Arbeitsblatt 1 – Grundlagen Elektronik

Übungen: Lösungen

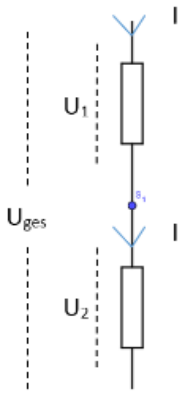
Aufgabe 1: Ohm'sches Gesetz

E12

1 Ω	10 Ω	100 Ω	1 kΩ	10 kΩ	100 MΩ	1 MΩ	10 MΩ
1,2 Ω	12 Ω	120 Ω	1,2 kΩ	12 kΩ	120 MΩ	1,2 MΩ	12 MΩ
1,5 Ω	15 Ω	150 Ω	1,5 kΩ	15 kΩ	150 MΩ	1,5 MΩ	15 MΩ
1,8 Ω	18 Ω	180 Ω	1,8 kΩ	18 kΩ	180 MΩ	1,8 MΩ	18 MΩ
2,2 Ω	22 Ω	220 Ω	2,2 kΩ	22 kΩ	220 MΩ	2,2 MΩ	22 MΩ
2,7 Ω	27 Ω	270 Ω	2,7 kΩ	27 kΩ	270 MΩ	2,7 MΩ	27 MΩ
3,3 Ω	33 Ω	330 Ω	3,3 kΩ	33 kΩ	330 MΩ	3,3 MΩ	33 MΩ
3,9 Ω	39 Ω	390 Ω	3,9 kΩ	39 kΩ	390 MΩ	3,9 MΩ	39 MΩ
4,7 Ω	47 Ω	470 Ω	4,7 kΩ	47 kΩ	470 MΩ	4,7 MΩ	47 MΩ
5,6 Ω	56 Ω	560 Ω	5,6 kΩ	56 kΩ	560 MΩ	5,6 MΩ	56 MΩ
6,8 Ω	68 Ω	680 Ω	6,8 kΩ	68 kΩ	680 MΩ	6,8 MΩ	68 MΩ
8,2 Ω	82 Ω	820 Ω	8,2 kΩ	82 kΩ	820 MΩ	8,2 MΩ	82 MΩ
Grundwerte	→ über Multiplikatoren →						

a)

- 9 V liegen an, 2 V fallen an der LED ab, verbleiben 7 V, die am Widerstand abfallen sollen.
- Es liegt eine Reihenschaltung vor, demnach fließt der Sollstrom von 20 mA = 0,02 A durch die LED und den Vorwiderstand.



Es berechnet sich demnach

$$U = R \cdot I \Leftrightarrow R = \frac{U}{I} = \frac{7}{0,02} = \underline{\underline{350\ \Omega}} \approx 330\ \Omega \quad \text{(E-Reihe)}$$

b)

5 Volt Vcc:

$$U = R \cdot I \Leftrightarrow R = \frac{U}{I} = \frac{5-2}{0,02} = \frac{3}{0,02} = \underline{\underline{150\ \Omega}} = \underline{\underline{150\ \Omega}} \quad \text{(E-Reihe)}$$

3 Volt Vcc:

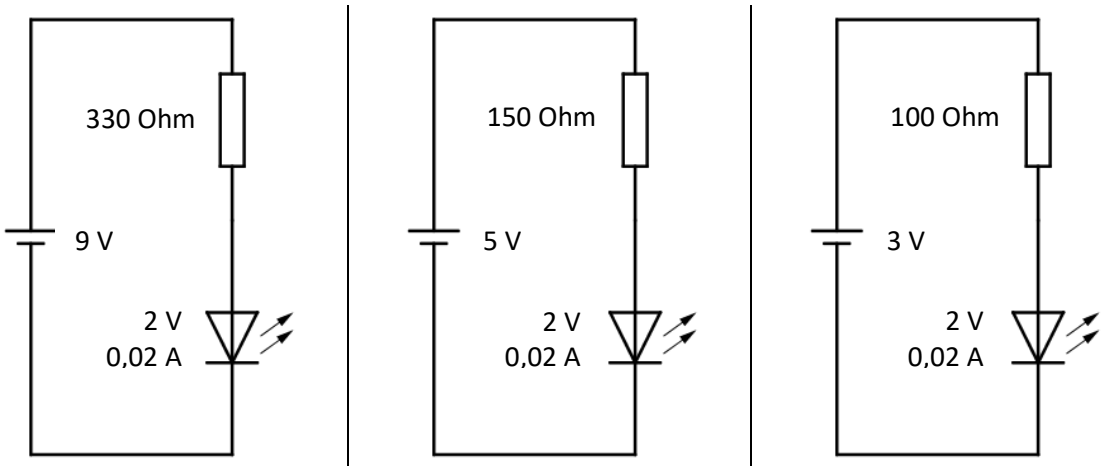
$$U = R \cdot I \Leftrightarrow R = \frac{U}{I} = \frac{3-2}{0,02} = \frac{1}{0,02} = \underline{\underline{50\ \Omega}} \approx 100\ \Omega \quad \text{(E-Reihe)}$$

c)
siehe (b)

150 Ohm (E12)
100 Ohm (E12)

d) Ergänzen Sie die unvollständige Schaltbilder mit Ihren Werten aus (c). Sie haben damit die Dimensionierung einer Standardschaltung vorgenommen und diese in Zukunft als Vorlage zur Verfügung:

>>Eine rote LED wird mit einem Vorwiderstand von 330 Ohm an einer (Mikrocontrollerspannung) von 3 Volt betrieben.<<



e) Realer Aufbau für Arduino

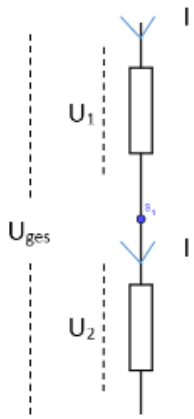
U=R*I → I=U/R = 3,3 Volt : 330 Ohm = 10 mA fließen durch die LED --- ideal für Dauerbetrieb

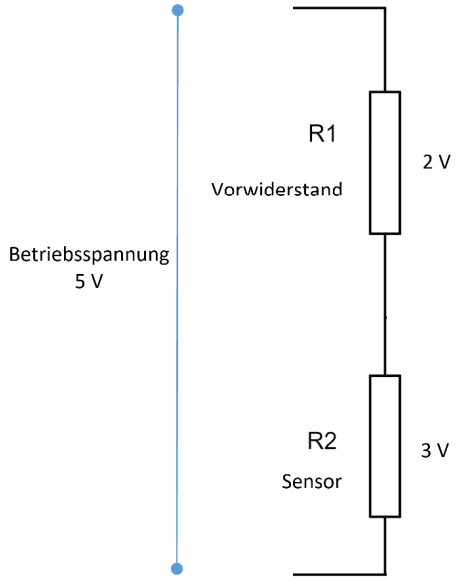
U=R*I → I=U/R = 5 Volt : 220 Ohm = 23 mA fließen durch die LED --- geht grade noch für Dauerbetrieb

U=R*I → I=U/R = 5 Volt : 330 Ohm = 15 mA fließen durch die LED --- ideal für Dauerbetrieb

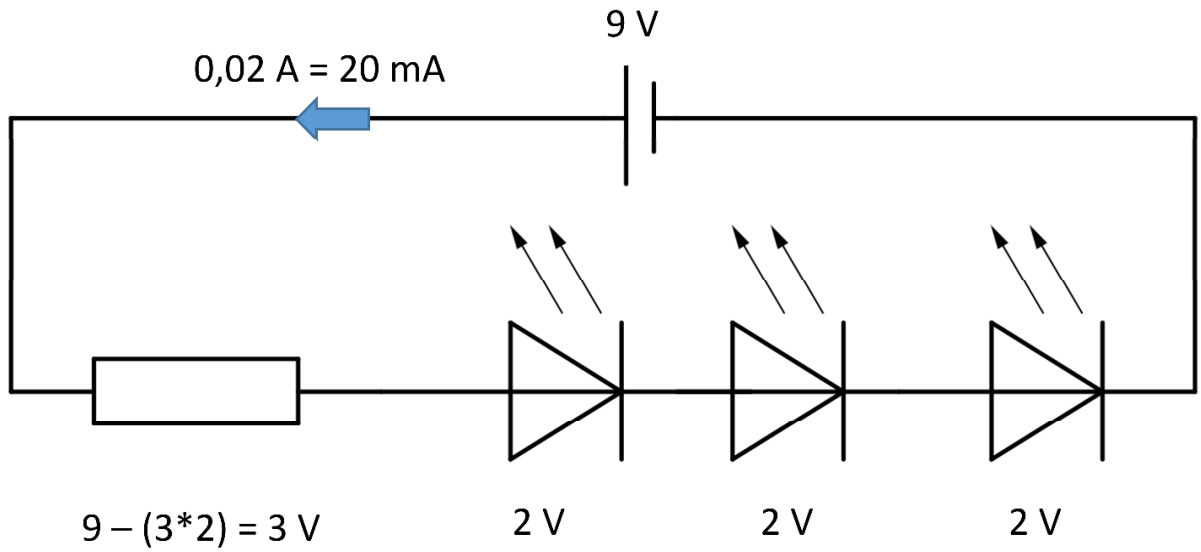
f) Spannungsteiler

Ein Bauelement verlangt weniger als die volle Betriebsspannung, was fast immer das Fall ist. Demnach ist ein weiteres Bauelement vorzuschalten, an dem die Spannungsdifferenz zwischen Betriebsspannung benötigter Spannung abfällt. Das dafür geeignete Bauelement ist der Widerstand. An ihm müssen 5-3=2 Volt abfallen. Wie ist er zu verschalten? Die Gesamtspannung (hier: Betriebsspannung) von 5 Volt, $U_{ges.}$, soll sich aufteilen in die Spannung am Sensor und an die überschüssige Spannung am Widerstand. Demnach ist eine Reihenschaltung notwendig, es entsteht ein **Spannungsteiler**.





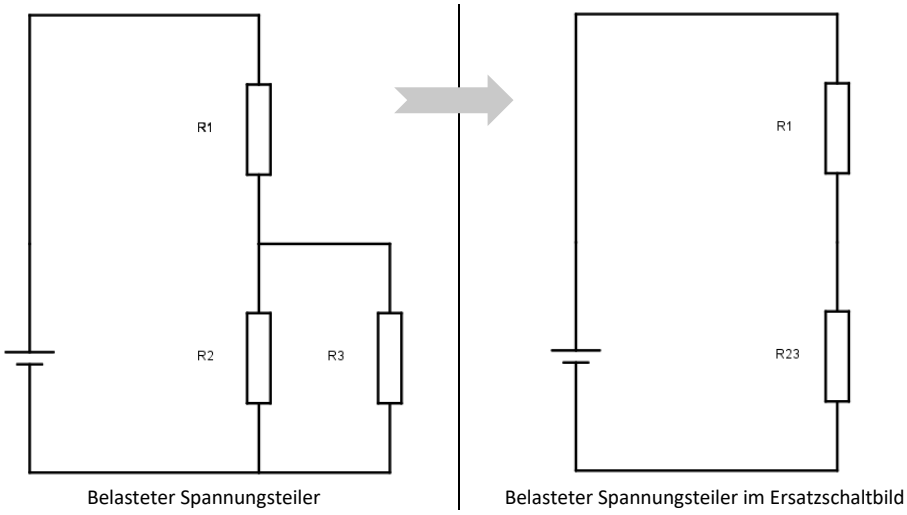
g)



Entsprechend

U=R*I → R=U/I = 3 Volt : 0,02 A = **150 Ohm = R**

Aufgabe 2:



Belasteter Spannungsteiler | Belasteter Spannungsteiler im Ersatzschaltbild

a) Berechnen Sie den Ersatzwiderstand R_{23} für eine Spannungsquelle von 9 V und R_2=R_3=470 Ohm, R_1=100 Ohm.

$$\frac{1}{R_{ges}} = \frac{1}{R_1} + \frac{1}{R_2} \Leftrightarrow \frac{1}{R_{ges}} = \frac{R_1 + R_2}{R_1 \cdot R_2} \Leftrightarrow R_{ges} = \frac{R_1 \cdot R_2}{R_1 + R_2}$$

$$R_{23} = \frac{R_1 \cdot R_2}{R_1 + R_2} = \frac{470 \cdot 470}{470 + 470} = \frac{470 \cdot \cancel{470}}{2 \cdot \cancel{470}} = \frac{470}{2} = \underline{\underline{235\ \Omega}}$$

Das geht auch kürzer **ohne Rechnung**. Zwei parallel geschaltete gleich große Widerstände halbieren sich in ihrem Wert, also ist der Ersatzwiderstand R_{23} direkt mit 235 Ohm angebbar.

b) Berechnen Sie den Ersatzwiderstand R_{12}, wenn R_3 nicht vorhanden ist.

Es liegt also eine Reihenschaltung von R_1 und R_2 vor. Der Ersatzwiderstand R12 berechnet sich zu $R_{12} = R_1 + R_2 = 100 + 470 = \underline{\underline{570\ \Omega}}$.

Aufgabe 3: Widerstand und Leistung

Üblich im Mikrocontrollerbereich sind ¼-Watt Widerstände. Es wurde im Text behauptet, dass ein an 3 Volt anliegender ¼-Watt Widerstand einen maximalen Stromfluss von 83 mA verträgt. Weisen Sie dies rechnerisch nach.

$P = U \cdot I$ → Mit $U = R \cdot I$ eingesetzt auch gilt auch $P = I^2 \cdot R$ oder $P = \frac{U^2}{R}$.

Leistung = Spannung · Stromstärke
P = U · I
¼ = 3 · I | · $\frac{1}{3}$

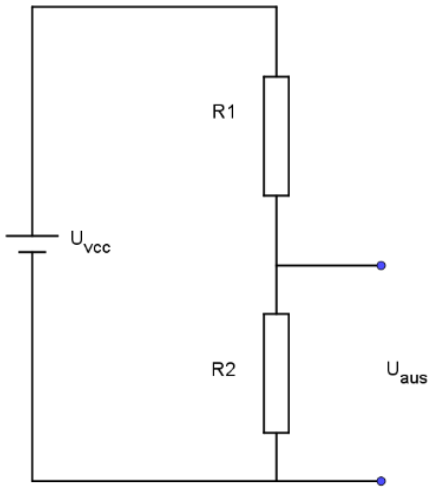
$$\frac{1}{4} \cdot \frac{1}{3} = I$$

$$I = \frac{1}{12}$$

$$I = \frac{1}{12} \approx 0{,}083\ A = 83\ mA$$

In der Praxis wird natürlich dafür gesorgt, dass dieser Wert bei Dauerbetrieb nicht erreicht wird, damit der Widerstand nicht am Limit betrieben wird. Alternativ könnte man auch auf etwa ½-Watt Widerstände umsteigen.

Aufgabe 4: Herleitung Spannungsteilerformel



Die Spannungen verhalten sich wie die dazu gehörigen Widerstände, also

$$\frac{U_{vcc}}{R_1 + R_2} = \frac{U_{aus}}{R_2} \Leftrightarrow R_2 \cdot \frac{U_{vcc}}{R_1 + R_2} = U_{aus} \Leftrightarrow \underline{\underline{U_{aus} = \frac{R_2 \cdot U_{vcc}}{R_1 + R_2}}}$$

$$U_{aus} = U_{ein} \cdot \frac{R_2}{R_1 + R_2}$$

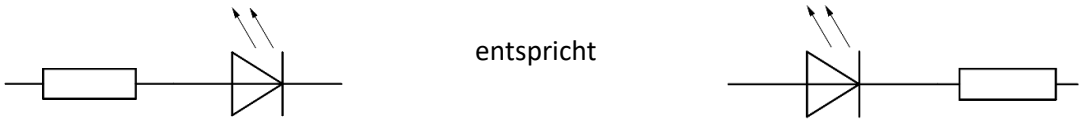
Aufgabe 5: Modelle

Modellentsprechungen:

- Strom - Wasserfluss
- Spannung - Wasserdruck
- Widerstand - Leitungsquerschnitt

Das Modell ist seit Urzeiten in der Lehre auf allen Niveaustufen bewährt, da es insbesondere sehr anschaulich ist. Ferner verdeutlicht es die Zusammenhänge im elektrischen Stromkreis auch inhaltlich leistungsfähig und weitestgehend korrekt.
Man kann sich Spannung durchaus als "Druck auf die Elektronen" vorstellen, je mehr Spannung (Druck) desto mehr Strom pro Zeit bei gleichem Querschnitt. Das wiederum führt zu einer größeren Wärmeentwicklung, worüber der Begriff zur elektrischen *Leistung*, P (in Watt), geschlagen wird.
Der Widerstand, vorgestellt als Leitungsquerschnitt einer Wasserleitung, ist ebenfalls geringer, wenn bei gleicher Wassermenge (Stromfluss) ein größeres Rohr genommen wird. Wird der Leitungsquerschnitt verringert, wird dem Strom-/Wasserfluss ein größerer Widerstand entgegen gesetzt.
Soweit passt alles sehr gut zusammen und ist so anschaulich wie es geht.

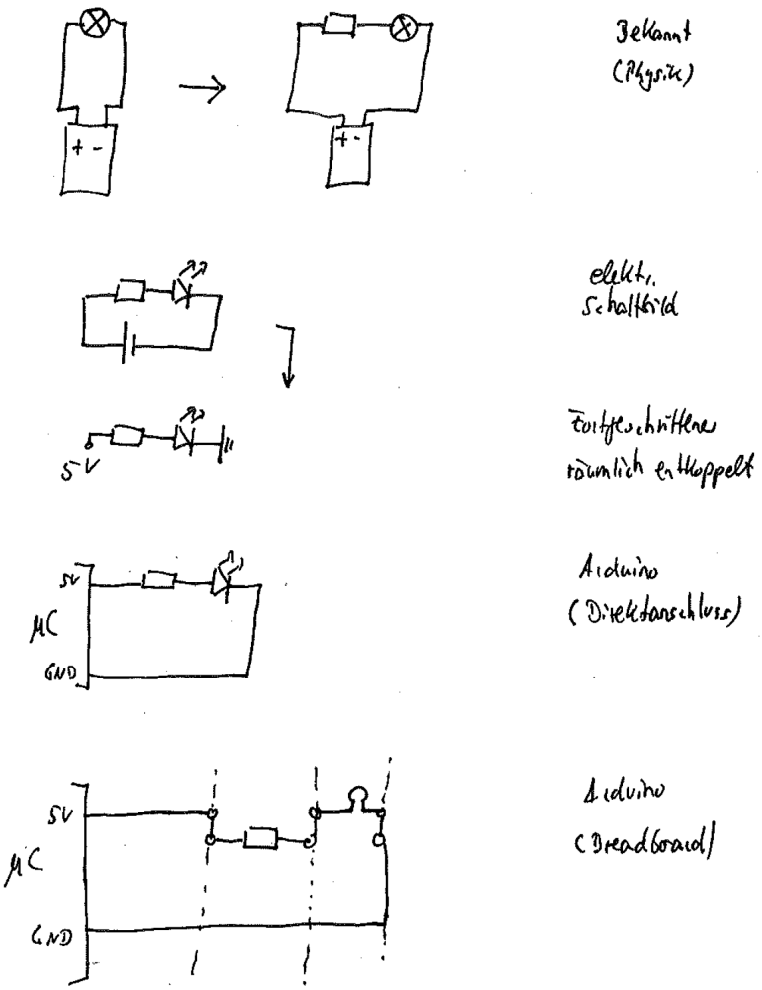
Ein echtes **Problem** ist die **zeitliche Abfolge**. Im Wassermodell fließt Wasser von A nach B, sequentiell. In der Elektronik **passiert alles gleichzeitig**. Dadurch wird es auch irrelevant, ob der "Vor"widerstand einer LED tatsächlich vor (bzgl. welchem Bezugspunkt?) oder hinter der LED geschaltet wird, was zunächst gänzlich unintuitiv ist. Ebenso wie die Fortführung dieser Zusammenhänge. Es ist nämlich egal in welcher Reihenfolge Bauelemente verschaltet werden, solange kein Knotenpunkt vorhanden ist, sprich, auf einer einzelnen Leitung lassen sich die Bauteile beliebig anordnen.



Zudem ist Spannung als Potential**differenz** zu verstehen. Potential (Vermögen; das Vermögen Arbeit zu verrichten) ist immer bezüglich eines Referenzpunktes angegeben. Dieser wird zuallermeist als Ground/Erde definiert und als 0 Volt betrachtet, so dass alle anderen Werte dann keine Differenzbildung mehr benötigen (x – 0 = x). Allerdings besteht zwischen einer Stelle von +3 Volt und einer mit +5 Volt (bzgl. der Erde) eben auch eine Potentialdifferenz – und damit eine Spannung – von 2 Volt. Das ist allerdings unterrichtlich durch Erklärung alles aufzufangen und kein Problem des Modells.

Die Reduktion von durch Modellvorstellungen hervorgerufene Fehlvorstellungen ist – wie bei allen Modellen – durch ein Verständnis sowohl des Modells wie auch seiner Grenzen zu erreichen. Es fließen Elektronen – auch wenn in der Elektronik "Löcher", also positive Ladungsträger, betrachtet werden – und kein Wasser, daher muss der Punkt kommen, wo sich Modell und Realität unterscheiden. Die explizite Betonung des Modellcharakters eines Modells und sein sinnvoller Einsatz obliegt der Lehrkraft. Explizite Thematisierungen der Modellgrenzen – bspw. durch Unterrichtsgespräch und Übungsaufgaben (wie diese) – helfen dabei.

Aufgabe 6: Schaltbildebenen und Schaltbildarten



Bekannt aus dem experimentellen Aufgabe des Physikunterrichts der Mittelstufe sind Schaltbilder in der "Physikdarstellung", welche nicht direkt der der Elektronik/Elektrotechnik entspricht. Hier sind – noch recht einfache – Gewöhnungsleistungen zu erbringen und durch Gebrauch und Übungen zu verinnerlichen.

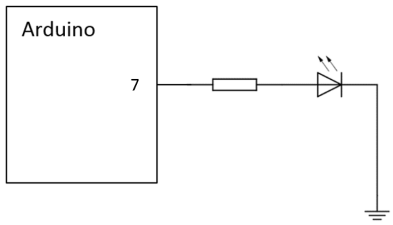
Problematischer wird es bei der räumlich entkoppelten Darstellung. Da Schaltbilder sehr schnell komplex werden, spart man - notwendiger Weise - zur Komplexitätsreduktion Leitungen ein, in dem die Zugänge zu den Polen (+ und -) nicht mehr tatsächlich als Leitung zur Quelle ausgeführt werden, sondern mit den entsprechenden Symbolen (GND) und (Vcc) enden. Das ist gewöhnungsbedürftig und muss intensiver beübt werden. Ist das nicht gewünscht, wäre es auch möglich stets bei vollständigen Schaltbildern "klassischer Art" (2. von oben) zu bleiben, wobei das eine von der Stufe abhängige, wichtige didaktische Entscheidung darstellt. Hier ist klar zu betonen, dass damit das verständige Lesen von realistischen Schaltbildern nicht wirklich abgedeckt werden würde.

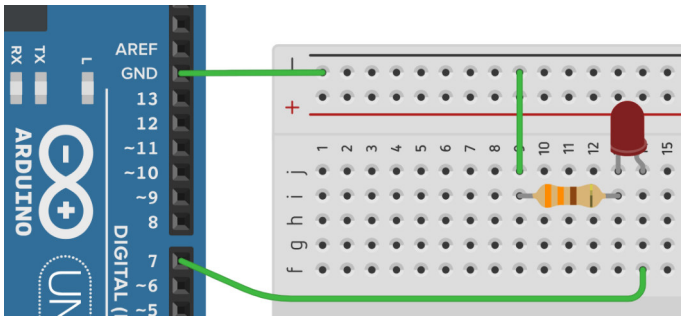
Der Anschluss an einen Mikrocontroller (uC) verlangt nochmals eine Abstraktionsleistung, da Vcc und GND nicht mehr sichtbar sind, sie werden durch den uC verdeckt. Wird dann noch ein **Steckbrett** (Breadboard) benutzt, kommt als **große Problemdimension** hinzu, dass sich Schaltbilder nicht mehr direkt auf das Steckbrett übertragen lassen (Spalten sind dort durchkontaktiert), was eine zusätzliche Abstraktions- und Zwischenebene darstellt.
Unterrichtlich muss das schrittweise, ausführlich erklärt und immer wieder beübt werden. Explizite Übungen zum Transfer einfacher Schaltungen auf einen Steckbrettaufbau empfehlen sich. Ferner ist hier auf sauberen Aufbau mit genügend Abständen und einem Aufbau möglichst nahe am Schaltbild bei Steckbrettnutzung zu achten.

Übungen: Lösungen

Aufgabe 1:

a)





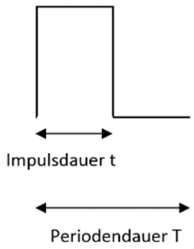
b)

0010_AB3_externeSchaltungen_LEDblinkt_lsg

```
int ledPin = 7;                  //globale Variable, die eingebaute LED

void setup()
{
  pinMode(ledPin, OUTPUT);      //eingebaute LED als Ausgabe definieren
}

void loop()
{
  digitalWrite(ledPin, HIGH);   // LED anschalten – HIGH-Pegel (5V)
  delay(1000);                  // 1000 ms = 1 sec. lang warten
  digitalWrite(ledPin, LOW);    // LED ausschalten – LOW-Pegel (0V)
  delay(1000);                  // 1000 ms = 1 sec. lang warten
}
```

c)

Ziel ist den Begriff zum Leben zu erwecken als Verhältnis von HIGH zu LOW. Die ‘Periodendauer‘ wird als Gesamtheit der beiden Anteile erfasst.
Damit sich ‘möglichst wenig Änderungen im Code‘ ergeben sind einfach zwei Konstanten zu definieren und mit 1000 zu multiplizieren, um auf die geforderten Sekunden von den ms in der delay-Funktion zu kommen.



0010_AB3_externeSchaltungen_tastgrad_lsg

```
/*
LED blinkt mit definiertem Tastgrad ( Tastgrad = t:T; Verhältnis HIGH zur
Periodendauer)
500/1500 als Tastgrad würde bspw. bedeuten, dass 1/3 der Zeit (500 von 1500
msec.) ein HIGH vorliegt
Zeitangaben alle in Millisekunden (msec.)
Tastgrad wird hier als Verhältnis von Zeiten codiert und NICHT in Prozent
betrachtet
*/

int ledPin = 13;      //globale Variable, die eingebaute LED
int highanteil_t = 100;
int periodendauer_T = 2000;
int lowanteil = periodendauer_T - highanteil_t;

void setup()
{
  pinMode(ledPin, OUTPUT);  //eingebaute LED als Ausgabe definieren
}

void loop()
{
  digitalWrite(ledPin, HIGH);  // LED anschalten – HIGH-Pegel (5V)
  delay(highanteil_t);                  //x sec. lang HIGH
  digitalWrite(ledPin, LOW);   // LED ausschalten – LOW-Pegel (0V)
  delay(lowanteil);                 // x sec. lang LOW
}
```

d)

Aufbau: machen (analog (a))

Widerstandswerte für verschiedenfarbige LEDs wären unterschiedlich, da U und I für diese LEDs verschieden sind. Als groben Richtwert für Prototypen geht bei 330 Ohm aber bei keiner LED etwas zu Bruch, daher ist der gleiche Vorwiderstandswert für alle Farben ok.

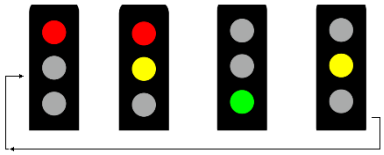
Man kann auch einfach drei rote LEDs verwenden, wie in der Aufgabe für ausreichend befunden, am Quellcode ändert das nichts.

Code: wie (b) bzw. AB2

0010_AB3_externeSchaltungen_3ampelLEDsAbwechselndBlinken_lsg

```
//lässt drei (Ampel)LEDs abwechselnd blinken

int ledPinRot = 8;
int ledPinGelb = 9;
int ledPinGruen = 10;
int delaytime = 1000;   //Verzögerungszeit zwischen dem LED-Umschalten


void setup()
{
  pinMode(ledPinRot, OUTPUT);
  pinMode(ledPinGelb, OUTPUT);
  pinMode(ledPinGruen, OUTPUT);
}

void loop()
{
  phase1();
  delay(delaytime);
  phase2();
  delay(delaytime);
  phase3();
  delay(delaytime);
}



void phase1()   //rot
{
    digitalWrite(ledPinRot, HIGH);
    digitalWrite(ledPinGelb, LOW);
    digitalWrite(ledPinGruen, LOW);
}

void phase2()   //gelb
{
    digitalWrite(ledPinRot, LOW);
    digitalWrite(ledPinGelb, HIGH);
    digitalWrite(ledPinGruen, LOW);
}

void phase3()   //gruen
{
    digitalWrite(ledPinRot, LOW);
    digitalWrite(ledPinGelb, LOW);
    digitalWrite(ledPinGruen, HIGH);
}
```

e) Erweitern sie Ihr voriges Programm so, dass die Ampelphasen[1] durchlaufen werden.



0010_AB3_externeSchaltungen_Ampelphasen_lsg

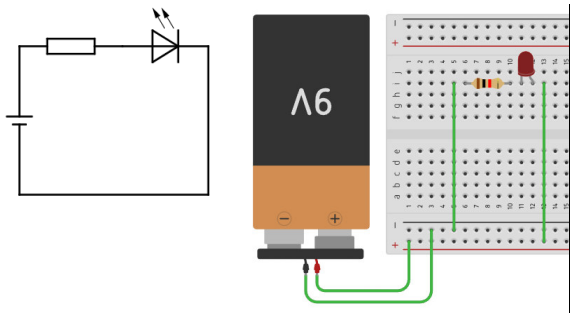
wie (d), ergänzt um eine 4. Phase

```
//lässt drei (Ampel)LEDs gemäß der Ampelphasen blinken

int ledPinRot = 8;
int ledPinGelb = 9;
int ledPinGruen = 10;
int delaytime = 1000;   //Verzögerungszeit zwischen dem LED-Umschalten


void setup()
{
  pinMode(ledPinRot, OUTPUT);
  pinMode(ledPinGelb, OUTPUT);
  pinMode(ledPinGruen, OUTPUT);
}

void loop()
{
  phase1();
  delay(delaytime);
  phase2();
  delay(delaytime);
  phase3();
  delay(delaytime);
  phase4();
  delay(delaytime);
}


void phase1()   //rot
{
    digitalWrite(ledPinRot, HIGH);
    digitalWrite(ledPinGelb, LOW);
    digitalWrite(ledPinGruen, LOW);
}

void phase2()   //rot-gelb
{
    digitalWrite(ledPinRot, HIGH);
    digitalWrite(ledPinGelb, HIGH);
    digitalWrite(ledPinGruen, LOW);
}

void phase3()   //gruen
{
    digitalWrite(ledPinRot, LOW);
    digitalWrite(ledPinGelb, LOW);
    digitalWrite(ledPinGruen, HIGH);
}

void phase4()   //gelb
{
    digitalWrite(ledPinRot, LOW);
    digitalWrite(ledPinGelb, HIGH);
    digitalWrite(ledPinGruen, LOW);
}
```

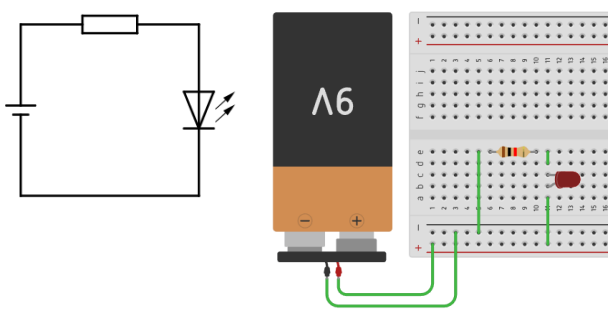
[1] https://commons.wikimedia.org/wiki/File:Cjam-neopixels-uk-traffic.svg

f)

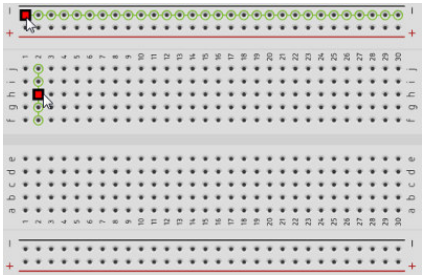
f.1)



Schaltung 1 und Aufbau

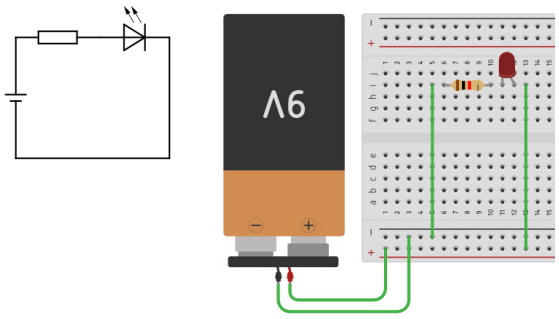


Schaltung 2 und Aufbau

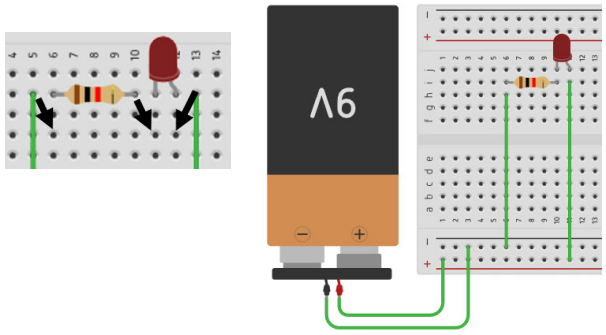


Ein Steckbrett hat eine besondere Struktur, die hier nicht berücksichtigt wurde, der häufigste Fehler – neben Wackelkontakten – beim Arbeiten mit Steckbrettern. Eine Spalte ist Durchverbunden, die Spannungsversorgungsreihen '+' und '–', sind ebenfalls durchverbunden:

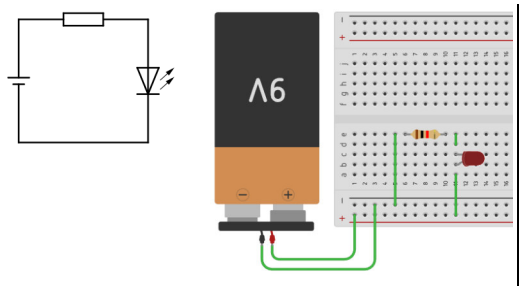
Es wurde in beiden Fällen die **räumliche Anordnung** der Bauteile gemäß Schaltplan 1:1 auf das Steckbrett übertragen, was nicht funktioniert. Die Bauteile müssen für die korrekte Durchkontaktierung alle um eine Spalte verschoben werden. Eine Kontrolle ist über den Abgleich zwischen Leitungen auf dem Schaltplan und Durchkontaktierung auf dem Steckbrett möglich. In der 2. Schaltung wurde zudem der *klassische Kurzschluss* erzeugt, in dem beide Beine der LED in eine – durchkontaktierte – Spalte gesteckt wurden.



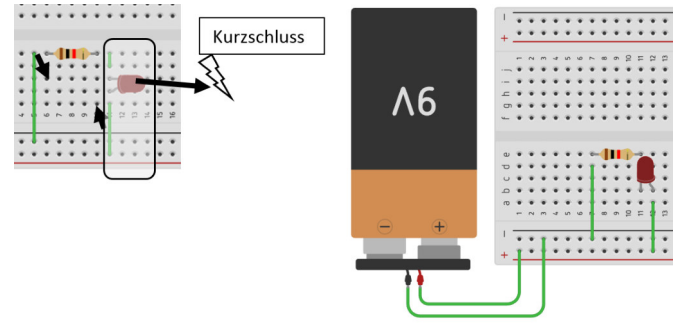
Schaltung 1 und Aufbau



Schaltung 1 - Korrekt



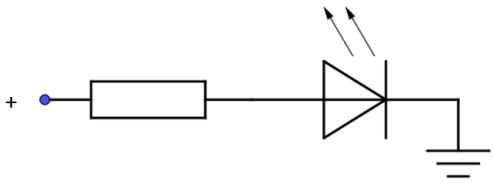
Schaltung 2 und Aufbau

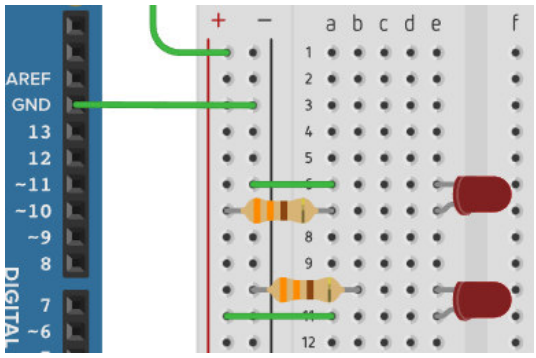


Schaltung 2 - Korrekt

f.2)
Beide Aufbauten auf dem Steckbrett funktionieren, die obere ist allerdings besser, da sie 1:1 das Schaltbild umsetzt. Das ist ein sehr wichtiger Faktor, insbesondere auch in der Lehre.

Ob der Vorwiderstand vor oder hinter der LED verbaut ist, ist solange egal, wie kein Verzweigungspunkt betroffen ist. Zudem sind die Begriffe ‘vor‘ und ‘hinter‘ relativ. Der Widerstand ist hier ‘vor‘ der LED, wenn von + nach – geschaut wird, wie in der Elektronik üblich.





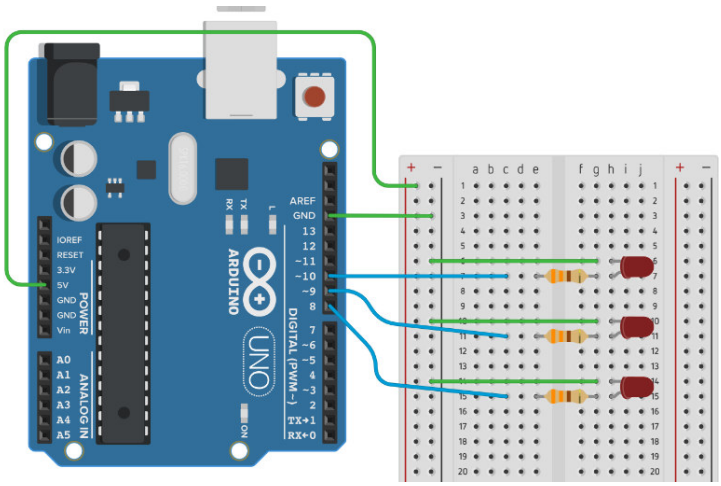
g) 4-Bit-Binärzähler

Aufbau wie (f) mit einer LED mehr, also 3 LEDs für die Binärzahlen

0010_AB3_externeSchaltungen_3BitBinaerzaehler_lsg

```
//3-Bit-Binärzähler
//CBA mit A=LSB=2^0, B=2^1, C=2^2

int A = 8;
int B = 9;
int C = 10;
int delaytime = 1000;   //Verzögerungszeit zwischen dem LED-Umschalten

void setup()
{
  pinMode(A, OUTPUT);
  pinMode(B, OUTPUT);
  pinMode(C, OUTPUT);
}

void loop()
{
  null();
  delay(delaytime);
  eins();
  delay(delaytime);
  zwei();
  delay(delaytime);
  drei();
  delay(delaytime);
  vier();
  delay(delaytime);
  fuenf();
  delay(delaytime);
  sechs();
  delay(delaytime);
  sieben();
  delay(delaytime);
}


void null()
{

    digitalWrite(A, LOW);
    digitalWrite(B, LOW);
    digitalWrite(C, LOW);
}

void eins()
{

    digitalWrite(A, HIGH);
    digitalWrite(B, LOW);
    digitalWrite(C, LOW);
}

void zwei()
{

    digitalWrite(A, LOW);
    digitalWrite(B, HIGH);
    digitalWrite(C, LOW);
}
```

0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111



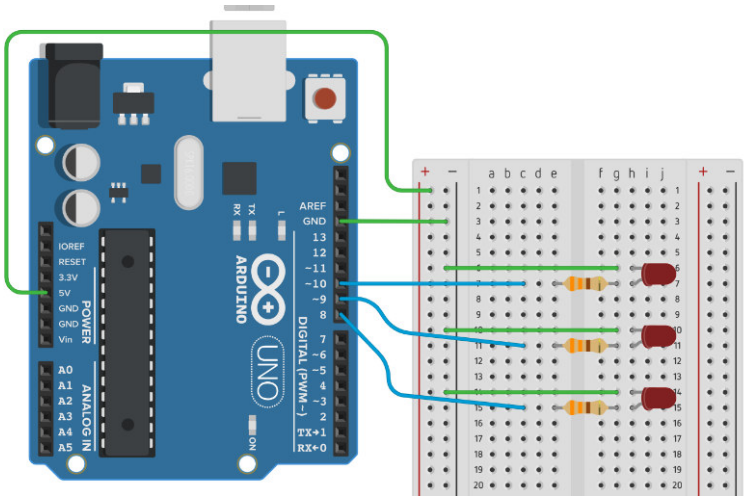
…

```
void sieben()
{

    digitalWrite(A, HIGH);
    digitalWrite(B, HIGH);
    digitalWrite(C, HIGH);
}
```

Erster Ansatz mit Schülern, aber
DAS GEHT SCHÖNER, man nutze ein Array…

0010_AB3_externeSchaltungen_3BitBinaerzaehler_mitArray_lsg

```
//3-Bit-Binärzähler
//CBA mit A=LSB=2^0, B=2^1, C=2^2

int A = 8;
int B = 9;
int C = 10;
int delaytime = 1000;   //Verzögerungszeit zwischen dem LED-Umschalten

int binaerzahlen[8][3] = {
  {0,0,0},    //0
  {0,0,1},    //1
  {0,1,0},
  {0,1,1},
  {1,0,0},
  {1,0,1},
  {1,1,0},
  {1,1,1},     //7
  };

void setup()
{
  pinMode(A, OUTPUT);
  pinMode(B, OUTPUT);
  pinMode(C, OUTPUT);
}
```



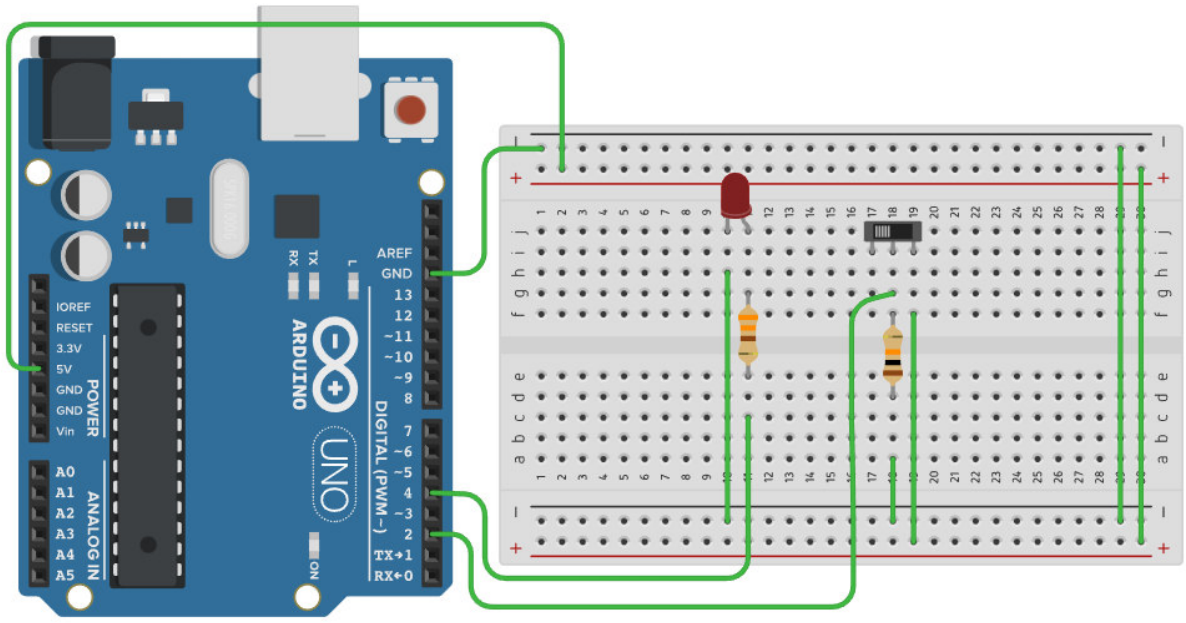
```
void loop()
{
  for( int dezimalzahl=0; dezimalzahl<8; dezimalzahl++)
  {
    zeigeBinaerziffer(dezimalzahl);
    delay(delaytime);
  }
}

//liest aus dem Array, in dem die Binärcodierung für die Dezimalzahlen 0-7
//stehen, die passenden Pegel für die LEDs A-C und steuert diese an
void zeigeBinaerziffer(int dezimalzahl)   //Reihenfolge CBA beachten
{
    digitalWrite(C, binaerzahlen[dezimalzahl][0] );  //MSB = C
    digitalWrite(B, binaerzahlen[dezimalzahl][1] );
    digitalWrite(A, binaerzahlen[dezimalzahl][2] );  //LSB = A
}
```

Ausführliche Bildreferenz:

Manuel Strehl (original)maix (dark version)Minoa (code optimisation) (https://commons.wikimedia.org/wiki/File:Traffic_lights_dark_yellow.svg), „Traffic lights dark yellow“, https://creativecommons.org/licenses/by-sa/2.5/legalcode

Übungen: Lösungen

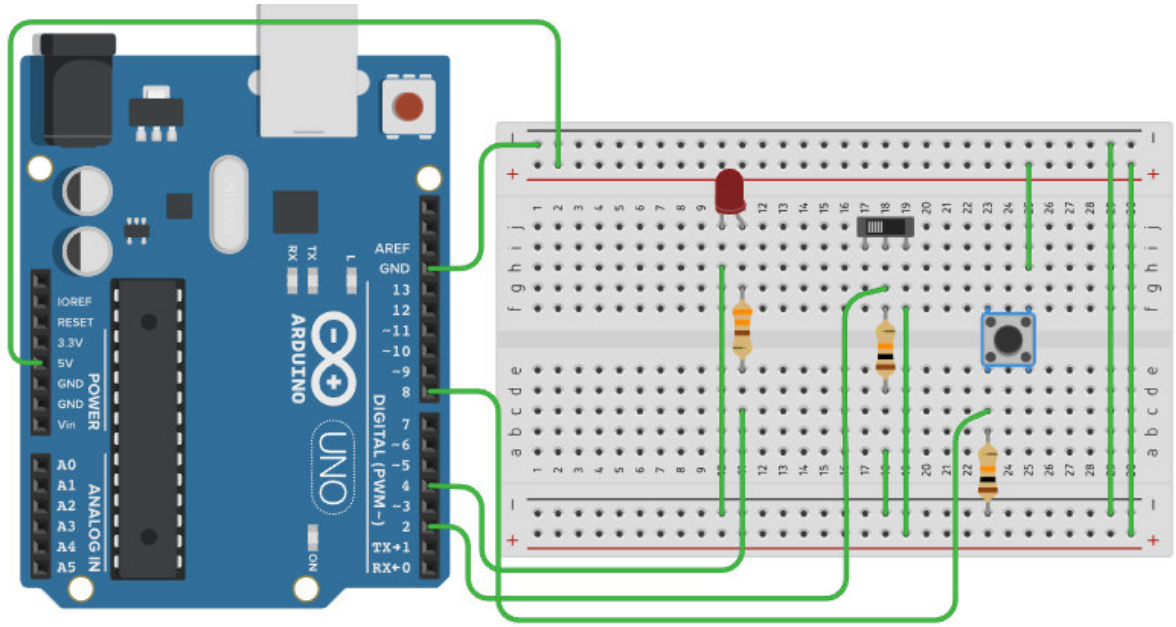
Aufgabe 1:
a)



0010_AB4_Taster_LEDmitSchalterAnschalten_lsg

```
//LED anschalten, solange Schalter geschlossen,
//ausschalten, wenn Schalter geöffnet

int ledpin = 4;
int schalterstatus= 2;

void setup()
{
  pinMode(ledpin, OUTPUT);
  pinMode(schalterstatus, INPUT);
  digitalWrite( ledpin, LOW);
}

void loop()
{
  if( digitalRead(schalterstatus) == LOW)
  {
    digitalWrite( ledpin, LOW);
  }
  else
  {
    digitalWrite( ledpin, HIGH);
  }
}
```

b)



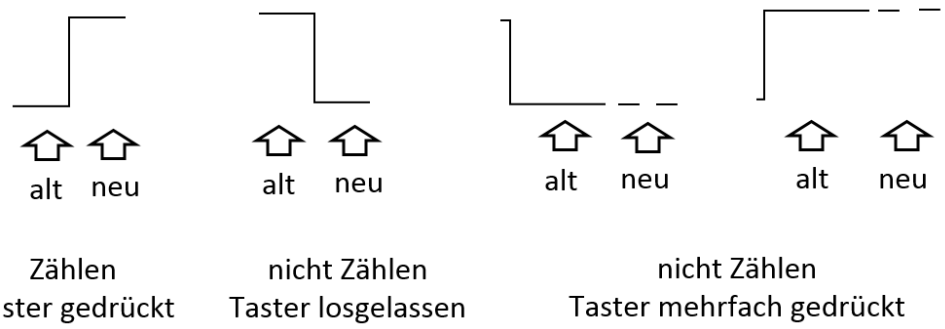
0010_AB4_Taster_LEDmitTasterAnschalten_lsg

```
//LED anschalten, solange Taster geschlossen,
//ausschalten, wenn Taster geöffnet

int ledpin = 4;
int schalterstatus = 2;
int tasterstatus = 8;

void setup()
{
  pinMode(ledpin, OUTPUT);
  pinMode(schalterstatus, INPUT);
  pinMode(tasterstatus, INPUT);
  digitalWrite( ledpin, LOW);
}

void loop()
{
  if( digitalRead(tasterstatus) == LOW)
  {
    digitalWrite( ledpin, LOW);
  }
  else
  {
    digitalWrite( ledpin, HIGH);
  }
}
```

c) Tastendrücke zählen:

```
int tasterPin = 2;              //Taster-Pin 2
int tasterWert = 0;             //Tasterstatus
int zaehler = 0;

void setup() {
 pinMode(tasterPin, INPUT);
}

void loop() {
 tasterWert = digitalRead(tasterPin);
 if(tasterWert == HIGH) {
  zaehler++;
 }
}
```

Der Code präsentiert die naheliegende Lösung, einfach jeden Tastendruck durch den entsprechenden HIGH-Pegel zu erfassen und den Zähler um einen hoch zu zählen. Das funktioniert allerdings nicht im Sinne des Erfinders, denn der Mikrocontroller arbeitet die loop-Schleife *immer wieder und* **sehr schnell** ab. Ein einzelner Tastendruck – insbesondere ein längeres drücken – kann also viele male erfasst werden und zu falschen Zählerwerten führen. Er muss über einen Umweg erfasst werden.

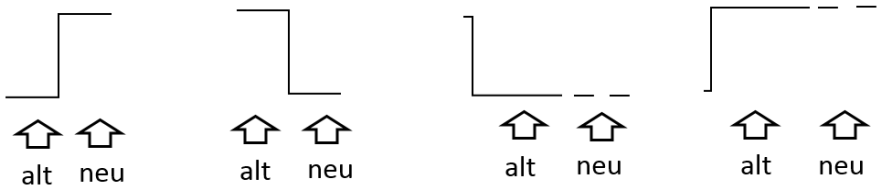


Nach der Abfrage muss der Pegel des Tasters in einer *Statusvariablen* gespeichert werden und bei der nächsten Abfrage wird der aktuelle Status mit dem vorigen verglichen. Sind bei Pegel gleich wird der Zählen nicht erhöht, denn hier liegt dann die besagte Mehrfacherfassung vor, die es zu vermeiden gilt. Sind die Pegel unterschiedlich muss noch geprüft werden, ob der Taster eingeschaltet oder ausgeschaltet wurde. Erfasst werden soll das Einschalten, daher muss der neue Wert einem HIGH entsprechen. Diesen jetzt aktuellen Wert in der Statusvariablen zwischenspeichern und alles beginnt von vorne.

d) Toggeln

Die Lösung greift die Idee aus (c) auf, der Zustand der LED sowie der Pegelwechsel müssen gespeichert und verglichen werden.

Pegelwechsel:



Zählen	nicht Zählen	nicht Zählen
Taster gedrückt	Taster losgelassen	Taster mehrfach gedrückt

LED-Zustände: low → high, high → low

Verfahren:

1. Relevant für den LED-Zustandswechsel sind nur Pegel*wechsel* (alterZustand != neuerZustand)
2. Wenn es einen Wechsel gab kann der Taster losgelassen oder gedrückt worden sein
3. Wir interessieren uns nur für den gedrückten Fall, was einem Wechsel LOW→HIGH entspricht
4. Hat ein Wechsel stattgefunden und war es der richtige, dann LED toggeln:
   Lang: wenn alterZustand==LOW dann neuerZustand=HIGH …
   Kurz: ledZustand = **!**ledZustand

0010_AB4_Taster_LEDtoggeln_lsg

V1: Erste Version – ausführlich, direkt

```
//LED toggeln
//V1

int ledpin = 4;
int tasterpin = 8;

int ledStatus = LOW;     //aktuellen Status (an, aus) der LED merken für den
Wechsel (toggeln)
int tasterstatusAktuell;
int tasterstatusAlt;


void setup()
{
  pinMode(ledpin, OUTPUT);
  pinMode(tasterpin, INPUT);
  digitalWrite( ledpin, LOW);
}

void loop()
{
  tasterstatusAlt = tasterstatusAktuell;          //letzten Zustand merken
  tasterstatusAktuell = digitalRead(tasterpin);   //neuen Zustand einlesen

  if( (tasterstatusAktuell == HIGH) && (tasterstatusAlt == LOW) )
  {
    // Umschalten (toggeln) der LED
    if(ledStatus == LOW)
    {
```

```
      ledStatus = HIGH;
    }
    else
    {
      ledStatus = LOW;
    }

    //LED umschalten  (toggeln)
    digitalWrite(ledpin, ledStatus);
  }
}
```

V2: Ergänzt um Ausgaben über den seriellen Monitor (Optional)

```
//LED toggeln
//V2 mit Testausgaben über den seriellen Monitor

int ledpin = 13;
int tasterpin = 8;

int ledStatus = LOW;    //aktuellen Status (an, aus) der LED merken für den
Wechsel (toggeln)
int tasterstatusAktuell;
int tasterstatusAlt;


void setup()
{
  Serial.begin(9600);
  pinMode(ledpin, OUTPUT);
  pinMode(tasterpin, INPUT);
  digitalWrite( ledpin, LOW);
}

void loop()
{
  tasterstatusAlt = tasterstatusAktuell;          //letzten Zustand merken
  tasterstatusAktuell = digitalRead(tasterpin);   //neuen Zustand einlesen

  if( (tasterstatusAktuell == HIGH) && (tasterstatusAlt == LOW) )
  {
    Serial.print("The button is pressed: ");

    // Umschalten (toggeln) der LED
    if(ledStatus == LOW)
    {
       ledStatus = HIGH;
       Serial.println("Turning LED on");
    }
    else
    {
      ledStatus = LOW;
      Serial.println("Turning LED off");
    }

    //LED umschalten  (toggeln)
    digitalWrite(ledpin, ledStatus);
  }
}
```

V3: Optimierter Code

```
//LED toggeln - optimierter Code
//V3

int ledpin = 13;
int tasterpin = 8;

int ledStatus = LOW;     //aktuellen Status (an, aus) der LED merken für den
Wechsel (toggeln)
int tasterstatusAktuell;
int tasterstatusAlt;


void setup()
{
  pinMode(ledpin, OUTPUT);
  pinMode(tasterpin, INPUT);
  digitalWrite( ledpin, LOW);
}

void loop()
{
  tasterstatusAlt = tasterstatusAktuell;          //letzten Zustand merken
  tasterstatusAktuell = digitalRead(tasterpin);   //neuen Zustand einlesen

  if( (tasterstatusAktuell == HIGH) && (tasterstatusAlt == LOW) )
  {
    //ledStatus = ( (ledStatus == HIGH) ? LOW : HIGH ); //Alternativer Code
    ledStatus = !ledStatus;           //Status umschalten (toggeln)
    digitalWrite(ledpin, ledStatus);  //LED umschalten  (toggeln)
  }
}
```

Je nach Taster kann es manchmal zu Flackern kommen, da die Taster, die Beschaltung und der Code das Bouncing-Problem noch ignorieren.

V4: mit Debouncing (per einfacher Verzögerung)

```
…
  if( (tasterstatusAktuell == HIGH) && (tasterstatusAlt == LOW) )
  {
    ledStatus = !ledStatus;           //Status umschalten (toggeln)
    delay(delaytime);  //10-50 ms reichen
    digitalWrite(ledpin, ledStatus);  //LED umschalten  (toggeln)
  }
```

Das Problem dabei ist, dass delay() das Programm blockiert, entsprechend auch nicht auf Tastereingaben in der Zeit reagiert. Die Lösung per Wartezeit ist daher nicht schön.

e)

0010_AB4_Taster_BlinkenUndLEDmitTasterAnschalten_millis_lsg

```
//eine LED anschalten, wenn Taster geschlossen
//eine zweite LED blinkt dabei dauerhaft
//
//V1 - hier ist die Programmblockade noch moderat aber schon erkennbar (100
msec.)

int ledpin = 13;
int ledSchaltbarPin = 11;
int tasterpin = 5;
int intervallzeit = 100;

void setup()
{
  pinMode(ledpin, OUTPUT);
  pinMode(ledSchaltbarPin, OUTPUT);
  pinMode(tasterpin, INPUT);
}

void loop()
{

  digitalWrite(ledpin, LOW);
  delay(intervallzeit);
  digitalWrite(ledpin, HIGH);
  delay(intervallzeit);

  if( digitalRead(tasterpin) == LOW)
  {
    digitalWrite( ledSchaltbarPin, LOW);
  }
  else
  {
    digitalWrite( ledSchaltbarPin, HIGH);
  }
}
```

Bei größeren Werten für `intervallzeit` reagiert das Programm fast gar nicht mehr auf den Tasterdruck, da die Chance den nicht blockierten Zeitrahmen zu treffen immer geringer wird. Man muss ihn ganz gedrückt halten, damit man überhaupt eine Chance hat die LED zu aktivieren…

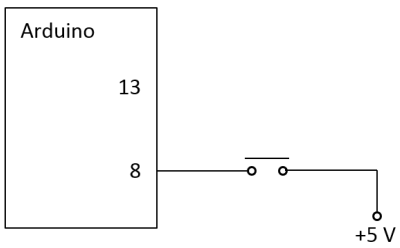
Das geht so nicht, eine Programmblockade ist in aller Regel unerwünscht. Stattdessen wird mit der Idee der (*verstrichenen Zeit > Schranke*) gearbeitet. UNSIGNED LONG bei `millis()` beachten.

```
//eine LED anschalten, wenn Taster geschlossen
//eine zweite LED blinkt dabei dauerhaft
//
//V2 - ohne delay(), keine Programmblockade mehr

int ledpin = 13;
int ledSchaltbarPin = 11;
int tasterpin = 5;
int intervallzeit = 1000;
unsigned int zeit;  //für Zeitvergleich mit aktueller Zeit
int ledStatus = LOW;

void setup()
{
  pinMode(ledpin, OUTPUT);
  pinMode(ledSchaltbarPin, OUTPUT);
```

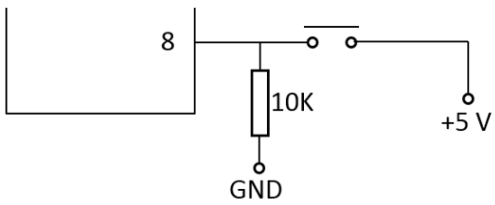
```
  pinMode(tasterpin, INPUT);
  zeit = millis();  //mit Startzeit laden
}

void loop()
{
  //wenn  mehr als intervallzeit vergangen ist LED umschalten
  //keine Pause mehr, nur eine if-Abfrage
  if( (millis() - zeit) > intervallzeit)
  {
    zeit = millis();    //neuer Zeitstempel
    ledStatus = !ledStatus; //LED toggeln
    digitalWrite(ledpin, ledStatus);
  }

  if( digitalRead(tasterpin) == LOW)
  {
    digitalWrite( ledSchaltbarPin, LOW);
  }
  else
  {
    digitalWrite( ledSchaltbarPin, HIGH);
  }
}
```

f) Rauschen, prellen und serieller Monitor

Der naive Betrieb eines Schalters ohne pull-up/down Widerstand funktioniert nicht zuverlässig.



Naiver Aufbau



Zuverlässiger Aufbau

f.1) Rauschen

0010_AB4_Taster_rauschen_seriellerMonitor_lsg

```
Aufbauen, abtippen, staunen
```

Ein Tasterdruck sorgt in jedem Fall für eine stabile 1, ohne pull-down R jedoch liegt der Port offen und bei geöffnetem Taster sieht man 0 und 1 wild wechseln. Ein pull-down R sorgt für einen dauerhaften, stabilen 0 Pegel bei geöffnetem Taster.

RAUSCHEN wird durch einen pull-up/down Widerstand unterbunden, Prellen (bouncing) nicht.

f.2) Prellen

```
Aufbauen, abtippen, staunen
```

Es gibt verschiedene SW-Lösungen für das Entprellen, das ist aber PROBLEMATISCH
Interessanter Fachartikel: https://www.skillbank.co.uk/arduino/switchbounce.htm

Pellen ist beim Schließen des Kontaktes größer als beim öffnen

FAIBLE.nrw

ORCA.nrw
Das Landesportal für Studium und Lehre.

Beteiligte Hochschulen:

- RWTH-Aachen
- Westfälische Wilhelms-Universität Münster
- Universität Duisburg-Essen
- Universität Bonn
- Universität Paderborn
- Technische Universität Dresden
- Carl von Ossietzky Universität Oldenburg

Ein Kooperationsvorhaben empfohlen durch die:

DIGITALE HOCHSCHULE NRW

INNOVATION DURCH KOOPERATION

gefördert durch:

Ministerium für Kultur und Wissenschaft des Landes Nordrhein-Westfalen