8 ARDUINO

8.1 Einleitung

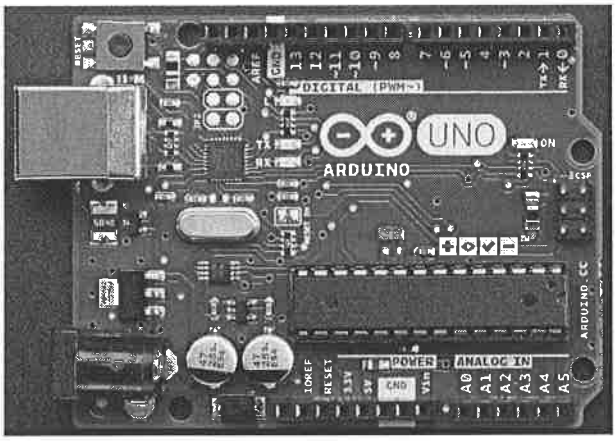

Abb. 8.1.1 Der ARDUINO UNO Mikrokontroller

Der Arduino ist für ElektronikerInnen so etwas wie der Zauberstab für Harry Potter: Je nach Arbeitskraft, Können und Phantasie kann man mit ihm Unglaubliches anstellen. Kurz gesagt, können Sie beispielsweise dem Arduino alles *programmieren*, was Sie in Kapitel 7 mit „Käfern" zusammengestellt haben. Natürlich ist der Arduino wie jeder Computer im „Innersten" aus NAND-Gattern, Flip-Flops etc. aufgebaut. Als Verbindung zur „Aussenwelt", der *Peripherie* verfügt der Arduino Uno über einen 10 bit-ADC (Analog-Digital-Wandler), und 14 Digital-Anschlüsse (Pins), welche wahlweise, als Ein- oder Ausgänge betrieben werden können. Weiter verfügt er über eine serielle Leitung, mit welcher er via USB-Anschluss mit einem Computer verbunden werden kann; über diesen Anschluss wird er vom Computer aus auch programmiert. Ist dann das Programm einmal auf den Arduino geladen, kann er völlig eigenständig funktionieren. Die Stromversorgung geht über USB oder wahlweise über separate Anschlüsse und hat mit 5 Volt TTL-Standard.

Tabelle 8.1 Technische Daten des Arduino UNO[12]

Microcontroller	ATmega328	DC Current per I/O Pin	40 mA
Operating Voltage	5V	DC Current for 3.3V Pin	50 mA
Input Voltage (recommended)	7-12V	Flash Memory	32 KB
Input Voltage (limit)	6-20V	Flash Memory for Bootloader	0.5 KB
Digital I/O Pins	14	SRAM	2 KB
PWM Digital I/O Pins	6	EEPROM	1 KB
Analog Input Pins	6	Clock Speed	16 MHz

Die *Struktur* von Arduino-Schaltungen ist folgende:

- Man hat elektronische Bauteile *vor* dem Arduino (beispielsweise einen LDR, oder Sensoren für Temperatur, Beschleunigung, Potentiometer um Spannungen einzustellen, Schalter und vieles mehr).
- Im Zentrum steht der Arduino, mit dessen Hilfe man die Signale einliest, verarbeitet und ausgibt.
- *Hinter* dem Arduino sind wieder elektronische Bauteile, wie LED, Anzeigen, Schieberegister, Motoren, Servos, und vieles mehr.
- Dann existieren viele so genannte *Shields*, welche man auf den Arduino steckt und mit deren Hilfe man beispielsweise etwas steuern, drahtlos kommunizieren und wieder alles Mögliche mehr kann. Über eine serielle Leitung ist die Kommunikation mit dem Computer möglich (d.h. nicht nur zum Laden der Programme!).

Wir betrachten im Folgenden eine Handvoll nicht zu aufwendiger Projekte, damit Sie einige Möglichkeiten kennen lernen. Interessierte können sich das *Arduino Starter Kit* anschaffen[13]. Weiter existiert viele Literatur[14] zum Arduino und natürlich das ganze Online-Universum.
Auch hier erarbeiten wir die notwendigen Fähigkeiten anhand der Projekte. Mit einer Seite, der *Elektronik* sind Sie nun sehr hinreichend vertraut; in die andere Seite die *Programmierung* müssen Sie sich noch etwas einarbeiten.

[12] www.arduino.cc
[13] beispielsweise bei Distrelec für CHF 84.80 erhältlich (Februar 2016)
[14] beispielsweise BRÜHLMANN, THOMAS: Arduino Praxiseinstieg, Mönchengladbach 2015

8.2 Projekt *Blink*

Hier werden wir eine Leuchtdiode zum Blinken bringen und anschliessend das Projekt etwas erweitern.
Zunächst zur Schaltung – bauen Sie diese ungefähr gemäss Abb. 8.2.1. Wählen Sie für die sechs LED (Farbe nach Wahl) je *gleiche* Widerstände im Bereich von 270 Ω bis 390 Ω. Achten Sie darauf, dass die *Kathoden* der LED (flache Gehäuseseite) auf der Masse liegen. Ansteuerung mit den Digital-Pins 7 bis 12.

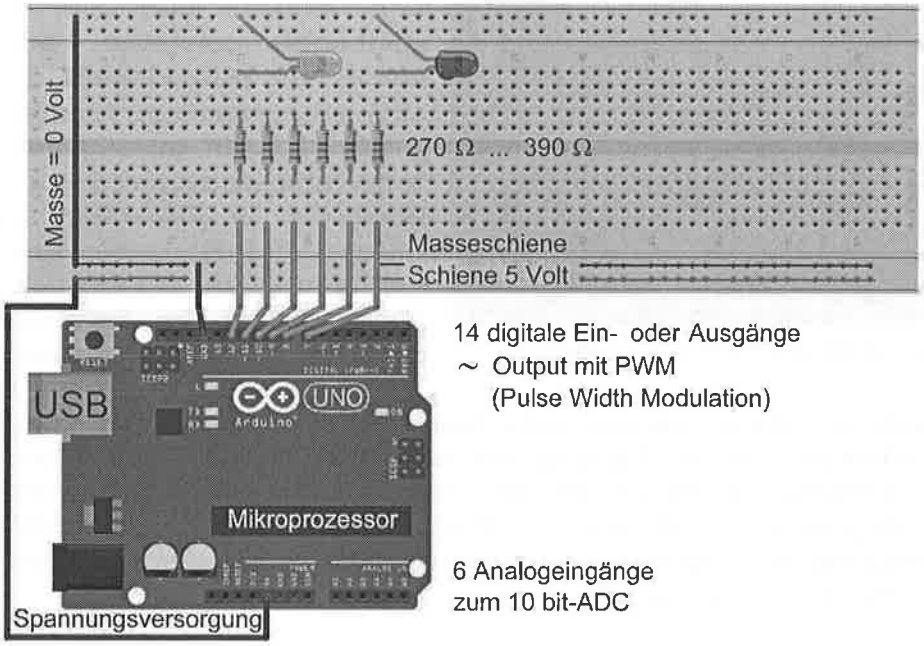

Abb. 8.2.1 Schaltung mit ARDUINO und 6 LED

Sie benötigen *keine* spezielle Spannungsversorgung – die Schaltung wird über die USB-Leitung gespeist. Schliessen Sie also den Arduino über USB an den Computer an. Kontrollieren Sie noch, ob die LED's richtig gepolt sind, indem Sie die je die Anschlussdrähte in einen Pin der 5V-Schiene stecken – die LED sollten dann leuchten.

8.2.1 Anlauf No. 1 – eine LED blinkt

Nun geht's zum ersten ARDUINO-Programm – das Programmfenster sieht nach dem Öffnen zunächst so aus:

Ein **ARDUINO-Sketch** ist grundsätzlich in zwei Teilen strukturiert.

- In *setup* kommen alle Befehle vor, welche nur *einmal* ausgeführt werden sollen, beispielsweise die Initialisierung von Ausgangs-Pins.
- In *loop*, der Schleife, stehen alle Befehle, welche *immer wieder* ausgeführt werden. Diese Schleife durchläuft das Programm immer und immer wieder, bis man ARDUINO den Stecker zieht, die RESET-Taste drückt, oder ein neues Programm hochlädt.

Abb. 8.2.2 Das ARDUINO Programmfenster nach dem Öffnen

Im nächsten Schritt schreiben Sie den ARDUINO-Sketch aus Abb. 8.2.3. Gross- und Kleinschreibung beachten, jedes Zeichen muss stimmen! Grundsätzlich wird im Sketch jeder Befehl mit einem Strichpunkt [;] abgeschlossen – vergisst man gerne. Es können auch mehrere Befehle durch [;] abgetrennt, in einer Zeile stehen, das wird aber schnell unübersichtlich.
Die **Kommentare** zu den Befehlen werden mit [//] abgetrennt, diese werden bei der Programmausführung *nicht* beachtet. Es ist sehr nützlich, zu kommentieren, weil man dann nach Monaten noch weiss, was man sich gedacht hat – und natürlich auch als Erläuterungen für andere. Die Kommentare in Abb. 8.2.3 sind allerdings seeeeehr ausführlich gehalten.
Sprach-**Referenz**: In der Menuleiste können Sie auf (HILFE – REFERENZ) klicken, dann finden Sie mit Erläuterungen und Beispielen versehen die Befehle für die ARDUINO Programme. Jede Zeile ist im Programmeditor nummeriert – aus unerfindlichen Grünen lassen sich die Zeilennummern leider nicht ins pdf drucken, daher *fehlen* die Nummern in Abb. 8.2.3 .

```
void setup() {              //Beginn Setup
  pinMode(8, OUTPUT);       //initialisiert Pin 8 als Output
}                           //Ende Setup

//Hauptroutine
void loop() {               //Schleifenbeginn
  digitalWrite(8, HIGH);    //Zustand HIGH (5 Volt) auf Pin 8
  delay(500);               //Stoppt Programmausführung während 0.5 sec
  digitalWrite(8, LOW);     //Zustand LOW (0 Volt) auf Pin 8
  delay(500);               //Stoppt wieder während 500 msec = 0.5 sec
}                           //springt zu Schleifenbeginn
```

Abb. 8.2.3 ARDUINO-Sketch, um LED No. 8 im Sekundentakt blinken zu lassen

Kompilieren: Wenn Sie glauben, alles korrekt abgeschrieben zu haben, können Sie den Sketch mit dem (GUT-Häkchen)-Knopf ganz links *kompilieren* (dann wird das Programm in die Maschinensprache übersetzt). Falls unten im Programmfenster eine Fehlermeldung erscheint: korrigieren!
Nun ist das Programm bereit zum **Hochladen**. Dafür muss der ARDUINO mit einem USB-Kabel an den Computer angeschlossen sein. Es bleibt noch, den *Kommunikations-Port* zu kontrollieren. Dazu klicken Sie in der Menu-Leiste auf (WERKZEUGE – PORT) und dann den COM, welcher mit einem Häkchen versehen ist. Der aktuelle Kommunikations-Port erscheint dann ganz unten rechts, siehe Abb. 8.2.2: „Arduino Uno on COM3"
Wenn auch das getan ist, drücken Sie zum Hochladen den (→)-Knopf. Falls jetzt die LED No. 8 im Sekundentakt blinkt: Herzliche Gratulation!
Sie können noch ein wenig mit dem Programm spielen, indem Sie die Zeiten in den Zeilen 8 und 10 ändern.

Notizen

8.2.2 Anlauf No. 2 – zwei LEDs blinken

Nun lassen wir zwei LED's wechselblinken. Sie können dazu das Programm aus Abb. 8.2.3 einfach abändern, indem Sie Befehlszeilen mittels COPY-PASTE kopieren und einfügen. Neu in diesem Sketch wird für die Zeitdauer(n) in den Zeilen 1 und 2 je eine *Integer*-Variable [int dauerH] und [int dauerL] definiert. Solche ganzzahlige Variablen benötigen 16 bit Speicherplatz (im Prinzip 15 bit, das höchste bit für das Vorzeichen). Jede Variable muss definiert werden, bevor sie im Programm vorkommt, beziehungsweise spätestens dann, wenn sie auftaucht. In folgendem Sketch werden beiden Variablen auch bereits Werte zugewiesen, dies ist aber grundsätzlich nicht notwendig. Der Vorteil dieses Vorgehens ist offensichtlich: Sie müssen die numerische Zeitdauer nur einmal wählen.

```
1   int dauerH = 500;        //Integer Variable für Dauer HIGH setze 0.5 sec
2   int dauerL = 0;          //Integer Variable für Dauer LOW setze 0 msec
3   void setup() {           //Beginn Setup
4     pinMode(8, OUTPUT);    //initialisiert Pin 8 als Output
5     pinMode(9, OUTPUT);    //initialisiert Pin 9 als Output
6   }                        //Ende Setup
7
8   //Hauptroutine
9   void loop() {            //Schleifenbeginn
10    digitalWrite(8, HIGH); //Zustand HIGH (5 Volt) auf Pin 8
10    delay(dauerH);         //Stoppt Programmausführung während def. Zeit
12    digitalWrite(8, LOW);  //Zustand LOW (0 Volt) auf Pin 8
13    delay(dauerL);         //Stoppt Programmausführung während def. Zeit
14    digitalWrite(9, HIGH); //Zustand HIGH (5 Volt) auf Pin 9
15    delay(dauerH);         //Stoppt Programmausführung während def. Zeit
16    digitalWrite(9, LOW);  //Zustand LOW (0 Volt) auf Pin 9
17    delay(dauerL);         //Stoppt Programmausführung während def. Zeit
18  }                        //springt zu Schleifenbeginn
```

Abb. 8.2.4 Wechselblinker mit einstellbarer Zeit

Auch hier: Spielen Sie mit verschiedenen Zeiten.
Grundsätzlich handelt es sich allerdings hierbei nicht gerade um ein elegantes Programm, weil jede Diode explizit initialisiert und anschliessend einzeln in der Schleife aufgeführt wird. So macht man es also besser nicht, sondern...

8.2.3 Anlauf No. 3 – Lauflicht mit 6 LED

```
1   //Lauflicht
2   int dauerH = 5;        //Integer Variable für Dauer HIGH setze 5 msec
3   int dauerL = 100;      //Integer Variable für Dauer LOW setze 100 msec
4
5   void setup() {
6     //For-Next-Schleife um Pin1 bis Pin13 als Output zu initialisieren
7     for (int pinNr = 1; pinNr <= 13; pinNnr++) {
8       pinMode(pinNr, OUTPUT);
9     }                      //Ende For-Schleife
10  }                        //Ende Setup
11
12  //Hauptroutine
13  void loop() {
14    //For-Schleife um von Pin12 absteigend bis Pin7 zu schreiben
15    for (int pinNr = 12; pinNr >= 7; pinNr--) {
16      digitalWrite(pinNr, HIGH);
17      delay(dauerH);
18      digitalWrite(pinNr, LOW);
19      delay(dauerL);
20    }                       //Ende For-Schleife
21  }
```

Abb. 8.2.5 6-LED-Lauflicht mit einstellbarer Zeit

Obwohl jetzt dreimal so viele LED's blinken, ist das Programm kürzer – warum?
Im setup-Teil werden die Pins mit Hilfe einer [for]-Schleife initialisiert (Zeilen 7/8):

```
for (int pinNr = 1; pinNr <= 13; pinNr++) {
    pinMode(pinNr, OUTPUT);
  }
```

Die Variable [int pinNr] wird direkt im Befehl definiert. Anschliessend wird in der [for]-Schleife [pinNr] = 1 gesetzt und dann Pin 1 auf [OUTPUT] gestellt. Im folgenden Schritt (Zeile 9) springt das Programm dann wieder an den Anfang der [for]-Schleife. Nun wird [PinNr] um 1 hinaufgesetzt d.h. [pinNr] = 2; der Befehl dazu lautet [pinNr++]), und so wird weiter hinaufgezählt bis die *Bedingung* [pinNr] = 13, Befehl [pinNr <= 13] erreicht und ausgeführt ist. Anschliessend springt das Programm aus der [for]-Schleife.
Solches Vorgehen spart natürlich viele Programmzeilen, weil nicht jede LED einzeln initialisiert werden muss. Dasselbe Prozedere finden Sie in der Hauptschleife, nur wird dort mit Hilfe von Befehl [pinNr--] von Pin 12 auf Pin 7 heruntergezählt, um die LED ein- und auszuschalten (Zeile 15).

8.2.4 Anlauf No. 4 – Lauflicht steuern mit LDR

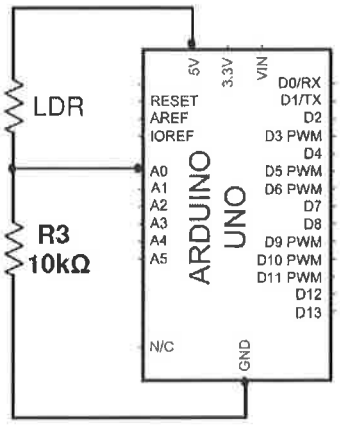

Abb. 8.2.6 Schaltung gemäss Abb. 8.2.1 modifiziert mit LDR in Serie mit 10 kΩ-Widerstand.

Bis jetzt haben wir von der in Kap. 8.1 aufgeführten Struktur des ARDUINO nur die letzten beiden Teile angewendet: *Programm* und *Output*. In diesem Schritt werden wir die Lauflicht Schaltung mittels eines LDR (Light-Dependent-Resistor) steuern.
Modifizieren Sie also Ihre bestehende Schaltung gemäss Abb. 8.2.6. Die 6 LED's mit ihren Seriewiderständen sind hier *nicht* eingezeichnet.

Wie Sie wissen, hängt der Widerstand des LDR sehr stark von der Beleuchtung ab (siehe Kapitel 2.2.3) und variiert von einigen 10 Ω bis weit über 10 MΩ. Damit fallen via Spannungsteiler über R3 je nach Beleuchtung einige 100 mV bis fast die gesamte Versorgungsspannung von 5 Volt ab. Diese Spannung wird auf den Analog-Eingang A0 des ARDUINO gegeben und vom internen 10-bit-ADC (Analog-to-Digital-Converter) in eine *Zahl* zwischen 0 und 1024 verwandelt, die man nun rechnerisch weiter verarbeiten kann. Zum Aufbau eines ADC siehe Kapitel 5.4.
Im Sketch aus Abb. 8.2.7 wird in der Hauptroutine (Zeile 14) mit dem Befehl [sensorwert = analogRead(A0)]der Wert aus dem ADC in die Variable [sensorwert] eingelesen. Anschliessend wird das Programm mit [substeuerung()] in eine Subroutine geschickt (Zeile 15). Man sollte die Hauptroutine in Programmen aus Übersichtlichkeitsgründen möglichst frei von Detail-Befehlen halten und diese in Subroutinen ausführen lassen. In der Subroutine (Zeilen 20/21) werden noch die Zeiten angepasst mit dem Befehl [dauer = sensorwert / 10].

Notizen

```
1   //6-LED Lauflicht
2   void setup() {
3                                   //digiale Pins 0 bis 13 auf Output
4     for (int pinnr = 0; pinnr <= 13; pinnr++) {
5       pinMode(pinnr, OUTPUT);
6     }
7   }
8   //div. Variablen setzen
9   int sensorwert;
10  int dauer;  int Delay;
11
12  //Hauptroutine
13  void loop() {
14    sensorwert = analogRead(A0); //Einlesen LDR mit ADC Analog-Eingang A0
15    substeuerung();              //Subroutine Ansteuerung 6 LED
16  }
17
18  //Subroutine Ansteuerung 6 LEDs
19  void substeuerung() {
20    dauer = sensorwert / 10;      // Dauer anpassen
21    Delay = sensorwert / 20;      // Verzögerung anpassen
22    for (int pinnr = 12; pinnr >= 7; pinnr--) {
23      digitalWrite(pinnr, HIGH);
24      delay(dauer);
25      digitalWrite(pinnr, LOW);
26      delay(Delay);
27    }
28  }
```

Abb. 8.2.7 Lauflicht mit Zeit-Ansteuerung durch einen LDR

8.2.5 Anlauf No. 5 – Daten in den seriellen Monitor schreiben

ARDUINO besitzt einen einfachen seriellen Monitor, in den Daten geschrieben werden können. Dies ist manchmal während der Programmentwicklung nützlich, um beispielsweise errechnete Werte zu kontrollieren. Um den Monitor einzuschalten, klicken Sie im Programmfenster ganz oben rechts auf das (LUPE)-Symbol.
Im setup des Programms wird der Monitor mit dem Befehl [`Serial.begin(9600)`] initialisiert (Zeile 4). „9600" steht für die *Baud*-Übertragungsrate (hier in bit/sec), mit welcher der ARDUINO die Daten über die serielle Leitung an den Computer sendet und muss mit der auf dem Monitor (unten rechts im Monitorfenster) eingestellten übereinstimmen – sonst „verstehen" sich Computer und ARDUINO nicht.
Das Programm in Abb. 8.2.8 ist wieder auf demjenigen von Abb. 8.2.7 aufgebaut, einfach mit einer weiteren Subroutine [`subschreiben`] (Zeilen 24ff) und weiteren Variablen, welche definiert werden müssen, siehe auch Kommentar.
Damit nur bei einer *Änderung* des ADC-Wertes geschrieben wird, ist folgende Bedingung gesetzt (Zeile 27):

```
[ if (sensorwert > (sw0 + 1) || sensorwert < (sw0 - 1))) { ]
```

wobei [`||`] dem logischen ODER entspricht

Der Wert wird daher nur geschrieben, wenn er sich um +/- 2 geändert hat.
Die Befehle [`Serial.print( )`] dienen zum Schreiben von Strings oder Zahlen hintereinander in eine Zeile des seriellen Monitors (Zeilen 32ff).
Konkret wird die vom ADC ausgegebene Zahl (zwischen 0 und 1024) und die daraus umgerechnete Spannung über R3 dargestellt.
Mit dem Befehl [`Serial.println( )`] in Befehlszeile 37 wird *nach* dem Schreiben ein *line-feed*, also ein Zeilenabsatz im Monitor erzeugt und die folgenden Daten in eine neue Zeile geschrieben.

```
1   //6-LED Lauflicht
2
3   void setup() {
4     Serial.begin(9600);    //seriellen Monitor initialisieren 9600 baud
5     //digiale Pins 0 bis 13 auf Output
6     for (int pinnr = 0; pinnr <= 13; pinnr++) {
7       pinMode(pinnr, OUTPUT);
8     }
9   }
10                                  //div. Variablen setzen
11  int sw0 = 0;                    //vorheriger Sensorwert
12  int sensorwert; float sensorwertf; float spannung;
13  int dauer;  int Delay;
14
15  //Hauptroutine
16  void loop() {
17    sensorwert = analogRead(A0);  //Einlesen LDR mit ADC Analog-Eingang A0
18    subschreiben();               //Subroutine schreiben (serieller Monit.)
19    substeuerung();               //Subroutine Ansteuerung 6 LED
20  }
21
22  /* Subroutine Sensorwert und Spannung in
23  seriellen Monitor schreiben falls Aenderung */
24  void subschreiben() {
25    /*Bedingung: schreiben falls Sensorwert
26    geändert um mehr als +/- 1  */
27    if (sensorwert > (sw0 + 1) || sensorwert < (sw0 - 1)) {
28      sensorwertf = sensorwert;   //Sensorw. in Fliesskomma-Variable verw.
29                                  //Spannung berechnen aus ADC-Wert
30      int spannung = sensorwertf / 1024 * 4890;
31                                  //in seriellen Monitor schreiben
32      Serial.print("LDR-Sensorwert = ");
33      Serial.print(sensorwert);
34      Serial.print("/1024    ");
35      Serial.print("Spannung = ");
36      Serial.print(spannung);
37      Serial.println("  mV");
38    }
39    sw0 = sensorwert;             // neuen Sensor-Vergleichswert sw0 setzen
40  }
41
42  //Subroutine Ansteuerung 6 LEDs
43  void substeuerung() {
44    dauer = sensorwert / 10;
45    Delay = sensorwert / 20;
46    for (int pinnr = 12; pinnr >= 7; pinnr--) {
47      digitalWrite(pinnr, HIGH);
48      delay(dauer);
49      digitalWrite(pinnr, LOW);
50      delay(Delay);
51    }
52  }
```

Abb. 8.2.8 Mit LDR gesteuertes 6 LED-Lauflicht und Möglichkeit, Daten in den Seriellen Monitor zu schreiben

Notizen

8.3 Temperatur messen

Die sehr preiswerte Temperatur-Sonde TMP-36 von *Analog-Devices* sieht aus wie ein Transistor – ist aber keiner.

Pin 1: 5 V
Pin 2: Signalausgang
Pin 3: Masse (GND)

Abb. 8.3.1 Temperatursonde TMP 36

Sie lässt sich direkt an den ARDUINO anschliessen, Pin 2 an Analog-Input A1. Die Schaltung mit den LED können Sie stehen lassen.

Wichtigste Daten	
Anschlussspannung:	2.7 V bis 5.5 V
Spannung an Pin 2 bei 25°C:	750 mV
Skalierungsfaktor:	10 mV/°C

Das Programm baut wieder auf den vorangegangenen auf. Man muss nun mit obigen Parametern die Temperatur als Funktion der Spannung ausrechnen.

Aufgabe 8.3.1 Zeigen Sie, dass die Funktion lautet $T(U) = 0.10 \cdot U - 50$, falls die Spannung in mV gemessen wird. Die Spannung wird mit dem Ausgabewert des ADC ausgerechnet. Im Programm aus Abb. 8.3.2 wird mit 10 bit = 1024 = 4890 mV gerechnet, Sie sollten aber die Spannung des ARDUINO nachmessen und gegebenenfalls den Wert im Programm korrigieren.

In der loop-Schleife wird das Programm zunächst in die Subroutine [`submeanT()`] geschickt (Zeile 38), dort wird 10'000 mal (dies können Sie mit der Variablen [`int Stopp`] in Zeile 4 einstellen) über den ADC-Wert gemittelt und das Mittel in der Variablen [`meanT`] ausgegeben.
Dann springt das Programm in die Hauptschleife zurück und wird sofort in die Subroutine [`subschreib()`] geschickt (Zeilen 18ff). Dort steht wieder eine Bedingung: Es wird nur geschrieben, wenn [`meanT`] sich mehr als 0.1 ändert. Wenn Sie Nachrechnen, sehen Sie, das dies einer Temperaturänderung von 0.05 °C entspricht.
Anschliessend (Zeilen 24ff) folgt eine kleine Übung, um auf genau Zehntelgrad darzustellen (weil ein Befehl fehlt, um Nachkommastellen abzuschneiden) und schliesslich wird in den Monitor geschrieben.

```
1    //Variablen definieren
2    float tempC = 0; float senswT; float meanT0 = 0;
3    float senswTf; float meanT;
4    int stopp = 10000; int k;
5
6    //Setup
7    void setup() {
8      Serial.begin(9600);
9    }
10
11   //Hauptroutine
12   void loop() {
13     submeanT();
14     subschreib();
15   }
16
```

```
17   //Subroutine in Monitor schreiben
18   void subschreib() {
19     //nur schreiben, wenn um 0.05°C geändert
20     if (meanT > (meanT0 + 0.1) || meanT < (meanT0 - 0.1)) {
21       /*Fühler TMP 36 mit 750mV bei 25°C und Rate 0.10°C/mV
22       ergibt Funktion tempC = mx + b = 0.1 * U[mV] - 50 */
23       tempC = 0.1 * (meanT / 1024 * 4890) - 50;
24       int tempC1 = tempC * 10;  //auf Zehntelgrad genau schreiben
25       tempC = tempC1;
26       tempC = tempC / 10;
27       Serial.print("Sensorwert = ");
28       Serial.print(meanT);
29       Serial.print("    ");
30       Serial.print("Temperatur = ");
31       Serial.print(tempC);
32       Serial.println("  Grad C");
33     }
34     meanT0 = meanT;
35   }
36
37   //Subroutine Sensorwert einlesen und mitteln
38   void submeanT() {
39     for (k = 1; k <= stopp; k++) {
40       senswT = senswT + analogRead(1);
41     }
42     meanT = senswT / (k - 1);
43     senswT = 0;
44   }
```

Abb. 8.3.2 Temperaturwerte von der Sonde TMP 36 ermitteln und auslesen

8.4 Zusatz: Datenübertragung auf PROCESSING

Der ARDUINO wurde eigentlich geschaffen, um als einfacher und autonom operierender Prozessor Dinge zu steuern und zu messen. Sein Befehlssatz und seine Möglichkeiten, auf einem Monitor Dinge darzustellen sind dementsprechend etwas beschränkt. So ist der Arduino-Monitor eher für Kontrollzwecke bei der Programmentwicklung gedacht. Mit Hilfe des Programms PROCESSING lassen sich nun die Möglichkeiten der Darstellung und der Datenverarbeitung bedeutend erweitern. Allerdings dient dann der Arduino eher als Interface, denn als selbständiger Rechner. Die Programmiersprache ist sehr ähnlich derjenigen von Arduino.

```
void setup() {
Serial.begin(115200);
}

void loop() {
Serial.println(analogRead(A0));
delay(50);
}
```

Abb. 8.4.1 Arduino-Sketch zum Auslesen von A0

Wir zeigen an einem Beispiel noch die Prinzipien. Zunächst der Arduino-Sketch:
Wie oben erwähnt, gestaltet sich dieser sehr kurz; im Wesentlichen geht es nur darum, den ADC auszulesen. Wir gehen davon aus, dass an A0 immer noch der LDR aus Kapitel 8.2.4 angeschlossen ist. Zur Abwechslung schreiben wir einmal mit der Höchstgeschwindigkeit von 115200 Baud in die serielle Leitung, natürlich überhaupt nicht nötig bei einem delay von 50 msec. Der Schreibbefehl ist etwas modifiziert, mit [`Serial.println()`] werden *Strings*, also Ketten aus Buchstaben (ASCI-Code) mit dem *Zeilenabsatz*-Befehl in die Leitung geschrieben.
Es folgt nun, ohne viel Kommetar das PROCESSING-Programm. Im Wesentlichen wird darin der String ausgelesen, in eine Zahl verwandelt und dann in ein Fenster geschrieben. Achtung, das Programm ist für Port COM3 geschrieben, dies müssen Sie gegebenenfalls abändern. Zusätzlich werden unten links im Monitorfenster die von Arduino gesendeten Strings gezählt. Zentral dabei ist, dass bei jedem Absatz (linefeed = 10), Zeile 51, mit Lesen aufgehört und auf den nächsten String gewartet wird.

PROCESSING-Sketch *Abb.* 8.4.2

```
1     Strings von Arduino in Processing
2     schreiben und in Monitor anzeigen,
3     Variante 20
4      2016-02-11  */
5     import processing.serial.*;
6
7     //Setup
8     void setup() {
9       subbildformat();
10      size(1000, 300);
11      //Port initialisieren ACHTUNG !! welcher ?!?!
12      COM=new Serial(this, "COM3", 115200);
13    }
14
15    //Variablen einlesen
16    Serial COM;
17    int linefeed = 10; int ardubyte = 0; //ASCI linefeed = 10
18    String ardustring; float ardustringfl;
19    int arduint; int i=0; int j=0;
20    int ypostext=40; int xpostext=40;
21    boolean flag1=true;
22
23
24
25    //Hauptschleife
26    void draw() {
27        subtaste();
28        sublesen();
29        subzuruecksetzen();
30      }
31
32    //Subroutine: Bild- und Textformat
33    void subbildformat() {
34      colorMode(HSB, 255);
35      fill(0, 0, 0);
36    }
37
38    //Subroutine: Taste einlesen: stop oder weiter
39    void subtaste() {
40      if (key=='s') {
41        flag1=false;
42      }
43      if (key=='w') {
44        flag1=true;
45      }
46    }
47
48    //Subroutine:  lesen falls etwas im COM
49    void sublesen() {
50      if (COM.available()>0) {
51        ardustring =COM.readStringUntil(linefeed);
52        //einlesen bis linefeed-Zeichen (ASCI=10) kommt
53        if (flag1==true) {
54          //schreiben, falls String nicht gleich 0
55          if (ardustring !=null) {
56            ardustringfl=float(ardustring); //in float verw.
57            arduint=int(ardustringfl);  // in Integer verw.
58            ardubyte=int(arduint/4); //auf ein byte skalieren
59            subschreiben();
60            j++;
61          }
62        }
63      }
64    }
65
```

```
66    //Subroutine: in Monitor schreiben
67    void subschreiben() {
68      background(ardubyte, 255, 255);
69      textSize(40);
70      text (j, xpostext+20, ypostext+220);
71      textSize(100);
72      text("ADC-Wert = ",xpostext+20,ypostext+120);
73      text(arduint, xpostext+700, ypostext+120);
74    }
75
76    //Subroutine: Zurücksetzen falls über 10^6
77    void subzuruecksetzen() {
78      i++;
79      if (i>10000000) {
80        i=0;
81        j=0;
82      }
83    }
```

Mit den Befehlen [colorMode(HSB, 255)] in Zeile 34 und [background(ardubyte, 255, 255] in Zeile 68 wird die Hintergrundfarbe des Monitors entsprechend dem ADC-Wert laufend verändert – sieht lustig aus. Sie können ja noch die Hintergrundfarbe für die Temperaturmessung entsprechend „designen".

8.5 Motoren und Servos

8.5.1 Motorsteuerung mit Pulsweitenmodulation PWM

Gewöhnlich wird ein Elektromotor an eine Spannungsquelle angeschlossen und mit Veränderung der Spannung die Drehzahl variiert. Ganz anders funktioniert die Ansteuerung mit PWM (Pulse-Width-Modulation). Hierbei gibt der ARDUINO Rechteckimpulse *fester* Frequenz aus, welche auch eine *konstante* Spannung von 5 V aufweisen. Hingegen ist die *Breite* der Rechteckimpulse veränderlich. Die beiden Extremfälle sind Breite 0 und volle Breite, d.h. einfach eine konstante Spannung von 5 Volt. Typisch für PWM ist das Summen der Motoren, wenn sich die Frequenz im Hörbereich befindet.

Aufgabe 8.5.1
a) Bauen Sie die Schaltung aus Abb. 8.5.1. Der Motor selber kann mit einem MOSFET IRF 530 angesteuert werden. Separate Speisung von 5 V bis 10 V. Details siehe Schaltplan. Sehr wichtig ist die *antiparallel* geschaltete Diode über dem Motor. Die Rechteckimpulse können bei der *abfallenden* Flanke in den Spulen des Motors hohe Induktionsspannungen erzeugen (erklären Sie das!) – diese müssen unbedingt mit der Diode kurzgeschlossen werden, sonst kann der IRF 530 oder sogar der ARDUINO Schaden nehmen! Auf der Vorderseite des ARDUINO wird die Spannung am 10kΩ-Poti in den Analogeingang A4 eingelesen. Wenn Sie genügend Platz haben, können Sie also die andern Schaltungen auf dem Breadboard stehen lassen.
b) Schliessen Sie den KO an Port D3 des ARDUINO an und beobachten Sie, wie sich mit Drehen des Potis die Pulsweite verändert!

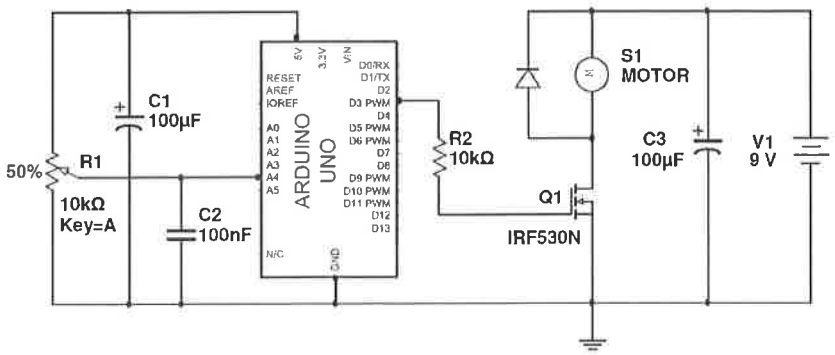

Abb. 8.5.1 mit ARDUINO pulsweitengesteuerter Motor

Mit der PWM-Steuerung können Sie den Motor äusserst *sanft* anfahren - ModellbauerInnen schätzen das sehr!

```
1     //Motor PWM_11
2     //Variablen definieren/einlesen
3     int potPin = A4; // Analogin Motor
4     int potVal;  int potValneu;  //Motor
5     int analogPin = 3; // Analogout Motor
6     int PWMfwert = 4; //(f = 488 Hz) Motor
7
8     //Motor anschliessen und RS232
9     void setup() {
10      pinMode(analogPin, OUTPUT); //Motor
11      Serial.begin(9600);
12      //PWM Frequenz einstellen
13      TCCR2B = (TCCR2B & 0xF8) | PWMfwert;
14    }
15
16    void loop() {
17      potVal = analogRead(potPin); //Einlesen Motor
18      //nur schreiben wenn Wert +/- 1 verändert
19      if (potVal < potValneu - 2 ||
20          potVal > potValneu + 2) {
21        Serial.print("PotiWert = ");
22        Serial.println(potVal);
23      }
24      potValneu = potVal; // Poti Referenzwert neu
25      analogWrite(analogPin, potVal / 4); //Puls an Motor
26      delay(2);
27     }
```

Abb. 8.5.2 ARDUINO Sketch zur PWM-Steuerung eines Elektromotors

In Zeilen 6/13 kann die PWM-Frequenz eingestellt werden (Timer 2, Port 3 und 11)

Wert	Frequenz [Hz]	Wert	Frequenz [Hz]
1	31372	5	245
2	3921	6	122
3	976	7	31
4	488		

Randbemerkung: Indem man PWM mit einer sehr hohen Frequenz - *oversampling* beispielsweise 44.1 kHz - durchführt, kann man digitale Musiksignale mit einem anschliessenden Tiefpassfilter wieder „analogisieren". Neuerdings werden auch Leistungsverstärker (Klasse D-Verstärker) zunehmend mit dieser Methode gebaut. Der grosse Vorteil ist, dass man in diesem Fall die Leistungstransistoren nur als Schalter braucht und nicht an einem Arbeitspunkt betreibt (siehe Kapitel 4.4), mit entsprechend deutlich weniger Wärmeentwicklung.

8.5.2 Steuerung eines Servos

Mit Servos können mechanische Bewegungen erzeugt werden. Im Modellbau werden diese Stellmotoren beispielsweise dazu verwendet, um bei Flugzeugen das Höhen- oder das Seitenruder zu bewegen. Einige weitere von unzähligen Möglichkeiten sind der Einschlag einer Auto-Lenkung, die Bewegung eines Roboterarms. Bei dieser Bewegung kann der *Drehwinkel* (meistens in einem Bereich von 0° bis 180°) sehr genau eingestellt werden. Servos enthalten intern Elektronik, welche den Soll-Drehwinkel des Eingangssignals mit der aktuellen Position (Ist-Wert) vergleicht und ihn auf den vorgegebenen einregelt. Die Erfassung des Drehwinkels erfolgt mit einem Potentiometer (je nach Qualität eine Quelle von Unregelmässigkeiten).
Die Signale werden mittels PWM, meist mit einer Frequenz von 50 Hz, übermittelt und zwar dergestalt, dass für den linken Anschlag eine Pulsweite von 1 msec, Mittelposition 1.5 msec und den rechten Anschlag 2 msec erforderlich sind.

Servos haben drei Anschlüsse; für den EMAX ES08A gilt

rot	Versorgungsspannung	4.8 bis 6.0 Volt
braun	Masse GND	
gelb	Signaleingang für PWM-Ansteuerung (bezogen auf GND)	

Achtung: Die Anschlussbelegung ist bei unterschiedlichen Servos *nicht* einheitlich.

Aufgabe 8.5.2
Bauen Sie die Schaltung gemäss Abb. 8.5.3. Sie können den Servo direkt am ARDUINO anschliessen. Hängen Sie an den PWM-Output D9 auch noch den KO zwecks Beobachtung der PWM-Signale.

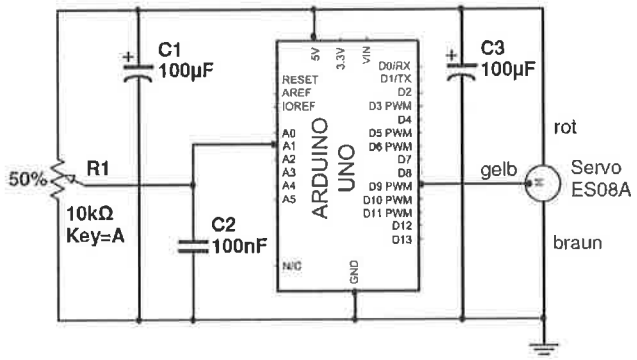

Abb. 8.5.3 Ansteuerung eines Sevos

Die Kondensatoren dienen zur Eliminierung unerwünschter Störungen (Rechtecksignale gemeinsam mit Spulen sind grosse Produzenten derjenigen).
Ein interessanter Befehl im Programm steht in Zeile 19: [`winkel = map(potValS, 0, 1000, 0, 175)`]. Mit dessen Hilfe wird der Wertebereich von 0 bis 1000 des ADC auf einen Bereich von 0 bis 175, Winkelbereich des Servos, skaliert.

```
1   //Bibliothek Servo
2   #include <Servo.h>
3   //Variablen definieren/einlesen
4   Servo Servo1; // Servo EMAX ES08A
5   int potPinS = A1; // Analogin Servo
6   int potValS;  int potValneuS; //Servo
7   int winkel;
8
9   //Servo anschliessen und RS232
10  void setup() {
11    Servo1.attach(9); //Analogout Servo Pin 9
12    Serial.begin(9600);
13  }
14
15  void loop() {
16    potValS = analogRead(potPinS); //Einlesen Servo
17    if (potValS < 1000) {
18      //Winkel auf 0 bis 175° mappen und Servo ansteuern
19      winkel = map(potValS, 0, 1000, 0, 175);
20      Servo1.write(winkel);
21    }
22    //nur schreiben wenn Wert +/- 1 verändert
23    if (potValS < 1000 && (potValS < potValneuS - 1 ||
24                           potValS > potValneuS + 1)) {
25      Serial.print("PotiWert = ");
26      Serial.print(potValS);
27      Serial.print("    Winkel = ");
28      Serial.println(winkel);
29    }
30    potValneuS = potValS; // Poti Referenzwert neu
31    delay(2);
32  }
```

Abb. 8.5.4 ARDUINO-Sketch zur Ansteuerung eines Sevos

8.6 Töne

ARDUINO kann Rechteckschwingungen beliebiger Frequenz erzeugen und an einen der Digital-Pins ausgeben. Mit einer aufwendigeren Beschaltung ist es (natürlich!!) auch möglich, einen digitalen *Sinusgenerator* zu bauen.
Zum Schluss entwerfen wir ein sehr primitives digitales *Theremin*. Das „richtige" Theremin ist ein physikalisch faszinierender Generator phantastischer Klänge, der eigentliche Vorläufer des Synthesizers. Dieses erste elektronische Musikinstrument wurde um 1920 von dem in die USA emigrierten Russen Lew Thermen (LEON THEREMIN) erfunden. Hören Sie, wie Theremin das Instrument spielt[15] oder Peter Pringle „Over The Rainbow"[16] – es ist der melancholische (Ab?)Gesang einer analogen Welt... ROBERT MOOG, genialer Konstrukteur des Moog-Synthesizers, hat in seinen jungen Jahren Theremine gebaut, bevor dann seine Instrumente in den Sechzigerjahren noch nie gehörte Klänge im musikalischen Universum auftauchen liessen. Selbstverständlich ist ein „richtiges" Theremin bereits Thema der ARDUINO-Gemeinde.[17] Manchmal kann man den Sound des „singenden Säge" auch in Science-Fiction-Filmen oder Horrorstreifen wahrnehmen.
Es geht bei diesem Projekt darum, dass ein Ton, bzw. ein Klang erzeugt wird, dessen Höhe von der Beleuchtung eines LDR abhängt.

Aufgabe 8.6.1
Bauen Sie die Schaltung gemäss Abb. 8.6.1. Vor dem ARDUINO befindet sich wieder eine Spannungsteilerschaltung mit einem LDR. Am Digital-Pin D8 gibt ARDUINO Rechteckschwingungen variabler Frequenz aus, LOW = 0 V, HIGH = 5 V. Beobachten Sie das Signal an dieser Stelle mit dem KO. Mit Hilfe des Elektrolytkondensators C4, dem Kopplungskondensator, wird ein allfälliger Gleichspannungsanteil weggefiltert. Anschliessend folgt eine Verstärkerschaltung mit dem 5-Watt Verstärker LM384 von *Texas Instruments*. Dieser ist auf eine fixe Verstärkung von +34 dB ausgelegt, würde also durch das Eingangssignal hoffnungslos übersteuert. Die Spannungsteiler-Schaltung aus R6 und R7 dämpft das Signal um 22 dB. Mit dem anschliessenden 10kΩ-Poti kann die Lautstärke reguliert werden, dabei beträgt die Verstärkung höchstens noch +34 dB – 22 dB = +12 dB. Bezogen auf die Eingangsspannung (peak-to-peak) von 5 V ergibt das eine maximale Ausgangsspannung (peak-to-peak) von 19 V. Falls Sie nicht mehr genau wissen, wie man mit dB-Pegeln rechnet, schauen Sie's in PAM Akustik, Kapitel 6.2 nach. Der LM 384 selber wird durch eine separate Versorgungsspannung von ca. 20 V gespeist und hat einen *Ausgangs-Ruhepegel* von der Hälfte der Versorgungsspannung; der Lautsprecher muss daher mit dem Kopplungskondensator C3 in der Gegend von 500µF bis 1000µF angeschlossen werden – auf die Polarität achten!

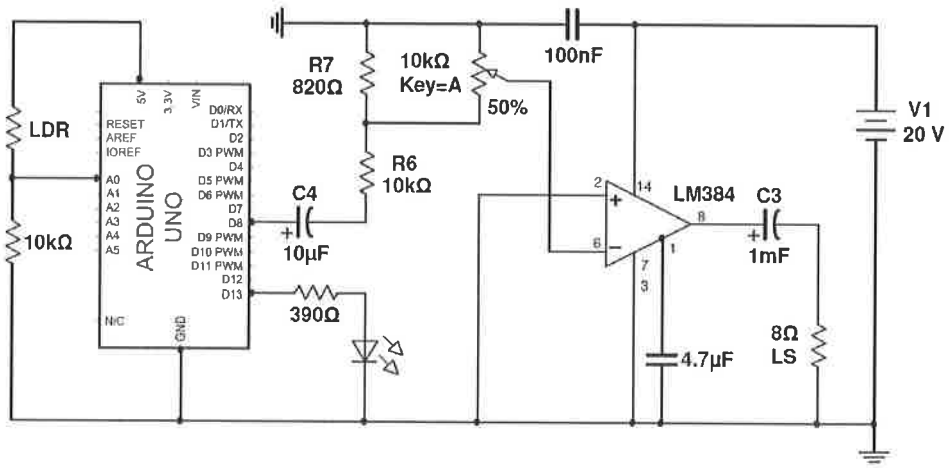

Abb. 8.6.1 „Theremin" mit nachgeschaltetem 5 Watt-Verstärker

Wichtige Bemerkung: Beim LM384-Käfer (DIL-14-Gehäuse, siehe Abb.7.2.10) sind die Pins 3,4,5,7,10,11 und 12 Masseanschlüsse (GND); in Ihrer Schaltung müssen Pin 7 und *mindestens* einer der anderen mit der Masse verbunden sein!

[15] https://www.youtube.com/watch?v=w5qf9O6c20o
[16] https://www.youtube.com/watch?v=K6KbEnGnymk
[17] www.gaudi.ch/Open**Theremin**/index.php?

```
1     //Variablen setzen/einlesen
2     int sensorWert; int pitch;
3     int sensorLow = 1023; int sensorHigh = 0;
4     const int LEDPin = 13; //LED Anzeige
5     // untere und obere Frequenz einstellen
6     int f_unten = 440; int f_oben = 880;
8     void setup() {
9       Serial.begin(9600);
10      pinMode(LEDPin, OUTPUT);
11      digitalWrite(LEDPin, HIGH); /*LED leuchtet und...
12      waehrend 5 sec von dunkel bis hell messen aus ADC*/
13      while (millis() < 5000) {
14        sensorWert = analogRead(A0);
15        if (sensorWert > sensorHigh) {
16          sensorHigh = sensorWert; // hellster Wert
17        }
18        if (sensorWert < sensorLow) {
19          sensorLow = sensorWert; // dunkelster Wert
20        }
21      }
22      digitalWrite(LEDPin, LOW); // LED abschalten
23      Serial.print("Oberer Wert   "); // in ser. Monitor
24      Serial.print(sensorHigh);
25      Serial.print("    Unterer Wert   ");
26      Serial.println(sensorLow);
27    }
29    void loop() {
30      sensorWert = analogRead(A0); //aktueller Wert einl. von ADC
31      // skalieren von unterer bis oberer Frequenz
32      pitch = map(sensorWert, sensorLow, sensorHigh, f_unten, f_oben);
33      //abschneiden bei unterer und oberer Frequenz
34      pitch = constrain(pitch, f_unten, f_oben);
35      tone(8, pitch);  //Ton auf Output 8
36      delay(10);
37    }
```

Abb. 8.6.2 ARDUINO-Sketch zum "Theremin"

Sie erkennen im Sketch aus Abb. 8.6.2 einen ungewöhnlich langen setup-Teil. Dort durchläuft das Programm während 5 Sekunden die Schleife [`while (millis() < 5000) { `], Zeilen 13ff, während der die LED an Pin D13 eingeschaltet ist. Vor allem aber werden in dieser Schleife der höchste und tiefste Wert des Spannungsteilers an Input A0 in die Variablen [`sensorHigh`] und [`sensorLow`] eingelesen. Dazu müssen Sie den LDR innerhalb der 5 Sekunden einmal *abdecken* und ein anderes Mal *hell beleuchten*, Reihenfolge spielt keine Rolle. Die eingelesenen Werte können im seriellen Monitor betrachtet werden.
In der Hauptschleife wird dann mit dem Skalierungsbefehl in Zeile 32 [`pitch = map(sensorWert, sensorLow, sensorHigh, f_unten, f_oben)`] der *aktuelle* Wert in das vorher gewählte Frequenzband eingeteilt. Im Beispiel beträgt die untere Frequenz [`f_unten`] = 440 Hz und die obere 880 Hz; diese können in Zeile 6 festgelegt werden. Mit dem Befehl [`pitch = constrain(pitch, f_unten, f_oben)`] in Zeile 34 werden die Frequenzgrenzen absolut limitiert. Mit anderen Worten: es werden *keine* Frequenzen tiefer als 440 Hz und *keine* höher als 880 Hz abgestrahlt, auch wenn der LDR aktuell weniger oder stärker beleuchtet wird als das bei der „Eichung" während der 5 Sekunden zu Beginn geschehen ist. Diesen Befehl können Sie auch streichen. Wenn Sie neu Eichen möchten, drücken Sie die RESET-Taste am ARDUINO.

Tonhöhen: **a** = 220 Hz; **b** = 233.1 Hz (Halbton 16 : 15); **h** = 246.9 Hz (Ganzton 9 : 8);
c' = 261.6 Hz (kleine Terz 6 : 5); **cis'** = 277.2 Hz (grosse Terz 5 : 4); **d'** = 293.7 Hz (Quarte 4 : 3);
e' = 329.6 Hz (Quinte 3 : 2); **f'** = 349.2 Hz (kl. Sexte 8 : 5); **fis'** = 370.0 Hz (gr. Sexte 5 : 3) ;
g' = 392.0 Hz (kl. Septime 16 : 9); **gis'** = 415.3 Hz (gr. Septime 15 : 8); **a'** = 440 Hz (Oktave 2 : 1) ;

- Halbtonschritt-Faktor = $\sqrt[12]{2} \approx 1.05946$ für die chromatische Tonleiter,
- ganzzahlige Verhältnisse entsprechen den reinen Intervallen der pythagoreischen Stimmung