Arduino-Grundkurs

Julian Heinzel

2. April 2022

Inhaltsverzeichnis

1 Einleitung

Arduino ist eine sogenannte „Physical computing"-Plattform, die auf einem einfachen Mikrocontroller und einer dazugehörigen *IDE* (Integrated Development Environment) basiert. Damit können leicht Projekte umgesetzt werden, die mit ihrer Umwelt interagieren, Messungen vornehmen und Geräte steuern. Die folgenden Aufgaben sollen eine Einführung in die Programmierung dieser Plattform darstellen und beziehen sich auf den *Arduino Uno*.

Als Programmiersprache kommt C/C++ zum Einsatz, welches um die Arduino-Bibliothek erweitert wurde. Alle darin enthaltenen Befehle können auf der Homepage arduino.cc unter dem Menüpunkt „Documentation > Reference" eingesehen werden. Auf der Webseite befinden sich außerdem zahlreiche Beispiele und Tutorials.

Ein leeres Arduinoprojekt enthält mindestens zwei Funktionen:

```
1 void setup()
2 {
3 }
4
5 void loop()
6 {
7 }
```

Diese werden von der Arduino-Bibliothek deklariert und müssen stets vorhanden sein. *setup* wird dabei einmal zu Anfang ausgeführt und sollte Anweisungen enthalten, die den Arduino wie benötigt konfigurieren. *loop* wird daraufhin unablässig wiederholt.

2 Digitaler Output

Der Arduino Uno besitzt 13 digitale Ein- bzw. Ausgänge. Digital heißt, dass diese nur zwischen zwei Zuständen unterscheiden können: HIGH oder LOW, bzw. null oder fünf Volt. Damit kann zum Beispiel eine LED ein- und ausgeschaltet werden.

In den Arduino ist auf Port D13 bereits eine LED eingebaut, es kann aber genauso eine externe LED angeschlossen werden. Dabei muss jedoch unbedingt ein Vorwiderstand verwendet werden, um den Strom durch die LED auf circa $20\,\text{mA}$ zu begrenzen: Ansonsten löst diese oder der Arduino sich in magischen Rauch auf.

Zuerst muss in der Setupfunktion definiert werden, dass der betreffende Port als Output verwendet werden soll, also dass damit ein anderes Gerät mit Strom und Spannung versorgt werden soll. Dazu existiert die Funktion

```
void pinMode(int pin, int mode)
```

wobei *mode* entweder OUTPUT oder INPUT sein kann.

Danach kann in der Loop mit *digitalWrite* der verwendete Pin auf HIGH oder LOW gesetzt werden.

```
void digitalWrite(int pin, int value)
```

Die Ausführung des Programms lässt sich mit mit zwei Funktionen unterbrechen:

```
void delay(int milliseconds)
void delayMicroseconds(int microseconds)
```

Weitere Informationen über die genannten Funktionen sind auf der offiziellen Webseite arduino.cc zu finden.

- Schreibe ein Programm, das eine LED für einige Sekunden einschaltet und danach einige Sekunden ausschaltet.

3 Digitaler Input

Die digitalen Ports können auch als Input verwendet werden. Dabei gelten Spannungen über $3.3\,\text{V}$ als HIGH, Spannungen darunter als LOW. Folgende Funktion kann den Zustand eines Pins überprüfen:

```
int digitalRead(int pin)
```

Dabei wird man anfangs versucht sein, die Schaltung nach Abbildung 1 wie folgt aufzubauen:

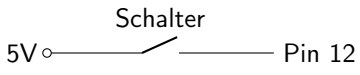

Abbildung 1: Schaltung ohne Pull-down-Widerstand

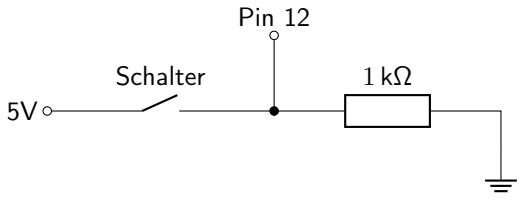

Abbildung 2: Schaltung mit Pull-down-Widerstand

Solange der Schalter geschlossen ist, wird man auf Pin 12 HIGH messen. Wird dieser jedoch geöffnet, so liegt der Pin nicht mehr auf HIGH - aber auch nicht auf LOW. Das Potential ist an dieser Stelle undefiniert, der Eingang ist „floating“, nimmt störende Felder aus seiner Umgebung auf und misst so eine mehr oder weniger zufällige Spannung. Abhilfe schafft Schaltung 2. Ist der Schalter geschlossen, so fällt die gesamte Spannung an dem *Pull-down-Widerstand* ab, am Eingang liegen wie zuvor $5\,\text{V}$. Im geöffneten Zustand jedoch wird der Eingang über den Widerstand auf Massepotential gezogen, an Pin 12 wird LOW gemessen. Analog lassen sich auch *Pull-up-Widerstände* einsetzen.

- Schreibe ein Programm, welches in jedem Durchgang der Hauptschleife überprüft, ob ein Schalter gedrückt ist und dementsprechend eine LED an- oder ausschaltet.

4 Funktionen

- Schreibe eine Funktion vom Datentyp *void*, welche einen Ton zufälliger Höhe auf einem Lautsprecher abspielt. Suche dafür in der Referenz auf arduino.cc geeignete Funktionen, um einen Ton zu erzeugen.

5 Analoger Input

Der Mikrocontroller kann nicht nur zwischen HIGH und LOW unterscheiden, sondern über die analogen Inputs A1 bis A6 auch die genaue Größe einer Spannung ermitteln. Die Spannung zwischen $0\,\mathrm{V}$ und $5\,\mathrm{V}$ wird als 10-Bit-Ganzzahl dargestellt, also als Wert zwischen 0 und 1023.

- Schließe ein Potentiometer zwischen $0\,\mathrm{V}$ und $5\,\mathrm{V}$ an und erstelle so eine variable Spannungsquelle.
- Schließe fünf LEDs an, von denen je nach Spannung jeweils eine leuchten soll. Vorwiderstand nicht vergessen!

6 Serielle Kommunikation

6.1 Daten senden

Über die serielle Schnittstelle kann der Mikrocontroller mit dem angeschlossenen Computer oder anderen Geräten kommunizieren. Dazu muss in der Setup-Funktion die Kommunikation mit einer bestimmten Baudrate (Übertragungsgeschwindigkeit/Symbole pro Sekunde) gestartet werden. Die am PC ankommenden Daten können dann im „Serial Monitor“ der Arduino-Umgebung eingesehen werden. Folgende Funktionen sind unter anderem wichtig:

```
void Serial.begin(speed)
```

und

```
long Serial.println(value)
```

- Wandle das Programm aus der letzten Aufgabe so ab, dass die aktuell gemessene Spannung auf dem Computer angezeigt wird.
- Falls du nicht weiter kommst, hilft dir die Website arduino.cc.

6.2 Daten empfangen

Es können nicht nur Daten gesendet werden - der Arduino kann auch Daten vom Computer empfangen. Dazu können die folgenden Funktionen verwendet werden:

```
int Serial.available()
```

```
int Serial.read()
```

- Der Arduino soll nun den seriellen Port auf ankommende Befehle überprüfen. Wird eine Zahl zwischen eins und fünf empfangen, soll die Messung des Potentiometers ignoriert und die entsprechende LED eingeschaltet werden. Wird das Potentiometer verstellt, so soll erneut die Spannungsmessung bestimmend für die LEDs sein.

7 PWM-Output

Im Gegensatz zu anderen Mikrocontrollern besitzt der Arduino Uno keine Digital-Analog-Wandler (DAC), um Spannungen zwischen null und fünf Volt zu erzeugen. In vielen Fällen kommt als Ersatz jedoch die Verwendung eines *PWM-Signals* in Frage. PWM steht für *Pulse Width Modulation* - dabei wird ein Rechtecksignal fester Frequenz erzeugt, jedoch die Zeit variiert, die das Signal auf HIGH-Pegel liegt:

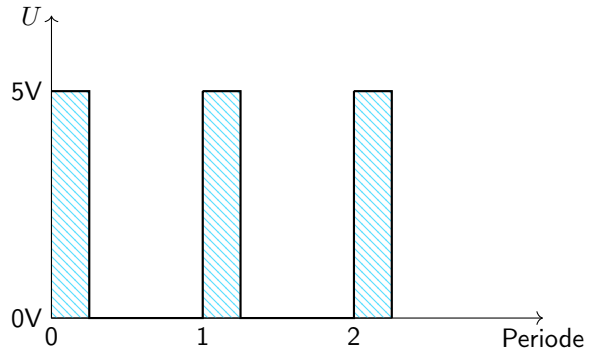

Abbildung 3: PWM-Signal mit 25% Pulsweite

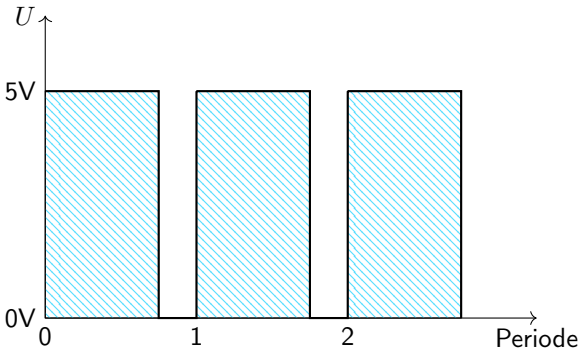

Abbildung 4: PWM-Signal mit 75% Pulsweite

Das Verhältnis von Ein- und Ausschaltdauer der Impulse nennt man *Pulsweite* oder *Duty Cycle*. Über die Modulation dieser Pulsweite lässt sich zum Beispiel die Helligkeit einer LED oder die Geschwindigkeit eines Motors regeln: Da das menschliche Auge bzw. die Mechanik zu träge ist, um die einzelnen Pulse aufzulösen und voneinander zu trennen, ergibt sich effektiv eine mittlere Spannung zwischen null und fünf Volt.

Alternativ ließe sich das Rechtecksignal mit einem großen Kondensator oder einer Spule glätten, die die Spannung puffern und die dahinter liegende Schaltung versorgen, solange das PWM-Signal LOW ist.

Servomotoren interpretieren das PWM-Signal im Gegensatz zu normalen Gleichstrommotoren als Position und drehen sich je nach Pulsweite in die entsprechende Position.

Nicht alle digitalen Pins eines Mikrocontrollers können ein PWM-Signal erzeugen. Bei einem Arduino Uno sind das zum Beispiel nur die Pins 3, 5, 6, 9, 10 und 11, das lässt sich auf der Website arduino.cc auch für andere Arduinomodelle nachschlagen. Diese Pins können verwendet werden, um die Funktion

```
void analogWrite(pin, value)
```

aufzurufen. *value* ist dabei die Pulsweite, angegeben als Zahl zwischen 0 (immer aus) und 255 (immer an). Um *analogWrite* verwenden zu können, muss der Pin übrigens nicht mit *pinMode* als Ausgang definiert werden.

- Baue eine Schaltung aus einer LED und einem Vorwiderstand an einem PWM-fähigen Pin.
- Schreibe eine Funktion, die als Parameter einen Pin und die gewünschte Helligkeit in Prozent annimmt und die LED via *analogWrite* anschaltet.

8 Interrupts

Die dimmbare LED aus der vorherigen Aufgabe soll nun via Knopfdruck durch einen *Interrupt* gesteuert werden. Interrupts sind hardwaremäßig in den Mikrocontroller eingebaut und können auf äußere Umstände reagieren. Sie können den aktuell ausgeführten Code unterbrechen („Interrupt“) und anstatt dessen „zwischendurch“ eine andere Funktion auslösen. Welche Funktion das ist und unter welcher Bedingung der Interrupt ausgelöst wird, kann mit der Methode

```
void attachInterrupt(int interrupt, void (*function)(void), int mode)
```

festgelegt werden. Mit ihr kann eine Funktion sozusagen „An einen Interrupt angeheftet“ werden. Der Arduino Uno verfügt über zwei interruptfähige Pins, welche von null an aufwärts nummeriert werden und den digitalen Pins zwei und drei entsprechen. Um diese beiden Nummerierungsarten verständlich und unabhängig vom verwendeten Mikrocontroller ineinander zu übersetzen, existiert die Funktion

```
int digitalPinToInterrupt(int pin)
```

Auf einem Arduino Uno wird durch diese Funktion also zum Beispiel zwei zu null und drei zu eins.

Der zweite Parameter der attachInterrupt-Funktion ist dabei einfach der Name der Funktion, die durch den Interrupt ausgeführt werden soll.

mode kann *RISING*, *FALLING*, *CHANGE* oder *LOW* sein und bestimmt, wann der Interrupt ausgelöst wird. RISING bedeutet in diesem Fall zum Beispiel, dass der Interrupt immer dann ausgelöst wird, wenn die Spannung am dazugehörigen Pin von LOW auf HIGH springt.

Codeblock 1: Ausschnitt aus dem Programmcode einer Messbox des Windkraftwerks

```
1  volatile int rounds = 0;
2  const int rpmPin = 3;
3
4  void setup()
5  {
6    pinMode(rpmPin, INPUT_PULLUP);
7    attachInterrupt(digitalPinToInterrupt(rpmPin), count, FALLING);
8  }
9
10 void loop() { }
11
12 void count()
13 {
14   rounds++;
15 }
```

Interrupts werden zum Beispiel bei der Drehzahlmessung des drachenbasierten fliegenden Windkraftwerks verwendet, das in Jahren 2014 bis 2016 am Aerospace Lab Herrenberg entwickelt wurde. Ein Auszug der dort verwendeten Software ist in Codeblock 1 dargestellt. In Zeile sechs wird der interne Pull-up-Widerstand von Pin drei aktiviert, der Pin liegt also im Normalfall über $20\,\mathrm{k\Omega}$ auf $5\,\mathrm{V}$ und ist mit der Lichtschranke verbunden. Wird die Lichtschranke jedoch nicht mehr blockiert, schaltet der darin befindliche Fototransistor durch und zieht Pin drei auf Nullpotential.

Da die Spannung von $5\,\mathrm{V}$ auf $0\,\mathrm{V}$ *abfällt* (FALLING), wird die Funktion *count* ausgelöst und die Variable *rounds* inkrementiert - wie in Zeile sieben definiert.

- Ergänze die Schaltung aus der vorherigen Aufgabe um einen Schalter an einem der Interruptpins. Der Schalter kann den Pin bei Aktivierung zum Beispiel mit $5\,\mathrm{V}$ verbinden. Vergiss in diesem Fall nicht einen Pull-Down-Widerstand wie in Aufgabe 3 zu verbauen, sodass bei deaktiviertem Schalter auch tatsächlich $0\,\mathrm{V}$ an dem Interruptpin anliegen.
- Schreibe eine Funktion *step*, die die LED mit jedem Knopfdruck etwas heller stellt. Du kannst dabei auf die Funktionen aus der vorherigen Aufgabe zugreifen.
- Wenn die LED mit maximaler Helligkeit leuchtet, soll der nächste Aufruf der Funktion *step* sie wieder ausschalten.

Verbesserungsvorschläge?

Gerne dürft ihr Fehler und Verbesserungsvorschläge auf `https://github.com/Jeinzi/Arduino-Grundkurs` melden - dort findet ihr auch die aktuellen LaTeX-Quelldateien und einen Link auf diese PDF :)