UNIVERSITÄT GREIFSWALD
Wissen lockt. Seit 1456

Arduino - Einführungskurs I

Programmierung und Elektronik

Michael Himpel

Institut für Physik,
Universität Greifswald

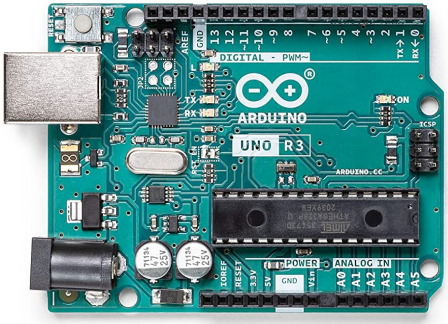
Sept. 2023

Übersicht

Arduino - Was ist das?

Einführung

- Microprozessor/Microcontroller/µC
- eingebettet in I/O-Konzept
- direkt programmierbar per PC/MAC
- nahezu beliebig erweiterbar durch sog. *Shields*
- verfügbar: Sensoren, Motoren, Displays, Speicherkarten, BT/WLAN, ...
- Idee: (Elektro-)Technik für Schüler/Studenten/Hobbyisten

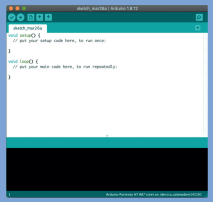


Arduino - Was ist das?

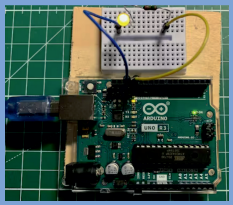
Konzept

Idee/
Problemstellung

Implementierung



Aufbau + Upload



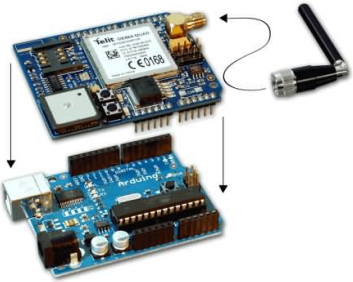
Arduino - Was ist das?

Arduino-Infrastruktur

- Erweiterbarkeit durch Shields
- Grove/Seeed als handlicher Einstieg in die Sensorik

! sämtliche Sensoren/Erweiterungen auch direkt anschließbar

- Ansteuerung der Erweiterungen durch Libraries (mit Beispielen)
- Stromversorgung durch Batterie(9V), USB, Hohlstecker/Netzteil



Arduino - Was ist das?

Arduino-Infrastruktur

Breadboard

+ volle Kontrolle über Alles
+ maximale Flexibilität
+ technische Details werden wichtig
+ realitätsnah
- ggf. “unordentliches” Layout

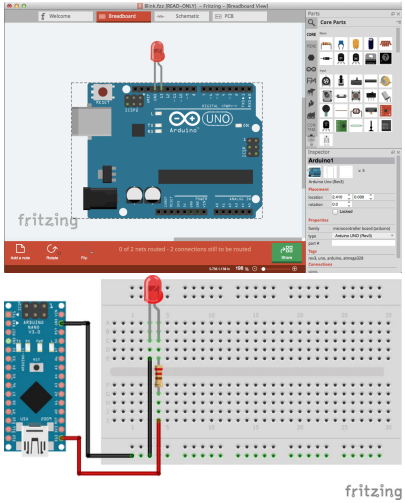
Seeed/Grove

+ vorgefertigte Steckverbindungen
+ geringere Hürde bis zum ersten Erfolg
+ vorbereitete Beispiele+Code
- reines Lehr/Lernkonzept

Arduino - Was ist das?

Fritzing - Schaltskizzen

- optisch gut verständliche Möglichkeit, Schaltbilder für Arduino-Aufbauten zu erzeugen
- optisch etwas ansprechender als üblich
- 1-zu-1 umsetzbar auf dem Breadboard



Arduino - Was ist das?

Kosten

Eher Elektroniklastig:

Arduino Starter Kit	90€
Breadboard	inkl.
Steckbrücken	inkl.
Elektronik Kleinteile	inkl.
Motor/Display usw.	inkl.
Gesamt:	90€

Eher Plug-and-Play:

Arduino Uno	25€
Arduino Sensor-Kit	33€
Grove-Kabel	inkl.
Sensoren+OLED	inkl.
SD-Shield + SD-Karte	11€
Gesamt:	70€

Außerdem: PC/MAC/Tablet(nicht empfohlen, aber geht zur Not...)

Programmierung

- Allgemeines zu C++
- Syntax
- Variablen
- Strukturen
- Funktionen
- Bibliotheken

Arduino - Programmierung

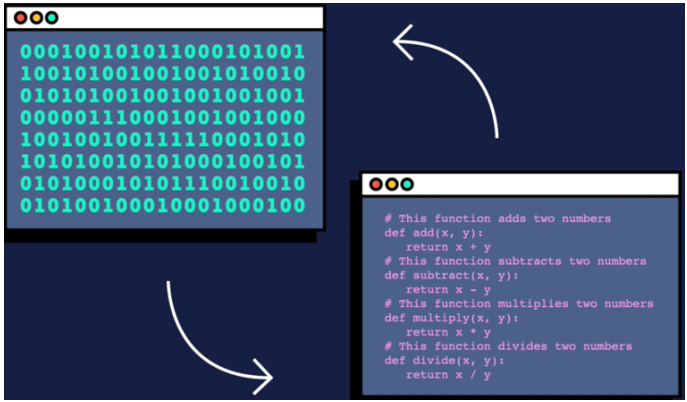
Arduino-IDE

- Editor+Compiler = IDE
- verfügbar für Windows/Linux/MAC/Web-Editor
- viele Beispiel-Codes bereits integriert
- Upload aufs Arduino-Board per Knopfdruck

Arduino - Programmierung

C/C++

+ plattformübergreifend
+ große Standardbibliothek
+ sehr verbreitet/viele Bibliotheken
+ breites Abstraktionsspektrum
+ objektorientiert
- eher komplizierte Syntax



Arduino - Programmierung

C/C++: Grundlagen

Was braucht man für C++ Programmierung?

- Texteditor (Notepad, Nano, EMACS, ...)
- Compiler (z.B. GCC)
- Glück für uns: Für Arduino-Programmierung ist dies in einer IDE zusammengefasst

```
void setup() {
  Serial.begin(9600);
  Serial.println("Hello World!");
}

void loop() {
  // put your main code here, to run repeatedly:
}
```

Arduino - Programmierung

C/C++: Grundlagen

- Anweisungen mit ";" abschließen
- Funktionen mit "{ }" einbetten
- "//" für Kommentare (wird vom Compiler ignoriert)
- Leerzeichen/Zeilenumbrüche nur für menschliche Lesbarkeit, werden vom Compiler ignoriert
- Case-Sensitiv! (Groß- und Kleinschreibung beachten)

```
c = a + b; // identisch: c=a+b;
c = A + b; // A nicht gleich a!
ergebnis = funktion(variable){
...
}
variablenBenennen = true;
```

Arduino - Programmierung

C/C++: Datentypen

- Variablen/Konstanten muss ein Datentyp zugeordnet werden!
- float, double für Dezimalzahlen
- sagt dem Compiler, wie die Bits im Speicher interpretiert werden sollen

```
int      positiveZahl = 1235;
unsigned long superRiesenZahl = 3;
float    dezimalZahl = 1.234;
double   hochPraezise = 3.1234567;
bool     stimmtDas = true;
```

Arduino - Programmierung

C/C++: Variablen

- Variablen müssen vor Gebrauch *deklariert* werden
- Am sichersten: Auch gleich initialisieren!
- Nicht-Initialisierte Variablen haben trotzdem irgendeinen Wert, kann zu Fehlern führen

```
int ganzeZahl; // nur Deklaration
ganzeZahl = 0; // dann Definition
int ganzeZahl = 0;
// Dekl. + Def. = Initialisierung
```

Arduino - Programmierung

C/C++: Konstanten

- Konstanten dürfen nicht mehr verändert werden
- Falls man das später doch tut, gibt es eine Fehlermeldung
- Also: Alles was konstant bleibt, auch als “const” deklarieren
- Am besten ganz nach vorn in den Quellcode

```
const int  pi = 3.14;
const bool ichHabeRecht = true;
const int  ledPin = 10;
```

Arduino - Programmierung

C/C++: Funktionen

- Rückgabetyp und Eingabetypen müssen festgelegt werden
- Rückgabe eines Wertes mit “return”
- Falls keine Rückgabe: Rückgabetyp “void”
- Am besten nach “loop” (also am Ende) die Funktionen definieren

```
int summe (int a, int b){ //I/O
return a+b; //Anweisungen...
}
double summe (double a, double b){ /
return a+b;
}
// Funktion aufrufen:
int smd1 = 4;
int smd2 = 5;
int ergAdd = summe(smd1, smd2);
```

Arduino - Programmierung

C/C++: Objekte/Methoden

- C++ ist eine objektorientierte Sprache
- Objekte werden definiert, dann erzeugt man nach Bedarf Instanzen davon
- diese Objekte haben eigene Methoden: objekt.methode(ggf. Parameter, Variablen...)

```
SD.begin(4); // SD Zugriff starten.
myFile = SD.open("test.txt",
FILE_WRITE);
myFile.println("Hallo Datei!");
myFile.close();
```

Arduino - Programmierung

C/C++: For-Schleifen

- For-Schleifen sind gut für gezählte Abfolgen
- Anzahl der Schleifendurchläufe muss bekannt sein
- nützlich: k++ identisch mit k=k+1
- Vergleiche: $==, !=, <, <=, >, >=$

```
for(start; abbruch? ; zaehlen){
// Anweisung
}

// Beispiel:
for (int k=1; k<=10 ; k++){
int c = 3+k;
}
```

Arduino - Programmierung

C/C++: While-Schleifen

- While Schleifen sind günstig um z.B. auf Eingaben zu reagieren (Knopf gedrückt usw.)
- Die Anzahl der Schleifendurchläufe ist i.d.R. vorher unbekannt
- Vergleiche: $==, !=, <, <=, >, >=$

```
while(Bedingung ist wahr){
// Anweisung
}

// Beispiel:
int k = 0;
while (k<=10){
k = k+1;
}
```

Arduino - Programmierung

C/C++: if. . . else

- um Bedingungen zu prüfen und ggf. Entscheidungen zu treffen
- z.B. Abfrage von Zustand an digitalem Eingang (Knopf gedrückt? Schalter geschlossen?)

```
if (logische Bedingung){
// Anweisung
}
else{
// Anweisung
}
```

Arduino - Programmierung

Arduino-Spezifisches: loop/setup

- setup(): Hier werden die Ports des Arduino festgelegt (Eingang oder Ausgang)
- setup(): Initialisierung/Einstellung von angeschlossenen Geräten
- loop(): eigentlicher Programmablauf als Dauerschleife

```
// hier ggf. Bibliotheken laden...

void setup() {
  // hierher kommen z.B.
  // I/O Definitionen
}

void loop() {
  // Dieser Programmteil ist
  // eine Dauerschleife
}
```

Arduino - Programmierung

Arduino-Spezifisches: pinMode

- die I/O-Ports sind auf dem Arduino nummeriert
- müssen in setup() als INPUT oder OUTPUT festgelegt werden
- auf Analog/Digital-Ports achten!

```
pinMode(12,INPUT);
pinMode(10,OUTPUT);
pinMode(LED_BUILTIN, OUTPUT);
```

Arduino - Programmierung

Arduino-Spezifisches: digitalRead/Write

- zur Umschaltung von HIGH/LOW (5V/GND) an digitalen Ausgängen
- ... müssen voher aus OUTPUT definiert worden sein
- ggf. Stromgrenzen beachten (i.d.R. $40\,\mathrm{mA}$)
- für Auslesen von HIGH/LOW per digitalRead muss der Pin als INPUT definiert sein.

```
void setup() {
pinMode(10,OUTPUT);
pinMode(9,INPUT);
}

int isActive = false;
void loop() {
  digitalWrite(10, HIGH);
  //an Pin 10 ab jetzt 5V
  isActive = digitalRead(9);
  //isActive wird nun 1 oder 0;
}
```

Arduino - Programmierung

Arduino-Spezifisches: analogRead/Write

- analogRead liest Spannung aus ($0 \ldots 1024 \equiv 0 \ldots 5\,\mathrm{V}$)
- analogWrite nutzt PWM für "Simulation" einer veränderbaren Gleichspannung (Werte 0...255)

```
void setup() {
pinMode(10,OUTPUT);
pinMode(A0,INPUT);
}

int spannungAnA0 = 0;
void loop() {
  analogWrite(10, 128);
  //an Pin 10 jetzt 2.5V RMS
  spannungAnA0 = analogRead(A0);
  //spannungAnA0 is nun 0...1024;
}
```

Arduino - Programmierung

Bibliotheken/Libraries

- definiert Funktionen, Objekte und Methoden für best. Geräte und Sensoren
- solche Funktionen sind meist sehr kompliziert ($\rightarrow$ nicht selber machen)
- ggf. in Hilfe zur Bibliothek nach Beispielen suchen
- Bibliotheken ganz vorn einfügen!

```
#include <LiquidCrystal_I2C.h>
//#include <LiquidCrystal.h>
#include <Wire.h>
LiquidCrystal_I2C lcd(0x27, 16, 2);

void setup() {
}

void loop() {
  lcd.init();
  lcd.setCursor(0, 0);
  lcd.print("HALLO");
  delay(10000);
  lcd.clear();
}
```

Arduino - Programmierung

Arrays

- Arrays sind Container für mehrere Werte – ähnlich Vektoren oder Matritzen
- Arrays können verschiedene Datentypen enthalten (int,double,char)
- ein “String” ist ein Array aus “Chars”
- Achtung: Es wird von 0 aus gezählt!

```
int myInts[6]; // noch irgendwelche
int myPins[] = {2, 4, 8, 3, 6}; // 
int mySensVals[5] = {2, 4, -8, 3, 2
char message[6] = "hello"; // array

// Arrays auslesen:
int readValue = myPins[1]; // ergib

// ...das ganze array:
for (byte i = 0; i < 5; i = i + 1) 
  Serial.println(myPins[i]);
}
```

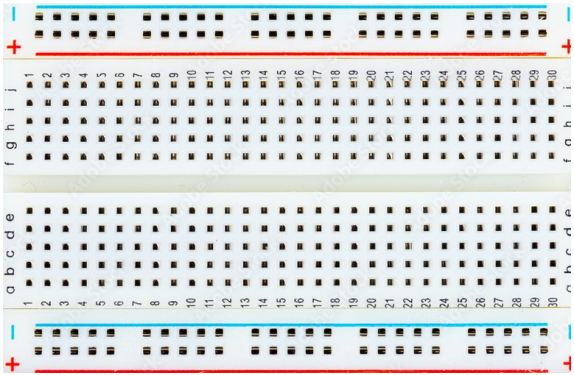
Elektronik

- Wie werden Schaltungen aufgebaut?
- Welche Bauteile werden (üblicherweise) verwendet?
- Wie funktionieren diese?
- Verwendung von Shields und Sensoren

Elektronik für den Arduino

Das Breadboard

- ermöglicht sehr freies Aufbauen von Schaltungen ohne Löten
- übersichtliches Arbeiten möglich
- Steckbrücken nötig
- a-e und f-j sind jeweils verbunden (Knoten)
- Spannung/Ground an Querleisten üblich
- bei großen Boards sind die Querleisten manchmal unterbrochen!



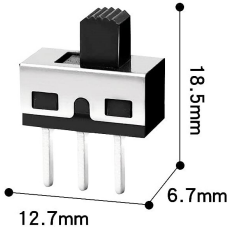
Das Breadboard

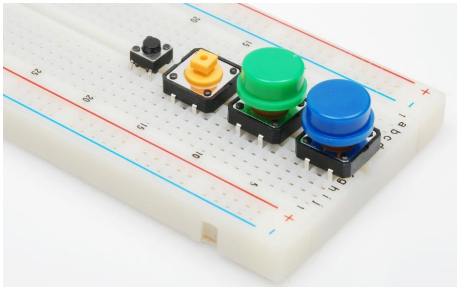
- Empfehlung: WAGO Klemmen für Bananenstecker/Litze Kombination
- Tischnetzteile können so gut mit dem Breadboard verbunden werden

Elektronik für den Arduino

Bauteile: Schalter/Taster

- Schalter ermöglichen Interaktionen der SuS
- Pinbelegungen am besten mit Multimeter (Durchgangstest) selbst erproben
- Achtung: günstige Schalter/Taster sind nicht entprellt

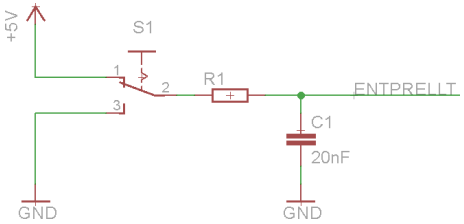




Elektronik für den Arduino

Bauteile: Schalter/Taster

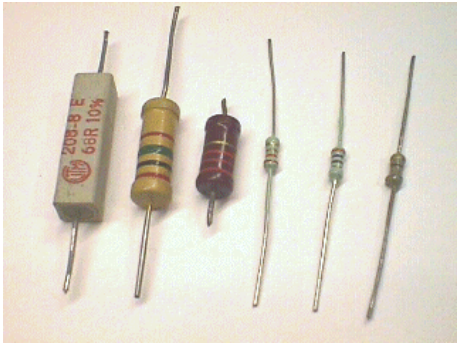
- Prellen kann mechanisch nie ganz unterbunden werden
- deutlich reduziert durch hochwertigere Taster (Digitast)
- durch Tiefpassfilter eliminierbar



Elektronik für den Arduino

Bauteile: Widerstand

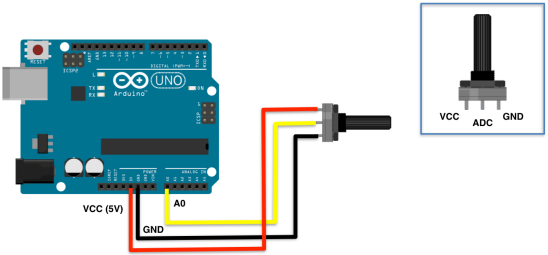
- DC/AC/PWM: $R = \frac{U}{I}$
- als Vorwiderstand für LEDs an 5V: $220\,\Omega$
- als Pulldown-Widerstand für Eingänge (ca. $10\,\mathrm{k}\Omega$)
- als Vorwiderstand bei Dioden(-Tests)

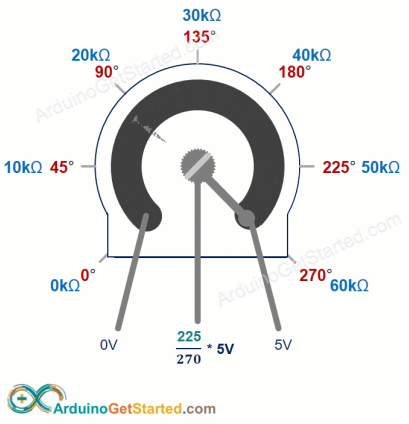


Elektronik für den Arduino

Bauteile: Potentiometer

- regelbarer Widerstand (meist mit Drehknopf)
- Achtung: immer mit Vorwiderstand! (sonst ggf. $R \approx 0\,\Omega$ möglich)
- kann als Eingabegerät für Arduino genutzt werden, indem man eine variable Spannung mit `analogRead()` ausliest (Spannungsteiler):





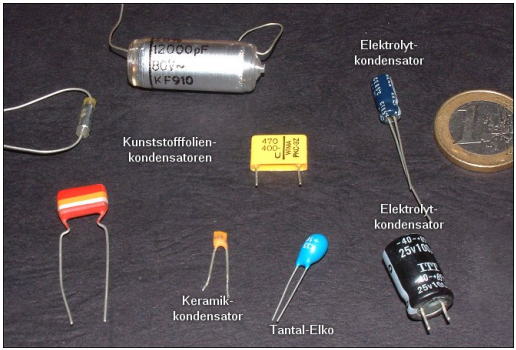
Elektronik für den Arduino

Bauteile: Kondensator

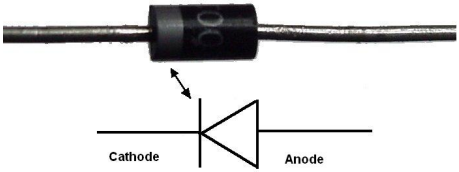
- Kapazität in $\mu\mathrm{F}, \mathrm{nF}, \mathrm{pF}$
- DC: Speichern Ladungen
- AC: wirkt wie Widerstand
- $|X_C| = \frac{1}{\omega C}$

→ durchlässig für hohe Frequenzen

- Elkos: hohe Kapazitäten, aber korrekte Polung wichtig!
- Anwendung: Hochpass, Tiefpass, Schwingkreis



- Dioden besitzen Sperrichtung/Durchlassrichtung
- vereinfacht: Bei Durchlassrichtung konstanter Spannungsabfall von $\approx 0.6\,\mathrm{V}$, unabhängig vom Strom
- in Durchlassrichtung $R \approx 0$ (immer Vorwiderstand bei Experimenten benutzen)
- Anwendung: Gleichrichtung von Wechselspannung, Signalverarbeitung



Elektronik für den Arduino

Bauteile: Leuchtdiode / LED

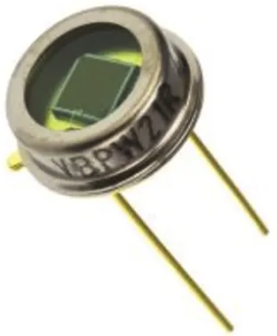
- zunächst: gleiche Eigenschaften wie Diode!
- zusätzlich aber mit Lichtemission
- Spannungsabfall: rot/$\pm 2\,\mathrm{V}$ bis blau/$\pm 3\,\mathrm{V}$
- für: Anzeige, Spiele, Debugging,...
- Vorwiderstand so dimensionieren, dass Stromstärke $\pm 20\,\mathrm{mA}$:

$$R = \frac{5\,\mathrm{V} - 1.7\,\mathrm{V}}{20\,\mathrm{mA}} \approx 200\,\Omega$$

Elektronik für den Arduino

Bauteile: Photodiode

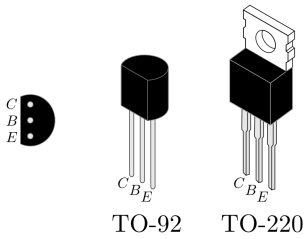
- durch Lichteinstrahlung geschaltete Diode
- sensitiv für vorgegebene Wellenlängen
- ermöglicht Reaktionen auf hell/dunkel Zustände

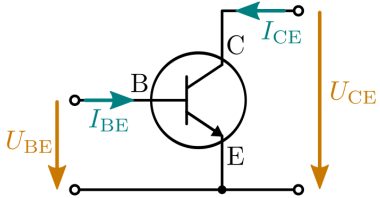


Elektronik für den Arduino

Bauteile: Transistor (bipolar)

- Prinzip: kleine Basisströme bewirken große Kollektorströme
- Betrieb als Schalter: ausreichend großer Basisstrom $> 200\,\mu\mathrm{A}$ schaltet CE-Strecke komplett durch (Achtung: Widerstand $R_{CE} \approx 0$)
- Transistoren sind sehr schnell, auch PWM-Verstärkung möglich bei Motoren (DC/Servo)
- Eingang/Ausgang invertiert

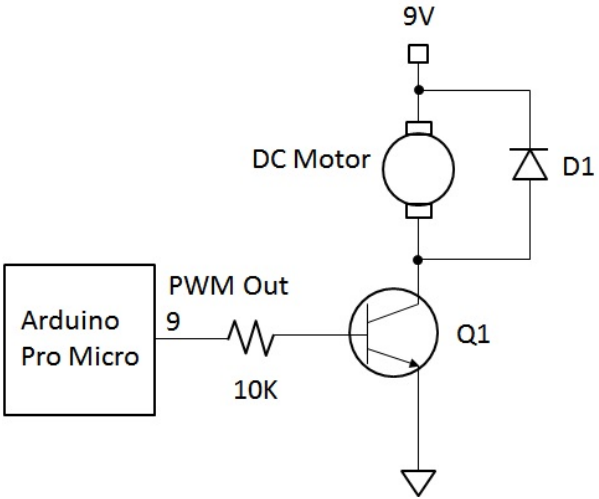




Elektronik für den Arduino

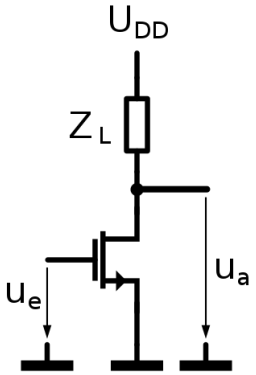
Bauteile: NPN Transistor für DC-Motor

- 10k-Widerstand für Begrenzung des Basisstroms
- Transistor sorgt per PWM Ansteuerung für Stromfluss von 9V-Block durch Motor
- Diode schützt vor Spannungsspitzen (Induktion in Motorspule)



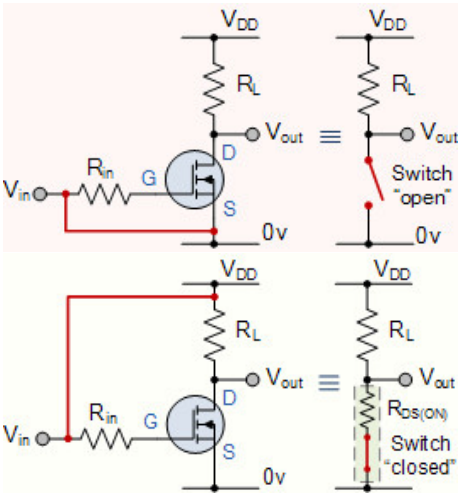
Bauteile: Transistor (MOSFET)

- so wie bipolar Transistor, aber mit Spannung statt Strom
- kleine Gate-Spannung erzeugt große Source-Spannung
- noch besser als Schalter geeignet, auch für große Leistungen



- geringe Spannung (z.B. GND) an Gate
- Transistor öffnet nicht

- ausreichende Spannung (z.B. Versorgungsspannung) an Gate
- Transistor schaltet komplett durch



Elektronik für den Arduino

PWM vs. Analogspannung

- Arduinos haben i.d.R. keine Analogausgänge
- stattdessen PWM (pulse width modulation) um DC-Spannung zu simulieren
- viele Bauteile setzen nur Zeitmittelwert (RMS) um, dann ist PWM ein guter Ersatz
- z.B DC-Motor, LEDs, Lüfter