Sensoren und Messwerterfassung mit dem Arduino

Fortbildung Sept. 2023 – Praxisteil

Michael Himpel
Universität Greifswald
https://physik.uni-greifswald.de/lehramt-exphy/
himpel@physik.uni-greifswald.de

Inhaltsverzeichnis

TEIL

I

Grundlagenversuche

Die hier gezeigten Anwendungen für den Arduino stammen maßgeblich aus den offiziellen Beispielen. Diese sind sehr übersichtlich und sollten die erste Anlaufstelle für Projektideen sein. Die Beispiele sind abrufbar unter

https://docs.arduino.cc/built-in-examples/

Abschnitt 1

LED zum Blinken bringen

Abschnitt 1.1

. . . integrierte LED

Der Arduino Uni besitzt eine auf dem Board integrierte LED. Dort ist der korrekte Vorwiderstand schon vorhanden und man muss die LED nur über den entsprechenden Pin ansteuern. Welcher das ist, hängt vom konkret genutzten Arduino ab. Die IDE kennt aber die Modelle und unter der Konstante `LED_BUILTIN` ist der richtige Pin bereits bekannt[1]. In Schaltplan 1 ist ansonsten der Anschluss einer externen LED mit Vorwiderstand gezeigt (ebenfalls an Pin 13). Der folgende Code lässt sich also sowohl mit interner als auch mit externer LED ausprobieren.

[1] *Bei den meisten Arduinos ist das der D13-Pin*

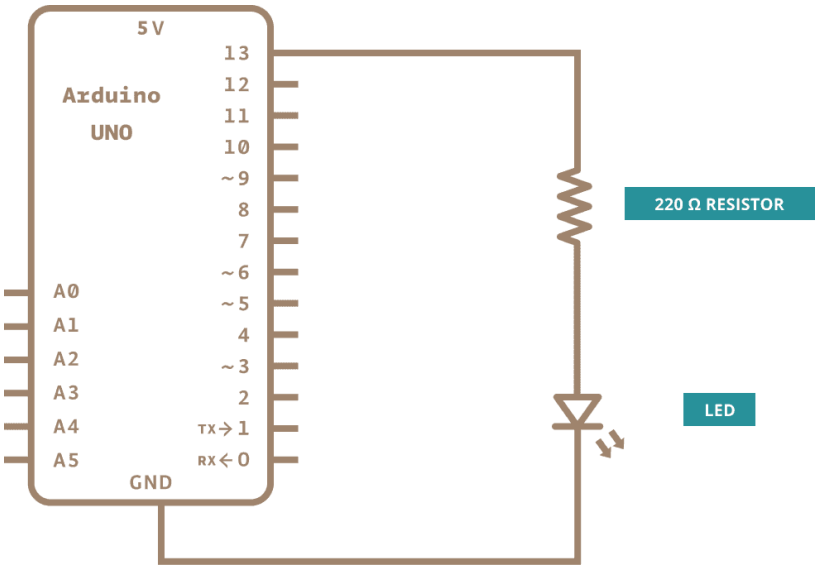


Abb. 1. Schaltplan zum Anschluss einer LED mit Vorwiderstand[1]

Abschnitt 1.2

. . . mit delay()

Wenn man in der Hauptfunktion void `loop(){}` die LED ein- und wieder ausschaltet, hat man bereits eine blinkende LED realisiert da ja die Schleife ständig wiederholt wird. Allerdings geht das viel zu schnell um dies mit dem Auge wahrzunehmen – die LED scheint ständig zu leuchten. Um also tatsächlich den Eindruck einer blinkenden LED zu bekommen muss man längere Pausen zwischen AN und AUS einfügen. Im folgenden Beispiel[1] sind das jeweils 1 s bzw. 1000 ms

```
/*
  Blink

  Turns an LED on for one second, then off for one second,
  repeatedly.

  Most Arduinos have an on-board LED you can control.
  On the UNO, MEGA and ZERO it is attached to digital pin 13,
  on MKR1000 on pin 6. LED_BUILTIN is set to
  the correct LED pin independent of which board is used.
  If you want to know what pin the on-board LED is connected
  to on your Arduino
  model, check the Technical Specs of your board at:
  https://www.arduino.cc/en/Main/Products

  modified 8 May 2014
  by Scott Fitzgerald
  modified 2 Sep 2016
  by Arturo Guadalupi
  modified 8 Sep 2016
  by Colby Newman

  This example code is in the public domain.

  https://www.arduino.cc/en/Tutorial/BuiltInExamples/Blink
*/

// the setup function runs once when you press
// reset or power the board
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_BUILTIN, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED_BUILTIN, HIGH);
  // turn the LED on (HIGH is the voltage level)
  delay(1000);
  // wait for a second
  digitalWrite(LED_BUILTIN, LOW);
  // turn the LED off by making the voltage LOW
  delay(1000);
  // wait for a second
```

```
}
```

Abschnitt 1.3

. . . mit Schleifen

Als Übung für die Strukturen in C++ kann man die das Blink-Beispiel entsprechend abändern. Es soll zum Beispiel verboten sein die delay-Funktion zu benutzen. Stattdessen könnte an auch in einer for-Schleife eine beliebige Rechenoperation oft durchführen um etwas Rechenzeit zu verbrauchen:

```
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_BUILTIN, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED_BUILTIN, HIGH);
  // turn the LED on (HIGH is the voltage level)
  int a = 1;
  int b = 2;
  for(int i=0; i<5000; i++){
      b = a+b;
  }
  // add a 5000 times on b to gain some time...

  digitalWrite(LED_BUILTIN, LOW);
  // turn the LED off by making the voltage LOW
  for(int i=0; i<5000; i++){
      b = a+b;
  }
  // again some time...
}
```

Hier wurde jetzt der gleiche Code zweimal verwendet um Zeit zu verbrauchen. Da bietet sich auch direkt eine Funktion an[2]. An den Anfang des Quellcodes stellt man die Funktionsdefinition:

[2] *Und man hat gleich noch eine C++-Struktur ausprobiert. . .*

```
void wasteTime(){
  int a = 1;
  int b = 1;
  for(int i=0; i<5000; i++){
      b = a+b;
  }
}
```

Später kann man dann durch das Aufrufen von `wasteTime();` die Verzögerung erreichen. Vielleicht liegt es nun auch nahe, den Zeitverzug zu steuern. Dadurch könnte man die Anzahl der Rechenschritte als Parameter an die Funktion übergeben. Dafür muss die Funktion nun etwas anders definiert werden:

```
void wasteTime(int wasteSteps){
  int a = 1;
  int b = 1;
```

```
  for(int i=0; i<wasteSteps; i++){
      b = a+b;
  }
}
```

Jetzt kann man durch Übergabe der Schrittzahl `wasteTime(10000);` nun die Länge der Zeitverschwendung steuern.

Abschnitt 1.4

. . . mit einem Potentiometer

Da wir nun schonmal eine variable Blinkfrequenz ins Programm eingebaut haben – entweder über `delay(...)` oder über `wasteTime(...)` – möchte man diese Frequenz vielleicht auch gern als Benutzer während des Betriebs ändern. Dafür kann man leicht ein Potentiometer nutzen. Dieses schließt man als Spannungsteiler2 an und liest die einstellbare Spannung am Analogeingang aus. Dadurch erhält man ein Integer vom Wert 0 bis 1024. Dieser Wert kann nun noch nach Bedarf vervielfältigt werden[3] um in einen sinnvollen Einstellbereich zu kommen. Mit dem folgenden Codeschnipsel[2] kann

[3] *vielleicht *10? Vielleicht quadrieren? Mal die Wucht der Exponentialfunktion kennenlernen?*

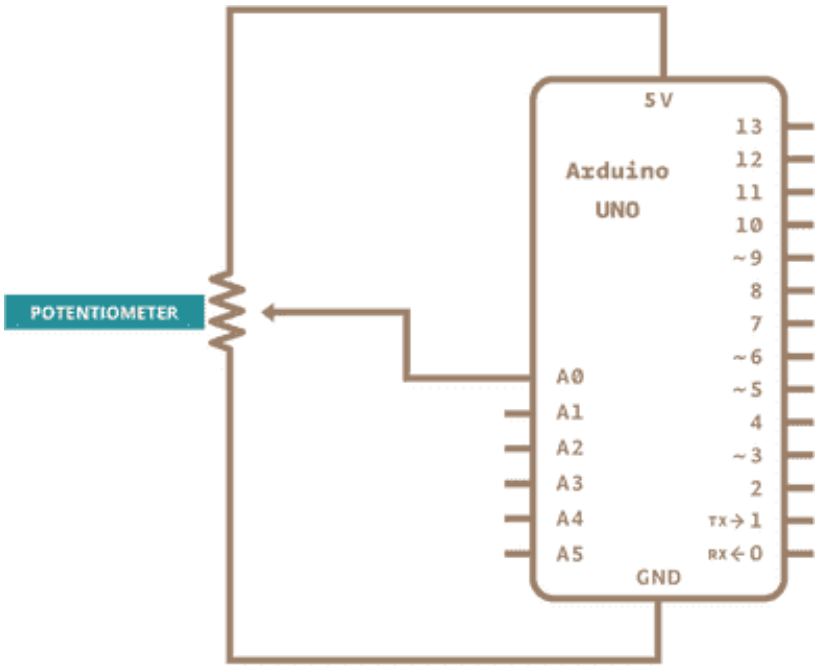


Abb. 2. Schaltplan zum Anschluss eines Potentiometers als Spannungsteiler[2]

man an den Wert kommen:

```
// the loop routine runs over and over again forever:
void loop() {
  // read the input on analog pin 0:
  int sensorValue = analogRead(A0);
  delay(1);  // delay in between reads for stability
}
```

Durch die Übergabe von `sensorValue` an die Wartefunktionen kann man nun die Blinkfrequenz der LED per Poti steuern.

Abschnitt 1.5

. . . mit einem Taster

Man kann auch einen Taster verwenden, um die LED ein- oder auszuschalten. Im Beispiel in Abb. 3 schaltet der Taster die 5 V auf den digitalen Pin 2. Es gibt hier aber ein Problem: Falls man nur den Schalter verwendet (und nicht noch den 10k-Widerstand) könnte es sein, dass bei nicht-gedrücktem Schalter ein zufälliger Pegel am Eingang 2 anliegt. In der Praxis flattert der Zustand dann zufällig hin und her. Um fest definierte Zustände zu haben, nutzt man einen sogenannten Pull-Down-Widerstand. Dieser ist so groß gewählt, dass bei

- betätigtem Schalter kein wesentlicher Strom von 5 V zu GND fließt. Es kann sich also leicht das HIGH-Potential am Pin 2 einstellen.
- nicht-betätigtem Schalter fließen eventuelle Ladungen, etwa durch EM-Einstrahlung, gegen GND ab und sorgen dafür, dass an Pin 2 auch wirklich GND-Potential anliegt.

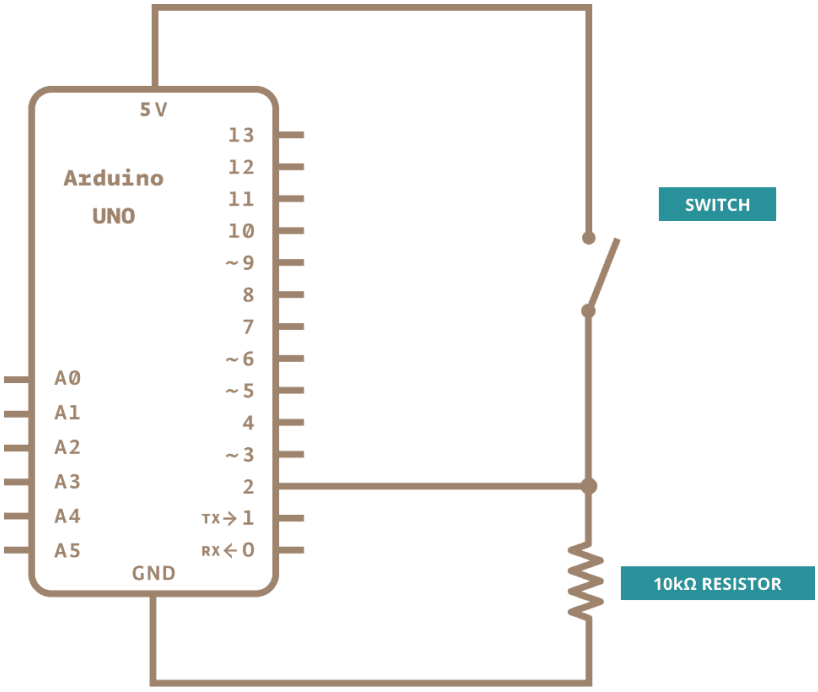


Abb. 3. Schaltplan zum Anschluss eines Tasters[3]

Mit dem folgendem Code [3] kann man den Taster Status (`buttonState`) auslesen und entsprechend die LED ein- oder ausschalten:

```
// constants won't change. They're used here to set pin numbers:
const int buttonPin = 2;   // the number of the pushbutton pin
const int ledPin = 13;     // the number of the LED pin

// variables will change:
int buttonState = 0;  // variable for reading the pushbutton status

void setup() {
```

```
  // initialize the LED pin as an output:
  pinMode(ledPin, OUTPUT);
  // initialize the pushbutton pin as an input:
  pinMode(buttonPin, INPUT);
}

void loop() {
  // read the state of the pushbutton value:
  buttonState = digitalRead(buttonPin);

  // check if the pushbutton is pressed. If it is,
  // the buttonState is HIGH:
  if (buttonState == HIGH) {
    // turn LED on:
    digitalWrite(ledPin, HIGH);
  } else {
    // turn LED off:
    digitalWrite(ledPin, LOW);
  }
}
```

Abschnitt 2

DC-Motor

Der DC-Motor läuft mit einer Betriebsspannung von etwa 9 V. Wenn man direkt eine 9 V-Batterie an den Motor anschließt dreht sich dieser. Die 5V des Arduino sollte man nicht nutzen, da der vom Motor benötigte Strom den Arduino überlastet. Man nutzt nun den Arduino, um den Motor etwas kontrollierter zu steuern. Mit einem Transistor als Schalter kann man gemäß Schaltung 4 die Batteriespannung zuschalten oder trennen. Die Basis des Transistors wird mit dem `anaklogWrite()`-Befehl angesteuert. Dieser erzeugt je nach Zielwert eine PWM-Welle am Ausgang mit variablem Duty-Cycle. Dies bedeutet mal längere und mal kürzere Versorgung des Motors mit 9 V. Die führt letztenendes zu einer regelbaren Geschwindigkeit des Motors, da bei diesem nur die zeitgemittelte Spannung in Bewegung umgesetzt wird. Es gibt aber auch einen minimalen Spannugnswert den der Motor benötigt um überhaupt anzulaufen. Im folgenden Beispielcode [4] wird bei Knopfdruck (`pushButton`) automatisch eine Spannungsrampe rauf- und dann wieder runtergefahren um die variable Geschwindigkeit des Motors zu illustrieren:

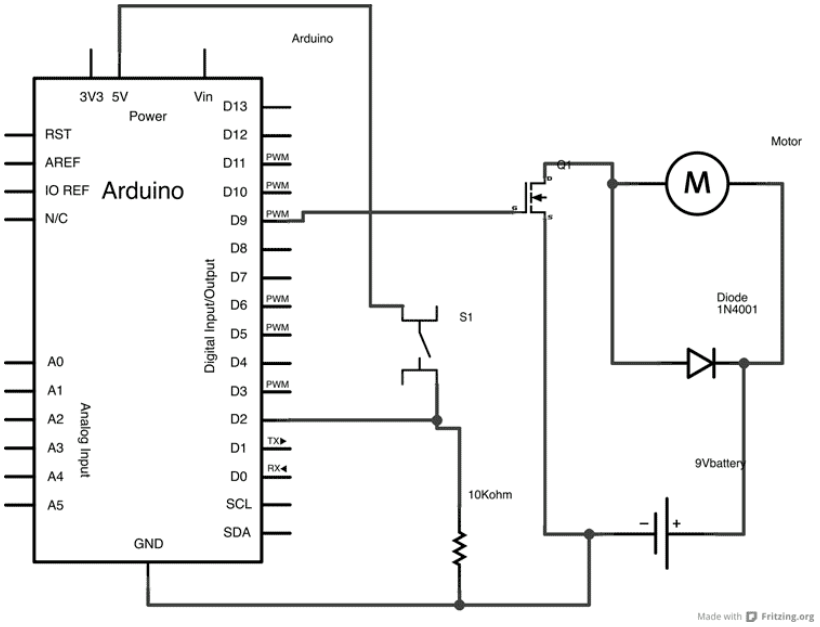


Abb. 4. Schaltplan zum Betrieb eines DC-Motors durch eine 9V-Batterie.[4]

```
/*
 Created on 03 January 2013
 by Scott Fitzgerald
*/

// give a name to digital pin 2, which has a
// pushbutton attached
int pushButton = 2;

// the transistor which controls the motor will
// be attached to digital pin 9
int motorControl = 9;
```

```
// the setup routine runs once when you press reset:
void setup() {
  // make the pushbutton's pin an input:
  pinMode(pushButton, INPUT);

  // make the transistor's pin an output:
  pinMode(motorControl, OUTPUT);
}

// the loop routine runs over and over again forever:
void loop() {

  // read the state of the button and check
  // if it is pressed
  if(digitalRead(pushButton) == HIGH){
    // ramp up the motor speed
    for(int x = 0; x <= 255; x++){
      analogWrite(motorControl, x);
      delay(50);
    }

    // ramp down the motor speed
    for(int x = 255; x >= 0; x--){
      analogWrite(motorControl, x);
      delay(50);
    }
  }
  // delay in between reads for stability:
  delay(1);
}
```

Die Kontrolle der Motorgeschwindigkeit kann aber auch mit einem Potentiometer vorgenommen werden. Dann wird wieder per `analogRead()` der "Wert" des Potis ermittelt und in einen sinnvollen Parameter für die PWM-Steuerung umgewandelt.

Abschnitt 3

Servo-Motor

Servo-Motoren bieten die Möglichkeit, durch eingehende Signale einen präzisen Stellwinkel einzunehmen. Die konkrete Kommunikation und Art des Signals übernimmt die Bibliothek `Servo.h`, die eine Vielzahl von Servomotoren ansprechen kann. Die Schaltung ist denkbar einfach: Der Servo-Motor wird an 5V, GND und die Signalleitung an einen PWM-Ausgang angeschlossen. Außerdem wird der Poti als variables Eingabegerät angeschlossen. Die Schaltung ist in Abb. 5 gezeigt.

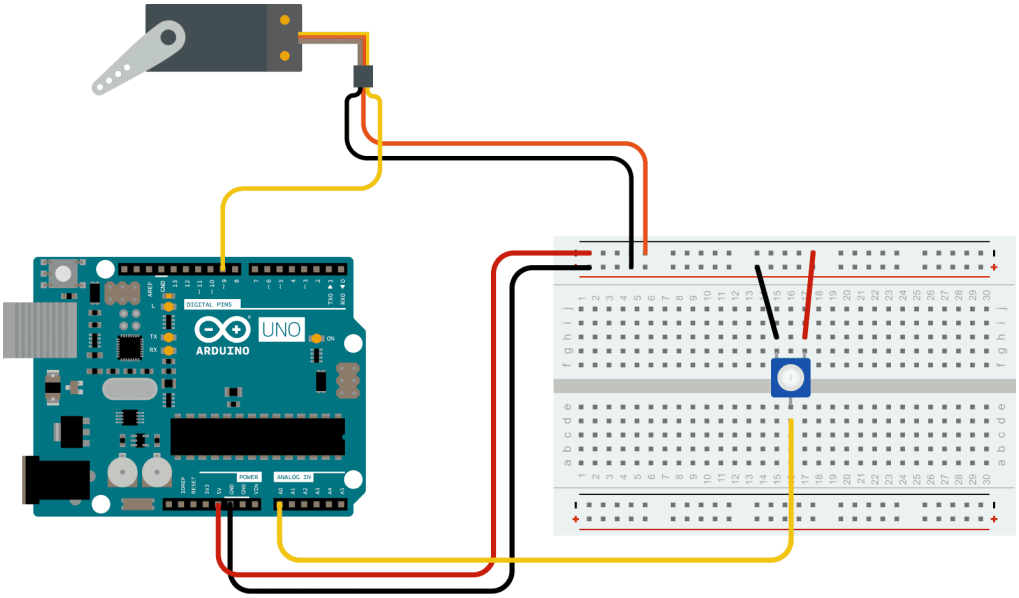


Abb. 5. Schaltplan zum Stellbetrieb eines Servo-Motors mit einem Potentiometer.[5]

Aus der Servo-Bibliothek werden nur die Methoden `Servo::attach(# Pin)` und `Servo::write(Stellwinkel in Grad)` verwendet. Der C++ Befehl `map` wird genutzt um den Wertebereich des Poti-Eingangs auf den Bereich 0 bis 180° abzubilden:

```
#include <Servo.h>

Servo myservo;  // create servo object to control a servo


int potpin = 0;
// analog pin used to connect the potentiometer
int val;
// variable to read the value from the analog pin

void setup() {
  myservo.attach(9);
  // attaches the servo on pin 9 to the servo object
}

void loop() {
  val = analogRead(potpin);
```

```
  // reads the value of the potentiometer
  // (value between 0 and 1023)

  val = map(val, 0, 1023, 0, 180);
  // scale it to use it with the servo
  // (value between 0 and 180)

  myservo.write(val);
  // sets the servo position according to the scaled value

  delay(15);
  // waits for the servo to get there
}
```

TEIL

II

Serielle Kommunikation

Für viele Anwendungen ist es nötig (oder interessant), dass man Informationen des Arduino während des Programmablaufes erhalten kann. Dies können Zwischenergebnisse bei Rechnungen oder auch die aktuellen Messwerte eines Sensors sein. Diese können dann entweder im OLED/LCD Display angezeigt werden oder per serieller Kommunikation in der Konsole angezeigt werden. Die Arduino IDE bietet auch die Möglichkeit, direkt Messwerte als Grafik zu plotten.

Eine fortgeschrittene Variante der Kommunikation besteht dann in einer sogenannten Handshake-Kommunikation, bei der der Arduino auf Anweisungen wartet und dann entsprechend mit dem Senden bestimmter Informationen reagiert.

Abschnitt 4

Serial Monitor

Der Serial Monitor ist in der Arduino IDE integriert. Er stellt eine Terminal-Verbindung zwischen IDE und Arduino her. So ist es möglich, Text vom Arduino zu empfangen. Das übliche Beispiel führt dazu, dass der Arduino ein “Hello World” auf der Konsole ausgibt:

```
void setup() {
Serial.begin(9600);
// start serial connection at 9600 Baud
}

void loop() {
Serial.println("Hello␣world!");
delay(1000);
}
```

Hier wird jetzt sekündlich ein “Hello World” auf der Konsole im Serial Monitor ausgegeben. Der Befehl `.println()` sendet den Text und gibt auch die Anweisung dazu, dass der Text vollständig ist und dargestellt werden soll. Im Gegensatz dazu kann man mit `.print()` mehr und mehr Text senden, der erst nach einem Aufruf von `.println()` dargestellt wird.

Der print-Befehl ist sehr anfängerfreundlich implementiert. So ist es sehr einfach, damit auch Variablenwerte wiederzugeben ohne dass Typenkonvertierungen stattfinden müssen[4]. Im folgenden sind die Beispiele aus der Befehlsreferenz [6] aufgelistet:

[4] *also etwa von Dezimalzahl zu Zeichenkette*

```
void setup() {
Serial.begin(9600);
// start serial connection at 9600 Baud
}

void loop() {

double x = 1.23456;

Serial.println(1.23456,0);
```

```
\\ gibt 1 aus
Serial.println(1.23456,2);
\\ gibt 1.23 aus
Serial.println(1.23456,4);
\\ gibt 1.2345 aus
Serial.println(x,2);
\\ gibt 1.23 aus
Serial.println("Test");
\\ gibt "Test" aus
Serial.println('Test');
\\ gibt "Test" aus
}
```

Abschnitt 5

Serial Plotter

Der Serial Plotter ist ebenfalls in der Arduino IDE integriert. Das Prinzip ist sehr ähnlich zum Serial Plotter: Man baut eine Verbindung auf und sendet dann Zahlenwerte. Diese werden im geöffneten Serial Plotter direkt grafisch dargestellt wie in Abb. 6.

```
void setup() {
  Serial.begin(9600);
}

void loop() {
  int y1 = analogRead(A0);

  Serial.println(y1);

  delay(100);
}
```

Um mehrere Linien in der Grafik zu zeichnen, müssen die Werte durch ein Whitespace (Leerzeichen oder Tab) getrennt sein. Wenn alle Werte für den aktuellen Zeitschritt gesendet wurden muss man mit `.println()` das Darstellen der Daten anfordern. In Abb. 6 ist ein Beispiel zu sehen.

```
Serial.print(variable_first);
Serial.print("\t"); // or Serial.print(" ")
Serial.print(variable_nth);
Serial.print("\t"); // or Serial.print(" ")
Serial.println(variable_last);
```

Abb. 6. (links) Plotter mit einer Variable. (rechts) Plotter mit mehreren Variablen.

Abschnitt 6

Serial Handshake

Die Kommunikation per Serial Handshake[5] ist sehr verbreitet in der Elektrotechnik. Laborausrüstung wie Druckmessköpfe, Generatoren, Stellmotoren, Oszilloskope usw. lassen sich nach diesem Prinzip regeln. Die Geräte warten dabei an ihrer seriellen Schnittstelle (RS232, USB, Bluetooth, WLAN) auf bestimmte ASCII-Zeichenketten und reagieren dann darauf. Meist wird der Empfang auch durch Echo[6] bestätigt. Die Reaktion kann etwa das Übermitteln von Messdaten oder das Starten einer Aufnahme usw. sein.

Für dieses Beispiel (siehe [7]) soll unterschieden werden ob nur ein einziges Zeichen oder mehrere Zeichen gesendet/empfangen werden. Der Unterschied besteht darin, dass man für mehrere Zeichen ein Array aus `char`-Elementen nutzen muss, was als etwas schwieriger anzusehen ist.

[5] *Es gibt dabei noch verschiedene Verfahren. Hier ist alles ziemlich vereinfacht dargestellt.*

[6] *also das Rücksenden des Empfangenen Textes*

Abschnitt 6.1

Echo eines einzelnen Chars

In diesem Beispiel soll der Arduino mit der Arduino IDE kommunizieren. Es gibt oberhalb des Serial-Monitor eine Eingabezeile wo mit man Zeichen an den Arduino schicken kann. Wenn ein Empfang auf dem Seriellen Kanal stattgefunden hat, wird `Serial.available()` den Wert wahr annehmen und man kann dann diesen Empfang mit `Serial.read()` auslesen:

```
char receivedChar;
boolean newData = false;

void setup() {
    Serial.begin(9600);
    Serial.println("<Arduino is ready>");
}

void loop() {
    recvOneChar();
    showNewData();
}

void recvOneChar() {
    if (Serial.available() > 0) {
        receivedChar = Serial.read();
        newData = true;
    }
}

void showNewData() {
    if (newData == true) {
        Serial.print("This just in ... ");
        Serial.println(receivedChar);
        newData = false;
    }
}
```

Der Arduino sendet also jedes empfangene Zeichen wieder durch `Serial.println()` an die Konsole zurück.

Abschnitt 6.2

Mehrere Zeichen übermitteln

Um mehr als ein Zeichen an den Arduino zu übermitteln, bereitet man sich ein Array aus Char-Einträgen vor. Dieses Array wird dann sukzessive mit den ankommenden Zeichen gefüllt bis ein verabredetes "Termination"-Zeichen folgt. Hier wird dies der newline-Charakter ("\n") sein. Damit das Eingabefeld der IDE auch den newline-Befehl beim Absenden der Nachricht übergibt, muss man dies ggf. unten im Fenster noch aus der Dropdown-Liste auswählen.

```
// Example 2 - Receive with an end-marker

const byte numChars = 32; // up to 32 chars can be read
char receivedChars[numChars];
// an array to store the received data

boolean newData = false;

void setup() {
    Serial.begin(9600);
    Serial.println("<Arduino is ready>");
}

void loop() {
    recvWithEndMarker();
    showNewData();
}

void recvWithEndMarker() {
    static byte ndx = 0;
    char endMarker = '\n';
    char rc;

    while (Serial.available() > 0 && newData == false) {
        rc = Serial.read();

        if (rc != endMarker) {
            receivedChars[ndx] = rc;
            ndx++;   // add one in every iteration
            if (ndx >= numChars) {
                ndx = numChars - 1;
            }
        }
        else {
            receivedChars[ndx] = '\0'; // terminate the string
            ndx = 0;
            newData = true;
        }
    }
}

void showNewData() {
    if (newData == true) {
        Serial.print("This just in ... ");
```

```
        Serial.println(receivedChars);
        newData = false;
    }
}
```

Wie man leicht sieht ist die Empfangs-Funktion `void recvWithEndMarker() {}` nun komplizierter geworden. Es wird mit einer Zählvariable `ndx` durch die einzelnen Elemente des Arrays gegangen und jeweils der Empfangene char `rc` dort eingetragen. Wenn als Zeichen jedoch `\n` empfangen wird (der Absender also die Eingabetaste im Nachrichtenfeld gedrückt hat), ist die Nachricht vollständig und wird ausgegeben. Außerdem wird wieder alles für den erneuten Empfang vorbereitet und die Zählvariable `ndx` zurückgesetzt. An der Ausgabe ändert sich nichts – println kann auch direkt per `Serial.println(receivedChars);` das ganze Array als String ausgeben.

Abschnitt 6.3

ASCII-Kommunikation in der Praxis

Bei der Steuerung von Geräten oder Sensoren wird oft ein Kommunikationsschema der Art `keyword = value` oder `keyword value` verwendet. Um aus diesem Text das Schlüsselwort und den Wert zu extrahieren nutzt man normalerweise entsprechende Bibliotheken[7]. Man muss nämlich daran denken, dass eventuell Leerzeichen an irgendwelchen Stellen oder sonstige unerwartete Zeichen gesendet werden könnten. Ein guter Parser kann dann mit allen Eventualitäten umgehen und gibt auch sinnvolle Fehlermeldungen zurück.

[7] *Man nennt das Extrahieren von Schlagwörtern auch* parsen.

TEIL

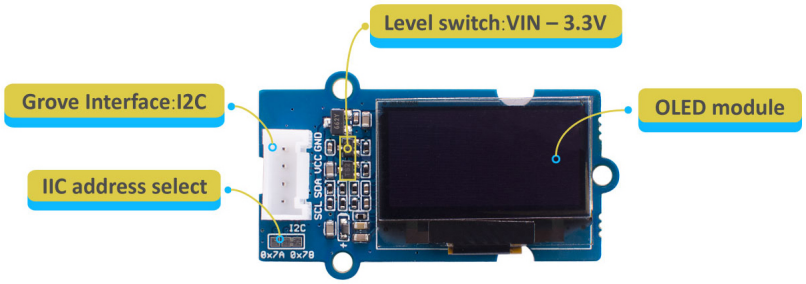
III

Sensoren und Messwerte

Abschnitt 7

OLED

Ein Display (OLED oder LCD) bietet eine Möglichkeit, dass der Arduino auch ohne angeschlossenen PC Informationen abgeben kann. Hier können zum Beispiel Messwerte, Texte usw. angezeigt werden. Die Ansteuerung erfolgt in der Regel über das IIC/I²C-Interface. Um das Display ansteuern zu können, muss deswegen dessen Hardware-Adresse bekannt sein. Oft ist ein Standard voreingestellt, ein Blick ins Datenblatt hilft hierbei.




```
#include <Wire.h>
#include <SeeedOLED.h>

void setup() {
    Wire.begin();
    SeeedOled.init();  //initialze SEEED OLED display

    SeeedOled.clearDisplay();
       //clear the screen and set start position to top left corner
    SeeedOled.setNormalDisplay();
       //Set display to normal mode (i.e non-inverse mode)
    SeeedOled.setPageMode();
       //Set addressing mode to Page Mode
}

void loop() {
    SeeedOled.setTextXY(0, 0);
       //Set the cursor to Xth Page, Yth Column
    SeeedOled.putString("Hello World!");
       //Print the String
}
```

Wenn man Variablenwerte als Text ausgeben möchte (anstelle der Zeichenkette “Hello World”) müssen diese entsprechen formatiert werden. Der Arduino muss wissen, wieviele Nachkommastellen usw. angegeben werden müssen. Im folgenden Beispiel sind einige Formatierungen gezeigt:

OLED

```
#include <Wire.h>
#include <SeeedOLED.h>

void setup() {
    Wire.begin();
    SeeedOled.init();  //initialze SEEED OLED display

    SeeedOled.clearDisplay();
       //clear the screen and set start position to top left corner
    SeeedOled.setNormalDisplay();
       //Set display to normal mode (i.e non-inverse mode)
    SeeedOled.setPageMode();
       //Set addressing mode to Page Mode
}

void loop() {
        float pi = 3.14159;
    SeeedOled.setTextXY(0, 0);
       //Set the cursor to Xth Page, Yth Column
    SeeedOled.putString("Hello␣World!");
       //Print the String
}
```

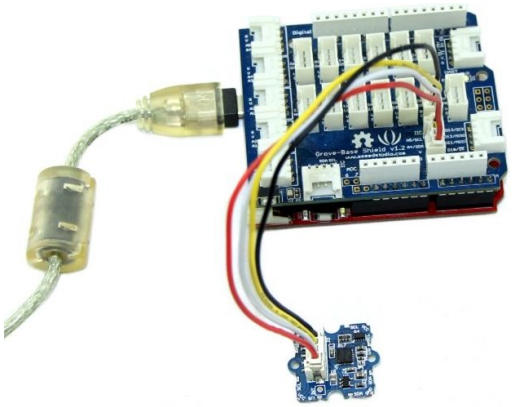
Abschnitt 8

Accelerometer

Accelerometer oder auch g-Meter nennt man Sensoren, die Beschleunigungen messen können. Meist werden 3-Achsen Sensoren angeboten mit denen man also den vollständigen Beschleunigungssensor bestimmen kann. Die Beschleunigungsdaten können zum Einen als Auslöser für Ereignisse genutzt werden (Aktion bei “Gerät fällt gerade vom Tisch”). Andererseits kann man die Daten auch speichern (SD-Shield siehe Abschnitt 18) und später mit dem PC die Beschleunigung integrieren (bzw. summieren) um über

$$v(t) = \int a\,dt \approx \sum_i a_i \Delta t \qquad s(t) = \int \left(\int a\,dt \right) dt \approx \sum_i v_i \Delta t$$

auf die Geschwindigkeit und die Wegstrecke zu schließen. Damit kann man etwa die Höhe eines Wurfes (ja, Arduino incl. Sensor und SD-Shield hochwerfen) ermitteln. So wird das z.B. auch bei Raketen gemacht.



Im Folgenden der Beispielcode des Herstellers zur Anzeige der Beschleunigungsdaten über das Serielle Interface:

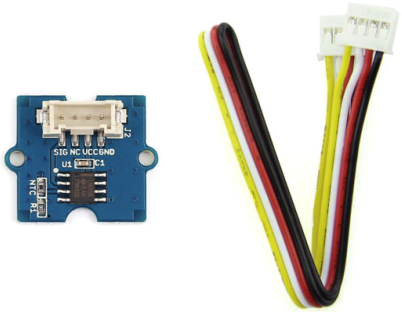
```
#include <Wire.h>
#include "MMA7660.h"
MMA7660 accelemeter;
void setup() {
    accelemeter.init();
    Serial.begin(9600);
}
void loop() {
    int8_t x;
    int8_t y;
    int8_t z;
    float ax, ay, az;
    accelemeter.getXYZ(&x, &y, &z);

    Serial.print("x␣=␣");
    Serial.println(x);
    Serial.print("y␣=␣");
    Serial.println(y);
    Serial.print("z␣=␣");
    Serial.println(z);
```

```
    accelemeter.getAcceleration(&ax, &ay, &az);
    Serial.println("accleration␣of␣X/Y/Z:␣");
    Serial.print(ax);
    Serial.println("␣g");
    Serial.print(ay);
    Serial.println("␣g");
    Serial.print(az);
    Serial.println("␣g");
    Serial.println("*************");
    delay(500);
}
```

Abschnitt 9

Temperatursensor

Die Temperatur wird durch einen Heißleiter (NTC) gemessen. Dieser hat die Eigenschaft, bei höherer Temperatur den Widerstand gegen Gleichspannung zu verringern. In einem gewissen Bereich ist dieser Zusammenhang annähernd linear, so dass man aus einer Kalibrierung heraus die gegenwärtige Temperatur des NTC in Grad Celsius oder Kelvin bestimmen kann. Der eingesetzte NTC kann von $-40°C$ bis $+125°C$ genutzt werden. Die Funktionsweise beruht darauf, dass die Versorungungsspannung am NTC angelegt wird und dann der Spannungsabfall gemessen und in einen Temperaturwert umgerechnet wird.

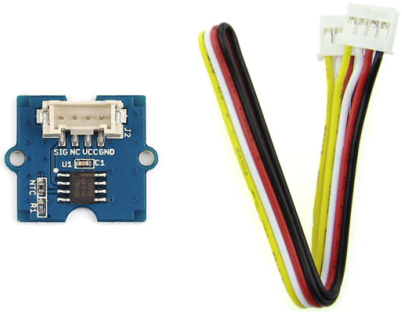



```
#include <math.h>

const int B = 4275;
  // B value of the thermistor (datasheet!)
const int R0 = 100000;
  // R0 = 100k
const int pinTempSensor = A0;
  // Grove - Temperature Sensor connect to A0

void setup()
{
    Serial.begin(9600);
}

void loop()
{
    int a = analogRead(pinTempSensor);

    float R = 1023.0/a-1.0;
    R = R0*R;

    float temperature = 1.0/(log(R/R0)/B+1/298.15)-273.15;
    // convert to temperature via datasheet

    Serial.print("temperature␣=␣");
    Serial.println(temperature);

    delay(100);
}
```

Abschnitt 10

Luftdrucksensor

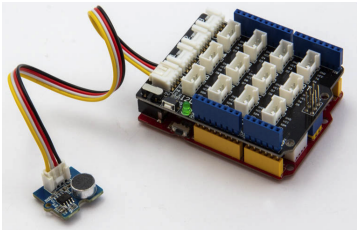
Der Luftdruck kann mit diesem Sensor von 300 hPa bis 1200 hPa ausgelesen werden mit einer Genauigkeit von ±0.002 hPa.



```
#include "BMP085.h"
#include <Wire.h>
float temperature;
float pressure;
float atm;
BMP085 myBarometer;
void setup() {
    Serial.begin(9600);
    while (!Serial);
    myBarometer.init();
}

void loop() {
    pressure = myBarometer.bmp085GetPressure(myBarometer.bmp085ReadUP());
      \\ get pressure from sensor

    atm = pressure / 101325;

    Serial.print("Pressure:␣");
    Serial.print(pressure, 0); //whole number only.
    Serial.println("␣Pa");

    Serial.print("Ralated␣Atmosphere:␣");
    Serial.println(atm, 4); //display 4 decimal places

    Serial.println();

    delay(1000); //wait a second and get values again.
}
```

Abschnitt 11

Sound-Sensor



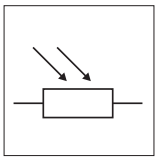
Der Sound Sensor erzeugt eine Spannung am Analogeingang (z.B. A0) abhängig von der empfangenen Geräuschlautstärke. Im Beispielcode wird vorgeschlagen, die Spannungswerte über je 32 Messwerte zu summieren um eine robustere Messung zu haben:

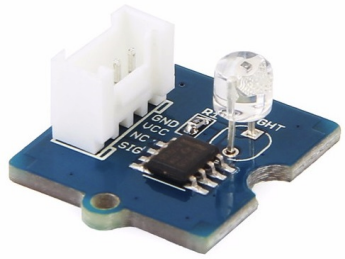
```
const int pinAdc = A0;

void setup()
{
    Serial.begin(115200);
    //Serial.println("Grove - Sound Sensor Test...");
}

void loop()
{
    long sum = 0;
    for(int i=0; i<32; i++)
    {
        sum += analogRead(pinAdc);
    }

    sum >>= 5;

    Serial.println(sum);
    delay(10);
}
```

Wenn man den Serial Plotter geöffnet hat, kann man nun als fortlaufenden Plot die gemessenen Lautstärkewerte ablesen.

Abschnitt 12

Lichtsensor



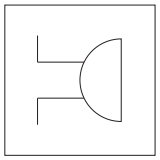


Das Lichtsensor-Modul wandelt die eingehende Lichtintensität per ADC in Zahlenwerte von 0 bis 255 um. Er ist damit geeignet für Lichtmessung, Lichterkennung und lichtgesteuerte Schalter.

```
#include <math.h>

float Rsensor; //Resistance of sensor in K

void setup() {
Serial.begin(9600);
  //Start the Serial connection
}

void loop() {
 int sensorValue = analogRead(0);
 Rsensor=(float)(1023-sensorValue)*10/sensorValue;

 Serial.println("the␣analog␣read␣data␣is␣");
 Serial.println(sensorValue);
 Serial.println("the␣sensor␣resistance␣is␣");
 Serial.print(Rsensor,DEC);
 Serial.println("␣K␣Ohm");
 //show the light intensity on the serial monitor;
 delay(500);
}
```

Abschnitt 13

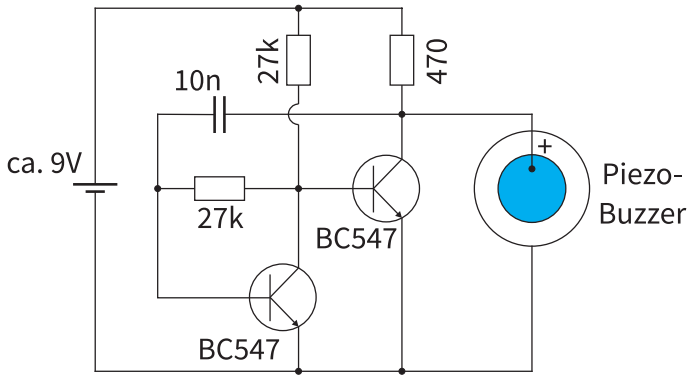
Lautsprecher (Buzzer)



Abschnitt 13.1

. . . als Bauteil

Ein Buzzer is ein Piezoelektrisches Bauteil. Durch eine angelegte Spannung verformt sich die “Lautsprechermembran”. Bei periodischer Spannungsänderung entsteht dadurch ein Ton. Um also einen hörbaren Ton mit einem Buzzer zu erzeugen kann man entweder den PWM-Ausgang des Arduino nutzen (die Frequenz entspricht dann der Tonhöhe, Periodendauer ca. 2 ms), oder die folgende Schaltung:

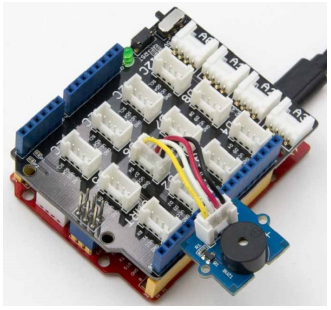


Hinweis: Welche Transistoren genutzt werden ist wohl relativ unwichtig. Auf die richtige Pinbelegung muss aber geachtet werden (siehe Datenblatt).

Abschnitt 13.2

. . . Grove-Variante

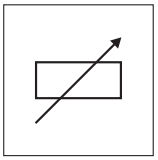
Der Grove-Buzzer besitzt bereits eine entsprechene Schaltung um den Piezzo-Lautsprecher anzusteuern. Man muss hier also nicht mehr für ein periodisches Signal sorgen, sondern nur noch den Buzzer aktivieren/deaktivieren.



```
void setup()
{
  pinMode(6, OUTPUT);
}

void loop()
{
  digitalWrite(6, HIGH);
    //erzeugt Dauerton...
  delay(1000);
  digitalWrite(6, LOW);
    //Ton aus
  delay(1000);
}
```

Abschnitt 14

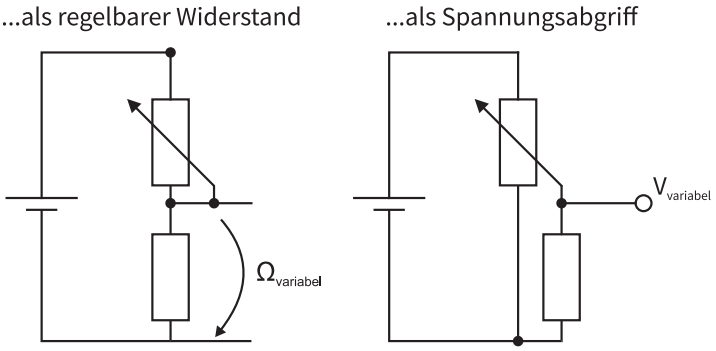
Potentiometer



Abschnitt 14.1

. . . als Bauteil

Potentiometer können als regelbare Widerstände oder auch zum abgreifen variabler Spannungen eingesetzt werden. Variable Spannungen können dann etwa vom Arduino ausgelesen und als Benutzer-Input interpretiert werden.

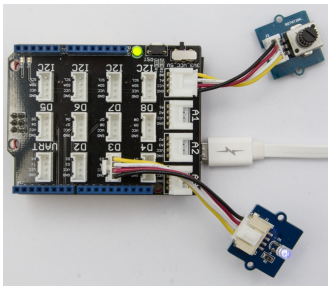


Wie hier gezeigt ist es immer ratsam, einen zweiten Widerstand einzubauen - dies verhindert die Möglichkeit, dass ungewollt ein zu großer Strom durch den Poti fließt (bei sehr klein eingestelltem Widerstand) und diesen zerstört. Im rechten Bild kann man dann den $V_{variabel}$-Anschluss mit dem A0-Input des Arduino verbinden und die Spannung durch `analogRead(A0)` auslesen.

Abschnitt 14.2

. . . Grove-Variante

Hier ist alles bereits fest für die Funktion als variabler Spannungsabgriff vorbereitet. Die eingestellte Position/Spannung kann dann (wie oben) über `analogRead(A0)` ausgelesen werden.

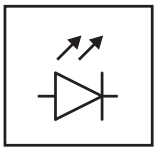


```
#define ROTARY_ANGLE_SENSOR A0
#define LED 3
  //the Grove - LED is connected to PWM pin D3 of Arduino
```

```
#define ADC_REF 5
  //reference voltage of ADC is 5v
#define GROVE_VCC 5
  //VCC of the grove interface is normally 5v
#define FULL_ANGLE 300
  //full value of the rotary angle is 300 degrees

void setup()
{
    Serial.begin(9600);
    pinMode(ROTARY_ANGLE_SENSOR, INPUT);
    pinMode(LED,OUTPUT);
}

void loop()
{
    float voltage;
    int sensor_value = analogRead(ROTARY_ANGLE_SENSOR);
    voltage = (float)sensor_value*ADC_REF/1023;
    float degrees = (voltage*FULL_ANGLE)/GROVE_VCC;
    Serial.println("The␣angle␣is:");
    Serial.println(degrees);

    int brightness;
    brightness = map(degrees, 0, FULL_ANGLE, 0, 255);
    analogWrite(LED,brightness);
    delay(500);
}
```

Abschnitt 15

LED



Abschnitt 15.1

. . . als Bauteil

LEDs sind Dioden, die in Durchlfussrichtung betrieben Licht aussenden. Der Spannungsabfall richtet sich dabei nach der Lichtfarbe: Man kann als Richtwert 1.8 V für rot und bis 3.3 V für blau annehmen. Dieser Spannungsabfall wird von der LED “erzwungen”. Damit dies nicht zu einem zu großen Stromfluss führt, benötigt man einen Vorwiderstand. Die meisten LEDs arbeiten mit Strömen von 10 bis 20 mA. Bei den 5V des Arduino muss der Vorwiderstand also für rot

$$\frac{5\,\mathrm{V} - 1.8\,\mathrm{V}}{10 \ldots 20\,\mathrm{mA}} \approx 200\,\Omega$$

und für blau

$$\frac{5\,\mathrm{V} - 3.3\,\mathrm{V}}{10 \ldots 20\,\mathrm{mA}} \approx 100\,\Omega$$

betragen. **Ohne Vorwiderstand wird entweder die LED oder der Arduino beschädigt.**

Abschnitt 15.2

. . . Grove-Variante

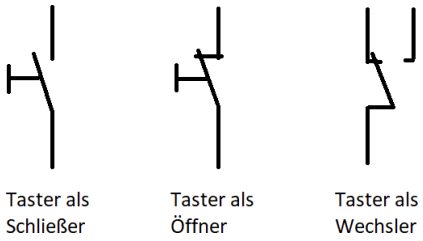
Es gibt Varianten mit fest verbauter LED und mit der Möglichkeit über einen regelbaren Widerstand verschiedene LEDs zu verbauen. In jedem Fall ist hier schon ein Vorwiderstand verbaut und es kann bei Verbidnung per Grove-Kabel kein Schaden an LED und Arduino entstehen.

```
#define LED 2 //connect LED to digital pin2
void setup() {
    // initialize the digital pin2 as an output.
    pinMode(LED, OUTPUT);
}

void loop() {
    digitalWrite(LED, HIGH);   // set the LED on
    delay(500);                // for 500ms
    digitalWrite(LED, LOW);    // set the LED off
    delay(500);
}
```

LED

Abschnitt 16

Taster

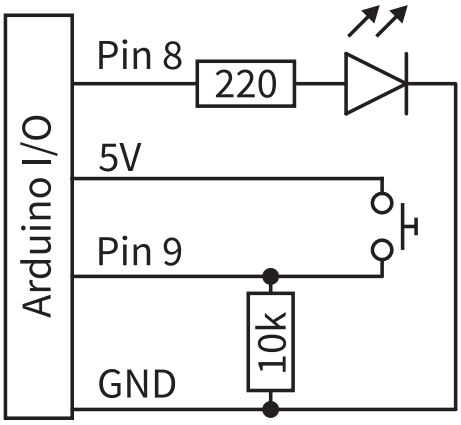


Abschnitt 16.1

. . . als Bauteil

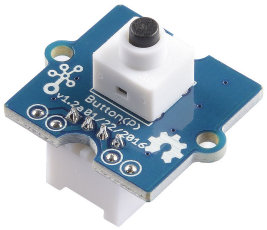
Taster mit mehr als 2 Kontakten können auch als Öffner oder manchmal auch als Wechsler verwendet werden. Eine gute Praxis ist es mir einem Ohmmeter als Durchgangsprüfer die Funktion der Pins durch "Ausprobieren" herauszufinden. Die kleinen Skizzen auf der Unterseite sind oft irreführend.

Wenn der Schalter als Eingabegerät genutzt werden soll, muss man darauf achten, dass der ausgelesene Pin immer einen definierten Zustand besitzt. Dafür nutzt man i.d.R einen Pulldown-Widerstand wie in der nebenstehenden Abbildung. Pin 9 wird bei gedrücktem Schalter mit 5V (HIGH) verbunden, ohne gedrückten Schalter fließen eventuell vorhandene Ladungen durch den 10k-Widerstand gegen GND ab und es stellt sich ein sicherer LOW-Zustand hergestellt. Mit `digitalRead(9)` bekommt man nun 1 oder 0 je nach Schalterzustand. In dieser Zeichnung könnten nun Pin 8 die LED zum Leuchten bringen, wenn der Knopf gedrückt wird.



Abschnitt 16.2

. . . Grove-Variante



Der folgende Beispielcode liest den Zustand des Tasters aus und lässt entsprechend eine LED leuchten:

```
const int buttonPin = 2;
// the number of the pushbutton pin
const int ledPin =  13;
// the number of the LED pin

int buttonState = 0;
  // variable for reading the pushbutton status

void setup() {
    // initialize the LED pin as an output:
    pinMode(ledPin, OUTPUT);
    // initialize the pushbutton pin as an input:
    pinMode(buttonPin, INPUT);
}

void loop(){
    // read the state of the pushbutton value:
    buttonState = digitalRead(buttonPin);

    // check if the pushbutton is pressed.
    // if it is, the buttonState is HIGH:
    if (buttonState == HIGH) {
        // turn LED on:
        digitalWrite(ledPin, HIGH);
    }
    else {
        // turn LED off:
        digitalWrite(ledPin, LOW);
    }
}
```

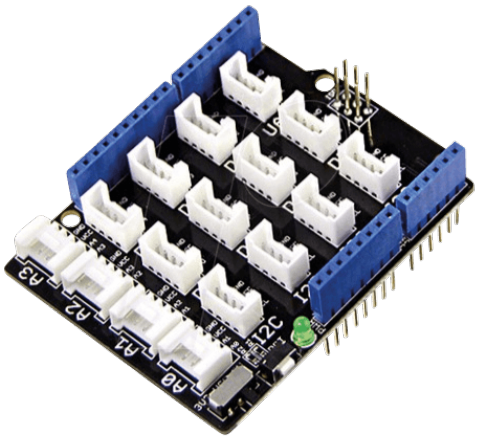
TEIL

IV

Shields

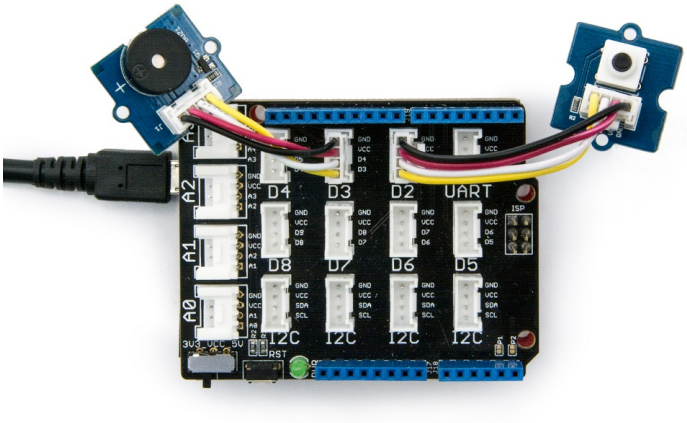
Abschnitt 17

Arduino Base-Shield



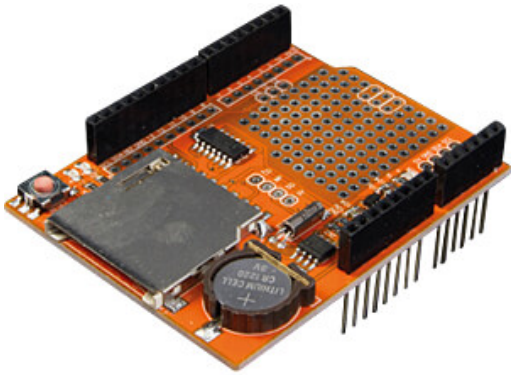
Das Arduino Base shield wird genutzt um Sensoren der Grove-Baureihe anzuschließen. Die verbauten Stecker sind leider etwas speziell und sind tatsächlich nur mit den zugehörigen Kabeln zu betreiben (10Stck ca. 5Euro). Die Schnittstellen selbst werden nicht wirklich erweitert sondern nur auf mehrere Steckkontakte aufgeweitet. So kann man nun mit den Grove-Steckern 4 Analog-Verbindungen (A0...A3), 7 Digital-Verbindungen, 1 UART und 4 I2C-Verbindungen verwenden.

An den Steckern ist jeweils eine Bezeichnung angebracht, so dass ersichtlich ist welchen Pin man im Arduino-Code verwenden muss.



Abschnitt 18

SD-Card Shield



Der Arduino hat selbst keinen dauerhaften Speicher. Lediglich der Programmcode wird nach dem Trennen von der Spannugnsquelle gespeichert und kann immer wieder ausgeführt werden. Gemessene Werte oder Variablen stehen nach einem Neustart nicht zur Verfügung. Wenn man also mit dem Arduino Messwerte aufnehmen will um sie etwa später zu Verarbeiten und Auszuwerten braucht man eine Speicherlösung. Dies kann etwa mit dem SD-Card-Shield realisiert werden. Damit kann man während des Programmablaufes Daten auf die SD-Karte schreiben und später mit einem anderen Gerät auslesen.

Beachten Sie: Die meisten SD-Shields verlangen, dass die SD-Karte mit einem **FAT32-Dateisystem** formatiert wurde. Dies muss man vor der Nutzung mit einem PC/MAC/Tablet bewerkstelligen.

Im folgenden soll in einem Beispielcode gezeigt werden wie man Messwerte kontinuierlich auf die SD-Karte schreibt. Der Code liest einen Temperatursensor aus und speichert die Temperaturwerte (zusammen mit der Zeit) auf der SD-Karte.

Der Tatsächliche Zugriff auf die SD-Karte sollte nicht zu oft erfolgen. Es ist sinnvoller erst etwas Daten zu sammeln und diese dann erst zu schreiben. Hier wird also erst die fertige Zeile für die csv-Datei gesammelt und dann komplett in die SD-Kartendatei geschrieben.

```
#include <SPI.h>  // fuer SPI-Verbindungen
#include <SD.h>   // zur Nutzung des SD-Shields
#include <Wire.h>
#include "RTClib.h"  // Real-time-clock

// A simple data logger for the Arduino analog pins

#define LOG_INTERVAL  1000
// 1000ms zwischen jedem Speichervorgang

#define SYNC_INTERVAL 1000
// mills between calls to flush() - to write data to the card
uint32_t syncTime = 0; // time of last sync()

#define ECHO_TO_SERIAL   1 // echo data to serial port
#define WAIT_TO_START    0 // Wait for serial input in setup()

// the digital pins that connect to the LEDs
#define redLEDpin 2
```

```
#define greenLEDpin 3

// Pin des Temperatursensors
#define tempPin 1                 // analog 1
// for the data logging shield, we use digital
// pin 10 for the SD cs line
const int chipSelect = 10;

RTC_DS1307 RTC; // define the Real Time Clock object

// the logging file
File logfile;

void error(char *str)
{
  Serial.print("error: ");
  Serial.println(str);

  // red LED indicates error
  digitalWrite(redLEDpin, HIGH);

  while(1);
}

void setup(void)
{
  Serial.begin(9600);
  Serial.println();

  // use debugging LEDs
  pinMode(redLEDpin, OUTPUT);
  pinMode(greenLEDpin, OUTPUT);

#if WAIT_TO_START
  Serial.println("Type any character to start");
  while (!Serial.available());
#endif //WAIT_TO_START

  // initialize the SD card
  Serial.print("Initializing SD card...");
  // make sure that the default chip select pin
  // is set to output, even if you don't use it:
  pinMode(10, OUTPUT);

  // see if the card is present ready:
  if (!SD.begin(chipSelect)) {
    error("Card failed, or not present");
  }
  Serial.println("card initialized.");

  // create a new file
  char filename[] = "LOGGER00.CSV";
  for (uint8_t i = 0; i < 100; i++) {
```

```
    filename[6] = i/10 + '0';
    filename[7] = i%10 + '0';
    if (! SD.exists(filename)) {
      // only open a new file if it doesn't exist
      logfile = SD.open(filename, FILE_WRITE);
      break;  // leave the loop!
    }
  }

  if (! logfile) {
    error("couldnt create file");
  }

  Serial.print("Logging to: ");
  Serial.println(filename);

  // connect to RTC
  Wire.begin();
  if (!RTC.begin()) {
    logfile.println("RTC failed");
#if ECHO_TO_SERIAL
    Serial.println("RTC failed");
#endif  //ECHO_TO_SERIAL
  }


  logfile.println("millis,stamp,datetime,temp");
  // Spaltenkoepfe in Datei schreiben
#if ECHO_TO_SERIAL
  Serial.println("millis,stamp,datetime,temp");
#endif //ECHO_TO_SERIAL

}

void loop(void)
{
  DateTime now;

  // delay for the amount of time we want between readings
  delay((LOG_INTERVAL -1) - (millis() % LOG_INTERVAL));

  digitalWrite(greenLEDpin, HIGH);

  // log milliseconds since starting
  uint32_t m = millis();
  logfile.print(m);           // milliseconds since start
  logfile.print(", ");
#if ECHO_TO_SERIAL
  Serial.print(m);         // milliseconds since start
  Serial.print(", ");
#endif

  // fetch the time
```

```
    now = RTC.now();
    // log time
    logfile.print(now.unixtime());
    // seconds since 1/1/1970
    logfile.print(",␣");
    logfile.print('"');
    logfile.print(now.year(), DEC);
    logfile.print("/");
    logfile.print(now.month(), DEC);
    logfile.print("/");
    logfile.print(now.day(), DEC);
    logfile.print("␣");
    logfile.print(now.hour(), DEC);
    logfile.print(":");
    logfile.print(now.minute(), DEC);
    logfile.print(":");
    logfile.print(now.second(), DEC);
    logfile.print('"');
  #if ECHO_TO_SERIAL
    Serial.print(now.unixtime());
    // seconds since 1/1/1970
    Serial.print(",␣");
    Serial.print('"');
    Serial.print(now.year(), DEC);
    Serial.print("/");
    Serial.print(now.month(), DEC);
    Serial.print("/");
    Serial.print(now.day(), DEC);
    Serial.print("␣");
    Serial.print(now.hour(), DEC);
    Serial.print(":");
    Serial.print(now.minute(), DEC);
    Serial.print(":");
    Serial.print(now.second(), DEC);
    Serial.print('"');
  #endif //ECHO_TO_SERIAL

    analogRead(tempPin);
    delay(10);
    int tempReading = analogRead(tempPin);

    float voltage = tempReading * 5.0 / 1024;
    float temperatureC = (voltage - 0.5) * 100 ;

    logfile.print(",␣");
    logfile.print(temperatureC);
  #if ECHO_TO_SERIAL
    Serial.print(",␣");
    Serial.print(temperatureC);
  #endif //ECHO_TO_SERIAL


    logfile.println();
```

```
#if ECHO_TO_SERIAL
  Serial.println();
#endif // ECHO_TO_SERIAL

  digitalWrite(greenLEDpin, LOW);

  // Now we write data to disk!
  // Don't sync too often - requires
  // 2048 bytes of I/O to SD card
  // which uses a bunch of power and takes time
  if ((millis() - syncTime) < SYNC_INTERVAL) return;
  syncTime = millis();

  // blink LED to show we are syncing
  // data to the card & updating FAT!
  digitalWrite(redLEDpin, HIGH);
  logfile.flush();
  digitalWrite(redLEDpin, LOW);
}
```

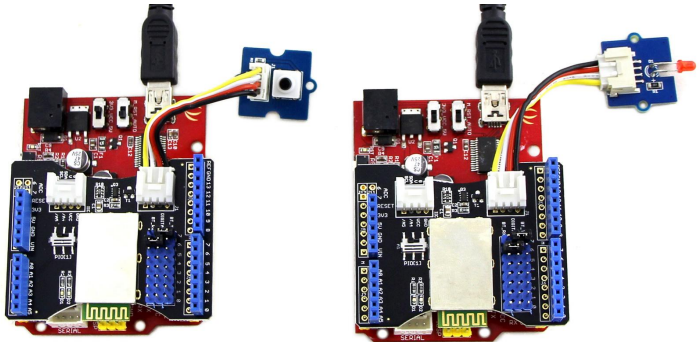
TEIL

V

Bluetooth-Verbindungen

Abschnitt 19

(Grove) Bluetooth Shield



Es gibt auch die Möglichkeit, den Arduino um die Möglichkeit zur Bluetooth-Kommunikation (und auch WLAN) zu erweitern. So kann dann also etwa ein Arduino mit BT-Modul mit einem anderen Arduino mit BT-Modul kommunizieren und Aktionen ausführen. Das Bild oben zeigt etwa die Möglichkeit, dass ein Knopfdruck an Arduino 1 die LED an Arduino 2 zum Leuchten bringt. Außerdem ist auch, leider dann per extra App, die Steuerung eines Arduino per Smartphone/PC/MAC/Tablet möglich. Auf der Website stehen die Codes für den Master(Button) und den Slave(LED) zur Verfügung. Das Prinzip einer BT-Kommunikation funktioniert dann wie folgt:

Master	Slave
sendet eine Anfrage zur Verbindung	schaltet sich “anfragbar”
verhandelt die Verbindungseigenschaften mit dem verfügbaren Gerät (Name vorgegeben)	
Verbindung herstellen	Verbindung herstellen
Sendet bei Knopfdruck das Zeichen “1” oder “0” als Kommando für LED	schaut nach, ob Daten empfangen werden und schaltet entsprechend die LED um

Die Quellcodes sind für Schüler, wenn überhaupt, nur schwer zu vestehen. Es ist aber vielleicht möglich, dass man vorgefertigte Codes nutzt und dann modifiziert um etwa statt des Knopfstatus einen Text an den Slave zu senden, den dieser dann auf einem Display anzeigt. Noch leichter wäre es z.B. die LED durch den Buzzer zu ersetzen. Dann kann man gewissermaßen eine Morse-Übertragung durchführen.

Zur Verbindung mit dem Smartphone nutzt man eine Buetooth SSP-App. SSP steht für Software Serial Port - also eine Art emulierter serieller Port auf Basis einer Bluetooth Verbindung. Auch hier wird durch den Austausch von Zeichen oder Zeichenketten kommuniziert: Das Smartphone schickt im Beispiel ein “t” an den Arduino und dieser antwortet darauf dann z.B. mit dem Text “temperature: 24”.

Abschnitt 20

Arduino-Android/iOS Verbindung

Die Verbindung von Arduino und einem Android-Gerät läuft ganz prinzipiell auch per serieller Schnittstelle und es werden Befehle gesendet und empfangen. Einziger Unterschied ist die modernere Darstellung. Die kostenlose und werbefreie App *com.thechampanurag.arduino* ist als *.apk verfügbar. Im Beispiel [8] soll nun gezeigt werden wie der Arduino ausgelesen und gesteuert werden kann. In dem Beispiel sind zwei LEDs an Pin 5 und Pin 6

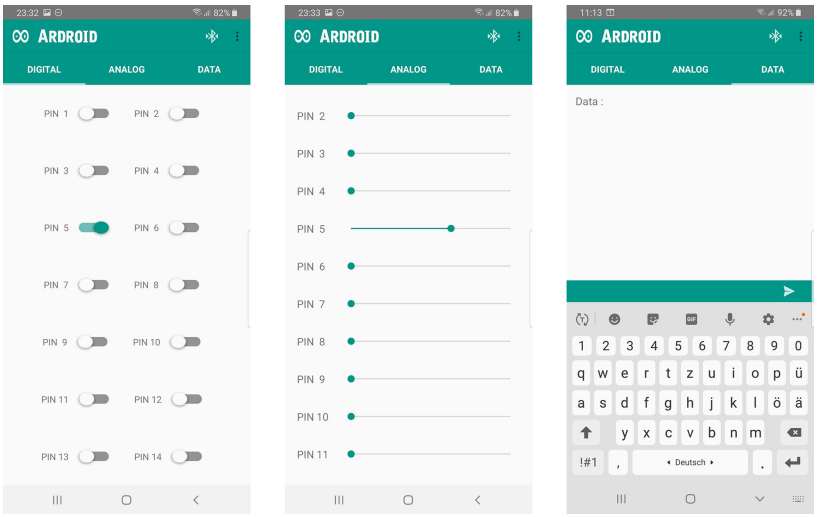


Abb. 7. So sieht etwa eine Steuerungs-App aus[9]. Es gibt aber eine Vielzahl an möglichen Alternativen...

angeschlossen, die dann per App gesteuert werden können. Der Arduino ist als “Slave” konfiguriert und wartet also ständig auf die Befehle der App, die ihrerseits Textkommandos sendet. Hier beginnt jeder Befehlt mit einem `*`, gefolgt von drei Zahlen für Kommando, Pinnummer und Wert. Damit kann man also die Beispielsweise die Nachricht `* 10 5 1` senden mit der Bedeutung: digitalWrite an Pin 1 auf den Wert 1 setzen.

```
/*
 PROJECT: Ardroid
 CODED BY: Anurag Goel
 PUBLIC Licence Free To Modify
*/

#define START_CMD_CHAR '*'
#define END_CMD_CHAR '#'
#define DIV_CMD_CHAR '|'
#define CMD_DIGITALWRITE 10
#define CMD_ANALOGWRITE 11
#define CMD_TEXT 12
#define CMD_READ_ARDROID 13
#define MAX_COMMAND 20
#define MIN_COMMAND 10
#define IN_STRING_LENGHT 40
#define MAX_ANALOGWRITE 255
#define PIN_HIGH 3
```

```
#define PIN_LOW 2

String inText;

void setup() {
  Serial.begin(38400);
  Serial.println("Ardroid␣By␣:␣Anurag␣Goel");
  Serial.flush();
}

void loop()
{
  Serial.flush();
  int ard_command = 0;
  int pin_num = 0;
  int pin_value = 0;

  char get_char = '␣';  //read serial

  // wait for incoming data
  if (Serial.available() < 1) return;
  // if serial empty, return to loop().

  // parse incoming command start flag
  get_char = Serial.read();
  if (get_char != START_CMD_CHAR) return;
  // if no command start flag, return to loop().

  // parse incoming command type
  ard_command = Serial.parseInt(); // read the command
  // parse incoming pin# and value
  pin_num = Serial.parseInt(); // read the pin
  pin_value = Serial.parseInt();  // read the value
  // 1) GET TEXT COMMAND FROM ARDROID
  if (ard_command == CMD_TEXT){
    inText =""; //clears variable for new input
    while (Serial.available())  {
   char c = Serial.read();

      //gets one byte from serial buffer
       delay(5);
       if (c == END_CMD_CHAR) {
       // if we the complete string has been read
         // add your code here
         Serial.println(inText);
         break;
       }
       else {
         if (c !=  DIV_CMD_CHAR) {
           inText += c;
           delay(5);
         }
       }
```

```
    }
  }

  // 2) GET digitalWrite DATA FROM ARDROID
  if (ard_command == CMD_DIGITALWRITE){
    if (pin_value == PIN_LOW) pin_value = LOW;
    else if (pin_value == PIN_HIGH) pin_value = HIGH;
    else return; // error in pin value. return.
    set_digitalwrite( pin_num,  pin_value);
    return;  // return from start of loop()
  }

  // 3) GET analogWrite DATA FROM ARDROID
  if (ard_command == CMD_ANALOGWRITE) {
    analogWrite(  pin_num, pin_value );
    // add your code here
    return;  // Done. return to loop();
  }

  // 4) SEND DATA TO ARDROID
  if (ard_command == CMD_READ_ARDROID) {
//    char send_to_android[] = "Place your text here." ;
//    Serial.println(send_to_android);   // Example: Sending text
    Serial.print("␣Analog␣0␣=␣");
    Serial.println(analogRead(A0));
    // Example: Read and send Analog pin value to Arduino
    return;  // Done. return to loop();
  }
}

// 2a) select the requested pin# for DigitalWrite action
void set_digitalwrite(int pin_num, int pin_value)
{
  switch (pin_num) {
  case 13:
    pinMode(13, OUTPUT);
    digitalWrite(13, pin_value);
    // add your code here
    break;
  case 12:
    pinMode(12, OUTPUT);
    digitalWrite(12, pin_value);
    // add your code here
    break;
  case 11:
    pinMode(11, OUTPUT);
    digitalWrite(11, pin_value);
    // add your code here
    break;
  case 10:
    pinMode(10, OUTPUT);
    digitalWrite(10, pin_value);
    // add your code here
```

```
      break;
    case 9:
      pinMode(9, OUTPUT);
      digitalWrite(9, pin_value);
      // add your code here
      break;
    case 8:
      pinMode(8, OUTPUT);
      digitalWrite(8, pin_value);
      // add your code here
      break;
    case 7:
      pinMode(7, OUTPUT);
      digitalWrite(7, pin_value);
      // add your code here
      break;
    case 6:
      pinMode(6, OUTPUT);
      digitalWrite(6, pin_value);
      // add your code here
      break;
    case 5:
      pinMode(5, OUTPUT);
      digitalWrite(5, pin_value);
      // add your code here
      break;
    case 4:
      pinMode(4, OUTPUT);
      digitalWrite(4, pin_value);
      // add your code here
      break;
    case 3:
      pinMode(3, OUTPUT);
      digitalWrite(3, pin_value);
      // add your code here
      break;
    case 2:
      pinMode(2, OUTPUT);
      digitalWrite(2, pin_value);
      // add your code here
      break;

  }
}
```

Wer Smartphones benutzen möchte muss die Apps natürlich vorher selber Testen. Auch die Kommunikation sollte vorher getestet werden. Es gibt einige Fallstricke wie etwa die Baudrate (die bei Master und Slave gleich sein muss).

Ich empfehle, auf bunte Knöpchen zu verzichten und stattdessen eine einfache App zur seriellen Kommunikation per Bluetooth zu verwenden. Das ist dann also einfach der Serial-Monitor der Arduino IDE die man auf dem Smartphone ausführt. Dort kann es keine tiefergehenden Probleme geben udn man “hat alles selbst in der Hand”. Das Stichwort hierfür lautet *Bluetooth SPP* – darunter findet man eine große Auswahl an Apps.

TEIL

VI

Arduino-Befehlsreferenz

Hier gibt es eine sehr kurze Übersicht über die wichtigsten Befehle beim Betreiben des Arduino. Am besten ist es, die Befehle bei Problemen/Fragen sowieso während des Programmierens im Netz nachzuschlagen. GGf. stößt man auch gleich auf nützliche Code-Schnipsel die die Arbeit erleichtern. Daran denken: Groß- und Kleinschreibung wird unterschieden!

Abschnitt 21

Bestandteil des Präprozessors:

const int meineKonstante = 1; ... definiert eine Konstante

const float pi = 3.14; ... definiert eine Konstante

#include <Servo.h> ... die bereits installierte Bibliothek installieren

#include SServo.h" ... die Bibliothek mit Namen "Servo.h" aus dem Projektpfad einfügen

Abschnitt 22

Bestandteil von void setup()

pinMode(# des Pins, INPUT) ... definiert den gegebenen Pin als Eingang

pinMode(# des Pins, OUTPUT) ... definiert den gegebenen Pin als Ausgang

Serial.begin(9600) ... Öffnet die serielle Schnittstelle und stellt die Datenrate auf 9600 Bit/s ein

Abschnitt 23

Bestandteil von void loop()

int meineVariable = 0; ... definiert eine Konstante

float dezimalzahl = 6.28; ... definiert eine Konstante

dezimalVariable = (float) meineVariable; ... zu float konvertieren

Serial.println("Textausgabe"); Sendet den Text (inklusive Zeilenumbruch) an den Seriellen Monitor.

delay(t in ms) ... unterbricht das Programm für t ms

if (Bedingung) ... bedingte Ausführung

for (int i = 0; i < 5; i++) ... Schleife für i = 0 bis i=4

while(Bedingung) ... Schleife wird wiederholt, solange "Bedingung" wahr ist oder mit "break" abgebrochen wird

digitalRead(# Pin) ... liest den Wert eines Digitaleingangs aus

analogRead(# Pin) ... liest den Wert eines Analogeingangs aus

digitalWrite(# Pin, HIGH) ... legt den Wert HIGH bzw. 5V an den Digitalausgang

digitalWrite(# Pin, LOW) ... legt den Wert LOW bzw. 0V an den Digitalausgang

analogWrite(# Pin, 0...255) ... gibt eine PWM-Welle am Ausgang aus

Literatur

[1] arduino.cc. Arduino Blink Example, 2023. URL https://docs.arduino.cc/built-in-examples/basics/Blink.

[2] arduino.cc. Arduino AnalogRead Example, 2023. URL https://docs.arduino.cc/built-in-examples/basics/AnalogReadSerial.

[3] arduino.cc. Arduino Pushbutton Example, 2023. URL https://docs.arduino.cc/built-in-examples/digital/Button.

[4] arduino.cc. Arduino Transistor Motor Control, 2023. URL https://docs.arduino.cc/learn/electronics/transistor-motor-control.

[5] arduino.cc. Arduino Servo Motor Basics, 2023. URL https://docs.arduino.cc/learn/electronics/servo-motors.

[6] arduino.cc. Arduino Serial.print(), 2023. URL https://www.arduino.cc/reference/de/language/functions/communication/serial/print/.

[7] arduino.cc. Arduino Basics, 2023. URL https://forum.arduino.cc/t/serial-input-basics-updated/382007.

[8] W. Ewald. Arduino mit dem Smartphone Steuern, 2019. URL https://wolles-elektronikkiste.de/arduino-mit-dem-smartphone-steuern.

[9] thechampanurag. Arduino-App. URL https://aapks.com/apk/thechampanurag-ardroid/.