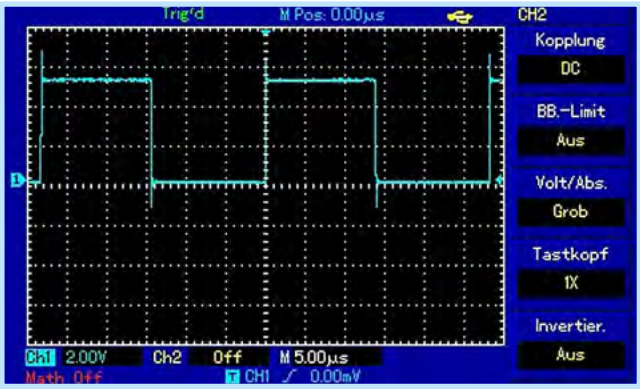

Abb. 2. Messung des Signals von Programmcode 2 am Oszilloskop. A0 gegen GND, 5 µs Einteilung der Zeitachse

Die Messung mit dem Oszilloskop (Abb. 3) zeigt, dass die Periodendauer jetzt sehr nah am Zielwert von 25 µs liegt (entspricht etwa 40 kHz). Zufriedenstellend ist diese Lösung allerdings nicht, weil die Frequenz immer noch nicht genau passt und der Ausgang außerdem länger eingeschaltet als ausgeschaltet ist.

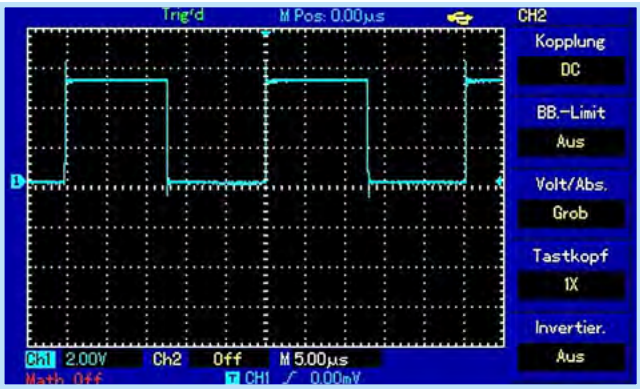

Abb. 3. Messung des Signals von Programmcode 3 am Oszilloskop. A0 gegen GND, 5 µs Einteilung der Zeitachse.

2 Erzeugung eines zeitkritischen Rechtecksignals mit Interrupts

Der in diesem Beispiel verwendete *Arduino Uno* besitzt einen Mikrocontroller vom Typ *ATmega328* und arbeitet mit einer Taktfrequenz von 16 MHz (s. Datenblatt Arduino Uno). Es ist möglich, eine Frequenz von 40 kHz zuverlässiger zu erzeugen, wenn man direkt auf diesen Systemtakt zugreift. Dies lässt sich durch sogenannte *Interruptfunktionen* umsetzen. Diese Funktionen werden nicht im Rahmen der *void loop()* – also der regulären "Programmschleife" – aufgerufen, sondern können stattdessen durch Signale an digitalen Eingängen oder durch den internen Taktgeber zeitkritisch aktiviert werden. *Interruptfunktionen* unterbrechen dann die Programmfolge im *void loop()* und führen stattdessen vorab definierte Befehle aus. Sie sind im Allgemeinen dazu vorgesehen, wenige Zeilen Code im "richtigen Moment" auszuführen. *Interruptfunktionen* lassen sich allerdings mit manchen Befehlen (wie z.B. *delay()*) nicht fehlerlos kombinieren.

Wir werden nun eine an den internen Taktgeber gekoppelte *Interruptfunktion* verwenden, um den Zustand eines Ausgangs umzuschalten. Die Architektur des *Arduino Uno* mit seinem *ATmega328* stellt dazu drei unterschiedliche *Timer* zur Verfügung, die sich untereinander leicht unterscheiden. Wir verwenden den Timer 1. Erklärungen und eine Übersicht über verschiedene Timer finden sich z.B. bei Ewald (o.J.).

Im folgenden Programmcode (Programmcode 4) werden dazu im Rahmen des Setups die Einstellungen dieses Timers angepasst. Diese Einstellungen sind in den *Registern* TCCR1A und TCCR1B hinterlegt. Für den vorliegenden Programmcode wird der CTC-Modus (*Clear Timer on Compare Modus*) aktiviert. In diesem Modus wird ein Zähler jedes Mal um eins erhöht, wenn ein Signal vom Systemtakt ankommt und mit einem frei wählbaren Vergleichswert abgeglichen. Sind beide Werte gleich, wird die *Interruptfunktion* ausgelöst und der Zähler wieder auf Null gesetzt. Durch Anpassen des Vergleichswertes kann also eine genaue Frequenz in Abhängigkeit von dem Systemtakt gesteuert werden.

Weiterhin ist im TCCR1B-Register ein *Prescaler* hinterlegt. Dieser sorgt dafür, dass der Zähler nicht mit jedem Systemtakt angehoben wird, sondern nur mit jedem *n*-ten Takt. Dabei steht *n* für den *Prescaler*. Für *Timer1* kann der *Prescaler* die Werte 1, 8, 64, 256 und 1024 annehmen. Außerdem können auch externe Taktgeber eingestellt werden. Durch den *Prescaler* ist es auch möglich, Frequenzen zu erreichen, die deutlich unter der Taktfrequenz des Prozessors liegen, ohne extrem große Vergleichswerte für den Zähler festlegen zu müssen.

```
void setup(){
  noInterrupts();               // Interrupts deaktivieren. So können die
                                // Einstellungen angepasst werden.
  TCCR1A = 0; TCCR1B = 0;       // Register leeren
  TCCR1B |= (1 << WGM12);       // CTC Modus (Clear Timer on Compare Modus).
  TCCR1B |= (0 << CS12)+(0 << CS11)+(1 << CS10);
                                // Prescaler auf 1 einstellen
  OCR1A = 199;                  // Compare Match Register setzen(16MHz/(1(199+1))
                                // = 80kHz)
  TIMSK1 |= (1 << OCIE1A);      // Wenn der Zähler voll ist, wird die
                                // Interruptfunktion ausgeführt.
  interrupts();                 // Interrupts aktivieren

  pinMode (A0, OUTPUT);         // A0 als Ausgang definieren
}
```

Programmcode 4, Teil 1. Einstellungen zum Timer

Insgesamt ergibt sich im CTC-Modus die folgende Formel für die Berechnung der *Interruptfrequenz*:

$$f_{Interrupt} = \frac{f_{CPU}}{P(V+1)}$$

Dabei steht *P* für den Wert des Prescalers und *V* für den Vergleichswert. Die "+1" ergibt sich daraus, dass in der Informatik in der Regel von 0 aufwärts gezählt wird.

Das Anpassen der Register und des Vergleichswerts im Setup ergibt den unten dargestellten Programmcode.
Die *Interruptfunktion* selbst ist vergleichsweise kurz (Programmcode 4, Teil 2): Zunächst wird der Ausgang beschrieben. Durch Einführen der Variable *OnOff* vom Typ *Boolean* (also nur die Werte 1 bzw. HIGH und 0 bzw. LOW) kann dafür gesorgt werden, dass der Ausgang mit dem gleichen Code eingeschaltet und ausgeschaltet werden kann. Die Variable *OnOff* wird dazu in der *Interruptfunktion* invertiert.

```
ISR(TIMER1_COMPA_vect)
{
  digitalWrite(A0, OnOff);      // A0 auf OnOff setzen
  OnOff = !OnOff;               // OnOff für die nächste Schleife
                                // umschalten
}
```

Programmcode 4, Teil 2. Einstellungen zum Interrupt

Eine weitere Messung mit dem Oszilloskop zeigt, dass sich mit diesem Programmcode ein sehr gleichmäßiges und zeitgenaues Rechtecksignal ergibt (Abb. 4).

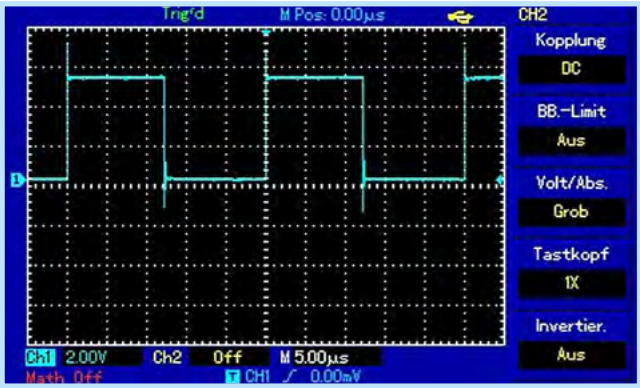

Abb. 4. Messung des Signals von Programmcode 4 am Oszilloskop, A0 gegen GND, 5 µs Einteilung der Zeitachse.

Damit das Experiment noch besser funktioniert, ist eine möglichst große Amplitude wünschenswert. Dies lässt sich umsetzen, indem das Rechtecksignal nicht nur zwischen einem Ausgang und GND abgenommen wird, sondern zwischen zwei Ausgängen, die invers geschaltet sind. So ergibt sich eine Spannungsdifferenz von $2 \cdot 5\ V = 10\ V$.
Es muss also ein zweites Rechtecksignal auf einen zweiten Ausgang gelegt werden, das um eine halbe Periode gegen das erste verschoben ist. Theoretisch lässt sich dies ergänzen, indem die *Interruptfunktion* um einen zweiten *digitalWrite()* Befehl ergänzt wird. Wir wissen aber schon aus dem vorherigen Kapitel, dass die benötigte Zeit zum Ausführen der Befehle hier Schwierigkeiten machen wird. Eine Messung zu dem Programmcode, der die zwei Befehle *digitalWrite()* nacheinander ausführt, bestätigen diese Annahme (Abb. 5).
Die Lösung ist, dass beide Ausgänge zeitgleich beschrieben werden müssen. Dies ist möglich, indem nicht erst ein einzelner Ausgang und dann der nächste Ausgang beschrieben wird,

```
byte OnOff = B10101010;          // jeder 2. Port bekommt das umgekehrte Signal
void setup(){
  noInterrupts();                // Interrupts deaktivieren. So können die
                                 // Einstellungen angepasst werden.
  TCCR1A = 0; TCCR1B = 0;        // Register leeren
  TCCR1B |= (1 << WGM12);        // CTC Modus (Clear Timer on Compare Modus).
  TCCR1B |= (0 << CS12)+(0 << CS11)+(1 << CS10);
                                 // Prescaler auf 1 einstellen
  OCR1A = 199;                   // Compare Match Register setzen(16MHz / (1(199+1))
                                 // = 80kHz)
  TIMSK1 |= (1 << OCIE1A);       // Wenn der Zähler voll ist, wird die
                                 // Interruptfunktion ausgeführt.
  interrupts();                  // Interrupts aktivieren
  pinMode (A0, OUTPUT);          // A0 als Ausgang definieren
  pinMode (A1, OUTPUT);          // A1 als Ausgang definieren
  pinMode (A2, OUTPUT);          // A2 als Ausgang definieren
  pinMode (A3, OUTPUT);          // A3 als Ausgang definieren
  pinMode (A4, OUTPUT);          // A4 als Ausgang definieren
  pinMode (A5, OUTPUT);          // A5 als Ausgang definieren
}
ISR(TIMER1_COMPA_vect)
{
  PORTC = OnOff; // Die Bits in OnOff an die Ausgänge senden
  OnOff = ~OnOff; // OnOffinvertieren für den nächsten Durchlauf
}
 void loop(){}
```

Programmcode 5. Finaler Programmcode des Arduino-Betriebsgerätes

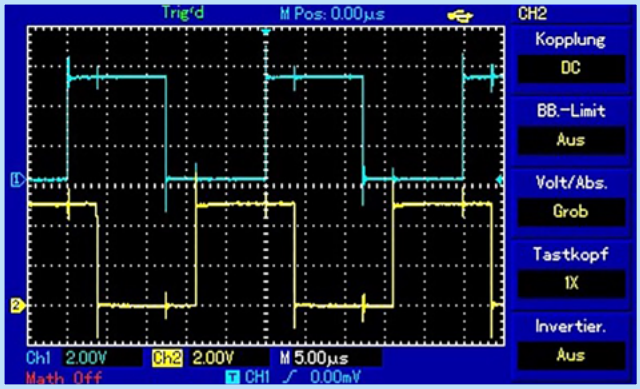

Abb. 5. Messung des Signals zweier nacheinander geschalteter Ausgänge am Arduino Uno am Oszilloskop, A0 und A1 gegen GND, 5 µs Einteilung der Zeitachse.

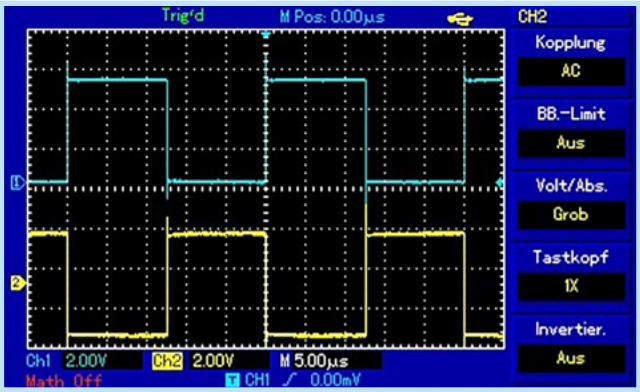

Abb. 6. Messung des Signals von Programmcode 5 am Oszilloskop, A0 und A1 gegen GND, 5 µs Einteilung der Zeitachse.

sondern stattdessen ein vollständiger Port. In einem Port sind mehrere *Ausgänge* zusammengefasst, die sich so gemeinsam steuern lassen (s. z.B. WALLER, o.J.). Im folgenden Beispiel wird *PORTC* verwendet, in dem die Pins A0 bis A5 zusammengefasst sind. *PORTC* wird mit 8 Bits beschrieben, die zu Beginn des Programmcodes in der Variablen vom Typ *Byte* names *OnOff* hinterlegt werden. Relevant für uns sind allerdings nur die ersten 6 Bits. Diese sind abwechselnd HIGH und LOW, wodurch man das Rechtecksignal also später jeweils zwischen einem geradzahligen und einem ungeradzahligen Pin abgreifen kann. In der *Interruptfunktion* wird dann nur noch *PORTC* mit dem *Byte OnOff* beschrieben und das gesamte *Byte* invertiert. Der vollständige Programmcode ist anschließend dargestellt (Programmcode 5). Mit dem Oszilloskop können dann die folgenden Rechtecksignale gemessen werden (Abb. 6).

Es ist gut zu erkennen, dass diese fast genau invers sind und sich in zufriedenstellender Näherung eine Frequenz 40 kHz ergibt.

3 Anwendung des Signals für ein KUNDT'sches Rohr

Nachfolgend stellen wir eine low-cost-Version des KUNDT'schen Rohres mit Ultraschallsendern (Abschnitt 3.1), die Verstärkung des 40 kHz Signals des Arduinos (Abschnitt 3.2) sowie Schülerexperimente (Abschnitt 3.3) vor.

3.1 Die low-cost-Version mit Ultraschallsendern

Ein KUNDT'sches Rohr kann mit Teilen aus dem 3D-Drucker und mit Ultraschallsendern sehr kostengünstig realisiert werden (HAVERKAMP et al., 2021). Die Ultraschallsender lassen sich z.B. aus dem verbreiteten und günstigen Ultraschallabstandssensor HC-SR04 herauslöten. Die Sender weisen einen sehr hohen Schalldruck von bis zu 125 dB bei 40 kHz auf (s. Datenblatt HC-SR04). Da die Wahrnehmungsgrenze beim Menschen bei ca. 16 – 20 kHz liegt, sind die Töne nicht mehr wahrnehmbar und der Betrieb dadurch angenehm lautlos. Allerdings ist der Frequenzbereich, in dem die Sender effektiv betrieben werden können, sehr klein, so dass der Schalldruck bereits bei Abweichungen von ± 1 kHz deutlich abfällt. Die Halterung aus dem 3D-Drucker (Abb. 7) nimmt die Anschlüsse für die Laborkabel auf. Auf der Oberseite lassen sich die durchsichtige Röhre (z.B. aus einem abgesägten Reagenzglas aus Plastik) sowie die Ultraschallsender einklemmen.

Die für den maximalen Schalldruck des verwendeten HC-SR04 (s. Datenblatt HC-SR04) erforderliche Frequenz von 40 kHz kann mit üblichen Leistungsfunktionsgeneratoren aus der Sammlung generiert werden. Eine Alternative ist die im nächsten Abschnitt vorgestellte Verstärkerschaltung mit Arduino und dem Programmcode für das 40 kHz Signal.

Abb. 7. Handliche low-cost-Version des KUNDT'schen Rohrs: Betriebsgerät (orange) und Halterung für Plexiglasröhre und Ultraschallsender (gelb)

3.2 Verstärkung des Signals

Der im Abschnitt 2 beschriebene Programmcode erzeugt ein 40 kHz Signal, indem jeweils die Ausgänge A0, A2 und A4 sowie A1, A3 und A5 wechselseitig von 0 V auf 5 V geschaltet werden. Diese Signale werden von dem einfachen 2-Kanal Verstärkerboard (eine doppelte H-Brücke) Typ L298N verstärkt (s. Datenblatt L298N Motortreiber). Der Arduino besitzt ebenso wie das Verstärkerboard einen Spannungsregler und kann daher statt mit üblicherweise 5 V über den USB-Anschluss auch mit bis zu 12 V über den Pin oder die Hohlbuchse betrieben werden. Das 3D-gedruckte Gehäuse hat zwei Eingänge für eine externe DC-Spannungsquelle (linke Seite in Abb. 8) sowie auch einen integrierten 9 V Block zum Betrieb des Arduinos und des Verstärkers. Die beiden 4 mm Ausgänge auf der rechten Seite liefern das verstärkte Signal für das KUNDT'sche Rohr oder auch ein Levitationsexperiment (s. dazu HAVERKAMP, SCHLUMMER, UBBEN, PUSCH, 2022). Eine stabile Plastikfolie (z.B. OHP-Folie) schützt die Komponenten und lässt gleichzeitig interessierte Blicke auf die Schaltung und die Komponenten zu.

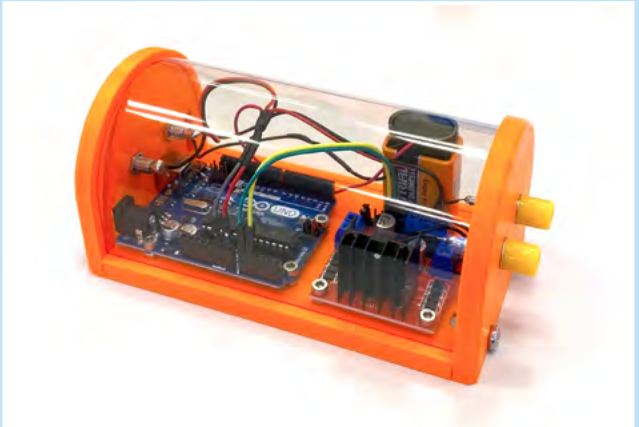

Abb. 8. Betriebsgerät mit Arduino im Gehäuse

3.3 Anwendung im Schulkontext und mögliche Schülerexperimente

Das KUNDT'sche Rohr bietet sowohl einen qualitativen als auch quantitativen Zugang zur Thematik von Schwingungen, stehenden Wellen und dem Zusammenhang zwischen Wellenlänge, Frequenz und Schallgeschwindigkeit. Eine grundlegende Anwendungsmöglichkeit des Rohres ist das Sichtbarmachen der mechanischen Schwingungen in Luft. Dies kann einen eindrucksvollen Zugang zum Thema "Schall" in der Sekundarstufe I bieten. Vertiefend sind dazu auch qualitative Diskussionen möglich. Eine schulgerechte Erklärung ist in der Online-Ergänzung zusammengefasst. Die hier vorgestellte Version des KUNDT'schen Rohres mit Ultraschallsendern ist günstig und einfach aufzubauen und eignet sich daher als Schülerexperiment. Gegenüber dem "klassischen" Aufbau mit einem Druckkammerlautsprecher hat es aber den Nachteil, dass die Wellenlänge wegen des festgelegten Frequenzbereichs der Ultraschallsender nicht variiert werden kann. Dennoch lassen sich einfache und eindrucksvolle Schülerexperimente umsetzen:

1. Ein Experiment zum Einstieg ist das Erzeugen der typischen Muster mit feinem Korkmehl oder Lycopodiumpulver. Die Schallquelle wird dazu so am Rohr positioniert, dass sich darin eine stehende Welle ausbildet. Die von der stehenden Welle hervorgerufene Bewegung der Luft im Rohr versetzt das Pulver an einigen Stellen sichtbar in Bewegung: Es bilden sich in regelmäßigen Abständen Bereiche aus, in denen das Pulver aufgewirbelt wird. Die

aufgewirbelten Bereiche des Pulvers bilden dabei eine charakteristische, lamellenartige Substruktur aus, die physikalisch gar nicht einfach zu erklären ist (für eine Erklärung dazu siehe z.B. SCHUBERT, 1981). Diese Bereiche wechseln sich ab mit Zonen, in denen das Pulver unbewegt liegen bleibt (Abb. 9 und 10). Bei der Durchführung kann helfen, die Bildung der Verwirbelungen an den Wellenbäuchen durch leichtes Klopfen oder Drehen des Röhrenstücks zu unterstützen. Wenn das Muster sich nicht deutlich abzeichnet, kann es außerdem helfen, die Menge an Pulver im Rohr zu reduzieren oder das Pulver gegen trockeneres Pulver zu ersetzen.

2. Die Musterbildung kann auch bei offenem Ende (einen Sender entfernen) sowie geschlossenem Ende (statt des entfernten Senders einen Stopfen einstecken) überprüft werden. Hierbei sind Bewegungen des Pulvers sichtbar, die Intensität ist für eine deutliche Ausprägung des Musters aber zu gering.

Abb. 9. Typisches Muster im KUNDT'schen Rohr

3. Quantitative Experimente zur Schallgeschwindigkeit in Luft sind durch Ausmessen der Abstände der aufgewirbelten Muster ebenfalls möglich. Der Abstand zwischen zwei Aufwirbelungen beträgt ca. 10 mm (über mehrere Aufwirbelungen messen!). Wie in der Online-Ergänzung beschrieben, ist dieser Abstand äquivalent zum Abstand zwischen zwei Schwingungsknoten der Schallschnelle. Durch Einsetzen in $c = \lambda \cdot f$ ergibt sich somit eine Schallgeschwindigkeit von etwa 400 m/s.

4. Mit dem hier vorgestellten Frequenzgenerator lassen sich nicht nur Schulexperimente zum KUNDT'schen Staubrohr realisieren, sondern auch das eindrucksvolle Phänomen der Ultraschalllevitation im Klassenraum demonstrieren. Im MNU Beitrag von HAVERKAMP et al. (2022) wurde ein Zugang zu stehenden Wellen mittels Ultraschallevitation vorgestellt, der an die hier zugrunde liegende Physik zur stehenden Welle und Ultraschall anschließt.
Das Levitationsexperiment kann sogar mit verkleinerten Dimensionen direkt am Arduino ohne Verstärker mit dem vorgestellten Programmcode betrieben werden und eignet sich somit auch als einfaches "Tüftel-Projekt" für Lernende, welches 3D-Druck, Löten und Mikrocontroller in einem physikalischen Experiment miteinander verbindet (Abb. 11).

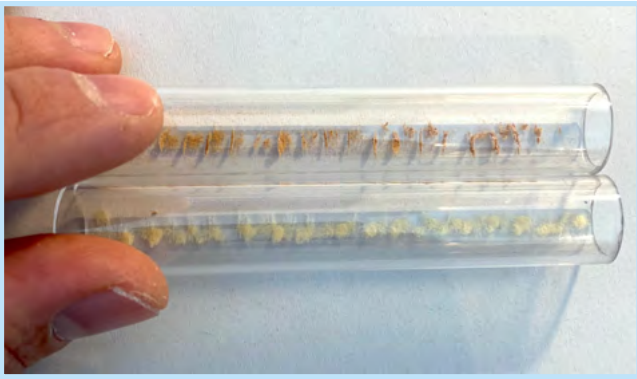

Abb. 10. Durch das Entfernen der Röhre können die Abstände einfacher bestimmt werden. Oben: feines Korkmehl; Unten: Lykopodiumpulver

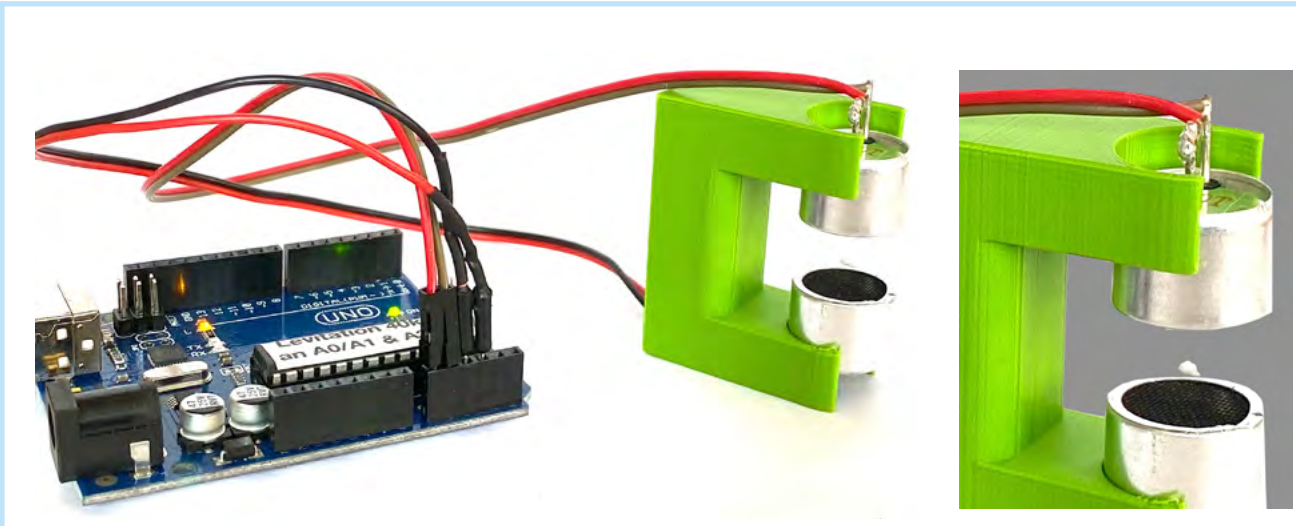

Abb. 11. Bei geringen Abständen reicht das unverstärkte Signal des Arduinos aus, um eine ausreichend starke stehende Welle zu erzeugen. Rechts: eingefärbter Hintergrund

4 Ausblick

Der Experimentieraufbau mit Arduino und Oszilloskop bietet die Chance, die elektro- und programmiertechnischen Aspekte der Signalerzeugung mit dem Arduino zu thematisieren und somit Physik und Informatik zu verbinden. So können die in den Abschnitten 1 und 2 dargestellten Varianten der Programmierung mit ihren Vor- und Nachteilen auch gemeinsam mit den Lernenden am eigenen Aufbau erprobt und verbessert werden. In diesem Zusammenhang können die Lernenden am einfachen Beispiel des Rechtecksignals auch an die Bedienung eines Oszilloskops herangeführt werden und so praktische Kenntnisse im Einsatz von Messgeräten erlangen. Der altbekannte Aufbau des KUNDT'schen Rohres gewinnt somit durch den Einsatz des Arduinos an neuen Facetten.

Die STLs, der Programmcode und die Materialliste finden sich unter: https://physikkommunizieren.de/3d-druck/3d-printed-kundts-tube/

In der Online-Ergänzung finden Sie eine schulgerechte Erklärung zum KUNDT'schen Rohr.

mit online www.mnu.de ergänzung

Literatur

Datenblatt HC-SR04 (o.J.). *Air Ultrasonic Ceramic Transducers. https://cdn-reichelt.de/documents/datenblatt/B400/UST%23PRT.pdf* (letzter Abruf 12.08.22)

Datenblatt L298N (o.J.). *Motortreiber. https://cdn-reichelt.de/documents/datenblatt/A300/ME089-N.pdf* (letzter Abruf 12.08.22)

Datenblatt Arduino Uno (o.J.). *Arduino Uno Rev3. https://store.arduino.cc/products/arduino-uno-rev3?selectedStore=eu* (letzter Abruf 12.08.22)

EWALD, W. (o.J.). *Timer und PWM - Teil 1 (8 Bit Timer0/2). https://wolles-elektronikkiste.de/timer-und-pwm-teil-1* (letzter Abruf 12.08.22)

HAVERKAMP, N., HAVEMANN, J., HOLZ, C., UBBEN, M., SCHLUMMER, P. & PUSCH A. (2021). A new implementation of Kundt's tube: 3D-printed low-cost set-up using ultrasonic speakers. *Physics Education, 56, 9.* doi: 10.1088/1361-6552/abd0d7.

HAVERKAMP, N., SCHLUMMER, P., UBBEN, M. & PUSCH, A. (2022). Ultraschalllevitation als Zugang zu stehenden Wellen. *MNU-Journal*, 75(1), 14–18.

KUNDT, A. (1866). Über eine neue Art akustischer Staubfiguren und über die Anwendung derselben zur Bestimmung der Schallgeschwindigkeit in festen Körpern und Gasen. *Annalen der Physik und Chemie. Band 203, Nr. 4,* S. 497–523, doi:10.1002/andp.18662030402

SCHUBERT, G. (1981). Staubfiguren im Kundt'schen Rohr. *Physik in unserer Zeit; Band 12 Heft 5*, 147–150.

Referenz zum *delay()*-Befehl. *https://www.arduino.cc/reference/de/language/functions/time/delay* (letzter Abruf 12.08.22)

WALLER, H. (o.J.). *LEDs schalten mit PORT/DDR. https://hartmut-waller.info/arduinoblog/leds-schalten-port-ddr* (letzter Abruf 12.08.22)

NILS HAVERKAMP, PAUL SCHLUMMER, Dr. MALTE UBBEN und Dr. ALEXANDER PUSCH lehren und forschen am Institut für Didaktik der Physik an der Westfälischen Wilhelms-Universität Münster.

MNU-Journal – Ausgabe 02.2023 – ISSN 0025-5866 – © Verlag Klaus Seeberger, Neuss