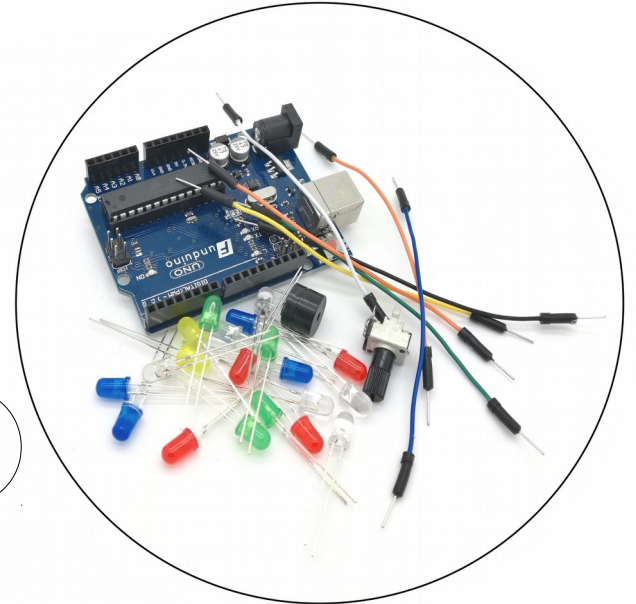
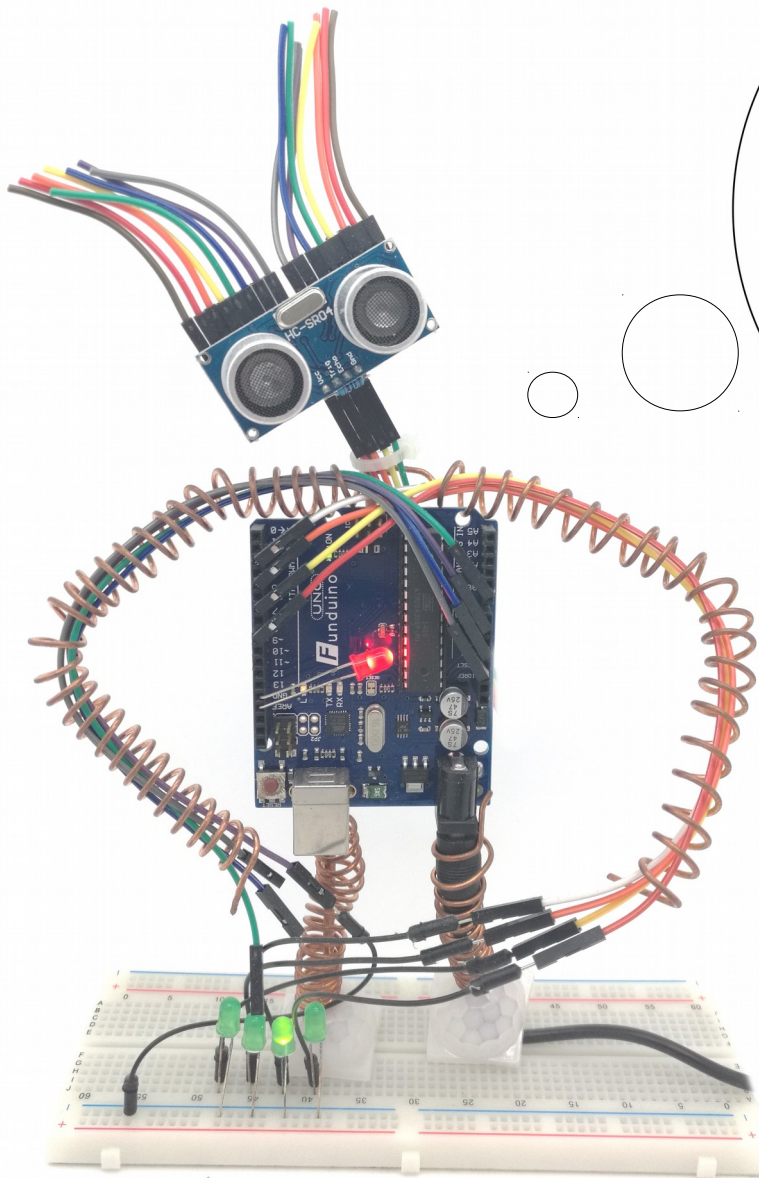


Funduino

Anleitungen für Arduino



Inhaltsverzeichnis

Vorwort.....	3
1.Vorwort zur Arduino Anleitung.....	3
2.Hardware und Software.....	4
2.1 Hardware.....	4
2.1.2 BESCHREIBUNG (ZUBEHÖR).....	5
2.1.2.1 DAS BREADBOARD.....	5
2.1.2.2 LEUCHTDIODEN, LED (LIGHT EMITTING DIODE).....	6
2.2 Software.....	7
2.2.1 INSTALLATION.....	7
2.2.1.1 Installation und Einstellung der Arduino-Software.....	7
2.2.1.2 Installation des USB-Treibers.....	8
3. Programmieren.....	9
Grundstruktur für einen Sketch.....	10
1. Variablen benennen.....	10
2. Setup (im Programm zwingend erforderlich).....	10
3. Loop (im Programm zwingend erforderlich).....	10
Anleitungen:.....	11
Eine blinkende LED.....	11
Der Wechselblinker.....	15
Eine LED pulsieren lassen.....	17
Gleichzeitiges Licht- und Tonsignal.....	19
Eine LED per Tastendruck aktivieren.....	21
Eine RGB-LED ansteuern.....	23
Der Bewegungsmelder.....	28
Helligkeitssensor auslesen.....	31
Drehregler.....	34
Temperaturen messen.....	36
Entfernung messen.....	40
Eine Infrarotfernbedienung zur Ansteuerung von Arduino Mikrocontrollern verwenden.....	45
Servo ansteuern.....	49
LCD Display.....	51
Relaiskarte.....	55
Schrittmotor.....	57
Feuchtigkeitssensor.....	60
Tropfsensor.....	63
RFID Kit.....	67
Zubehör für Arduino.....	75
Keypad shield.....	75
Anleitung zum LCD Display mit I2C Anschluss.....	78
Zwei I ² C Displays.....	82
Arduino-RTC.....	89
Tastenfeld.....	99
Tastenfeld – Schloss.....	103
Farbsensor.....	108
Töne erzeugen.....	114
Arduino Neigungssensor SW-520D.....	122
Temperatur messen mit dem LM35.....	125
Spannung messen.....	130
Vierstellige 7 Segment Anzeige.....	133

Vorwort

1. Vorwort zur Arduino Anleitung

Diese Anleitung soll als Grundlage zum Erlernen der Arduino-Plattform dienen. Sie soll Anfängern einen einfachen, interessanten und eng geleiteten Einstieg in die Arduino-Thematik geben. Die Anleitung orientiert sich dabei hauptsächlich an praxisorientierten Aufgaben mit einer theoretischen Einführung vorab. Diese sollte man vorher unbedingt lesen, um bei den späteren Praxisaufgaben nicht an Kleinigkeiten zu scheitern.

Diese Anleitung ist im Rahmen einer Unterrichtstätigkeit entstanden. Sie kann kostenlos zum Erlernen der Arduino-Plattform verwendet, jedoch nicht ohne Erlaubnis kopiert oder anderweitig verwendet werden. Die Anleitung wurde sorgfältig erstellt und wird kontinuierlich gepflegt, jedoch wird keine Garantie für die Richtigkeit und Vollständigkeit übernommen.

Für die praktischen Aufgaben sollte man mit einigen elektronischen Bauteilen versorgt sein. Auf dieser Internetseite können sie passende Arduino-Sets bestellen, die speziell auf diese Anleitung zugeschnitten sind.

Was ist eigentlich „Arduino“?

Arduino ist eine Open-Source-Elektronik-Prototyping-Plattform für flexible, einfach zu bedienende Hardware und Software im Bereich Mikrocontrolling. Es ist geeignet, um in kurzer Zeit spektakuläre Projekte zu verwirklichen. Viele davon lassen sich unter dem Begriff „Arduino“ bei Youtube finden. Es wird vor allem von Künstlern, Designern, Tüftlern und Bastlern verwendet, um kreative Ideen zu verwirklichen.

Aber auch in Schulen, Hochschulen und Universitäten wird die Arduino-Plattform zunehmend eingesetzt, um Lernenden einen kreativen und spannenden, aber vor allem auch einfachen Zugang zum Thema „Mikrocontrolling“ zu ermöglichen.

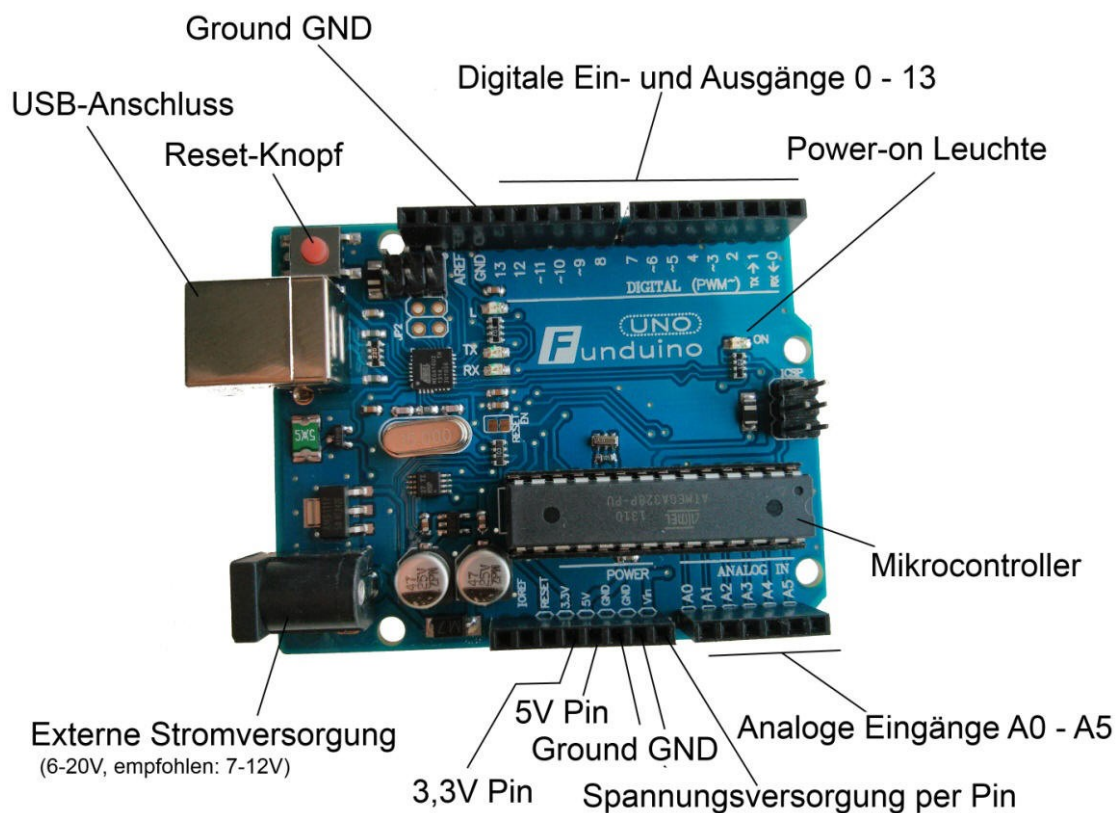
2. Hardware und Software

Der Begriff Arduino wird im allgemeinen Wortgebrauch gleichermaßen für die verschiedenen „Arduino-Boards“ (also die Hardware) als auch für die Programmierumgebung (Software) verwendet.

2.1 Hardware

Der „Arduino“ ist ein sogenanntes Mikrocontroller-Board (im weiteren Verlauf „Board“ genannt). Also im Grunde eine Leiterplatte (Board) mit jeder Menge Elektronik rund um den eigentlichen Mikrocontroller. Am Rand des Boards befinden sich viele Steckplätze (Pins genannt), an denen man die unterschiedlichsten Dinge anschließen kann. Dazu gehören: Schalter, LEDs, Ultraschallsensoren, Temperatursensoren, Drehregler, Displays, Motoren, Servos usw.

Es gibt verschiedene Versionen von Boards, die mit der Arduino-Software verwendet werden können. Dazu gehören sowohl viele verschiedene große und kleine „offizielle“ Boards mit der offiziellen „Arduino“ Bezeichnung als auch eine Vielzahl von häufig günstigeren aber gleichwertigen Arduino-„compatiblen“ Boards. Typische offizielle Boards heißen Arduino UNO, Arduino MEGA, Arduino Mini... etc. Compatible Boards heißen Funduino UNO, Funduino MEGA, Freeduino, Seeduino, Sainsmart UNO usw.



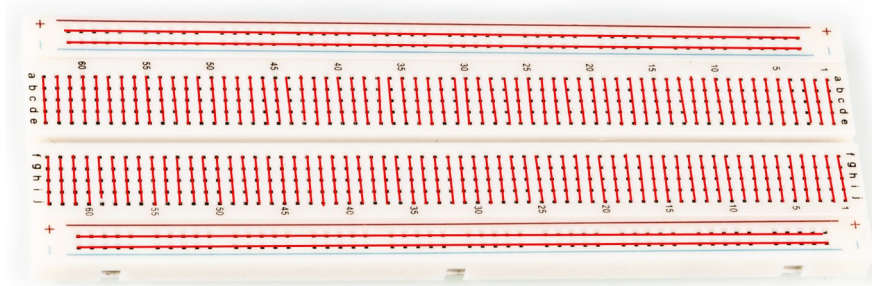
2.1.2 BESCHREIBUNG (ZUBEHÖR)

Neben Sensoren und Aktoren benötigt man als Basis für schnelle und flexible Versuchsaufbauten Steckkabel in Verbindung mit einem Breadboard. Dadurch erspart man sich zeitraubende Lötarbeiten. Des Weiteren eignen sich Leuchtdioden sehr gut, um die Signalausgabe des Boards zu überprüfen.

2.1.2.1 DAS BREADBOARD

Ein Breadboard oder auch „Steckbrett“ ist ein gutes Hilfsmittel, um Schaltungen aufzubauen ohne zu löten. In einem Breadboard sind immer mehrere Kontakte miteinander verbunden. Daher können an diesen Stellen viele Kabel miteinander verbunden werden, ohne dass sie verlötet oder verschraubt werden müssen.

Im folgenden Bild ist farbig dargestellt, welche Kontakte miteinander verbunden sind.



2.1.2.2 LEUCHTDIODEN, LED (LIGHT EMITTING DIODE)

Mit LEDs kann man sehr schnell die Ergebnisse eines Projekts testen. Daher sind sie für nahezu alle Arduino-Projekte nützlich. Über LEDs kann man vieles im Netz nachlesen. Hier nur die wichtigsten Infos.

> Der Strom kann nur in einer Richtung durch die LED fließen. Daher muss sie korrekt angeschlossen werden. Eine LED hat einen längeren und einen kürzeren Kontakt. Der längere Kontakt ist + und der kürzere ist -.

>Eine LED ist für eine bestimmte Spannung ausgelegt. Wird diese Spannung unterschritten, leuchtet die LED weniger hell oder sie bleibt aus. Wird die Spannung jedoch überschritten brennt die LED sehr schnell durch und wird an den Kontakten sehr heiß (ACHTUNG!).

>Typische Spannungswerte nach LED Farben: Blau:3,1V, Weiß:3,3V, Grün:3,7V, Gelb:2,2V, Rot:2,1V Die Spannung an den Digitalen Ports des Boards beträgt 5V. Beim direkten Anschluss an diese Ports gibt jede LED recht schnell den Geist auf. Daher muss ein Widerstand mit in den Stromkreis geschaltet werden. Im Internet gibt es unter dem Suchbegriff „Widerstandsrechner LED“ sehr gute Hilfen dazu.

>Unverbindliche Empfehlung für Widerstände an folgenden LEDs (Bei Anschluss an die 5V Pins des Mikrocontroller-Boards:

LED:	Weiß	Rot	Gelb	Grün	Blau	IR
Widerstand:	100 Ohm	200 Ohm	200 Ohm	100 Ohm	100 Ohm	100 OHM

2.2 Software

Die Software, mit welcher der Mikrocontroller programmiert wird, ist open-Source-Software und kann auf www.arduino.cc kostenlos heruntergeladen werden. In dieser „Arduino-Software“ schreibt man dann kleine Programme, die der Mikrocontroller später ausführen soll. Diese kleinen Programme werden „Sketch“ genannt.

Per USB-Kabel werden die fertigen Sketches dann auf den Mikrocontroller übertragen.

Mehr dazu später im Themengebiet „Programmieren“.

2.2.1 INSTALLATION

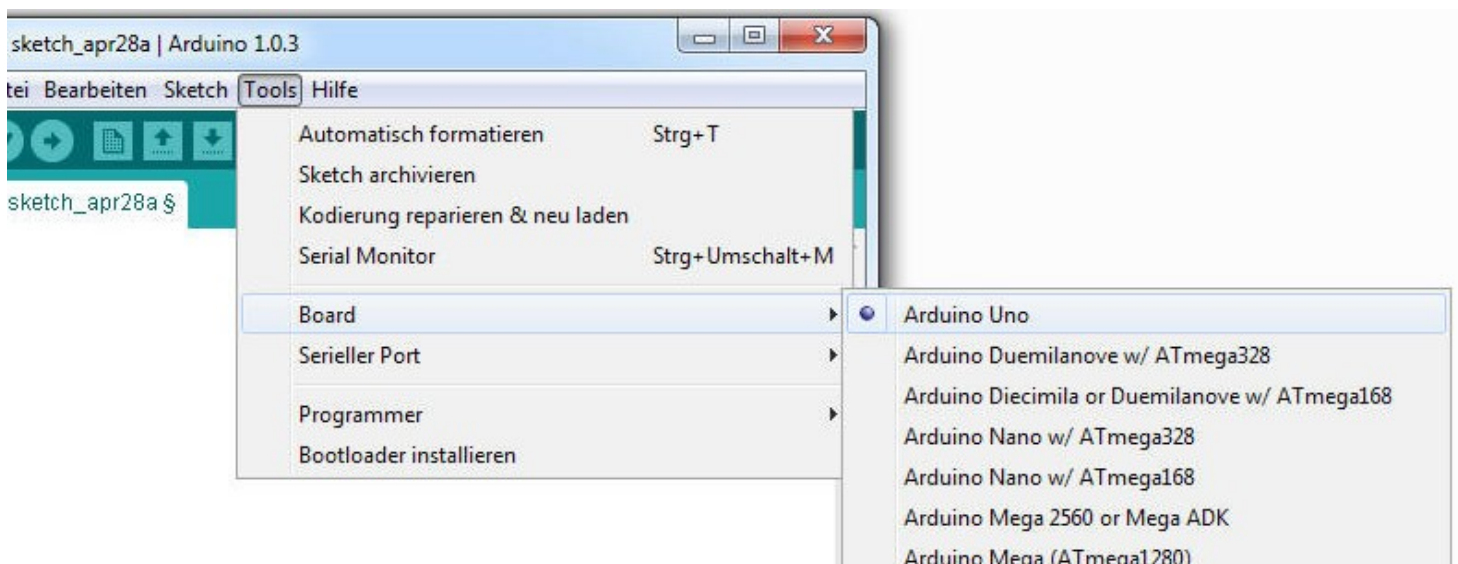
Nun muss nacheinander die Arduino-Software und der USB-Treiber für das Board installiert werden.

2.2.1.1 Installation und Einstellung der Arduino-Software

1. Software von www.arduino.cc downloaden und auf dem PC installieren (Das Arduino-Board noch NICHT angeschlossen). Danach öffnet man den Softwareordner und startet das Programm mit der Datei `arduino.exe`.

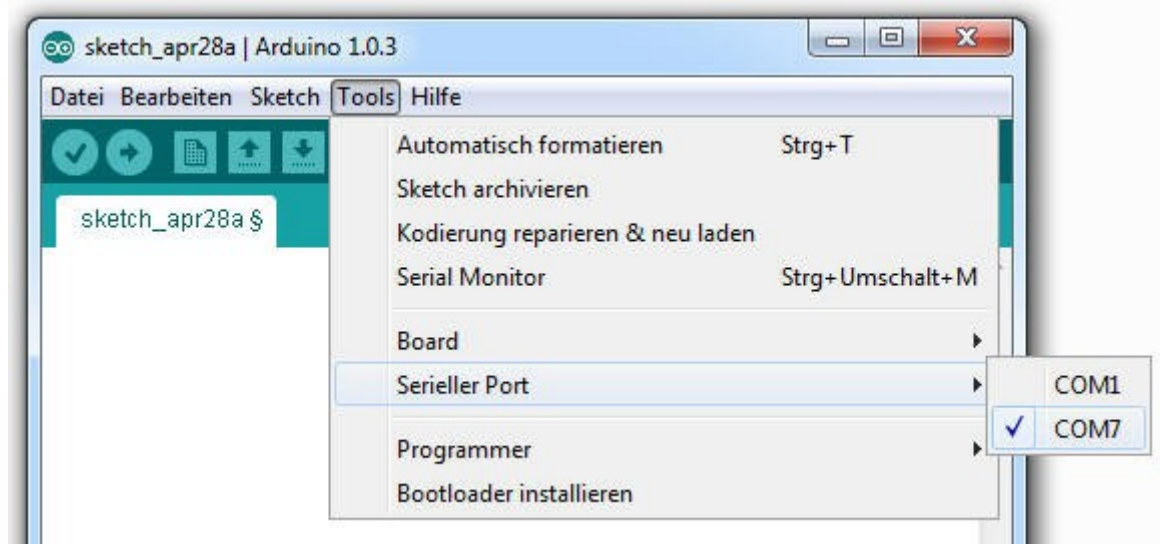
Zwei wichtige Einstellungen gibt es im Programm zu beachten.

a) Es muss das richtige Board ausgewählt werden, dass man anschließen möchte. Das „Funduino Uno“ Board wird hier als „Arduino Uno“ erkannt.



b) Es muss der richtige „Serial-Port“ ausgewählt werden, damit der PC weiß, an welchem USB Anschluss das Board angeschlossen ist. Dies ist jedoch nur möglich, wenn der Treiber richtig installiert wurde. Das kann folgendermaßen geprüft werden:

Zum jetzigen Zeitpunkt ist der Arduino noch nicht am PC angeschlossen. Nun klickt man in dem Untermenü der Software auf „Serial Ports“. Dort werden schon ein oder mehrere Ports zu sehen sein (COM1 / COM4 / COM7 / ...) Die Anzahl der angezeigten Ports ist dabei unabhängig von der Anzahl der USB-Anschlüsse. Wenn später das Board richtig installiert und angeschlossen ist, FINDET MAN HIER EINEN PORT MEHR!!!!



2.2.1.2 Installation des USB-Treibers

Idealer Ablauf:

1. Man schließt das Board an den PC an.
2. Der PC erkennt das Board und möchte einen Treiber installieren.

ACHTUNG: An dieser Stelle nicht zu schnell! Der Treiber wird in den meisten Fällen nicht automatisch erkannt und installiert. Man sollte im Verlauf der Installation den Treiber selber auswählen. Er befindet sich in dem Arduino-Programmordner in dem Unterordner „Drivers“.

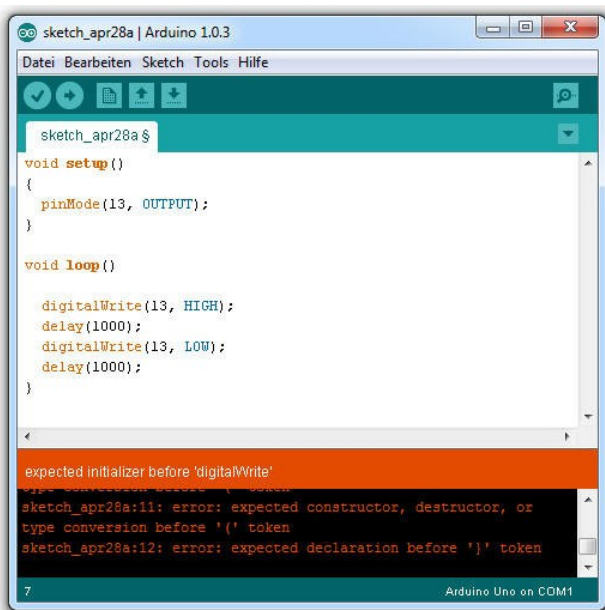
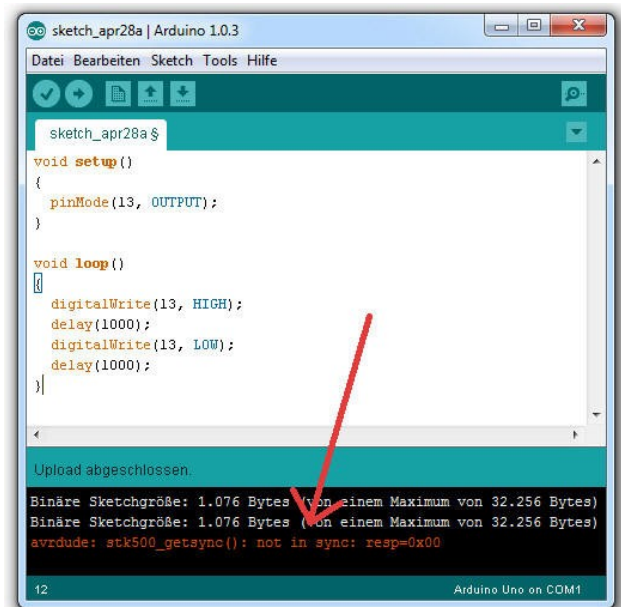
Kontrolle: In der Systemsteuerung des Computers findet man den „Gerätemanager“. Nach einer erfolgreichen Installation ist das Arduino-Board hier aufgelistet. Wenn die Installation nicht erfolgreich war, ist hier entweder nichts besonderes zu entdecken oder es ist ein unbekanntes USB-Gerät mit einem gelben Ausrufezeichen vorhanden. Für diesen Fall: Das unbekannte Gerät anklicken und auf „Treiber aktualisieren“ klicken. Jetzt kann man den Ablauf der manuellen Installation erneut durchführen

3. Programmieren

Jetzt geht es aber wirklich los. Ohne viel Theorie fangen wir direkt mit dem Programmieren an. „Learning by doing“ ist hier das Zauberwort. Während im linken Bereich die sogenannten „Sketche“ abgedruckt sind, befinden sich im rechten Bereich die Erklärungen zum Code. Wer sich nach diesem System durch die Programme arbeitet, wird den Programmcode in kurzer Zeit selber durchschauen und anwenden können. Später kann man sich dann selbst mit weiteren Funktionen vertraut machen. Diese Anleitung stellt nur einen Einstieg in die Arduino-Plattform dar. Alle möglichen Programmfunktionen bzw. Programmcodes werden auf der Internetseite „www.arduino.cc“ unter dem Punkt „reference“ genannt.

Vorab noch eine kurze Information zu möglichen Fehlermeldungen, die während der Arbeit mit der Arduino-Software auftauchen könnten. Die häufigsten sind die folgenden beiden:

1) Das Board ist nicht richtig installiert oder es ist ein falsches Board ausgewählt. Beim hochladen des Sketches wird im unteren Bereich der Software eine Fehlermeldung angezeigt, die in etwa so aussieht wie rechts abgebildet. Im Fehlertext befindet sich dann ein Vermerk „not in sync“.



2) Es gibt einen Fehler im Sketch. Beispielsweise ist ein Wort falsch geschrieben, oder es fehlt nur eine Klammer oder ein Semikolon. Im Beispiel links fehlt die geschweifte Klammer, die den Loop-Teil einleitet. Die Fehlermeldung beginnt dann häufig mit „expected...“. Das bedeutet, dass das Programm etwas erwartet, das noch nicht vorhanden ist.

Grundstruktur für einen Sketch

Ein Sketch kann zunächst in drei Bereiche eingeteilt werden.

1. Variablen benennen

Im ersten Bereich werden Elemente des Programms benannt (Was das bedeutet, lernen wir im Programm Nr.3). Dieser Teil ist nicht zwingend erforderlich.

2. Setup (im Programm zwingend erforderlich)

Das Setup wird vom Board nur einmal ausgeführt. Hier teilt man dem Programm zum Beispiel mit, welcher Pin (Steckplatz für Kabel) am Mikrocontrollerboard ein Ausgang oder ein Eingang ist.

Definiert als Ausgang: Hier soll eine Spannung am Board ausgegeben werden. Beispiel: Mit diesem Pin soll eine Leuchtdiode zum Leuchten gebracht werden.

Definiert als Eingang: Hier soll vom Board eine Spannung eingelesen werden. Beispiel: Es wird ein Schalter gedrückt. Das Board bemerkt dies dadurch, dass er an diesem Eingangspin eine Spannung erkennt.

3. Loop (im Programm zwingend erforderlich)

Der Loop-Teil wird von Board kontinuierlich wiederholt. Es verarbeitet den Sketch einmal komplett bis zum Ende und beginnt dann erneut am Anfang des Loop-Teils.

Anleitungen:

Eine blinkende LED

Aufgabe: Eine Leuchtdiode soll blinken.

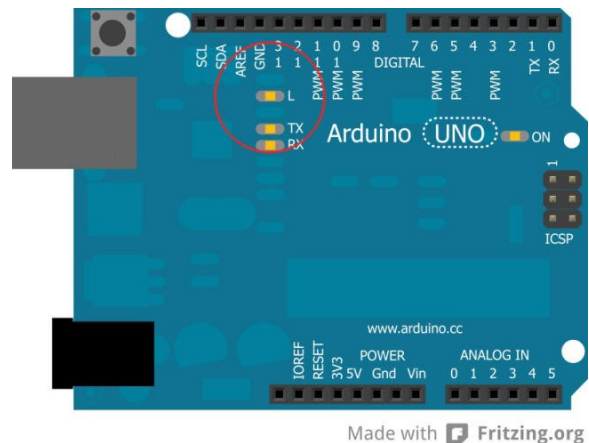
Material: Nur das Mikrocontrollerboard mit dem USB-Kabel!

Auf dem Arduino ist an Pin 13 bereits eine LED eingebaut (für Testzwecke). Häufig blinkt diese Lampe schon, wenn man ein neues Arduino-Board anschließt, da das Blink-Programm zum Testen des Boards je nach Hersteller bereits vorab installiert ist. Wir werden dieses Blinken jetzt selbst programmieren.

Schaltung:

Die auf dem Board vorhandene LED ist in dem Schaubild rot eingekreist.

Man muss nur das Board per USB-Kabel mit dem Computer verbinden.



Made with  Fritzing.org

1.1 Programmabschnitt 1: Variablen benennen

– Hier machen wir erstmal nichts!

1.2 Programmabschnitt 2: Setup

Wir haben nur einen Ausgang – An Pin13 soll eine Spannung ausgegeben werden (Die LED soll schließlich leuchten.).

Wir schreiben mitten in das weiße Eingabefeld der Arduino-Software:

```
void setup() //Hier beginnt das Setup
{ //geschweifte Klammer auf - Hier beginnt ein Programmabschnitt.
} //geschweifte Klammer zu - Hier ist ein Programmabschnitt beendet.
```

In den Teil zwischen den geschweiften Klammern werden nun die Setupinformationen eingebracht. In diesem Fall: „Pin13 soll ein Ausgang sein“.

```
void setup() //Hier beginnt das Setup
{ //Hier beginnt ein Programmabschnitt.
  pinMode(13, OUTPUT); // Pin 13 soll ein Ausgang sein.
} //Hier ist ein Programmabschnitt beendet.
```

1.3 Programmabschnitt 3: Loop (Hauptteil)

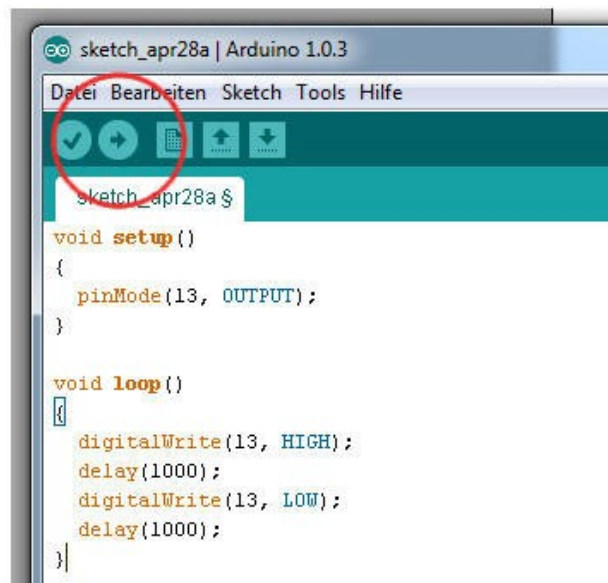
```
void setup() //Hier beginnt das Setup
{ // Hier beginnt ein Programmabschnitt.
  pinMode(13, OUTPUT); //Pin 13 soll ein Ausgang sein
} // Hier ist ein Programmabschnitt beendet.
void loop() //Hier beginnt das Hauptprogramm
{ //Hier beginnt ein Programmabschnitt.
} //Hier ist ein Programmabschnitt beendet.
```

Nun wird auch der Inhalt des Loop-Teils, also das Hauptprogramm, eingebracht:

DIES IST DER KOMPLETTE SKETCH:

```
void setup() //Hier beginnt das Setup
{ //Hier beginnt ein Programmabschnitt.
  pinMode(13, OUTPUT); //Pin 13 soll ein Ausgang sein.
} //Hier ist ein Programmabschnitt beendet.
void loop() //Hier beginnt das Hauptprogramm
{ //Programmabschnitt beginnt.
  digitalWrite(13, HIGH); //Schalte die die Spannung an Pin13 ein (LED an).
  delay(1000); //Warte 1000 Millisekunden (eine Sekunde)
  digitalWrite(13, LOW); //Schalte die die Spannung an Pin13 aus (LED aus).
  delay(1000); //Warte 1000 Millisekunden (eine Sekunde).
} //Programmabschnitt beendet.
```

Fertig. Der Sketch sollte nun exakt so aussehen, wie er auf dem Bild rechts dargestellt ist. Er muss jetzt nur noch auf das Board hochgeladen werden. Das funktioniert mit der rot umkreisten Schaltfläche (Oben links in der Software).



1.4 Das Programm kann jetzt noch variiert werden. Beispiel: Die LED soll sehr schnell blinken. Dazu verkürzen wir die Wartezeiten (Von 1000ms auf 200ms)

```
void setup() //Hier beginnt das Setup
{ //Hier beginnt ein Programmabschnitt.
  pinMode(13, OUTPUT); //Pin 13 soll ein Ausgang sein.
} //Hier ist ein Programmabschnitt beendet.
void loop() //Hier beginnt das Hauptprogramm
{ //Programmabschnitt beginnt.
  digitalWrite(13, HIGH); //Schalte die die Spannung an Pin13 ein (LED an).
  delay(200); //Warte 200 Millisekunden
  digitalWrite(13, LOW); //Schalte die die Spannung an Pin13 aus (LED aus).
  delay(200); //Warte 200 Millisekunden
} //Programmabschnitt beendet.
```

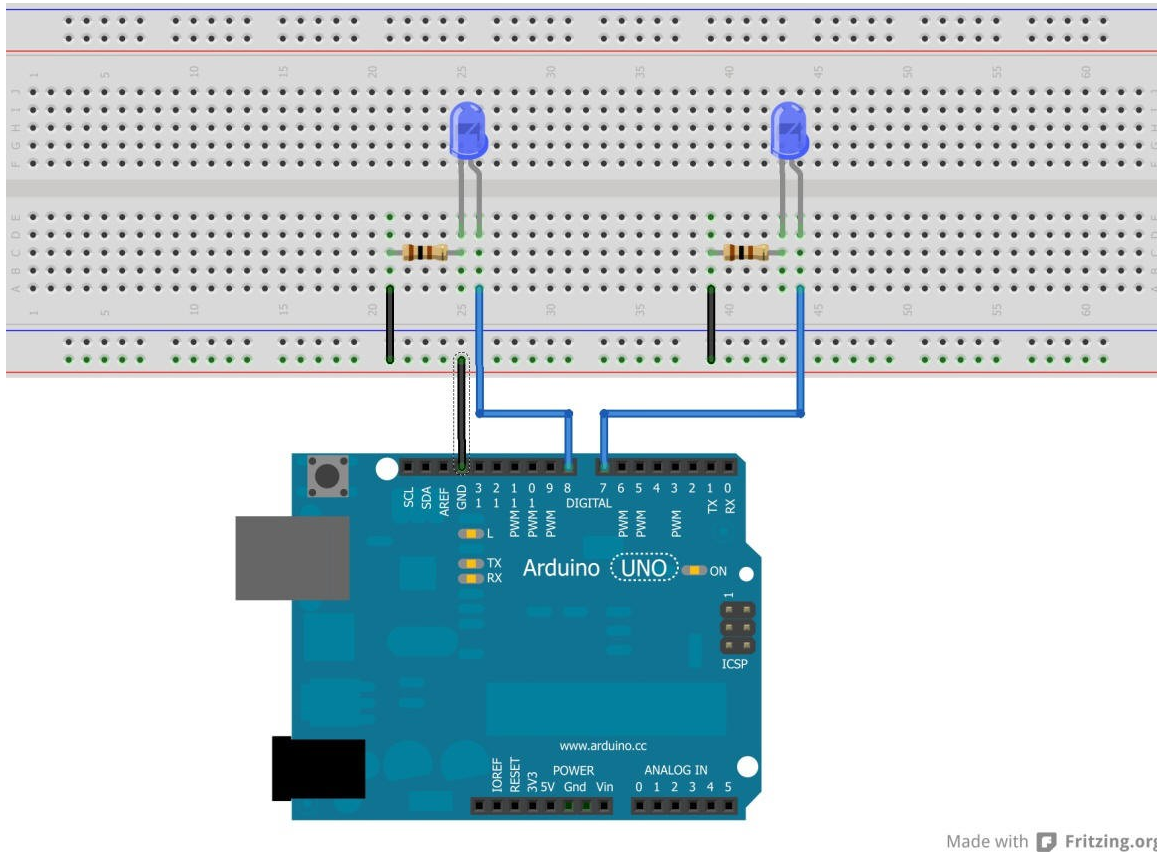
Der neue Sketch muss nun wieder auf das Board hochgeladen werden. Wenn alles geklappt hat, sollte die LED nun schneller blinken.

Der Wechselblinker

Aufgabe: Zwei Leuchtdioden sollen abwechselnd blinken.

Material: Arduino / zwei Leuchtdioden (blau) / Zwei Widerstände mit 100 Ohm / Breadboard / Kabel

Aufbau:



Sketch:

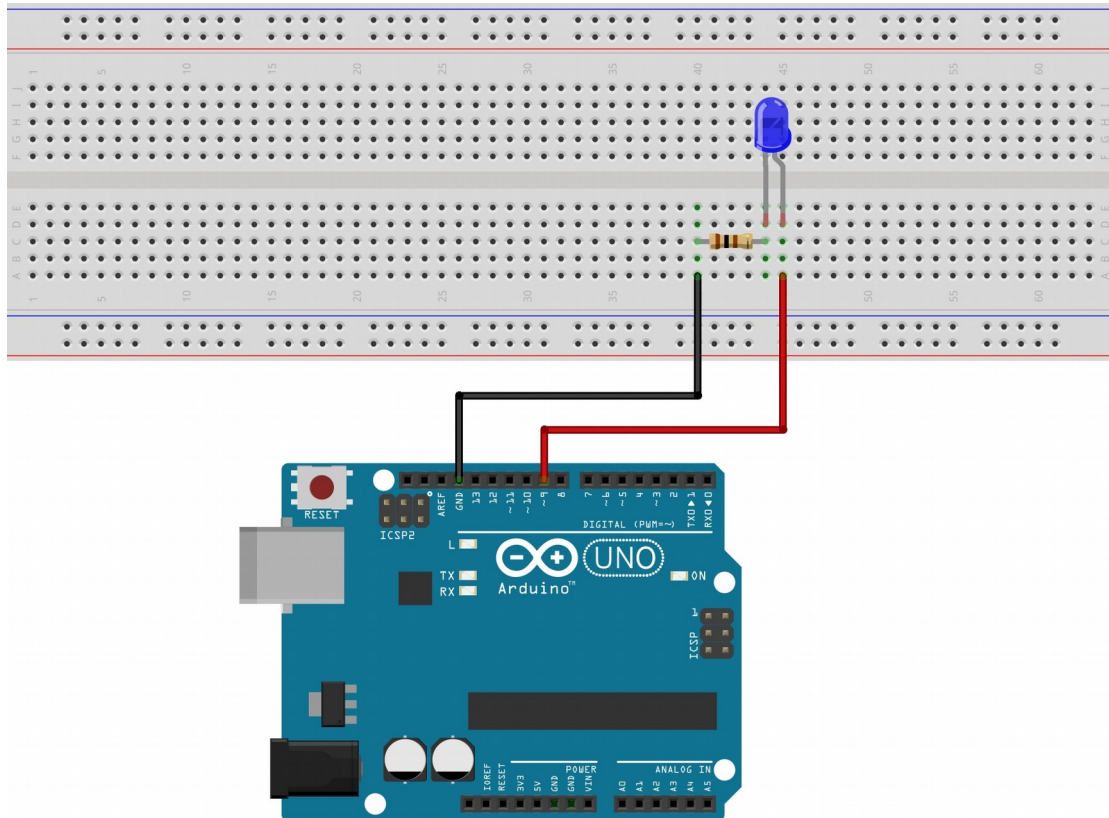
```
void setup()
{ // //Wir starten mit dem Setup
pinMode(7, OUTPUT); //Pin 7 ist ein Ausgang.
pinMode(8, OUTPUT); //Pin 8 ist ein Ausgang.
}
void loop()
{ // Das Hauptprogramm beginnt.
digitalWrite(7, HIGH); //Schalte die LED an Pin7 an.
delay(1000); //Warte 1000 Millisekunden.
digitalWrite(7, LOW); // Schalte die LED an Pin7 aus.
digitalWrite(8, HIGH); //Schalte die LED an Pin8 ein.
delay(1000); //Warte 1000 Millisekunden.
digitalWrite(8, LOW); //Schalte die LED an Pin8 aus.
} //Hier am Ende springt das Programm an den Start des Loop-Teils. Also..
//...schalte die LED an Pin7 an.
// ... usw... usw... usw...
```

Eine LED pulsieren lassen

Aufgabe: Eine LED soll pulsierend heller und dunkler werden. (Auch als engl. „faden“ bezeichnet)

Material: Arduino / eine LED (blau) / Ein Widerstand mit 100 Ohm / Breadboard / Kabel

Aufbau:



fritzing

Der Arduino ist ein digitaler Mikrocontroller. Er kennt an seinen Ausgängen nur „5 Volt an“ oder „5V aus“. Um die Helligkeit einer LED zu variieren, müsste man die Spannung jedoch variieren können. Zum Beispiel 5V wenn die LED hell leuchtet. 4 Volt, wenn sie etwas dunkler leuchtet usw. DAS GEHT AN DIGITALEN PINS ABER NICHT. Es gibt jedoch eine Alternative. Sie nennt sich Pulsweitenmodulation (PWM genannt). Die PWM lässt die 5V Spannung pulsieren. Die Spannung wird also im Millisekundenbereich ein und ausgeschaltet. Bei einer hohen PWM liegt das 5V Signal nahezu durchgehend am jeweiligen Pin an. Bei einer geringen PWM ist das 5V Signal kaum noch vorhanden (Da dies eine sehr kompakte Zusammenfassung ist, sollte man sich im Internet nach weiteren Erläuterungen umsehen). Mit dieser PWM kann man bei LEDs einen ähnlichen Effekt erreichen, als würde man die Spannung variieren. Nicht alle digitalen Pins am Board haben die PWM Funktion. Die Pins an denen die PWM funktioniert sind besonders gekennzeichnet, bspw. durch eine kleine Welle vor der Zahl mit der Pinnummer. Los geht's!

Sketch:

```

int LED=9; //Das Wort „LED“ steht jetzt für den Wert 9.

int helligkeit=0; //Das Wort „helligkeit“ steht nun für den Wert, der bei der
//PWM ausgegeben wird. Die Zahl 0 ist dabei nur ein beliebiger Startwert.

int fadeschritte=5; // fadeschritte: bestimmt die Geschwindigkeit des „fadens“

void setup()
{
    //Hier beginnt das Setup.

    pinMode(LED, OUTPUT);      ////Der Pin mit der LED (Pin9) ist ein Ausgang
}

void loop()
{

    analogWrite(LED, helligkeit); //Mit der Funktion analogWrite wird hier an dem
//Pin mit der LED (Pin9) die PWM Ausgabe aktiviert. Der PWM-Wert ist der Wert,
//der unter dem Namen „helligkeit“ gespeichert ist. In diesem Fall „0“ (Siehe
//ersten Programmabschnitt)

    helligkeit=helligkeit + fadeschritte; //Nun wird der Wert für die PWM-Ausgabe
//verändert. Unter dem Wert „helligkeit“ wird jetzt zur vorherigen helligkeit
//der Wert für die fadeschritte addiert. In diesem Fall: helligkeit=0+ 5. Der
//neue Wert für „helligkeit“ ist also nicht mehr 0 sondern 5. Sobald der Loop-
//Teil einmal durchgelaufen ist, wiederholt er sich. Dann beträgt der Wert für
//die Helligkeit 10. Im nächsten Durchlauf 15 usw. usw...

    delay(25); //Die LED soll für 25ms (Millisekunden), also nur ganz kurz die
//Helligkeit beibehalten. Verringert man diesen Wert, wird das Pulsieren
//ebenfalls schneller.

    if(helligkeit==0 || helligkeit== 255){ //Bedeutung des Befehls: Wenn die
//Helligkeit den Wert 0 ODER 255 erreicht hat, wechselt der Wert für die
//„fadeschritte“ von positiv zu negativ bzw. andersrum. Grund: Die LED wird
//zunächst bei jedem Durchlauf des Loop-Teils immer ein bisschen heller.
//Allerdings ist irgendwann der Maximalwert für die PWM-Ausgabe mit dem Wert 255
//erreicht. Die LED soll dann wieder Schritt für Schritt dunkler werden. Also
//wird der Wert für die „fadeschritte“ an dieser Stelle negatiert (Ein
//Minuszeichen wird davor gesetzt)

    fadeschritte= -fadeschritte; //Das bedeutet für den nächsten Durchlauf, dass in
//der Zeile „helligkeit = helligkeit + fadeschritte;“ die helligkeit abnimmt.
//Beispiel: „helligkeit=255+(-5)“. Der Wert für Helligkeit ist ab dann 250. Im
//nächsten Durchlauf 245 usw. usw... Sobald der Wert für Helligkeit bei 0
//angekommen ist, wechselt wieder das Vorzeichen. (Man bedenke die alte
//mathematische Regel: „minus und minus ergibt plus“.)

}

} // Mit dieser letzten Klammer wird der Loop-Teil geschlossen.

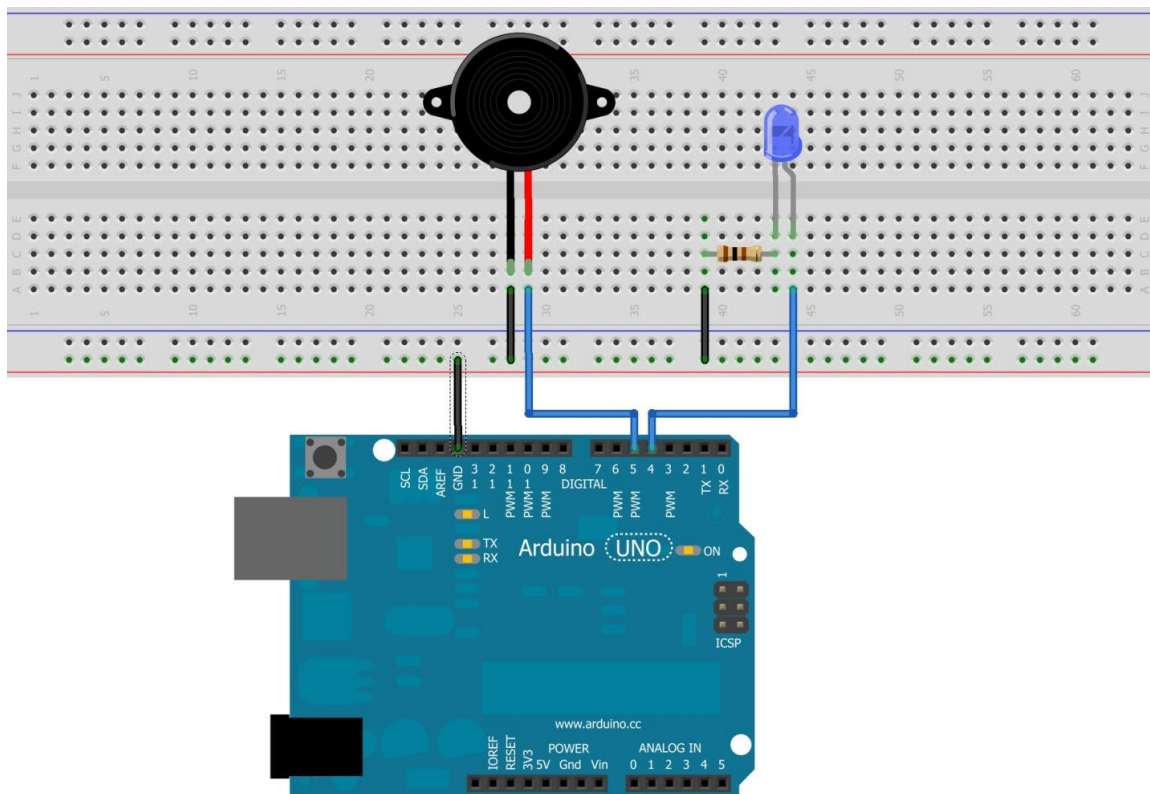
```

Gleichzeitiges Licht- und Tonsignal

Aufgabe: Eine LED und ein Piezo-Lautsprecher sollen kontinuierlich blinken bzw. piepen.

Material: Arduino / eine LED / Ein Widerstand mit 200 Ohm / Ein Piezo-Speaker / Breadboard / Kabel

Aufbau:



Made with  Fritzing.org

Sketch:

```
// Dieses Mal nutzen wir auch den ersten Programmabschnitt. Hier werden
//Variablen eingetragen. Das bedeutet, dass sich hinter einem Buchstaben oder
//einem Wort eine Zahl verbirgt. Bei uns ist die LED an Pin 4 angeschlossen und
//der Piezo-Speaker an Pin 5. Damit man die beiden Pins später nicht
//verwechselt, benennen wir Pin4 und Pin5 einfach um.

int LED=4; // Das Wort „LED“ steht jetzt für die Zahl „4“.
int pieps=5; // Das Wort „Pieps“ steht jetzt für die Zahl „5“.
void setup()
{ // Wir starten mit dem Setup.
pinMode(LED, OUTPUT); // Pin 4 (Pin „LED“) ist ein Ausgang.
pinMode(pieps, OUTPUT); // Pin 5 (Pin „Pieps“) ist ein Ausgang.
}
void loop()
{ // Das Hauptprogramm beginnt.
digitalWrite(LED, HIGH); //Schalte die LED an
digitalWrite(pieps, HIGH); // Schalte den Piezo-Lautsprecher an.
delay(1000); //Warte 1000 Millisekunden. (Es piepst und leuchtet.)
digitalWrite(LED, LOW); // Schalte die LED aus.
digitalWrite(pieps, LOW); //Schalte den Piezo aus.
delay(1000); //Warte 1000 Millisekunden. (kein Lärm, kein Licht)
} // Hier am Ende springt das Programm an den Start des Loop-Teils. also wird
//es gleich wieder piepsen und leuchten. Wenn man die Pause (delay) verkleinert
//oder vergrößert, piepst und leuchtet es schneller oder langsamer.
```

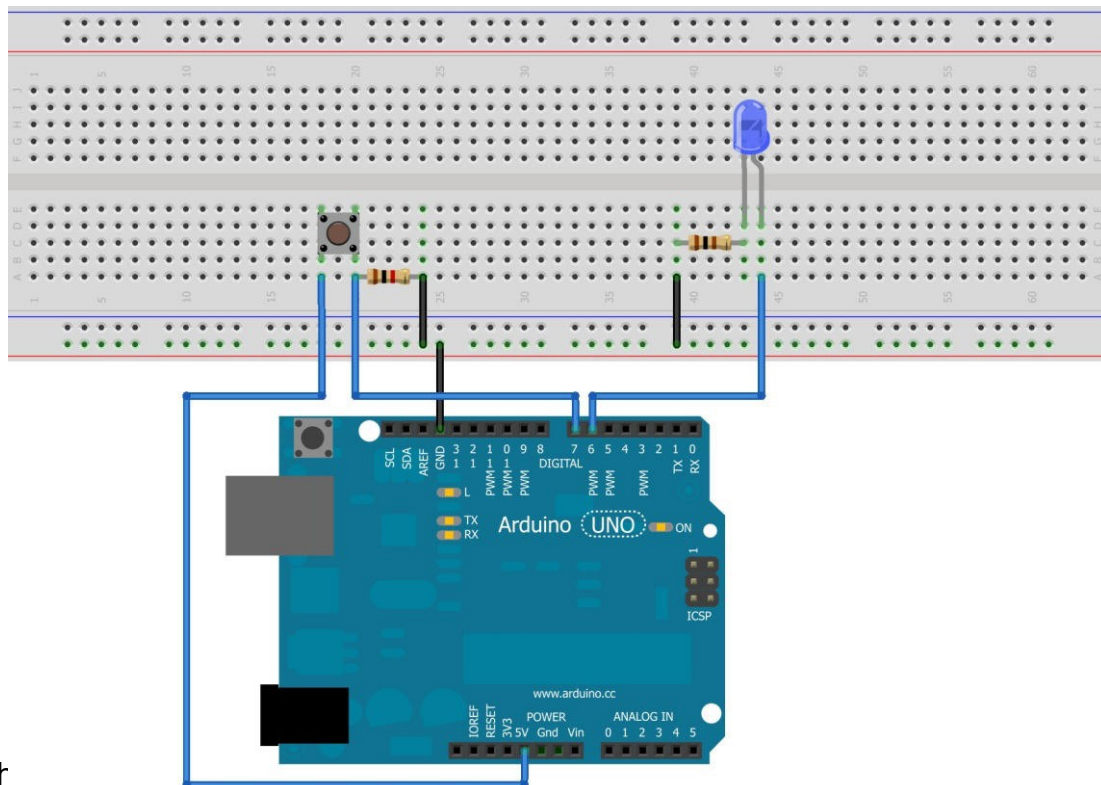
Eine LED per Tastendruck aktivieren

Aufgabe: Eine LED soll für 5 Sekunden leuchten, wenn ein Taster betätigt wurde.

Material: Arduino / eine LED (blau) / Ein Widerstand mit 100 Ohm / Ein Widerstand mit 1K Ohm (1000 Ohm) / Breadboard / Kabel / Taster

Der Mikrocontroller kann an seinen digitalen Pins nicht nur Spannungen ausgeben, sondern auch auslesen. Dies wollen wir in diesem Programm ausprobieren. Bei dem Aufbau gibt es jedoch eine Besonderheit. Wenn man den Taster einfach nur mit dem Mikrocontroller verbindet, dann liegt an dem Pin des Mikrocontrollers eine Spannung an, sobald der Taster gedrückt wird. Man kann sich das so vorstellen, als würden an dem besagten Pin ganz viele Elektronen herumschwirren. Wenn der Taster dann losgelassen wird, kommen keine neuen Elektronen mehr zu dem Pin am Mikrocontroller hinzu. Jetzt kommt der Knackpunkt. Die Elektronen, die es sich vorher auf dem Pin gemütlich gemacht haben, sind dann immer noch da und entweichen nur ganz langsam über kleine Kriechströme. Der Mikrocontroller denkt dann also, dass der Taster nicht nur kurz gedrückt wird sondern dass er ganz lange gedrückt wird. Nämlich so lange, bis sich keine Elektronen mehr auf dem Pin aufhalten. Dieses Problem kann man dadurch beheben, dass man den Pin über einen Widerstand mit ca. 1000 Ohm (1 K Ohm) erdet. Die Elektronen können dadurch recht schnell vom Pin abfließen und der Mikrocontroller erkennt, dass der Taster nur kurz „angetastet“ wurde. Da der Widerstand die Spannung an dem Eingangspin immer auf 0V „herunter zieht“, wird er auch als „PULLDOWN“-Widerstand bezeichnet. ACHTUNG: Wenn man dafür einen zu kleinen Widerstand verwendet, kann beim Drücken des Tasters ein Kurzschluss auf dem Mikrocontroller entstehen.

Aufbau:



Sketch

Made with  Fritzing.org

```

int LEDblau=6;    //Das Wort „LEDblau“ steht jetzt für den Wert 6.

int taster=7;    //Das Wort „taster“ steht jetzt für den Wert 7.

int tasterstatus=0; //Das Wort „tasterstatus“ steht jetzt zunächst für den
//Wert 0. Später wird unter dieser Variable gespeichert, ob der Taster gedrückt
//ist oder nicht.

void setup()
{    //Hier beginnt das Setup.
pinMode(LEDblau, OUTPUT); //Der Pin mit der LED (Pin 6) ist jetzt ein Ausgang.
pinMode(taster, INPUT); //Der Pin mit dem Taster (Pin 7) ist jetzt ein Eingang.
}

void loop()
{    //Mit dieser Klammer wird der Loop-Teil geöffnet

tasterstatus=digitalRead(taster); //Hier wird der Pin7 ausgelesen
//(Befehl:digitalRead). Das Ergebnis wird unter der Variable „tasterstatus“ mit
//dem Wert „HIGH“ für 5Volt oder „LOW“ für 0Volt gespeichert.

if (tasterstatus == HIGH) //Verarbeitung: Wenn der taster gedrückt ist (Das
//Spannungssignal ist hoch)..

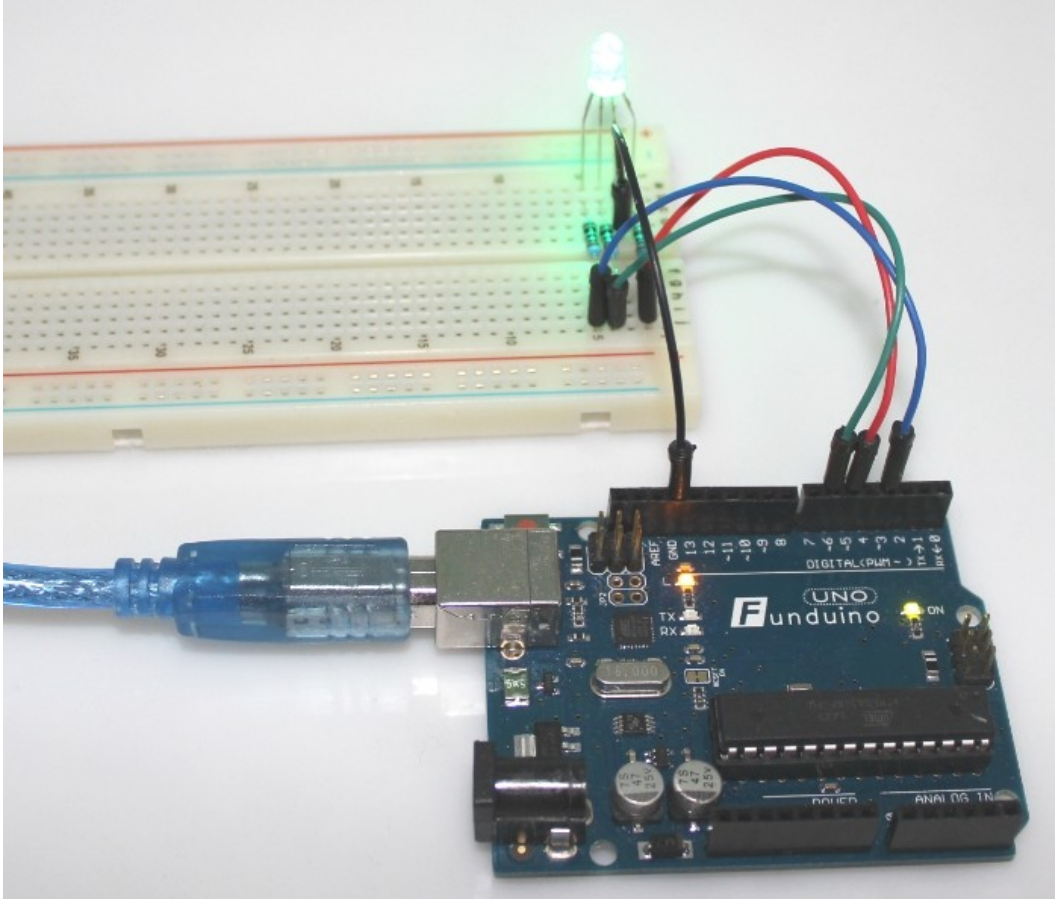
{    //Programmabschnitt des IF-Befehls öffnen.
digitalWrite(LEDblau, HIGH); //..dann soll die LED leuchten..
delay(5000); //..und zwar für für 5 Sekunden (5000 Millisekunden).
digitalWrite(LEDblau, LOW); //danach soll die LED aus sein.
} //Programmabschnitt des IF-Befehls schließen.
else // ...ansonsten...
{    //Programmabschnitt des else-Befehls öffnen
digitalWrite(LEDblau, LOW); // ...soll die LED aus sein.
} //Programmabschnitt des else-Befehls schließen
} // Mit dieser letzten Klammer wird der Loop-Teil geschlossen.

```

Eine RGB-LED ansteuern

Aufgabe: Eine RGB LED soll in verschiedenen Farben leuchten

Material: Arduino-Board / eine RGB-LED / drei Widerstände mit je 200 Ohm / Breadboard / Kabel

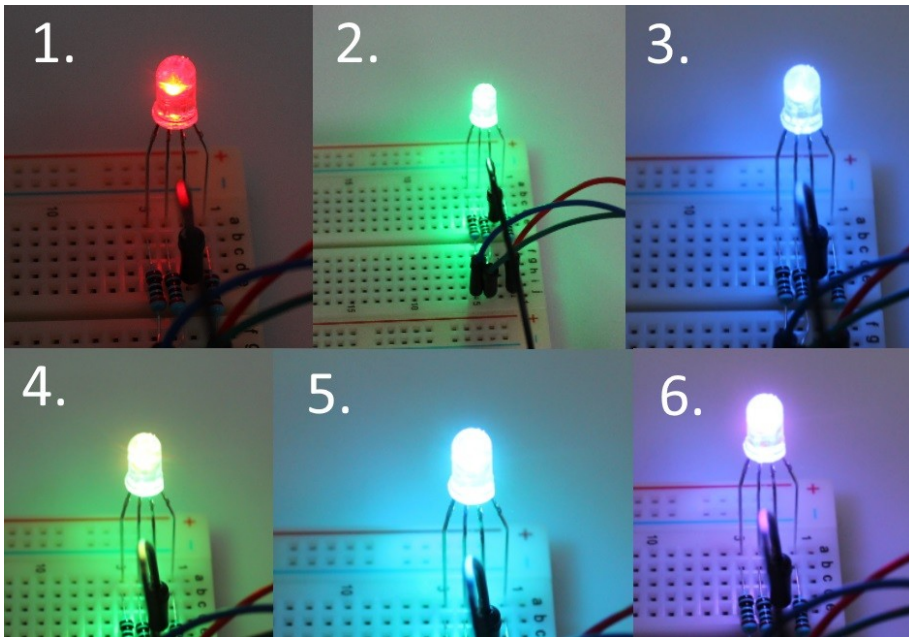


Was ist eine RGB-LED? Eine RGB-LED ist eine LED die in verschiedenen Farben leuchten kann. Hinter der Bezeichnung RGB verbergen sich die Farben „Rot“, „Grün“ und „Blau“. Die LED besteht im Inneren aus drei einzeln ansteuerbaren LEDs, die in den drei Farben leuchten. Deswegen hat eine RGB-LED auch so viele Beinchen, nämlich genau vier. Das längste der vier Beinchen ist je nach Version die gemeinsame Anode (+) bzw. Kathode (-). Mit den drei kürzeren Beinchen werden die einzelnen Farben der RGB-LED angesteuert.

Version a: „Common cathode“ – Das längste Beinchen der LED ist „-“ und die drei kürzeren Beinchen werden über „+“ (Spannung) angesteuert.

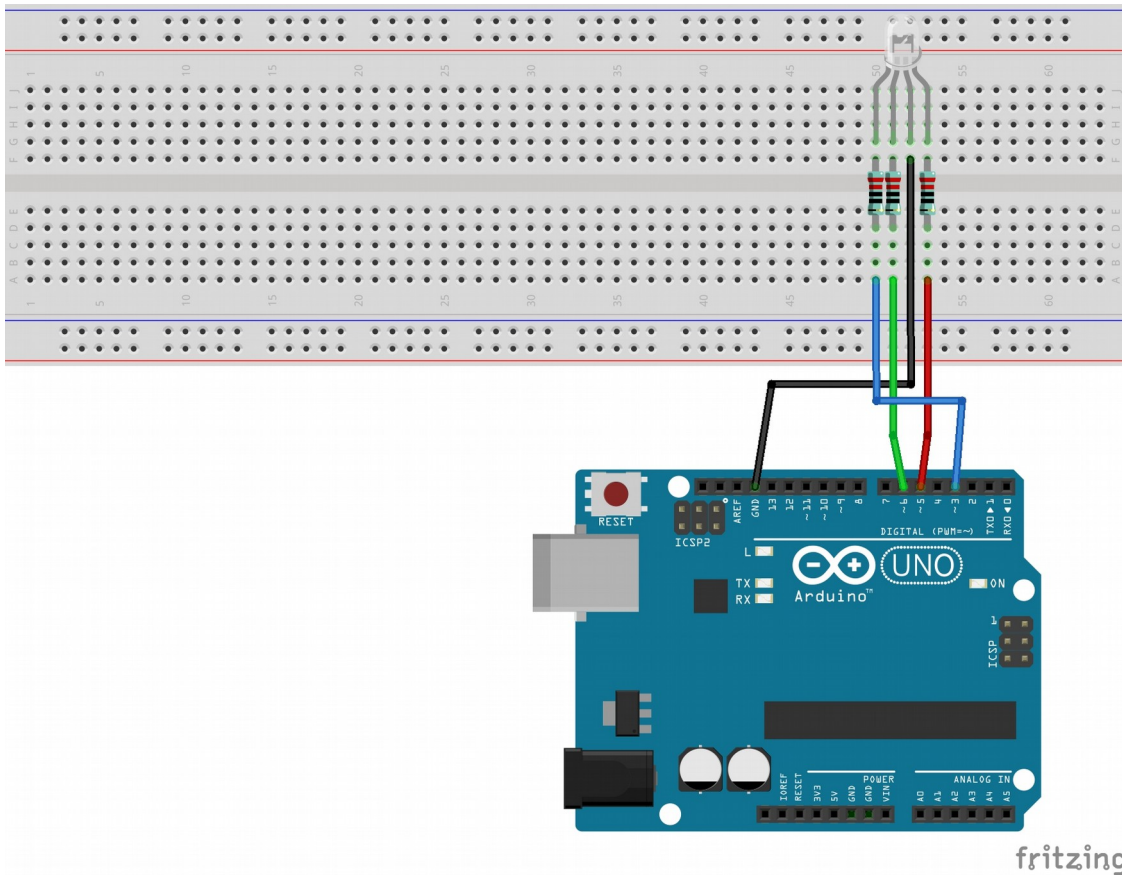
Version b) „Common anode“ – Das längste Beinchen der LED ist „+“ und die drei kürzeren Beinchen werden über „-“ (GND) angesteuert.

Durch eine Mischung der Farben können noch sehr viele weitere Farben erzeugt werden. Zum Beispiel entsteht durch die Ansteuerung der Farben „Blau“ und „Grün“ die Farbe „Türkis“.

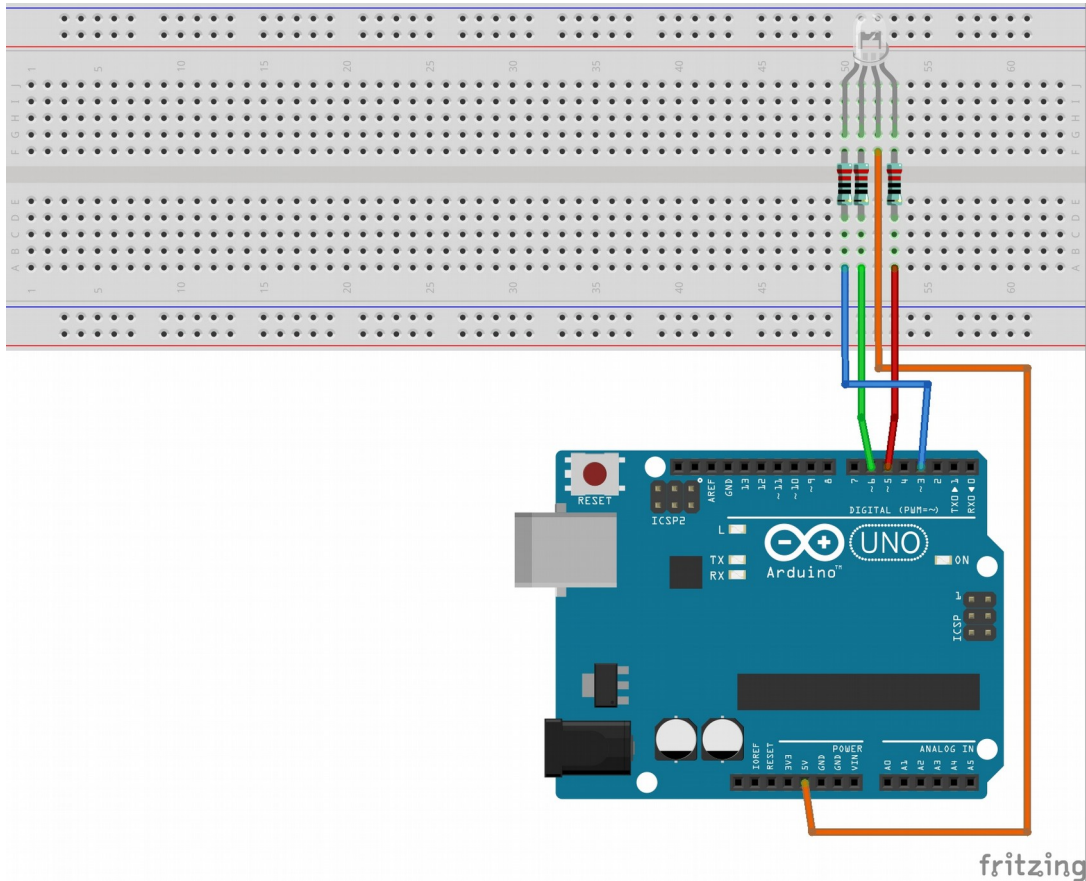


Welche Version man selber hat, findet man durch einfaches umstecken von „+“ und „-“ an der LED heraus (Info: Eine LED leuchtet nur bei richtigem Anschluss)

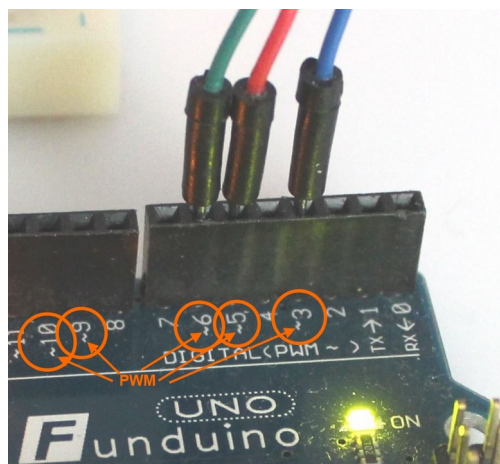
Aufbau für RGB LED Version a):



Aufbau für RGB LED Version b):



Der Arduino ist ein digitaler Mikrocontroller. Er kennt an seinen digitalen Ausgängen nur „5 Volt an“ oder „5V aus“. Damit man mit einer RGB-LED die vielen verschiedenen Farben erzeugen kann müssen die einzelnen Farben der LED jedoch genauer angesteuert werden. Dazu verwendet man die Pulsweitenmodulation. Die PWM (Pulsweitenmodulation) kann an den digitalen Pins verwendet werden, an denen auf dem Board eine kleine Welle aufgedruckt ist.



Die PWM lässt die Spannung zwischen +5V und 0V pulsieren. Die Spannung wird also im Millisekundenbereich ein und ausgeschaltet. Bei einer hohen PWM liegt das 5V Signal nahezu durchgehend am jeweiligen Pin an. Bei einer geringen PWM ist das 5V Signal

kaum noch vorhanden (Da dies eine sehr kompakte Zusammenfassung ist, sollte man sich im Internet nach weiteren Erläuterungen umsehen). Mit dieser PWM kann man bei LEDs einen ähnlichen Effekt erreichen, als würde man die Spannung variieren.

Die folgenden Codes funktionieren für beide RGB-Versionen gleichermaßen. Es muss nur eine Sache beachtet werden: Bei der LED Version b (Common anode) muss der Wert für „dunkel“ auf 255 gesetzt werden. Das hat zur Folge, dass dann nicht nur am gemeinsamen Pluspol der LED eine positive Spannung anliegt, sondern auch an der entsprechenden Farbe. Dann kann zwischen den beiden Kontakten der LED kein Strom mehr fließen und die jeweilige Farbe der LED bleibt aus. Aus diesem Grund ist auch für die Farbmischung zu beachten, dass bei dieser Version der Leuchtdiode, die Farbe heller wird, wenn der Wert kleiner wird. So leuchtet die Farbe blau an Pin 3 in diesem Sketch hell, wenn der Code für die blaue Farbe so gewählt wird:

```
int brightness1a = 0;
```

Sketch 1:

In diesem Code werden die drei einzelnen Farben nacheinander ein- und ausgeschaltet.

```
int LEDblau = 3; // Farbe blau an Pin 3
int LEDrot = 5; // Farbe rot an Pin 5
int LEDgruen=6; // Farbe gruen an Pin 6
int p=1000; // p ist eine Pause mit 1000ms also 1 Sekunde

int brightness1a = 150; // Zahlenwert zwischen 0 und 255 - gibt
//die Leuchtstärke der einzelnen Farbe an

int brightness1b = 150; // Zahlenwert zwischen 0 und 255 - gibt
//die Leuchtstärke der einzelnen Farbe an

int brightness1c = 150; // Zahlenwert zwischen 0 und 255 - gibt
//die Leuchtstärke der einzelnen Farbe an

int dunkel = 0; // Zahlenwer 0 bedeutet Spannung 0V - also LED aus

void setup()
{
  pinMode(LEDblau, OUTPUT);
  pinMode(LEDgruen, OUTPUT);
  pinMode(LEDrot, OUTPUT);
}
void loop()
{
  analogWrite(LEDblau, brightness1a); // blau einschalten
  delay(p); // pause
  analogWrite(LEDblau, dunkel); // blau ausschalten
  analogWrite(LEDrot, brightness1b); // rot einschalten
  delay(p); // pause
  analogWrite(LEDrot, dunkel); // rotausschalten
```

```

analogWrite(LEDgruen, brightness1c); // gruen einschalten
delay(p); // pause
analogWrite(LEDgruen, dunkel); // gruenausschalten
}

```

Sketch 2:

In diesem Code werden die drei einzelnen Farben jeweils paarweise nacheinander ein- und ausgeschaltet. Dadurch entstehen die Farbmischungen Gelb, Türkis und Lila.

```

int LEDblau = 3; // Farbe blau an Pin 3
int LEDrot = 5; // Farbe rot an Pin 5
int LEDgruen=6; // Farbe gruen an Pin 6
int p=1000; // p ist eine Pause mit 1000ms also 1 Sekunde
int brightness1a = 150; // Zahlenwert zwischen 0 und 255 - gibt
//die Leuchtstärke der einzelnen Farbe an
int brightness1b = 150; // Zahlenwert zwischen 0 und 255 - gibt
//die Leuchtstärke der einzelnen Farbe an
int brightness1c = 150; // Zahlenwert zwischen 0 und 255 - gibt
//die Leuchtstärke der einzelnen Farbe an
int dunkel = 0; // Zahlenwert 0 bedeutet Spannung 0V - also LED
//aus
void setup()
{
pinMode(LEDblau, OUTPUT);
pinMode(LEDgruen, OUTPUT);
pinMode(LEDrot, OUTPUT);
}
void loop()
{
analogWrite(LEDgruen, brightness1c); // gruen und rot ein = gelb
analogWrite(LEDrot, brightness1b);
delay(p);
analogWrite(LEDgruen, dunkel); // gruen und rot aus = gelb aus
analogWrite(LEDrot, dunkel);
analogWrite(LEDgruen, brightness1c); // gruen und blau ein =
//türkis
analogWrite(LEDblau, brightness1b);
delay(p);
analogWrite(LEDgruen, dunkel); // gruen und blau aus = türkis aus
analogWrite(LEDblau, dunkel);
analogWrite(LEDrot, brightness1b); // rot und blau ein = lila
analogWrite(LEDblau, brightness1b);
delay(p);
analogWrite(LEDrot, dunkel); // rot und blau aus = lila aus
analogWrite(LEDblau, dunkel)
}

```

Der Bewegungsmelder

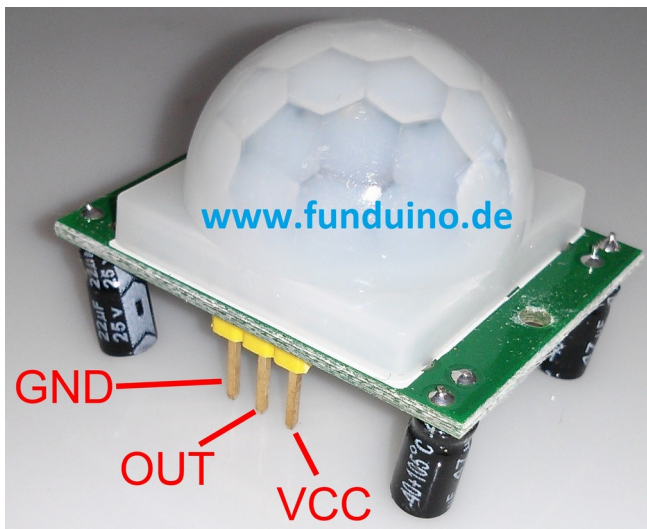
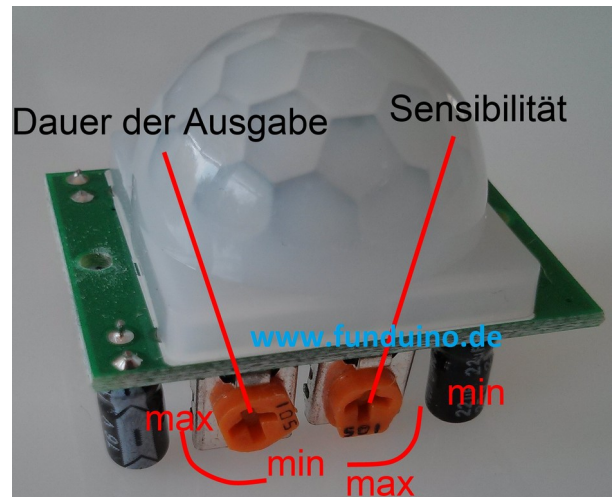
Aufgabe: Ein Piezo-Lautsprecher soll piepen, sobald eine Bewegung registriert wird.

Material: Arduino / Bewegungsmelder / Breadboard / Kabel / Piezo-Lautsprecher

Lerninhalt: Spannung eines Bewegungsmelders auslesen und für eine Ausgabe verwenden.

Der Bewegungsmelder, auch PIR Sensor genannt, ist sehr einfach konstruiert. Sobald er eine Bewegung detektiert, gibt er auf einem Pin eine Spannung von 5 Volt aus. Diese muss nur ausgelesen und vom Mikrocontroller verarbeitet werden.

Die Dauer des Ausgangssignals und die Sensibilität (Reichweite) kann über zwei Drehregler eingestellt werden (Bild rechts).



An der Unterseite des Bewegungsmelders befindet sich zudem ein Jumper, mit dem man zwischen zwei Modi wechseln kann.

Die Kunststofflinse ist nur leicht gesteckt. Wenn man sie abhebt kann man den Infrarotdetektor erkennen und man sieht anhand der Beschriftung unter der Linse, wie der Sensor verkabelt werden muss: GND (-), OUT (Ausgang des Signals), VCC (+).

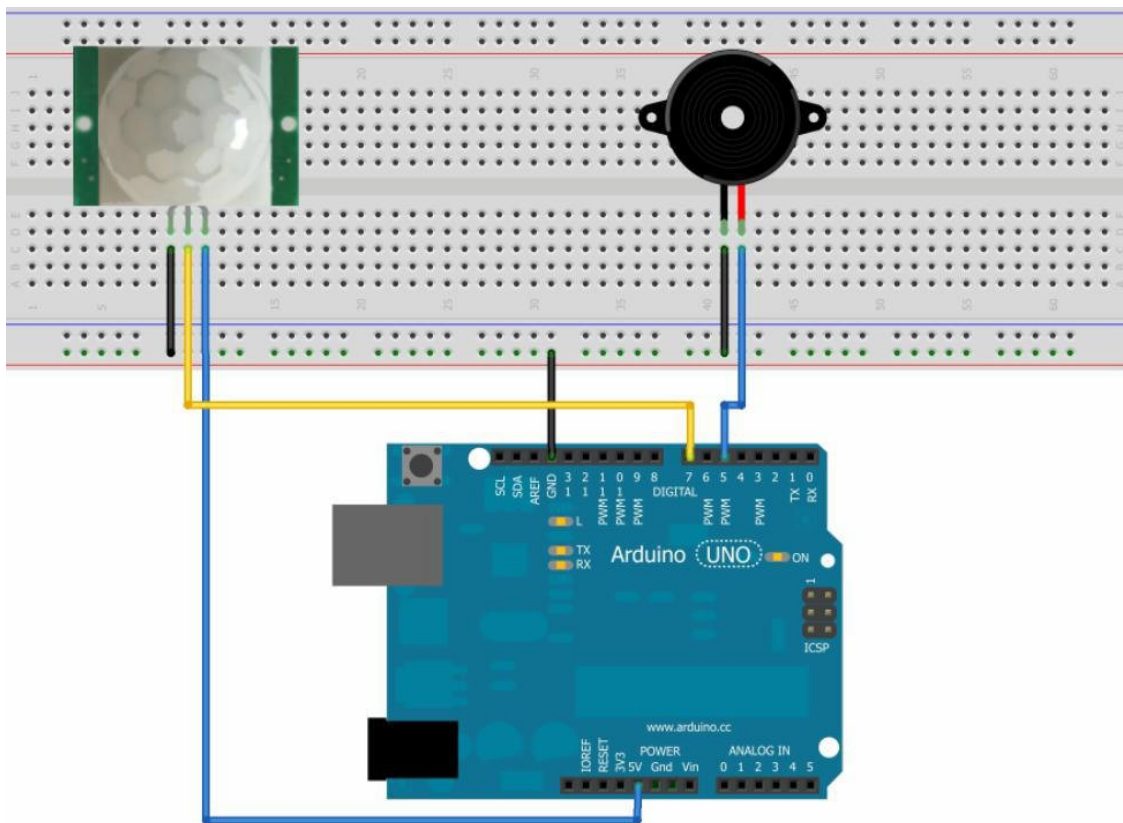
Links sind die Kontakte noch einmal als Bild.



1) Jumper ist ganz außen: Das Ausgangssignal wird nachdem eine Bewegung detektiert wurde für eine gewisse Zeit aufrecht erhalten und danach auf jeden Fall wieder deaktiviert, auch wenn im Aktionsbereich des Bewegungsmelders noch eine Bewegung detektiert werden könnte. Nach einer gewissen Zeit wird das Ausgangssignal erneut erzeugt.

2) Der Jumper ist wie auf dem Abbild rechts leicht nach innen versetzt. Das Ausgangssignal bleibt pausenlos aktiv, so lange vom Bewegungsmelder eine Bewegung detektiert wird. Dieser Modus wird für Arduinoprojekte empfohlen.

Aufbau:



Made with  Fritzing.org

Sketch:

```
int piezo=5; //Das Wort „piezo“ steht jetzt für den Wert 5.

int bewegung=7; //Das Wort „bewegung“ steht jetzt für den Wert 7.

int bewegungsstatus=0; //Das Wort „bewegungsstatus“ steht jetzt zunächst für den
//Wert 0. Später wird unter dieser Variable gespeichert, ob eine Bewegung
//erkannt wird oder nicht.

void setup() //Hier beginnt das Setup.

{

pinMode(piezo, OUTPUT); //Der Pin mit dem Piezo (Pin 5) ist jetzt ein Ausgang.

pinMode(bewegung, INPUT); //Der Pin mit dem Bewegungsmelder (Pin 7) ist jetzt
//ein Eingang.

}

void loop() //Der Loop-Teil beginnt

{ //Mit dieser Klammer wird der Loop-Teil geöffnet.

bewegungsstatus=digitalRead(bewegung); //ier wird der Pin7 ausgelesen. Das
//Ergebnis wird unter der Variablen „bewegungsstatus“ mit dem Wert „HIGH“ für
//5Volt oder „LOW“ für 0Volt gespeichert.

if (bewegungsstatus == HIGH) //Verarbeitung: Wenn eine Bewegung detektiert wird
//(Das Spannungssignal ist hoch)

{ //Programmabschnitt des IF-Befehls öffnen.

digitalWrite(piezo, HIGH); //dann soll der Piezo piepsen.

delay(5000); //...und zwar für für 5 Sekunden.

digitalWrite(piezo, LOW); //...danach soll er leise sein.

} //Programmabschnitt des IF-Befehls schließen.

else //ansonsten...

{ //Programmabschnitt des else-Befehls öffnen.

digitalWrite(piezo, LOW); //...soll der Piezo-Lautsprecher aus sein.

} //Programmabschnitt des else-Befehls schließen.

} //Mit dieser letzten Klammer wird der Loop-Teil geschlossen.
```

Helligkeitssensor auslesen

Jetzt wird's etwas komplizierter! Erst durchatmen, neues Getränk holen!

Aufgabe: Eine LED soll leuchten, wenn es dunkel wird bzw. wenn ein Fotowiderstand abgedeckt wird.

Material: Arduino / eine LED / ein Widerstand mit 200 Ohm / ein Widerstand mit 10K Ohm / Breadboard / Kabel / Fotowiderstand

Spannungen auslesen:

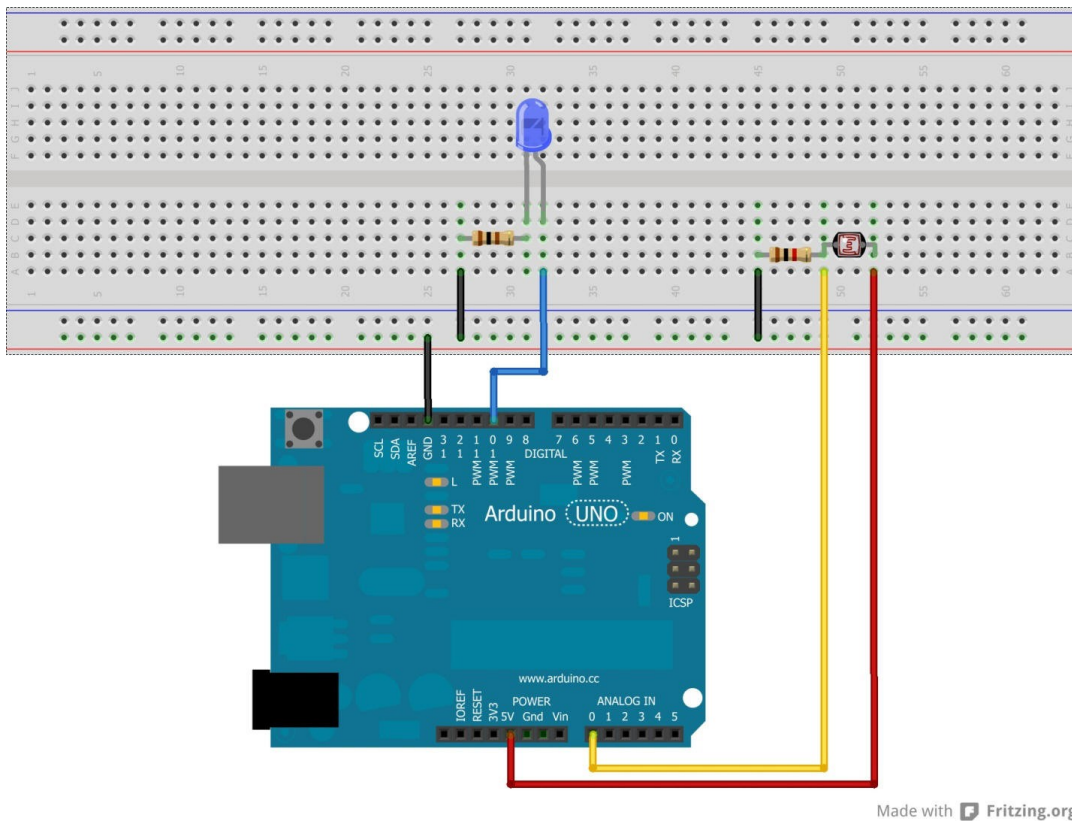
Der Mikrokontroller soll über einen Fotowiderstand auslesen, wie hell es ist. Dazu nutzt man ein einfaches physikalisches Prinzip. Wenn in einem Stromkreis zwei Verbraucher hintereinander angeschlossen sind (Reihenschaltung), dann „teilt“ sie sich auch die gemeinsam anliegende Spannung. Ein Beispiel: Zwei gleiche Lampen sind in Reihe geschaltet und es wird eine Spannung von 6 Volt angelegt. Dann kann man mit einem Spannungsmessgerät feststellen, dass an den Lampen jeweils nur 3 Volt anliegen. Wenn zwei ungleiche Lampen angeschlossen werden (Eine hat einen geringeren Widerstand), dann kann man zwei unterschiedliche Spannungen an den beiden Lampen messen, bspw. 1,5 Volt und 4,5 Volt. Ein Fotowiderstand ändert seinen Widerstand in Abhängigkeit der Lichtstärke. Diesen Effekt nutzt man, um anhand der an ihr anliegenden Spannung einen Wert für Helligkeit bzw. Dunkelheit in Form von verschiedenen Spannungen abzulesen. Damit man hier überhaupt eine Spannungsteilung erzeugen kann, schließt man den Fotowiderstand und einen Widerstand (1 – 10 K Ohm, je nach verwendetem Fotowiderstand. Der Widerstand sollte einen ähnlichen Widerstandswert wie der Fotowiderstand haben) in Reihe an und verbindet sie mit 5 Volt und der „Erdung“ (Ground / GND) – siehe Aufbau. Das Mikrocontroller-Board ist in der Lage, analoge Signale (Spannung) zu messen und diese zu verarbeiten. Dies geschieht mit den analogen Eingängen auf dem Board. Dieser wandelt den gemessenen Spannungswert in eine Zahl um, die dann weiter verarbeitet werden kann. 0 Volt entspricht dabei der Zahl 0 und der höchste Messwert 5 Volt entspricht der Zahl 1023 (0 bis 1023 entspricht 1024 Zahlen = 10 Bit). Beispiel: Es wird eine Spannung von 2,5 Volt gemessen, dann liefert der Mikrokontroller den Wert 512 ($1024 : 2$).

Der „serial monitor“:

Der „serial monitor“ ist ein wichtiger Bestandteil der Arduino-Software. Mit diesem „serial monitor“ kann man sich am PC Daten anzeigen lassen, die das Mikrocontroller-Board an den PC sendet (Zahlen oder Texte). Das ist sehr sinnvoll, da man nicht immer ein LCD Display am Mikrocontroller angeschlossen hat, auf dem man bestimmte Werte ablesen könnte. In diesem Sketch wird der „serial monitor“ verwendet, um die Werte anzeigen zu lassen, die das Board von dem Fotowiderstand einliest. Wozu ist das sinnvoll? Mal angenommen, die LED soll erst bei beginnender Dunkelheit anfangen zu leuchten. Dann muss es im Sketch einen Bereich geben, der die Funktion hat: „Wenn der Wert des Fotowiderstandes den Wert x unterschreitet, dann soll die LED leuchten“. Dazu müsste man wissen wie groß der Wert x bei beginnender Dämmerung ist.

Lösung: Ich sende den ausgelesenen Wert „x“ der Spannung an dem Fotowiderstand bei entsprechender Helligkeit (bspw. Dämmerung) an den „serial monitor“ und lasse ihn mir dort anzeigen. Mit diesem Wissen kann ich später das Programm in der folgenden Form abändern. „Wenn der Spannungsausgabewert des Fotowiderstandes einen Wert von „x“ unterschreitet, dann schalte die LED an.“

Aufbau:



Sketch:

```
int eingang=A0; //Das Wort „eingang“ steht jetzt für den Wert „A0“ (Bezeichnung
//vom Analogport 0)
int LED=10; //Das Wort „LED“ steht jetzt für den Wert 10
int sensorwert=0; //Variable für den Sensorwert mit 0 als Startwert

void setup() //Hier beginnt das Setup.
{
  Serial.begin(9600); //Die Kommunikation mit dem seriellen Port wird
//gestartet. Das benötigt man, um sich den tatsächlich ausgelesenen Wert später
//im serial monitor anzeigen zu lassen.

  pinMode (LED,OUTPUT); //Der Pin mit der LED (Pin 10) ist jetzt ein Ausgang
}

void loop()
{ //Mit dieser Klammer wird der Loop-Teil geöffnet.

  sensorwert=analogRead(eingang); //Die Spannung an dem Fotowiderstand auslesen
//und unter der Variable „sensorWert“ abspeichern.
  Serial.print("Sensorwert="); //Ausgabe am Serial-Monitor: Das Wort „Sensorwert:“

  Serial.print(sensorwert); // Ausgabe am Serial-Monitor. Mit dem Befehl
//Serial.print wird der Sensorwert des Fotowiderstandes in Form einer Zahl
//zwischen 0 und 1023 an den serial monitor gesendet.

  if(sensorwert > 512) //Wenn der Sensorwert über 512 beträgt...
  {
    digitalWrite(LED,HIGH); //...soll die LED leuchten...
  }
  else
  {
    digitalWrite(LED,LOW); //...andernfalls soll sie nicht leuchten
  }

  delay(50); // Eine kurze Pause, in der die LED an oder aus ist
} //Mit dieser letzten Klammer wird der Loop-Teil geschlossen.

//Wenn nun der Sensorwert bei normaler Helligkeit bspw. nur den Wert 100 hat
//(Der Wert ist abhängig von den verwendeten Widerständen, von der Helligkeit
//und von der Stromrichtung), dann nimmt man anstelle des Wertes 512 einen
//wesentlich kleineren Wert, bei dem die LED zu leuchten beginnen soll. Bspw.
//nimmt man dann den Wert 90. Den aktuellen Sensorwert kann man sich nun mit
//Hilfe des „Serial monitor“ anzeigen lassen. Dazu klickt man oben auf „Tools“
//und anschließend auf „serial monitor“.
```

Drehregler

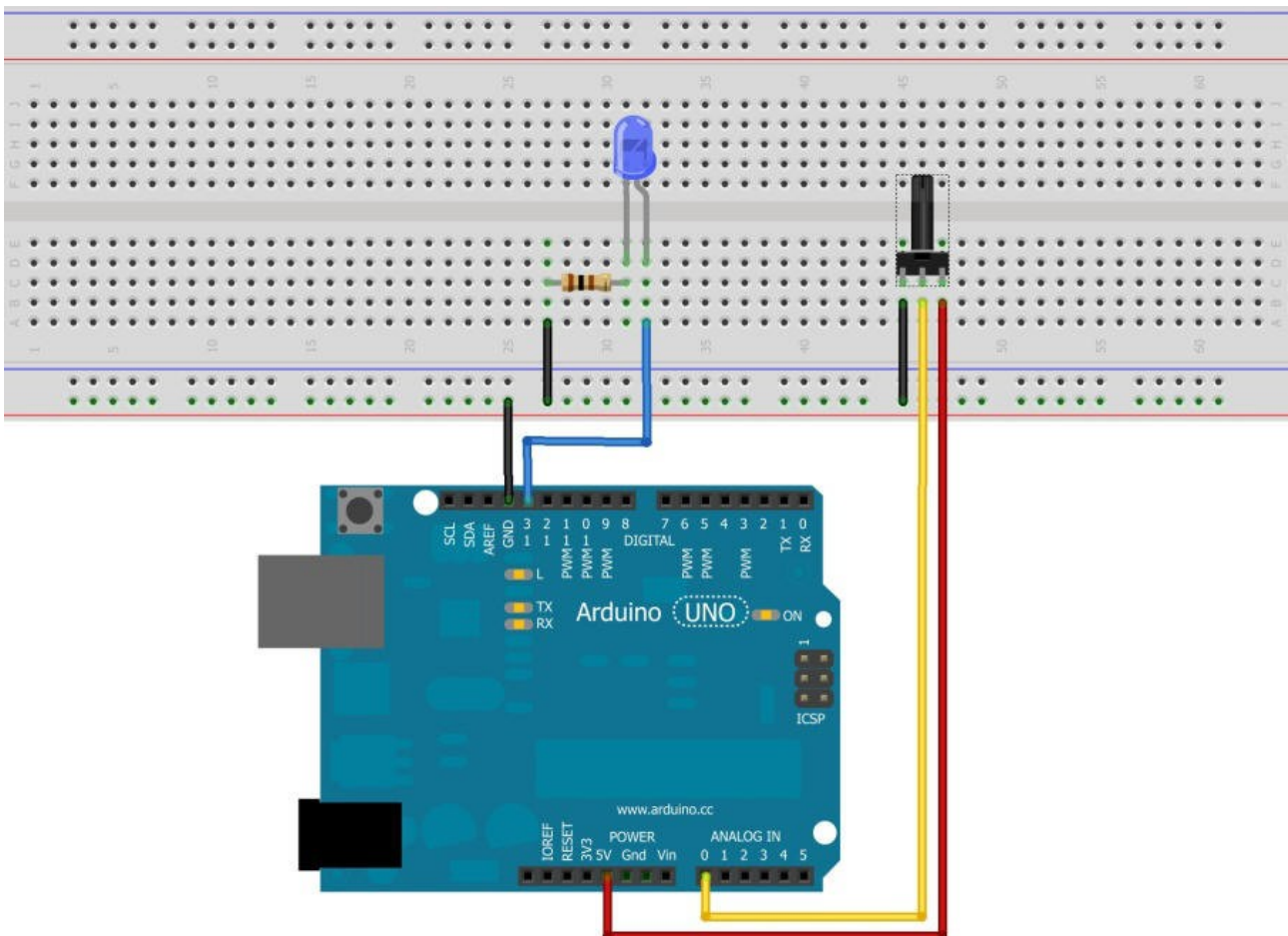
Aufgabe: Eine LED soll blinken. Die Blinkgeschwindigkeit soll mit einem Drehregler eingestellt werden.

Material: Arduino / ein Drehregler (Potentiometer) / Breadboard / Kabel

Lerninhalt: Spannung eines Drehreglers auslesen, Sensorwerte mathematisch verarbeiten und für eine Ausgabe verwenden (In diesem Fall für die Dauer einer Pause).

Ein Drehregler hat drei Anschlüsse. Außen wird + und – angeschlossen. Von dem mittleren Pin geht ein Kabel zu einem analogen Eingangspin am Mikrocontroller-Board. Wenn man den Drehregler dreht, dann gibt der mittlere Pin eine Spannung zwischen 0 und 5 Volt aus. Drehregler ganz links: 0 V und Drehregler ganz rechts: 5V, bzw. Seitenverkehrt, je nach Verkabelung.

Als LED, die blinken soll, verwenden wir wie im ersten Sketch die LED, die mit Pin13 am Mikrocontroller befestigt ist. Zusätzlich kann dort auch noch eine weitere LED angeschlossen werden, wie im Aufbau zu sehen ist



Made with  Fritzing.org

Sketch:

```
int eingang=A0; //Das Wort „eingang“ steht jetzt für den Wert „A0“ (Bezeichnung
//vom Analogport 0)

int LED=13; //Das Wort „LED“ steht jetzt für den Wert 13

int sensorwert=0; //Variable für den Sensorwert mit 0 als Startwert

void setup()
{
    //Hier beginnt das Setup.
    pinMode (LED,OUTPUT); //Der Pin mit der LED (Pin 13) ist jetzt ein Ausgang.
}

void loop()
{
    //Mit dieser Klammer wird der Loop-Teil geöffnet.

    sensorwert= analogRead(eingang); //Die Spannung am Drehregler wird auslesen
//und wie im vorherigen Sketch als Zahl zwischen 0 und 1023 unter der Variable
//„sensorwert“ gespeichert.

    digitalWrite (LED,HIGH); //Die LED wird eingeschaltet

    delay(sensorwert); // Die LED bleibt für so viele Millisekunden eingeschaltet,
//wie der Wert von „sensorwert“ es gespeichert hat.

    digitalWrite(LED, LOW); //Die LED wird ausgeschaltet.

    delay(sensorwert); // Die LED bleibt für so viele Millisekunden ausgeschaltet,
//wie der Wert von „sensorwert“ es gespeichert hat.

} // Mit dieser Klammer wird der Loop-Teil geschlossen

//Der Loop-Teil wird nun erneut gestartet. Wenn sich der Wert des ausgelesenen
//Drehreglers ändert, dann ändert sich auch die Zeit zwischen den Ein- und Aus-
//Phasen der LED. Das Blinken wird dadurch schneller und langsamer. Das längste
//delay beträgt in diesem Sketch 1023ms (Millisekunden). Wenn man längere delays
//benötigt, dann baut man eine kleine mathematische Zeile in den Code ein.
//Beispiel: Man ändert die Zeile „sensorwert =analogRead(eingang);“ in
//„sensorwert =analogRead(eingang)*2;“ Damit wird der abgespeicherte Sensorwert
//um den Faktor 2 vergrößert. Da längste delay wäre dann 2046ms usw...
```

Temperaturen messen

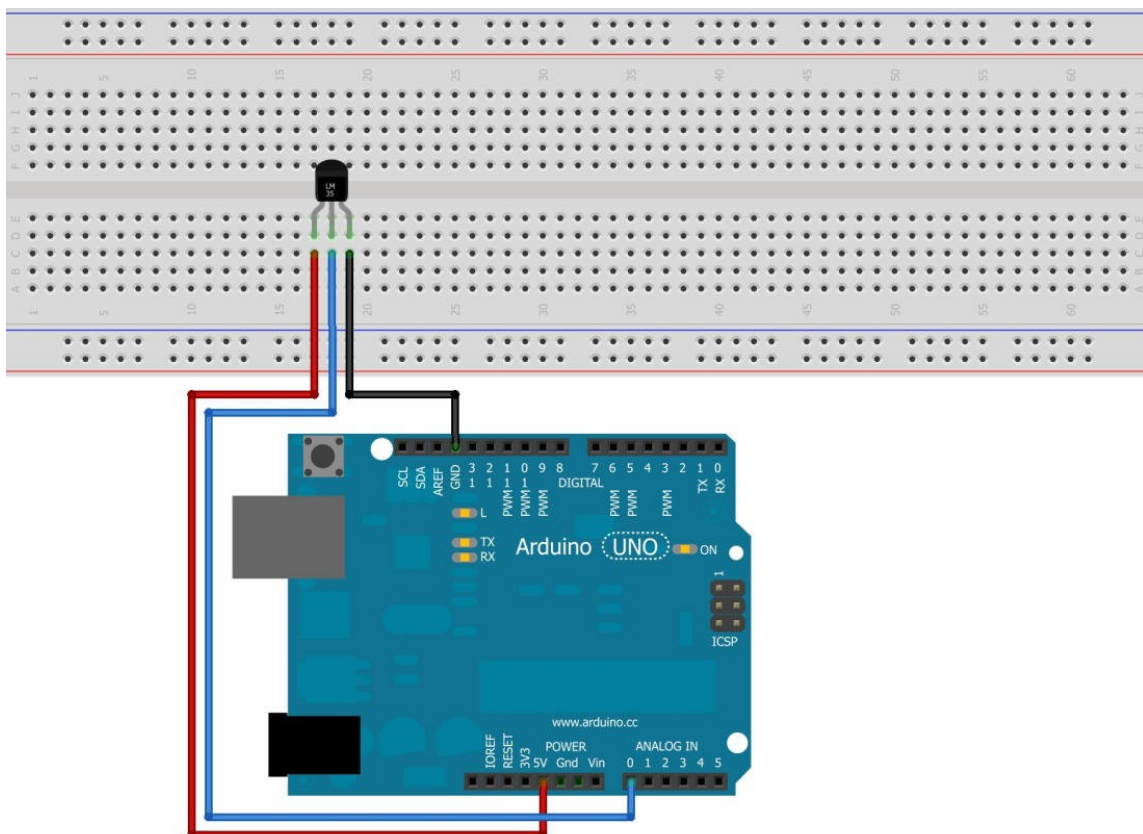
Aufgabe: Mit dem Temperatursensor TMP36 soll die Temperatur ausgelesen und mit dem serial-monitor angezeigt werden.

Material: Arduino / Breadboard / Kabel / Temperatursensor TMP36 / Externe Stromversorgung

Der Sensor hat drei Anschlüsse. Beim Blick auf die flache Seite des Sensors: links 5V, rechts GND und in der Mitte der Pin für das Temperatursignal. Auf diesem Pin gibt der Sensor eine Spannung zwischen 0 und 2,0 Volt aus. Wobei 0V -50°C entsprechen und der Wert 2,0V entspricht 150°C. Laut Hersteller ist der Sensor zwischen -40°C und +125°C einigermaßen genau ($\pm 2^\circ\text{C}$). Die Spannung dieses Pins muss vom Mikrocontroller-Board ausgelesen und in einen Temperaturwert umgerechnet werden.

– ACHTUNG: Wenn der Sensor falsch angeschlossen wird, brennt er durch!

– Bei dem Aufbau sollte nach Möglichkeit eine externe Stromversorgung verwendet werden, da dies die Sensorgenauigkeit wesentlich verbessert (9V Netzteil oder 9V-Batterie).



Made with  Fritzing.org

Sketch:

```
int TMP36 = A0; //Der Sensor soll am analogen Pin A0 angeschlossen werden. Wir
//nennen den Pin ab jetzt „TMP36“

int temperatur = 0; //Unter der Variablen „temperatur“ wird später der
//Temperaturwert abgespeichert

int temp[10]; //Damit man gute Werte erhält, liest man zunächst mehrere Werte
//aus und bildet dann den Mittelwert. Die eckige Klammer „[10]“ erzeugt hier
//gleich zehn Variablen mit den Namen „temp[0]“, „temp[2]“, „temp[3]“, ... bis ...
//„temp[9]“. Mit der Schreibweise [10] spart man also nur etwas Platz.

int time= 20; //Der Wert für „time“ gibt im Code die zeitlichen Abstände
//zwischen den einzelnen Messungen vor.

void setup() {

Serial.begin(9600); // Im Setup beginnen wir die serielle Kommunikation, damit
//wir uns die Temperatur später auch anzeigen lassen können. Über die serielle
//Kommunikation sendet das Board die Messwerte an den Computer. In der Arduino-
//Software kann man unter „Tools“ den „Serial Monitor“ starten um die Messwerte
//zu sehen.

}

void loop() {

temp[0] = map(analogRead(TMP36), 0, 410, -50, 150);
delay(time);

temp[1] = map(analogRead(TMP36), 0, 410, -50, 150);
delay(time);

temp[2] = map(analogRead(TMP36), 0, 410, -50, 150);
delay(time);

temp[3] = map(analogRead(TMP36), 0, 410, -50, 150);
delay(time);

temp[4] = map(analogRead(TMP36), 0, 410, -50, 150);
delay(time);

temp[5] = map(analogRead(TMP36), 0, 410, -50, 150);
delay(time);

temp[6] = map(analogRead(TMP36), 0, 410, -50, 150);
delay(time);

temp[7] = map(analogRead(TMP36), 0, 410, -50, 150);
delay(time);

}
```



```

temp[8] = map(analogRead(TMP36), 0, 410, -50, 150);

delay(time);

temp[9] = map(analogRead(TMP36), 0, 410, -50, 150); //Ab hier wird zehnmahl die
//Temperatur ausgelesen. Dazwischen ist je eine kleine Pause mit der Dauer
//„time“ in Millisekunden. Aber was passiert hier genau? Sehen wir uns den
//Befehl einmal genauer an.
//temp[1] = map(analogRead(TMP36), 0, 410, -50, 150);
//Der Reihe nach:
//temp[1] – ist der Name der ersten Variablen.
//„map ( a, b, c, d, e)“ – Dies ist der sogenannte „Map-Befehl“. Hiermit kann
//man einen eingelesenen Wert (a) bei dem die Werte zwischen (b) und (c) liegen,
//in einen anderen Zahlenbereich umwandeln, und zwar in den Bereich zwischen (d)
//und (e).
//In unserem Befehl passiert folgendes:
//Der Wert des Sensors wird unmittelbar im „Map-Befehl“ ausgelesen mit
//„analogRead(TMP36)“. Die Messwerte sollten zwischen 0 und 410 liegen. Das
//entspricht am analogen Port den Werten zwischen 0 V und 2V. Die Spannung gibt
//der Sensor bei Temperaturen zwischen -50°C und +150°C aus. Diese Werte am
//analogen Port werden nun durch den „Map-Befehl“ direkt in die °Celsius Werte
//zwischen -50 und 150 umgewandelt.

temperatur=(temp[0]+temp[1]+temp[2]+temp[3]+temp[4]+temp[5]+temp[6]+temp[7]+temp
[8]+temp[9])/10; //Alles in eine Zeile! Hier werden alle zehn ermittelten
//Temperaturwerte zusammengerechnet und durch zehn geteilt. Der
//durchschnittliche Wert wird unter der Variablen „temperatur“ gespeichert

Serial.print(temperatur); //Nun wird der Wert „temperatur“ über die serielle
//Kommunikation an den PC gesendet. Nun muss man den seriellen Monitor in der
//Arduino-Software öffnen um die Temperatur am Sensor abzulesen.

Serial.println("Grad");

}

```

Erweiterung des Programms:

Sobald die Temperatur von 30°C erreicht ist, soll ein Warnsignal ertönen (Piepston eines Piezo-Lautsprechers).

```

int TMP36 = A0;

int temperatur = 0;

int temp[10];

int time= 20;

int piezo=5; //Das Wort „piezo“ ist jetzt die Zahl 5, also wird an Port 5 der
//Piezo angeschlossen.

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

pinMode (piezo, OUTPUT); //Der Piezo-Lautsprecher an Pin5 soll ein Ausgang sein

```



```

//(Logisch, weil der ja vom Mikrokontroller-Board ja eine Spannung benötigt um
//zu piepsen.
}
void loop() {
temp[0] = map(analogRead(TMP36), 0, 410, -50, 150);

delay(time);

temp[1] = map(analogRead(TMP36), 0, 410, -50, 150);
temp[9] = map(analogRead(TMP36), 0, 410, -50, 150);

temperatur=(temp[0]+temp[1]+temp[2]+temp[3]+temp[4]+temp[5]+temp[6]+temp[7]+temp
[8]+temp[9])/10; // Alles in eine Zeile!

Serial.print(temperatur);

Serial.println("Grad");

if (temperatur>=30) //Wenn der Wert für die Temperatur über oder gleich 30
//ist, dann...
{
digitalWrite(piezo,HIGH); //...fange an zu piepsen.
}
else // Und wenn das nicht so ist...
{
digitalWrite(piezo,LOW); //...dann sein leise.
}
}
}

```

Entfernung messen

Aufgabe: Mit dem Ultraschallsensor HC-SR04 und einem Arduino Mikrocontroller soll eine Entfernung gemessen und mit dem „serial-monitor“ angezeigt werden.

Wie funktioniert der Ultraschallsensor? Der Sensor hat vier Anschlüsse.

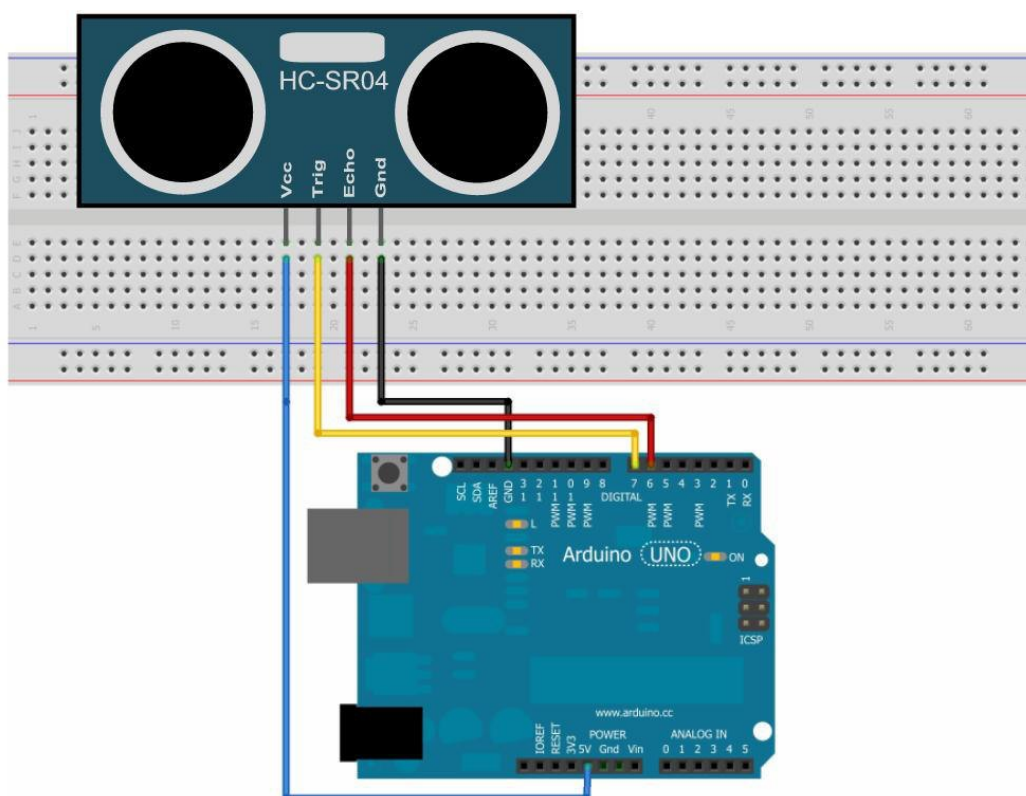
a) 5V(+) b) GND (-) c) echo d) trigger

Die Anschlüsse 5V und GND verstehen sich von selbst, sie versorgen den Sensor mit Energie.

Der Pin „trigger“ bekommt vom Mikrocontroller-Board ein kurzes Signal (5V), wodurch eine Schallwelle vom Ultraschallsensor ausgelöst wird. Sobald die Schallwelle gegen eine Wand oder sonstigen Gegenstand stößt, wird sie reflektiert und kommt irgendwann auch wieder zum Ultraschallsensor zurück. Sobald der Sensor diese zurückgekehrte Schallwelle erkennt, sendet der Sensor auf dem „echo“ Pin ein 5V Signal an das Mikrocontroller-Board. Dieser misst dann lediglich die Zeit zwischen dem Aussenden und der Rückkehr der Schallwelle und rechnet diese Zeit dann in eine Entfernung um. Auf gehts!

Material: Mikrocontroller-Board / Kabel / Breadboard / Hc-SR04 Ultraschallsensor

Aufbau:



Made with Fritzing.org

Code:

```
int trigger=7; //Das Wort trigger ist jetzt die Zahl 7.

int echo=6; //Das Wort echo ist jetzt die Zahl 6.

long dauer=0; //Das Wort dauer ist jetzt eine Variable, unter der die Zeit
//gespeichert wird, die eine Schallwelle bis zur Reflektion und zurück benötigt.
//Startwert ist hier 0.

long entfernung=0; //Das Wort „entfernung“ ist jetzt die variable, unter der
//die berechnete Entfernung gespeichert wird.

//Info: Anstelle von „int“ steht hier vor den beiden Variablen „long“. Das hat
//den Vorteil, dass eine größere Zahl gespeichert werden kann. Nachteil: Die
//Variable benötigt mehr Platz im Speicher.

void setup()
{
  Serial.begin (9600); //Serielle kommunikation starten, damit man sich später
//die Werte am serial monitor ansehen kann.

  pinMode(trigger, OUTPUT); //„trigger“(Pin7) ist ein Ausgang.

  pinMode(echo, INPUT); //„echo“(Pin6) ist ein Eingang.
}

void loop()
{
  digitalWrite(trigger, LOW); //Hier nimmt man die Spannung für kurze Zeit vom
//Trigger-Pin, damit man später beim senden des Trigger-Signals ein rauschfreies
//Signal hat.

  delay(5); //Dauer: 5 Millisekunden

  digitalWrite(trigger, HIGH); //Jetzt sendet man eine Ultraschallwelle los

  delay(10); //Dieser „Ton“ erklingt für 10 Millisekunden.

  digitalWrite(trigger, LOW); //Dann wird der „Ton“ abgeschaltet.

  dauer = pulseIn(echo, HIGH); //Mit dem Befehl „pulseIn“ zählt der
//Mikrokontroller die Zeit in Millisekunden, bis der Schall zum
//Ultraschallsensor zurückkehrt.

  entfernung = (dauer/2) / 29.1; //Nun berechnet man die Entfernung in
//Zentimetern. Man teilt zunächst die Zeit durch zwei (Weil man ja nur eine
//Strecke berechnen möchte und nicht die Strecke hin- und zurück). Den Wert
//teilt man noch durch 29.1 und erhält dann den Wert in Zentimetern, da der
//Schall in der Luft eine Geschwindigkeit von einem Zentimeter pro 29,1
//Mikrosekunden hat.
//Beispiel: Der Sensor misst eine Zeit von 2000 Mikrosekunden. Das sind dann
//1000 Mikrosekunden pro Strecke. Nun teilt man 1000 durch 29.1 (Weil der Schall
//ja 1cm pro 29,1 Mikrosekunden zurücklegt) und erhält den Wert 34,3cm
```

```

if (entfernung >= 500 || entfernung <= 0) //Wenn die gemessene Entfernung über
//500cm oder unter 0cm liegt,...

{

Serial.println("Kein Messwert"); //dann soll der serial monitor ausgeben „Kein
//Messwert“, weil Messwerte in diesen Bereichen falsch oder ungenau sind.
}

else //ansonsten..
{

Serial.print(entfernung); //...soll der Wert der Entfernung an den serial monitor
//hier ausgegeben werden.

Serial.println("cm");
}

delay(1000); //Das delay von einer Sekunde sorgt in ca. jeder neuen Sekunde für
//einen neuen Messwert.
}

```

Erweiterung des Programms:

Wenn ein Abstand unter 80cm gemessen wird, soll ein Piezo-Lautsprecher piepsen.

```

int trigger=12;

int echo=13;

long dauer=0;

long entfernung=0;

int piezo=5; //Das Wort piezo ist jetzt die Zahl 5

void setup()

{

Serial.begin (9600);

pinMode(trigger, OUTPUT);

pinMode(echo, INPUT);

pinMode(piezo, OUTPUT); //Der Piezo-Lautsprecher an Pin5 soll ein Ausgang sein
//(Logisch, weil der ja vom Mikrokontroller-Board ja eine Spannung benötigt um
//zu piepsen

}

void loop()

```

```

{
digitalWrite(trigger, LOW);
delay(5);
digitalWrite(trigger, HIGH);
delay(10);
digitalWrite(trigger, LOW);
dauer = pulseIn(echo, HIGH);
entfernung = (dauer/2) / 29.1;
if (entfernung >= 500 || entfernung <= 0)
{
Serial.println("Kein Messwert");
}
else
{
Serial.print(entfernung);
Serial.println("cm");
}
if (entfernung <= 80) //Wenn der Wert für die Entfernung unter oder gleich 80
//ist, dann...
{
digitalWrite(piezo,HIGH); //...fange an zu piepsen.
}
else //Und wenn das nicht so ist...
{
digitalWrite(piezo,LOW); //...dann sei leise.
}
delay(1000);
}

```

Erweiterung: Rückfahrwarner

Mit diesem Code lässt sich ein Rückfahrwarner konstruieren. Am Pin12 wird zusätzlich zu dem bereits angeschlossenen Ultraschallsensor aus dem Sketch 10 („Entfernung messen“ eine LED angeschlossen.

Ein Bild folgt... kannst du den Rückfahrwarner trotzdem schon konstruieren?

```
int trigger=7;
int echo=6;
long dauer=0;
int LED=12;
long entfernung=0;

void setup()
{
  Serial.begin (9600);
  pinMode(trigger, OUTPUT);
  pinMode(echo, INPUT);
  pinMode(12, OUTPUT);
}

void loop()
{
  digitalWrite(trigger, LOW);
  delay(5);
  digitalWrite(trigger, HIGH);
  delay(10);
  digitalWrite(trigger, LOW);
  dauer = pulseIn(echo, HIGH);
  entfernung = (dauer/2) / 29.1;

  if (entfernung >= 500 || entfernung <= 0)
  {
    Serial.println("Kein Messwert");
  }

  else
  {
    Serial.print(entfernung);
    Serial.println(" cm");
  }

  if (entfernung <= 40)
  {
    digitalWrite(LED, HIGH);
    delay(entfernung*3);
    digitalWrite(LED, LOW);
    delay(entfernung*3);
  }
}
```

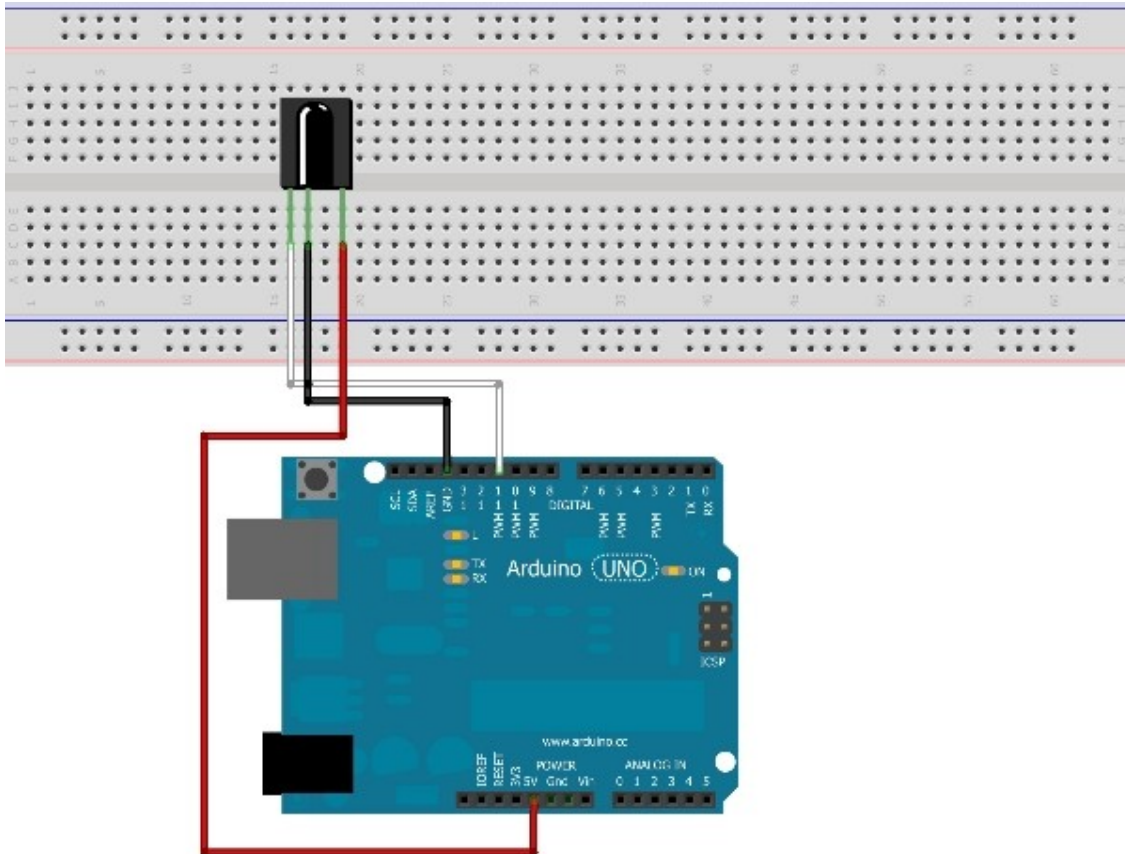
Eine Infrarotfernbedienung zur Ansteuerung von Arduino Mikrocontrollern verwenden.

Material: Arduino / Breadboard / Kabel / Infrarotsensor / Infrarotfernbedienung

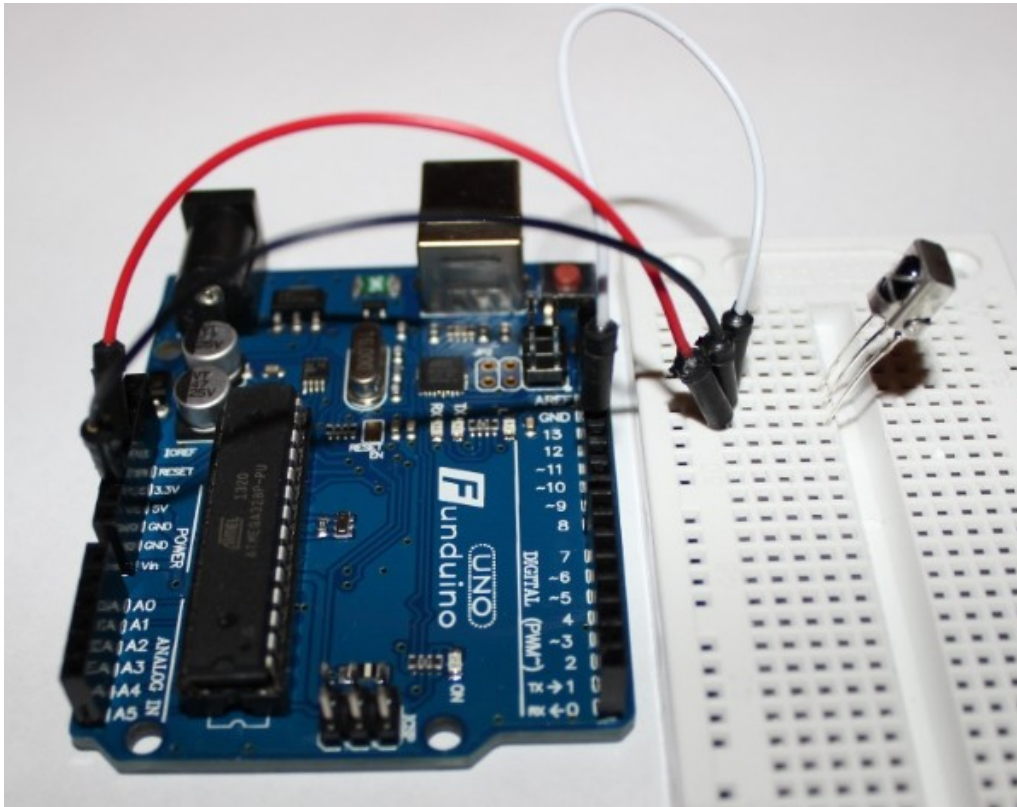
Mit Hilfe eines Infrarotempfängers kann das Arduinoboard die Befehle einer Infrarotfernbedienung auswerten. Die Daten werden dabei in Form von Infrarotlicht von der Fernbedienung zum Empfänger gesendet. Da unser Auge dieses Licht nicht wahrnehmen kann, können wir dieses Licht nicht sehen. Mit einem kleinen Trick kann man jedoch testen, ob bspw. eine Fernbedienung ein Infrarotsignal sendet. Dazu nimmt man eine Digitalkamera (zum Beispiel die vom Smartphone) und betrachtet über das Display die Infrarotdiode. Wenn nun die Fernbedienung betätigt wird, kann man die Infrarotdiode leuchten sehen. Das liegt daran, dass die Sensoren von Digitalkameras das Infrarotlicht wahrnehmen und darstellen können. Das Licht flackert leicht, da die Infrarotdiode sehr schnell an- und aus geht. Dahinter verbirgt sich ein ganz bestimmter Rhythmus, anhand dessen der Infrarotempfänger später auswerten kann, welche Taste an der Fernbedienung gedrückt wurde.



Aufbau:



Made with  Fritzing.org



Dieser Sketch ist eine leichte Abwandlung des Sketches „IRrecvDemo“, welcher sich nicht standardmäßig im Arduino-Hauptprogramm unter den Beispielen finden lässt. Dieser Sketch kann zusammen mit der wichtigen IR-Remote-library von der Internetseite <https://github.com/shirriff/Arduino-IRremote> heruntergeladen werden. Dazu klickt man auf der Seite unten rechts auf „Download ZIP“. Den Ordner in der gepackten Datei kopiert man in das Arduino Hauptprogramm in den Ordner „libraries“ und benennt ihn dort in den Namen „IRremote“ um.

Nun kann man den Sketch im Arduino-Programm unter Datei -> Beispiele -> IRremote -> IRrecvDemo aufrufen.

Dieser Sketch ist sehr kurz gehalten und bietet sich daher sehr gut für die ersten Versuche an.

Code:

```
/*
 * IRremote: IRrecvDemo - demonstrates
 * receiving IR codes with IRrecv
 * An IR detector/demodulator must be
 * connected to the input RECV_PIN.
 * Version 0.1 July, 2009
 * Copyright 2009 Ken Shirriff
 * http://arcfn.com
 */ //Informationen über das ursprüngliche Programm „IrrecvDemo“.
#include <IRremote.h> // Das Programm greift an dieser Stelle auf eine
//„Library“ zurück. Das erleichtert einem viel Arbeit. Denn das Infrarotlicht
//wird mit einem Code verschlüsselt gesendet. Um diesen Code selber auszulesen
//und in passende Werte umzuwandeln, wären sehr viele Zeilen Code erforderlich.

int RECV_PIN = 11; // Der Kontakt der am Infrarotsensor die Daten ausgibt,
//wird mit Pin 11 des Arduinoboards verbunden.

IRrecv irrecv(RECV_PIN); // An dieser Stelle wird ein Objekt definiert, dass
//den Infrarotsensor an Pin 11 ausliest.

decode_results results; // Dieser Befehl sorgt dafür, dass die Daten, die per
//Infrarot eingelesen werden unter „results“ abgespeichert werden.

void setup()
{
  Serial.begin(9600); //Im Setup wird die Serielle Verbindung gestartet, damit
//man sich die Empfangenen Daten der Fernbedienung per seriellen Monitor ansehen
//kann.

  pinMode (13, OUTPUT);

  irrecv.enableIRIn(); //Dieser Befehl initialisiert den Infrarotempfänger.
}

void loop()
{ //Der loop-Teil fällt durch den Rückgriff auf die „library“ sehr kurz aus.
  if (irrecv.decode(&results)) { //Wenn Daten empfangen wurden,
    Serial.println(results.value, DEC); //werden sie als Dezimalzahl (DEC) an den
//Serial-Monitor ausgegeben.
    irrecv.resume(); //Der nächste Wert soll vom IR-Empfänger eingelesen werden
  }
}
```

Ein Druck auf die Taste „1“ der Infrarotfernbedienung bewirkt, dass am Serialmonitor die Zahl „16724175“ ausgegeben wird. Dies ist der entschlüsselte Zahlencode, der sich hinter dieser Taste verbirgt.

Wenn man die Taste dauerhaft gedrückt hält, erscheint permanent die Zahl „4294967295“. Dies ist der Code der angibt, dass eine Taste dauerhaft gedrückt wird. Diese Zahl ist unabhängig davon, welche Taste gedrückt wird.

Es können aber auch noch andere Zahlen auftauchen, falls eine Taste nur ganz kurz oder pulsierend gerückt wird. In dem Fall kann der Sensor keinen eindeutigen Wert auslesen.

Erweiterung des Programms:

Beim Drücken der Taste“1“ soll eine LED an gehen und beim Drücken der Taste“2“ soll die LED aus gehen.

```
#include <Irremote.h>

int RECV_PIN = 11;

IRrecv irrecv(RECV_PIN);

decode_results results;

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

  pinMode (13, OUTPUT); //An Pin 13 wird eine LED angeschlossen.
  digitalWrite(13, LOW); //Diese soll zunächst aus sein
  irrecv.enableIRIn();
}

void loop() {
  if (irrecv.decode(&results)) {
    Serial.println(results.value, DEC);

    if (results.value == 16724175) //Wenn der Infrarotempfänger die Zahl
//„16724175“ ausgelesen hat (Entsprechend der Taste“1“ der Fernbedienung)

    {digitalWrite (13, HIGH);} //soll die LED an gehen.

    if (results.value == 16718055) //Wenn der Infrarotempfänger die Zahl
//„16718055“ ausgelesen hat (Entsprechend der Taste“2“ der Fernbedienung),

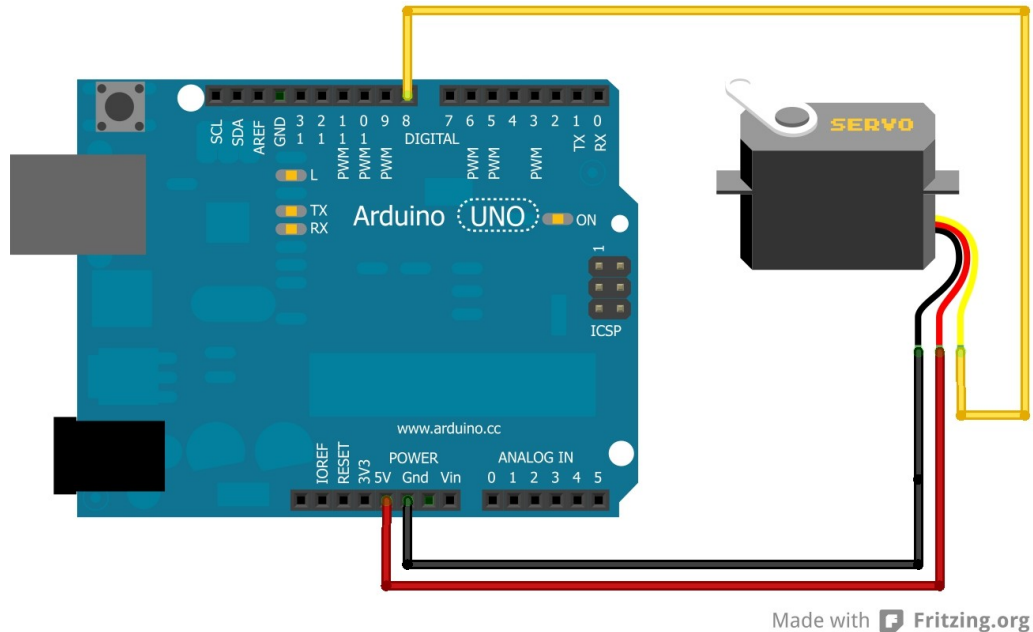
    {digitalWrite (13, LOW);} //soll die LED aus gehen.

    irrecv.resume();
  }
}
```

Servo ansteuern

Aufgabe: Ein Servo soll von einem Arduino-Mikrocontroller angesteuert werden. Der Servo soll dazu in diesem Beispiel drei verschiedene Positionen ansteuern und zwischen den Positionen eine kurze Zeit warten.

Material: Ein Mikrocontrollerboard, Ein Servo, drei Steckkabel



Code:

```
#include <Servo.h>    //Die Servobibliothek wird aufgerufen.

Servo servoblau;    //Erstellt für das Programm ein Servo mit dem Namen
//„servoblau“

void setup()
{
servoblau.attach(8);    //Servo wird mit Pin8 verbunden
}

void loop()
{
servoblau.write(0);    //Position 1 ansteuern mit dem Winkel 0°
delay(3000);    //Das Programm stoppt für 3 Sekunden
servoblau.write(90);    //Position 2 ansteuern mit dem Winkel 90°
delay(3000);    //Das Programm stoppt für 3 Sekunden
servoblau.write(180);    //Position 3 ansteuern mit dem Winkel 180°
delay(3000);    //Das Programm stoppt für 3 Sekunden
servoblau.write(20);    //Position 4 ansteuern mit dem Winkel 20°
delay(3000);    //Das Programm stoppt für 3 Sekunden
}
```

LCD Display

Aufgabe: Ein LCD Display soll mit einem Arduino Mikrocontroller angesteuert werden. Danach soll auf dem Display ein vorher festgelegter Text wie auf folgendem Beispielfoto erscheinen.

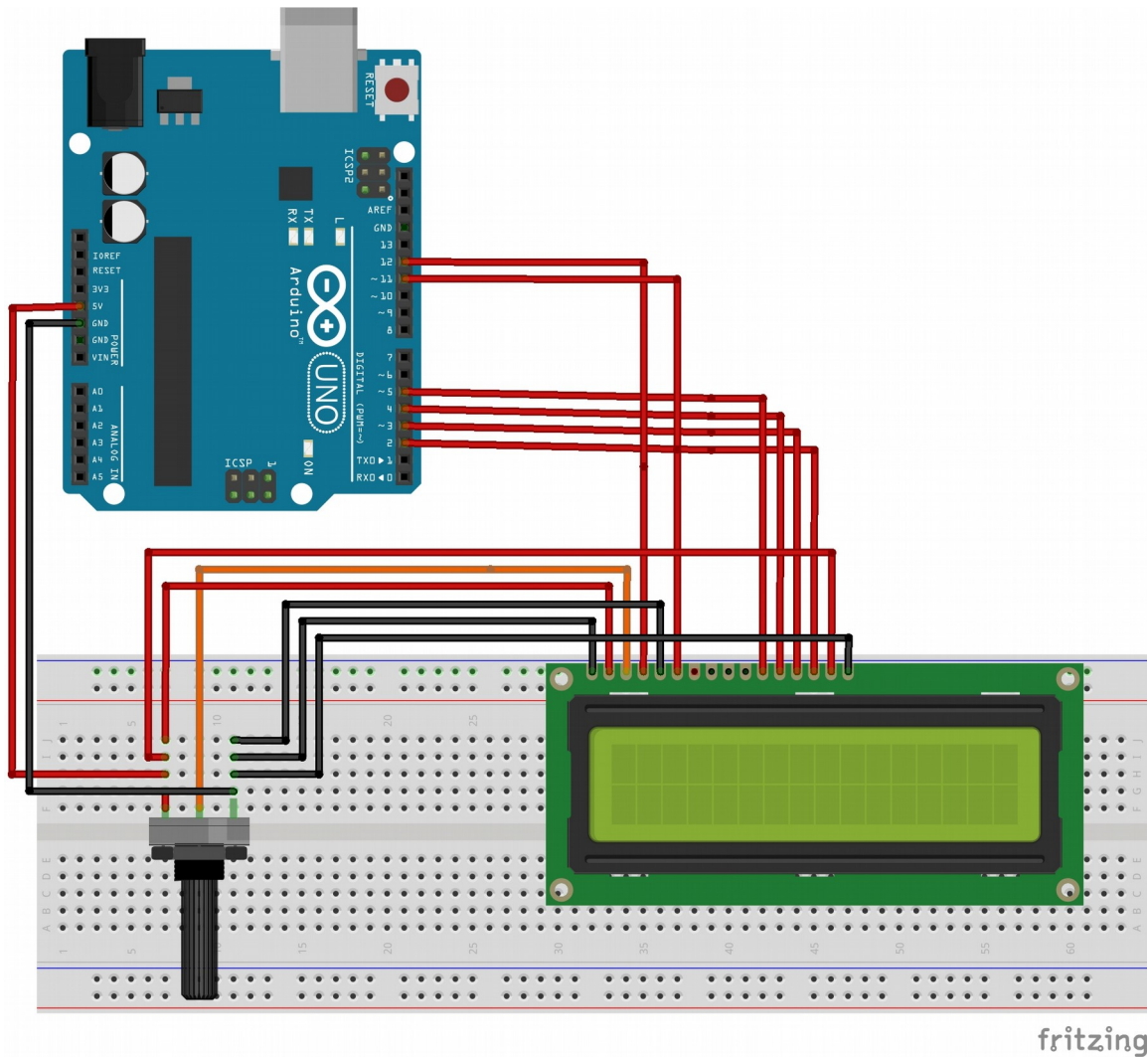
Material: Mikrocontrollerboard (In diesem Beispiel UNO R3), ein Drehregler (bzw. Potentiometer), 14 Jumperkabel, Breadboard



Anhand des Materials und der Skizze erkennt man schnell, dass die Verkabelung nicht so leicht ist. Das liegt daran, dass das LCD Display mit sehr vielen Kabeln verbunden werden muss. Eine weitere Schwierigkeit sind fehlende Buchsen mit denen man die Kabel anschließen könnte. Daher bietet es sich an, eine Steckbuchenleiste aufzulöten oder die Kabel direkt an das LCD anzulöten. Im zweiten Fall bietet es sich an, ein Flachbandkabel zu verwenden (Bspw. von alten IDE-Laufwerken wie Festplatten, CD- oder DVD Laufwerken). Ohne eine gelötete Verbindung ist es nahezu ausgeschlossen, gute Ergebnisse zu erzielen.

Der Drehregler ist dazu da, den Kontrast des LCD einzustellen. Die LED Hintergrundbeleuchtung wird wie in der Skizze dargestellt mit 5V versorgt. Da man in der Skizze die Beschriftung am LCD nicht erkennen würde, wird sie nicht dargestellt. Man muss also die Position der Kabel am LCD abzählen (Beispiel: Am LCD wird das erste Kabel von rechts mit GND verbunden. Das zweite Kabel von rechts wird mit 5V verbunden usw...). Info: Für schnelle Basteleien benutzen viele Bastler lieber ein LCD-Keypad-Shield oder ein I2C-LCD, da man sich bei den beiden Alternativen nicht mehr um die Verkabelung des LCD kümmern muss. Allerdings sind die beiden genannten Alternativen auch teurer.

Aufbau:



Wenn man die Verkabelung erfolgreich hergestellt hat, kann man sich mit der Software befassen. Wie bei vielen anderen Bauteilen, wird auch hier auf eine „Library“ zurückgegriffen. Die Library für das LCD Display ist Bestandteil der Arduino-Software und muss nicht zusätzlich installiert werden.

Code:

```
#include <LiquidCrystal.h> // LCD-Bibliothek laden

LiquidCrystal lcd(12, 11, 5, 4, 3, 2); // In dieser Zeile wird festgelegt,
//welche Pins des Mikrocontrollerboards für das LCD verwendet wird (Am besten
//erstmal nicht verändern).

void setup() {

  lcd.begin(16, 2); //Im Setup wird angegeben, wie viele Zeichen und Zeilen
  verwendet werden. Hier: 16 Zeichen in 2 Zeilen

}

void loop() {

  lcd.setCursor(0, 0); //Startposition der Darstellung auf dem LCD festlegen.
  //Bedeutet: Erstes Zeichen in der ersten Zeile.

  lcd.print("www.funduino.de"); //Dort soll der Text „www.funduino.de“
  //erscheinen.

  lcd.setCursor(0, 1); // lcd.setCursor(0,1) bedeutet: Erstes Zeichen in der
  //zweiten Zeile.

  lcd.print("Viel Erfolg!!!"); //Dort soll dann der Text „Viel Erfolg!!!“
  //auftauchen.

}
```

Eine Variation: Im Display soll abwechselnd erst oben und dann unten ein Text erscheinen. In diesem Beispiel der Text „Oben“ und „Unten“.

```
#include <LiquidCrystal.h>

LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

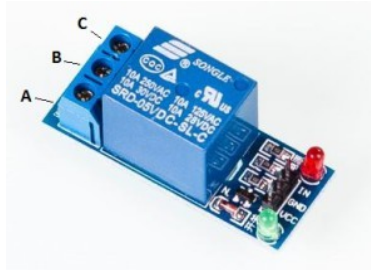
void setup() {
  lcd.begin(16, 2);
}

void loop() {
  lcd.setCursor(0, 0); //Beginn beim ersten Zeichen in der ersten Zeile mit dem
//Text „Oben“.
  lcd.print("Oben");
  delay (2000); //Zwei Sekunden warten.
  lcd.clear(); //Display löschen.
  lcd.setCursor(5, 1); //Erneuter Beginn beim fünften Zeichen in der zweiten
//Zeile mit dem Text „Unten“.
  lcd.print("Unten");
  delay (2000); //Zwei Sekunden warten.
  lcd.clear(); //Display löschen.
}
```

Ein LCD eignet sich besonders gut, um Sensorwerte oder andere Ausgaben des Mikrocontrollerboards anzuzeigen. Weitere Hilfe bekommt man z.B. in der Arduino-Software. Unter den Beispiel-Sketches findet man eine Reihe von verschiedenen Beispielen unter dem Menüpunkt „LiquidCrystal“.

Relaiskarte

Beim Basteln mit Arduino sind Relais sehr wichtig. Relais sind Schalter, die einen größeren Strom fließen lassen können, als es vom Mikrocontroller aus möglich wäre. Mit dem Arduino aktiviert man nun also nur noch über einen ganz kleinen Strom das Relais, welches dann auch größere elektrische Geräte betreiben kann.



An den Kontakten (auf dem Foto am rechten Bildrand, zwischen der roten und der grünen LED) wird die Relaiskarte mit dem Arduino verbunden. Die Karte ist im Betrieb dauerhaft mit 5V+ und GND (-) verbunden. Der Pin mit der Aufschrift „IN“ wird mit einem digitalen Pin des Arduinoboards verbunden. Solange an dem Pin „IN“ kein Signal anliegt (ausgegeben vom digitalen Pin des Arduino) sind die Schraubkontakte A und B miteinander verbunden. Sobald ein Signal anliegt, werden die Kontakte B und C miteinander verbunden.

ACHTUNG: Es gibt Relaiskarten die schalten, wenn an dem Pin „IN“ GND angelegt wird und es gibt Relaiskarten die schalten, wenn an dem Pin „IN“ eine Spannung von 5V+ angelegt wird. Welche Version man hat lässt sich leicht feststellen, indem man den „Signal-„Pin einmal mit GND und einmal mit 5V+ des Arduino Boards verbindet. Das Schalten der Relaiskarte ist durch ein lautes knacken deutlich zu erkennen.

An den unten Kontakten (A, B, C) kann ein elektrisches Gerät angeschlossen werden, bei dem ein höherer elektrischer Strom fließt, als es der Arduino liefern könnte. Zum Beispiel einen großen Elektromotor, eine große Lampe usw.

Als Beispielcode kann in diesem Fall der einfache „Blink-“ Code verwendet werden. Anstelle der LED schließt man den Ausgabepin des Arduinoboards an den „Signal-„Pin der Relaiskarte an. Das Relais wird dann im Sekundentakt schalten.

Code:

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

void loop()
{
  digitalWrite(6, HIGH); //An dieser Stelle würde das Relais einsschalten

  delay(1000); //...eine Sekunde warten

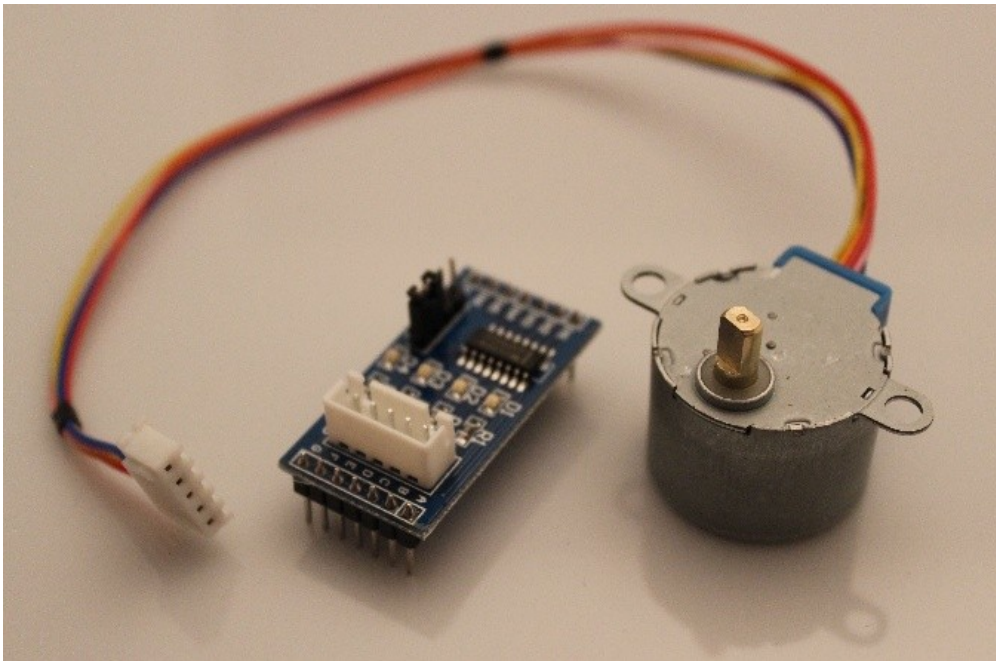
  digitalWrite(6, LOW); //Und wieder ausschalten

  delay(1000); //...und eine Sekunde warten.
}
```

Schrittmotor

Bei diesem Schrittmotor handelt es sich um einen Schrittmotor der sich speziell für kleine Anwendungen mit dem Arduino-Board eignet. Die Besonderheit liegt darin, dass er ohne eine externe Spannungsversorgung betrieben werden kann. Der Motor entwickelt dabei ein relativ hohes Drehmoment. Dies wird durch ein Getriebe realisiert, welches innerhalb des Metallgehäuses vor dem eigentlichen Schrittmotor verbaut wurde. Dadurch wird es in dieser kompakten Bauweise überhaupt erst möglich, dass sich eine ganze Umdrehung der Antriebswelle auf 2048 Einzelschritte aufteilen lässt. Ein kleiner daraus resultierender Nachteil ist die langsame maximale Drehgeschwindigkeit.

Der Schrittmotor wird an eine Motorsteuerungsplatine angeschlossen. Diese versorgt den Motor mit ausreichend elektrischer Energie, damit die Leistung nicht von den digitalen Pins des Arduino-Boards aufgebracht werden muss. Die Steuerungsplatine gibt es in zwei Versionen, bei denen die seitlich angebrachten Pins entweder nach oben oder nach unten aus der Platine herausragen. Die Anschlussbelegung ist jedoch gleich.



Verkabelung

PIN2 am Arduino-Board auf IN4 der Motorsteuerplatine **!!! 2 auf 4 !!!**

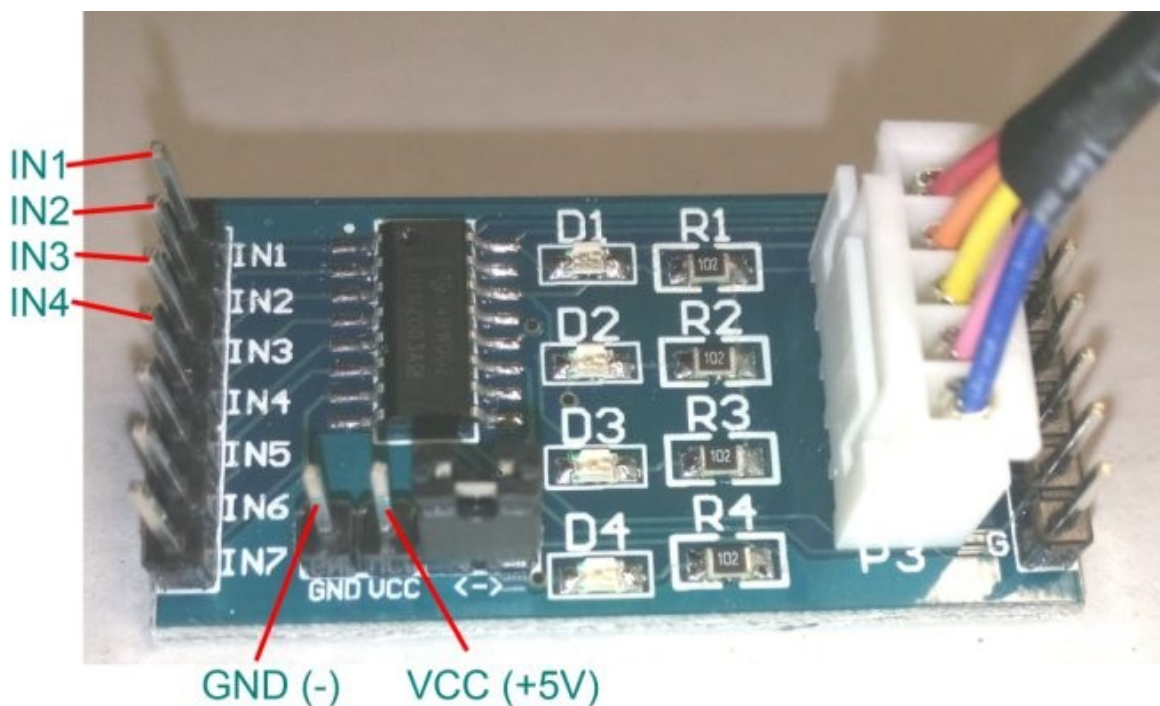
PIN3 am Arduino-Board auf IN2 der Motorsteuerplatine **!!! 3 auf 2 !!!**

PIN4 am Arduino-Board auf IN3 der Motorsteuerplatine **!!! 4 auf 3 !!!**

PIN5 am Arduino-Board auf IN1 der Motorsteuerplatine **!!!5 auf 1 !!!**

PIN „GND“ am Arduino-Board auf GND der Motorsteuerplatine

PIN „5V“ am Arduino-Board auf VCC der Motorsteuerplatine



Dies ist ein Beispielcode, der den Motor abwechselnd um 2048 Schritte (entspricht einer ganzen Umdrehung) vor- und zurückdrehen lässt.

Code

```
#include <Stepper.h>

int SPMU = 32;

Stepper myStepper(SPMU, 2,3,4,5);

void setup()
{
  myStepper.setSpeed(500);
}

void loop() {
  myStepper.step(2048);
  delay(500);
  myStepper.step(-2048);
  delay(500);
}
```

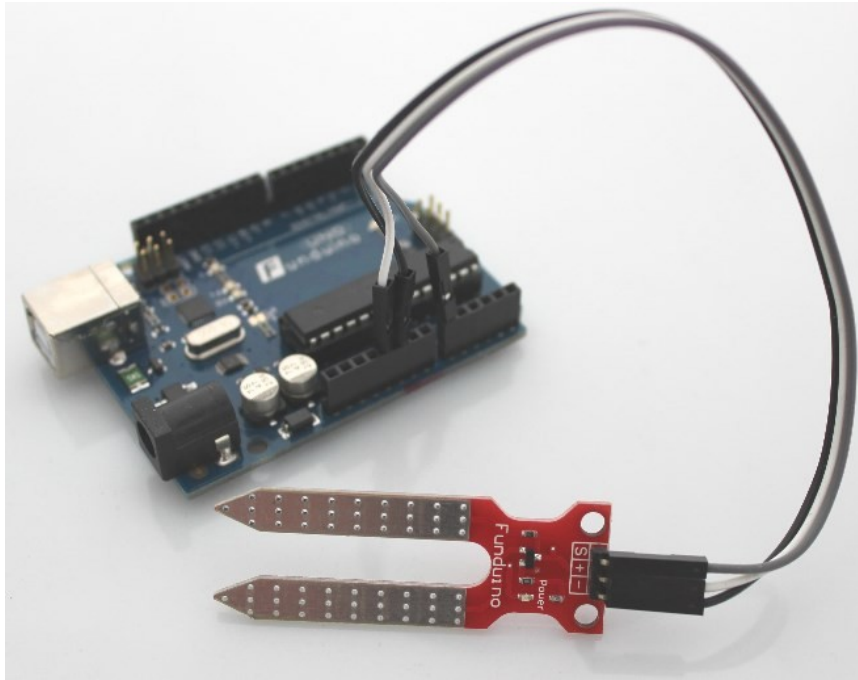
Feuchtigkeitssensor

Anleitung zum Feuchtigkeitssensor



Mit einem Feuchtigkeitssensor (Moisture Sensor) kann man wie es der Name schon sagt die Feuchtigkeit messen. Dies bezieht sich jedoch auf die direkt anliegende Feuchtigkeit, wie z.B. Hautfeuchtigkeit oder Bodenfeuchtigkeit, nicht jedoch von Luftfeuchtigkeit. Man kann ihn zum Beispiel verwenden, um die Feuchtigkeit im Boden von Pflanzen zu messen. Im Falle von Trockenheit könnte dann ein Alarmsignal ertönen, oder eine elektrische Wasserpumpe könnte die Pflanze automatisch mit Wasser versorgen. Der Sensor eignet sich ebenfalls, um den Wasserstand im Bereich des Sensors zu messen. Die Funktionsweise ist einfach. An den beiden Kontakten des Sensors liegt eine Spannung an. Je höher die Feuchtigkeit zwischen den beiden Kontakten ist, desto besser kann der Strom von einem Kontakt zum anderen fließen. Dieser Wert wird im Sensor elektronisch aufbereitet und in Form eines analogen Signals an einen analogen Eingang des Boards übermittelt. Da das Board, wie bereits in vorherigen Tutorials beschrieben, keine elektrische Spannung als solche messen kann, wandelt es die am analogen Pin anliegende Spannung in einen Zahlenwert um. 0 bis 12 Volt entspricht einem Zahlenwert von 0 bis 1023 (Das sind 1024 Zahlen, da die Null als erster Zahlenwert gezählt wird).

Bei dem Feuchtigkeitssensor liegt das obere Limit jedoch ca. bei dem Zahlenwert 800, wenn der Sensor komplett im Wasser eingetaucht ist. Die genaue Kalibrierung ist jedoch abhängig vom Sensor und von der Art der Flüssigkeit/Feuchtigkeit, die gemessen wird (bspw. hat Salzwasser eine bessere Leitfähigkeit und der Wert wäre entsprechend höher).
Aufbau:



Die Programmierung ist sehr einfach und ähnelt sehr stark dem Auslesen von Potentiometern, da einfach nur ein analoger Wert ausgelesen wird.

```
int messwert=0; //Unter der Variablen „messwert“ wird später der Messwert des
//Sensors gespeichert.

void setup()
{ // Hier beginnt das Setup.

Serial.begin(9600); //Die Kommunikation mit dem seriellen Port wird gestartet.
//Das benötigt man, um sich den ausgelesenen Wert im serial monitor anzeigen zu
//lassen.

}

void loop()
{ // Mit dieser Klammer wird der Loop-Teil geöffnet.

messwert=analogRead(A0); //Die Spannung an dem Sensor wird ausgelesen und
//unter der Variable „messwert“ gespeichert.

Serial.print("Feuchtigkeits-Messwert:"); //Ausgabe am Serial-Monitor: Das Wort
//„Feuchtigkeits-Messwert:“

Serial.println(messwert); //und im Anschluss der eigentliche Messwert

delay(500); //Zum Schluss noch eine kleine Pause, damit nicht zu viele
//Zahlenwerte über den Serial-Monitor rauschen.
}
```

Erweiterung des Programms:

Nun soll ein Alarmsignal mit Hilfe eines Piezo-Speakers ertönen, sobald die gemessene Feuchtigkeit einen bestimmten Grenzwert unterschreitet.

Als Grenzwert wird in diesem Fall der Wert „200“ festgelegt.

Code:

```
int messwert=0;

int PIEPS=6; //Mit dem Namen „PIEPS“ wird jetzt der Pin6 bezeichnet, an dem
ein Piezo-Speaker angeschlossen wird.

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

  pinMode (6,OUTPUT); //Im Setup wird der Pin6 als Ausgang festgelegt
}

void loop()

{
  messwert=analogRead(A0);

  Serial.print("Feuchtigkeits-Messwert:");
  Serial.println(messwert);

  delay(500);

  if ( messwert <200) //Hier beginnt die Abfrage: Wenn der Sensorwert kleiner
//als „200“ ist, dann..

  {

  digitalWrite(PIEPS, HIGH); //...soll der Piezo-Speaker piepsen

  }

  else //...ansonsten...

  {

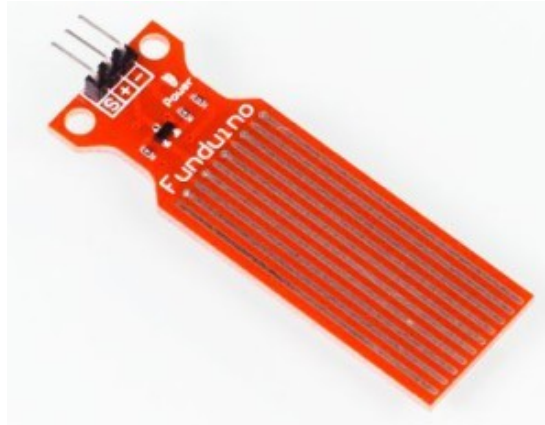
  digitalWrite(PIEPS, LOW); //...soll er leise sein.

  }

}
```

Tropfensensor

Mit einem Tropfensensor oder auch Flüssigkeitssensor kann man wie es der Name schon sagt eine Flüssigkeit detektieren. Dazu muss sich die Flüssigkeit direkt auf dem Sensor befinden. Es reicht bereits ein kleiner Tropfen aus, um einen eindeutigen Messwert zu erhalten.



Man kann den Sensor zum Beispiel als Regensensor verwenden. Sobald ein Tropfen auf den Sensor trifft, kann das Arduino-Board eine Aktion ausführen wie z.B. eine Markise einrollen, Jalousien schließen, einen Alarm auslösen oder einen Scheibenwischer betätigen.

Die Funktionsweise ist einfach. An den langen Kontaktstellen die den Sensor durchziehen liegt eine Spannung an (entweder + oder -). Sobald eine Flüssigkeit bspw. durch einen Tropfen zwei Kontakte verbindet, fließt ein kleiner Strom von einem Kontakt zum Anderen. Dieser Wert wird im Sensor elektronisch aufbereitet und in Form eines analogen Signals an einen analogen Eingang des Boards übermittelt. Da das Board, wie bereits in vorherigen Tutorials beschrieben, keine elektrische Spannung als solche messen kann, wandelt es die am analogen Pin anliegende Spannung in einen Zahlenwert um. 0 bis 12 Volt entspricht einem Zahlenwert von 0 bis 1023 (Das sind 1024 Zahlen, da die Null als erster Zahlenwert gezählt wird).

Bei dem Flüssigkeitssensor liegt der Wert im trockenen bei null „0“. Sobald ein Tropfen Wasser auf die Kontakte des Sensors trifft, liegt der Wert bei ca „480“. Je mehr Tropfen sich auf dem Sensor befinden, desto höher ist der Wert.

Im ersten Code geht es nur darum, den Sensorwert mit dem Arduino-Board auszulesen und mit dem „serial monitor“ darzustellen.

Aufbau: Siehe Bild Seite 57 (Die beiden oberen Kabel (rot und schwarz) und der Piezo-Speaker auf dem Breadboard werden erst später benötigt)

Die Programmierung ist sehr einfach und ähnelt sehr stark dem Auslesen von Potentiometern oder dem Auslesen des Feuchtigkeitssensors, da einfach nur ein analoger Wert ausgelesen wird.


```
int messwert=0; //Unter der Variablen „messwert“ wird später der Messwert des
//Sensors gespeichert.

void setup()
{ // Hier beginnt das Setup.

Serial.begin(9600); //Die Kommunikation mit dem seriellen Port wird gestartet.
//Das benötigt man, um sich den ausgelesenen Wert im serial monitor anzeigen zu
//lassen.

}

void loop()
{ // Mit dieser Klammer wird der Loop-Teil geöffnet.

messwert=analogRead(A0); //Die Spannung an dem Sensor wird ausgelesen und
//unter der Variable „messwert“ gespeichert.

Serial.print("Feuchtigkeits-Messwert:"); //Ausgabe am Serial-Monitor: Das Wort
//„Feuchtigkeits-Messwert:“

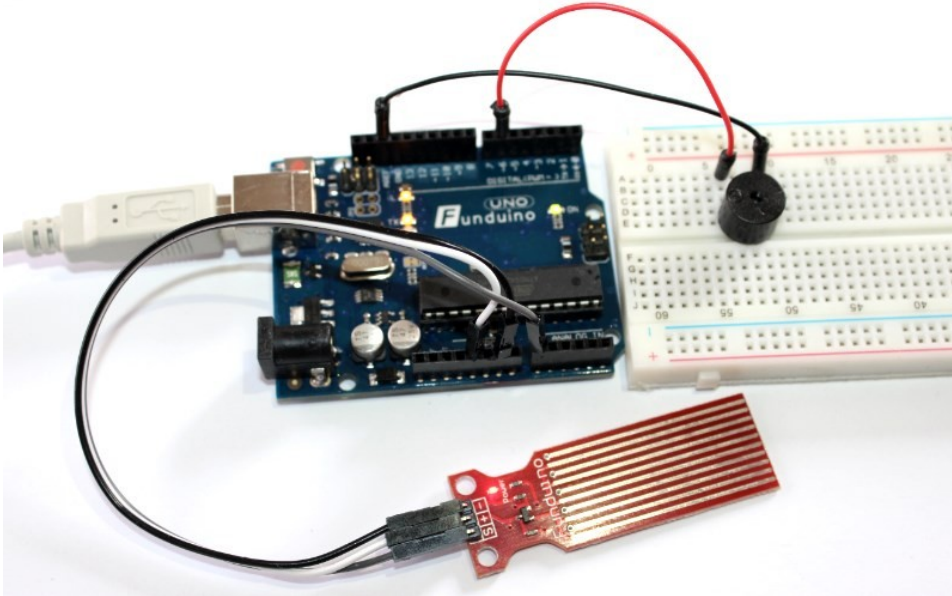
Serial.println(messwert); //und im Anschluss der eigentliche Messwert

delay(500); //Zum Schluss noch eine kleine Pause, damit nicht zu viele
//Zahlenwerte über den Serial-Monitor rauschen.
}
```

Erweiterung des Programms:

Nun soll ein Alarmsignal mit Hilfe eines Piezo-Speakers ertönen, sobald ein Regentropfen auf den Sensor trifft. Als Grenzwert wird in diesem Fall der Messwert 400 festgelegt, da bei einem Tropfen auf dem Sensor ein Messwert von ca. 480 zu erwarten ist.

Aufbau:



Code:

```
int messwert=0;

int PIEPS=6; //Mit dem Namen „PIEPS“ wird jetzt der Pin6 bezeichnet, an dem
ein Piezo-Speaker angeschlossen wird.

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

pinMode (6,OUTPUT); //Im Setup wird der Pin6 als Ausgang festgelegt
}

void loop()
{

messwert=analogRead(A0);

Serial.print("Feuchtigkeits-Messwert:");

Serial.println(messwert);

delay(500);

if ( messwert >400) //Hier beginnt die Abfrage: Wenn der Sensorwert kleiner
//als „400“ ist, dann...

{

digitalWrite(PIEPS, HIGH); //...soll der Piezo-Speaker piepsen

}

else //...ansonsten...

{

digitalWrite(PIEPS, LOW); //...soll er leise sein.

}

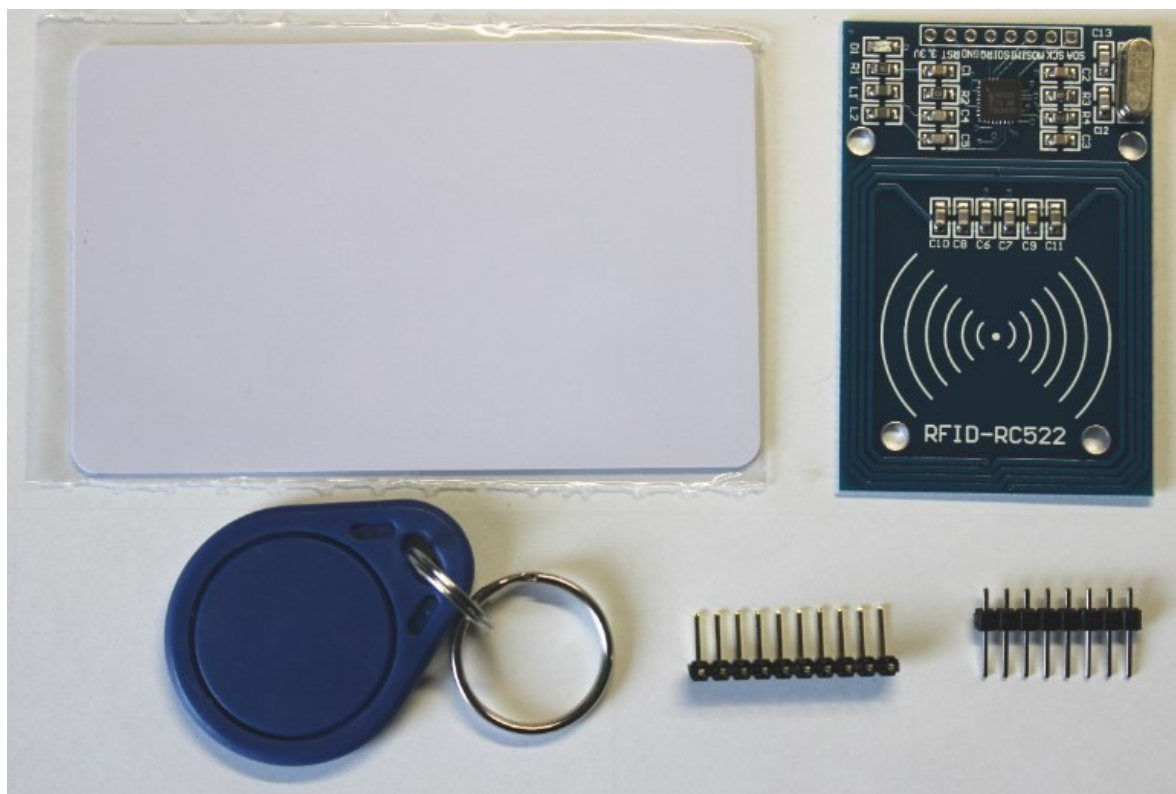
}
```

RFID Kit

Der RFID („radio-frequency identification“) Reader wird verwendet, um von RFID Sendern (auch „RFID Tags“ genannt) per Funk einen bestimmten Code auszulesen. Jeder Sender hat dabei nur einen einmaligen ganz individuellen Code. Damit lassen sich mit dem Arduino Schließanlagen oder ähnliche Projekte verwirklichen, bei denen sich eine Person mit einem Sender identifizieren soll.

RFID-TAGs können verschiedene Formen haben, wie z.B. Schlüsselanhänger oder Karten im Kreditkartenformat.

Auf dem folgenden Bild sieht man Links oben und unten zwei RFID-TAGs, rechts oben den RFID-Empfänger RFID-RC522 und unten rechts Stiftleisten zum anlöten an den Empfänger.



Wie funktioniert das ganze? Ein RFID-Empfänger enthält eine kleine Kupferspule, die ein magnetisches Feld erzeugt. Ein RFID-Sender enthält ebenfalls eine Kupferspule, die das magnetische Feld aufgreift und in dem Sender eine elektrische Spannung erzeugt. Diese Spannung wird dann verwendet um einen kleinen elektronischen Chip dazu zu bringen, per Funk einen elektrischen Code auszusenden. Dieser Code wird dann direkt vom Sender empfangen und so verarbeitet, dass der Arduino-Mikrocontroller den empfangenen Code weiterverarbeiten kann.

Es ist auch möglich, RFID-TAGs zu mit einem Code zu beschreiben. Dies wird aufgrund der Komplexität in dieser Anleitung nicht behandelt. Weiterführende Tutorials finden sich genügend im Netz.

Einen RFID-TAG mit Arduino auslesen und die Daten verarbeiten

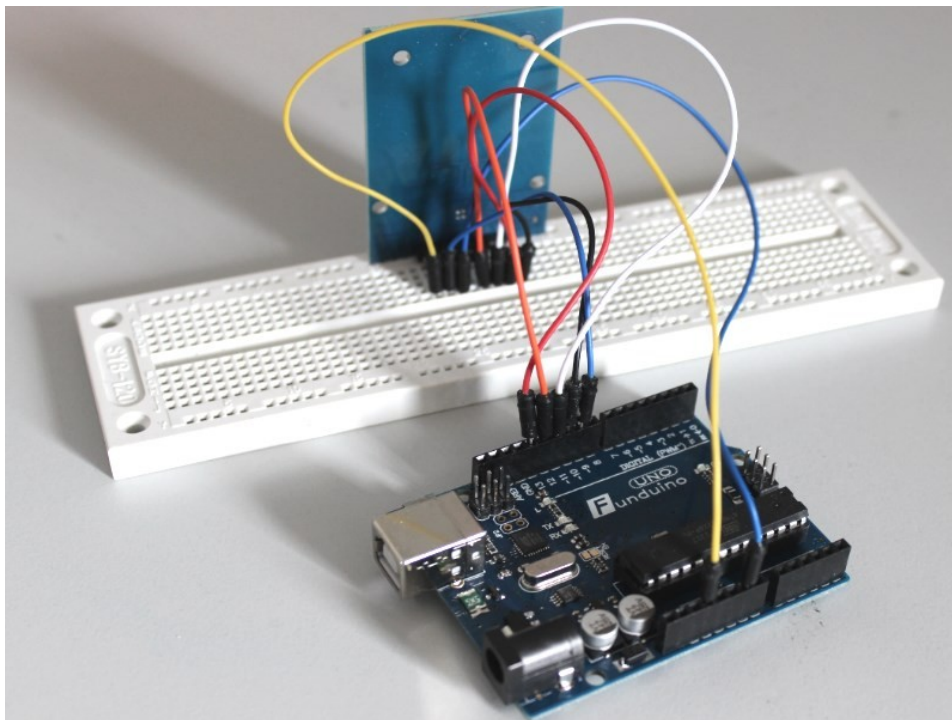
Material: Arduino UNO oder MEGA, RFID-READER, mindestens einen RFID-TAG, Breadboard, einige Breadboardkabel, eine LED, ein 200 Ohm Widerstand

Aufgabe: Mit Hilfe eines Arduino-Mikrocontrollers soll ein RFID-TAG ausgelesen werden. Sofern es sich um den richtigen TAG handelt, soll eine Leuchtdiode für 5 Sekunden leuchten.

Verkabelung des Arduino-Boards mit dem RFD-Empfänger:

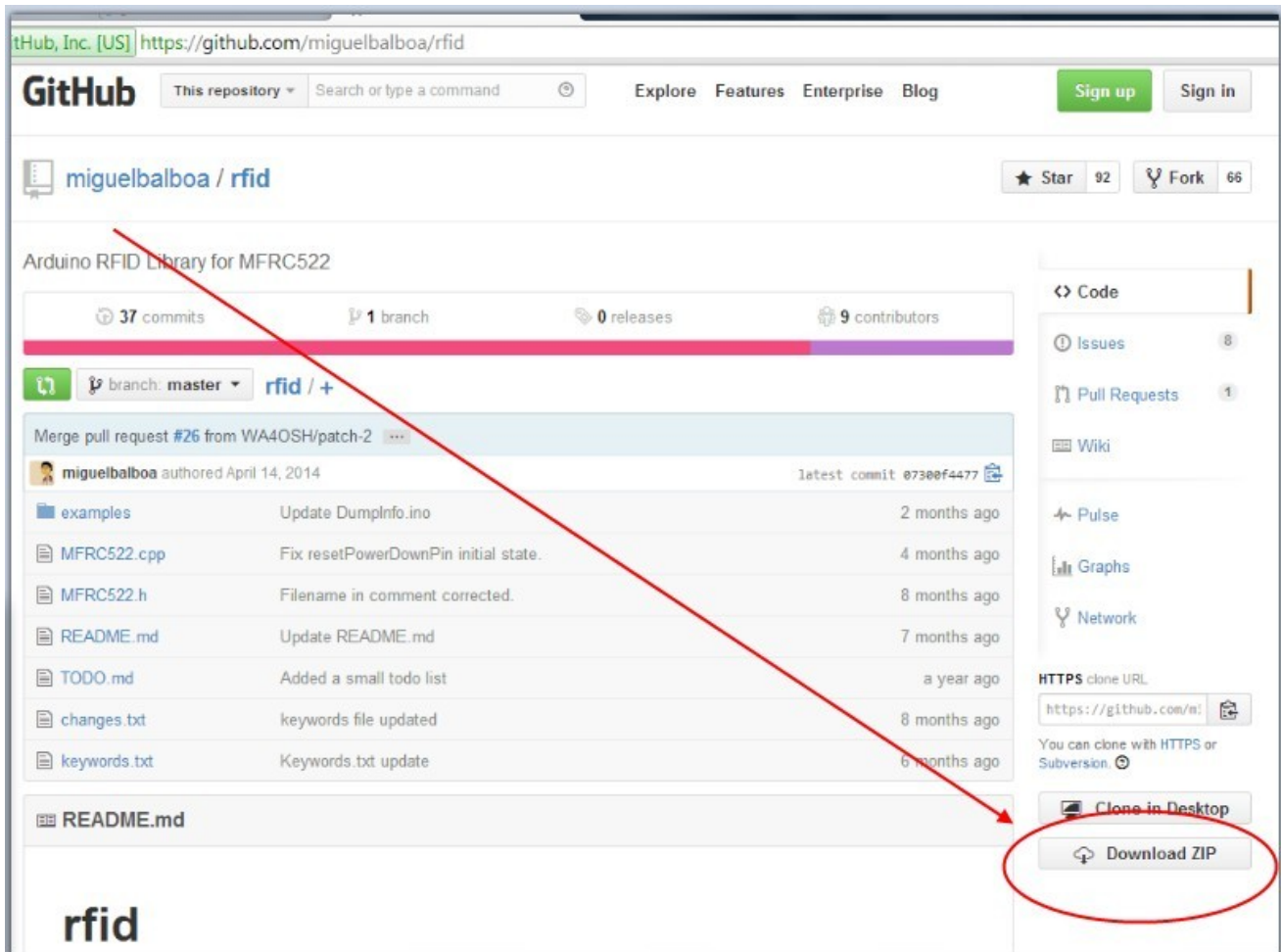
Board:	Arduino Uno	Arduino Mega	MFRC522-READER
Pin:	10	53	SDA
Pin:	13	52	SCK
Pin:	11	51	MOSI
Pin:	12	50	MISO
Pin:	unbelegt	unbelegt	IRQ
Pin:	GND	GND	GND
Pin:	9	5	RST
Pin:	3,3V	3,3V	3,3V

Foto vom Aufbau. In diesem Beispiel wurden an den RFID-Empfänger die um 90° gebogenen Kontaktstifte angelötet, damit der Empfänger senkrecht in ein Breadboard gesteckt werden kann:



Programmierung

Beim Auslesen und verarbeiten der Daten eines RFID-Empfängers wären wie auch bei anderen komplexen Aufgaben sehr viele Zeilen Quellcode erforderlich. Daher bedienen wir uns einer vorgefertigten Library aus dem Internet. Es gibt einige verschiedene im Netz zu finden. In dieser Anleitung haben wir uns für die Library von der Internetseite <https://github.com/miguelbalboa/rfid> entschieden. Damit man damit arbeiten kann, muss man sich die Library zunächst herunterladen und im Arduino Programmordner abspeichern. Auf der Internetseite klickt man dazu unten rechts auf „Download ZIP“ und speichert die gepackte Datei auf seiner Festplatte.



Danach entpackt man den Inhalt in die Arduino-Software auf der Festplatte und zwar in den Ordner „libraries“. Für gewöhnlich befindet sich der Ordner auf der Festplatte unter dem Pfad „C:Programmearduino\libraries...“ (Wenn man das Programm an einer anderen Stelle gespeichert hat, muss man dort den Ordner „libraries“ verwenden).

Sobald die Datei entpackt ist, entsteht im Ordner „libraries“ die Datei „rfid-master“. Jetzt MUSS zunächst der Bindestrich entfernt werden. Man benennt den Ordner also um in den Ordner „rfidmaster“. So, das wäre geschafft. Die Library kann in der Arduino-Software verwendet werden.

Vorbereitung – der erste Sketch mit dem RFID-READER

Zunächst werden wir die UID („Unique Identification Number“), also die individuelle Bezeichnung eines RFID-TAGs auslesen. Dazu verwenden wir das folgende Programm (Achtung, das Programm funktioniert nur, wenn die library wie oben beschrieben zur Arduino-Software hinzugefügt wurde). Das Programm ist für den UNO R3 Mikrocontroller vorgesehen. Bei MEGA2560 und anderen Controllern müssen die Pins entsprechend angepasst werden

```
#include <SPI.h> // SPI-Bibliothek hinzufügen

#include <MFRC522.h> // RFID-Bibliothek hinzufügen

#define SS_PIN 10 // SDA an Pin 10 (bei MEGA anders)

#define RST_PIN 9 // RST an Pin 9 (bei MEGA anders)

MFRC522 mfrc522(SS_PIN, RST_PIN); // RFID-Empfänger benennen

void setup() // Beginn des Setups:
{
  Serial.begin(9600); // Serielle Verbindung starten (Monitor)

  SPI.begin(); // SPI-Verbindung aufbauen

  mfrc522.PCD_Init(); // Initialisierung des RFID-Empfängers
}

void loop() // Hier beginnt der Loop-Teil
{
  if ( ! mfrc522.PICC_IsNewCardPresent() ) // Wenn eine Karte in Reichweite
  //ist...
  {
    return; // gehe weiter...
  }

  if ( ! mfrc522.PICC_ReadCardSerial() ) // Wenn ein RFID-Sender ausgewählt
  wurde
  {
    return; // gehe weiter...
  }

  Serial.print("Die ID des RFID-TAGS lautet:"); // "Die ID des RFID-TAGS
  //lautet:" wird auf den Serial Monitor geschrieben
  for (byte i = 0; i < mfrc522.uid.size; i++)
  {
    Serial.print(mfrc522.uid.uidByte[i], HEX); // Dann wird die UID ausgelesen,
    //die aus vier einzelnen Blöcken besteht und der Reihe nach an den Serial
    //Monitor gesendet. Die Endung Hex bedeutet, dass die vier Blöcke der UID
```

```
//als HEX-Zahl (also auch mit Buchstaben) ausgegeben wird

Serial.print(" "); // Der Befehl „Serial.print(" ");“ sorgt dafür, dass
//zwischen den einzelnen ausgelesenen Blöcken ein Leerzeichen steht.

}

Serial.println(); // Mit dieser Zeile wird auf dem Serial Monitor nur ein
//Zeilenumbruch gemacht.

}
```

Wenn alles funktioniert hat, sieht das Ergebnis am Serial-Monitor (Abgesehen von der eigenen UID) so aus:

```
MifareClassicValueBlock | Arduino 1.5.2
Datei Bearbeiten Sketch Werkzeuge Hilfe

MifareClassicValueBlock$
#include <SPI.h>
#include <MFRC522.h>

#define SS_PIN 10
#define RST_PIN 9
MFRC522 mfrc522(SS_PIN, RST_PIN);

void setup() {
  Serial.begin(9600);
  SPI.begin();
  mfrc522.PCD_Init();
}

void loop() {
  if ( ! mfrc522.PICC_IsNewCardPresent() )
  {
    return;
  }

  if ( ! mfrc522.PICC_ReadCardSerial() )
  {
    return;
  }

  Serial.print("Die ID des RFID-TAGS lautet:");
  for (byte i = 0; i < mfrc522.uid.size; i++)
  {
    Serial.print(mfrc522.uid.uidByte[i], HEX);
    Serial.print(" ");
  }
}

Hochladen abgeschlossen.
Binary sketch size: 5.780 bytes (of a 32.256 byte maximum) - 17% used
35 Arduino Uno on COM19
```

Mit dieser hintereinander geschriebenen HEX-Zahl lässt sich nicht gut arbeiten. Also ändern wir die Zeile „Serial.print(mfrc522.uid.uidByte[i], HEX);“ um in „Serial.print(mfrc522.uid.uidByte[i], DEC);“. Dann bekommt man als Ergebnis die einzelnen Blöcke des UID-Codes als Dezimalzahl ausgegeben:

RFID Sketch 2

Jetzt wird der UID-Code zwar als Dezimalzahl ausgegeben, aber er ist immernoch in vier Blöcke aufgeteilt. Wir verändern den Code jetzt mit ein wenig Mathematik dahingehend, dass wir für die UID eine einzige zusammenhängende normale Zahl erhalten (Dezimalzahl).

Warum machen wir das? Wenn wir später den Sketch verwenden wollen um etwas in Abhängigkeit eines richtig ausgelesenen RFID-TAGs auszulösen (Z.B. eine LED soll leuchten oder eine Servomotor soll sich um 90 Grad drehen...) können wir mit einer zusammenhängenden Zahl besser einen IF-Befehl verwenden. Zum Beispiel:

„Wenn der RFID-Code=1031720 ist, dann soll eine LED für 5 Sekunden leuchten“.

Schwerer wäre es dagegen mit dem Befehl „Wenn der erste Block 195 lautet und der zweite Block 765 lautet und der dritte Block 770 lautet und der vierte Block 233 lautet ... Dann schalte eine LED für 5 Sekunden an.“

Nachteil ist jedoch, dass der Sketch dadurch etwas unsicherer wird, weil nicht alle Zahlen der vier Blöcke (Max. 12 Ziffern) in einer Zusammenhängenden Zahl dargestellt werden können. Wenn es sicherer sein soll, müsste man jeden einzelnen Block als solchen abfragen.

```
#include <SPI.h>
#include <MFRC522.h>
#define SS_PIN 10
#define RST_PIN 9
MFRC522 mfrc522(SS_PIN, RST_PIN);

void setup()
{
  Serial.begin(9600);
  SPI.begin();
  mfrc522.PCD_Init();
}

void loop()
{
  if ( ! mfrc522.PICC_IsNewCardPresent())
  {
    return;
  }

  if ( ! mfrc522.PICC_ReadCardSerial())
  {
    return;
  }

  long code=0; // Als neue Variable fügen wir „code“ hinzu, unter welcher später
  die UID als zusammenhängende Zahl ausgegeben wird. Statt int benutzen wir jetzt
  den Zahlenbereich „long“, weil sich dann eine größere Zahl speichern lässt.

  for (byte i = 0; i < mfrc522.uid.size; i++)
  {
    code=((code+mfrc522.uid.uidByte[i])*10); // Nun werden wie auch vorher die
    vier Blöcke ausgelesen und in jedem Durchlauf wird der Code mit
    dem Faktor 10 „gestreckt“. (Eigentlich müsste man hier den Wert
    1000 verwenden, jedoch würde die Zahl dann zu groß werden.

  }
```

```
Serial.print("Die Kartenummer lautet:"); //Zum Schluss wird der Zahlencode  
(Man kann ihn nicht mehr als UID bezeichnen) ausgegeben.
```

```
Serial.println(code);  
}
```

Prima, jetzt können wir von einem RFID-TAG eine eindeutige Identifikationsnummer auslesen (Die Nummer wird am Serial Monitor angezeigt). In diesem Fall lautet die Identifikationsnummer dieses individuellen RFID-TAGs 1031720.

Und wie geht es weiter? Jetzt wollen wir eine LED für 5 Sekunden einschalten, falls der gewünschte RFID-TAG vor den RFID-READER gehalten wird.

Sketch 3

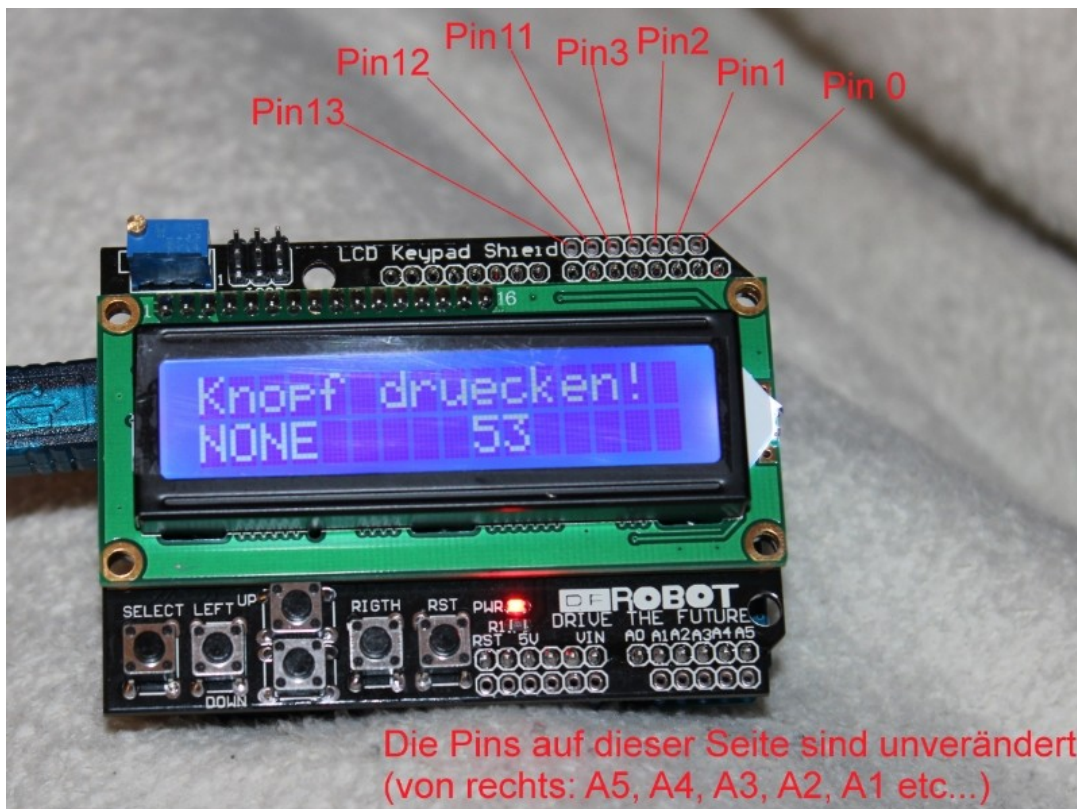
```
#include <SPI.h>  
  
#include <MFRC522.h>  
  
#define SS_PIN 10  
#define RST_PIN 9  
  
MFRC522 mfrc522(SS_PIN, RST_PIN);  
  
void setup()  
{  
  Serial.begin(9600);  
  SPI.begin();  
  mfrc522.PCD_Init();  
  
  pinMode(2, OUTPUT); // Der Pin 2 ist jetzt ein Ausgang (Hier wird eine LED  
  angeschlossen)  
}  
  
void loop()  
{  
  if ( ! mfrc522.PICC_IsNewCardPresent())  
  {  
    return;  
  }  
  
  if ( ! mfrc522.PICC_ReadCardSerial())  
  {  
    return;  
  }  
  
  long code=0;  
  
  for (byte i = 0; i < mfrc522.uid.size; i++)  
  {  
    code=((code+mfrc522.uid.uidByte[i])*10);  
  }  
}
```

```
}  
Serial.print("Die Kartennummer lautet:");  
Serial.println(code);  
// Ab hier erfolgt die erweiterung des Programms.  
if (code==1031720) //Wenn der Zahlencode 1031720 lautet...  
{ //Programmabschnitt öffnen  
digitalWrite (2, HIGH); //...dann soll die LED an Pin 2 leuchten...  
delay (5000); //für 5 Sekunden  
digitalWrite (2, LOW); // ... und danach wieder aus gehen  
} // Programmabschnitt schließen  
} //Sketch abschließen
```

Zubehör für Arduino

Keypad shield

Ein Keypadshield hat den Vorteil, dass man das LCD Display nicht in komplizierter Weise verkabeln muss und dass man zusätzlich sechs Taster hat, die man verwenden kann. Das besondere liegt bei den Tasten darin, dass sie im Programmcode alle über einen analogen Pin ausgelesen werden können. Denn die Taster sind über unterschiedliche Widerstände alle mit einem analogen Pin (A0) verbunden. Der Pin A0 kann daher nur eingeschränkt für andere Dinge verwendet werden. Aus dem Grund befindet sich auf dem Shield auch kein Steckplatz für den A0-Pin.



Das Keypadshield kann u.a. auf das UNO das MEGA Board aufgesteckt werden. Es wird so platziert, dass die Pins für die Spannungsversorgung genau in die Pins der Spannungsversorgung auf dem Arduino Board passt (auf dem Bild unten in der Mitte sieht man die Pins für die Spannungsversorgung. Ein Anhaltspunkt kann der Pin mit der Aufschrift „VIN“ sein). Die oberen Steckplätze des Arduinoboards werden ebenfalls durch das Shield verdeckt (Pin 0-13). Einige können eh nicht mehr verwendet werden, da das LCD Display sie verwendet. Die nicht mehr benötigten Pins wurden in einer Reihe von Steckplätzen zusammengefasst (Siehe Foto oben rechts).

Wenn man diese Steckplätze verwenden möchte, sollte man dort Kabelbuchsen auflöten.

Als Beispielcode kann der folgende verwendet werden:

Code:

```
//Sample using LiquidCrystal library

#include <LiquidCrystal.h>

/*****
This program will test the LCD panel and the buttons
Mark Bramwell, July 2010
*****/

// select the pins used on the LCD panel
LiquidCrystal lcd(8, 9, 4, 5, 6, 7);

// define some values used by the panel and buttons
int lcd_key = 0;
int adc_key_in = 0;
#define btnRIGHT 0
#define btnUP 1
#define btnDOWN 2
#define btnLEFT 3
#define btnSELECT 4
#define btnNONE 5

// read the buttons
int read_LCD_buttons()
{
  adc_key_in = analogRead(0); // read the value from the sensor
  // my buttons when read are centered at these valies: 0, 144, 329, 504, 741
  // we add approx 50 to those values and check to see if we are close
  if (adc_key_in > 1000) return btnNONE; // We make this the 1st option for speed
  reasons since it will be the most likely result

  if (adc_key_in < 50) return btnRIGHT;
  if (adc_key_in < 195) return btnUP;
  if (adc_key_in < 380) return btnDOWN;
  if (adc_key_in < 555) return btnLEFT;
  if (adc_key_in < 790) return btnSELECT;

  return btnNONE; // when all others fail, return this...
}

void setup()
{
  lcd.begin(16, 2); // start the library
  lcd.setCursor(0,0);
  lcd.print("Message"); // print a simple message
  pinMode (2, OUTPUT);
}

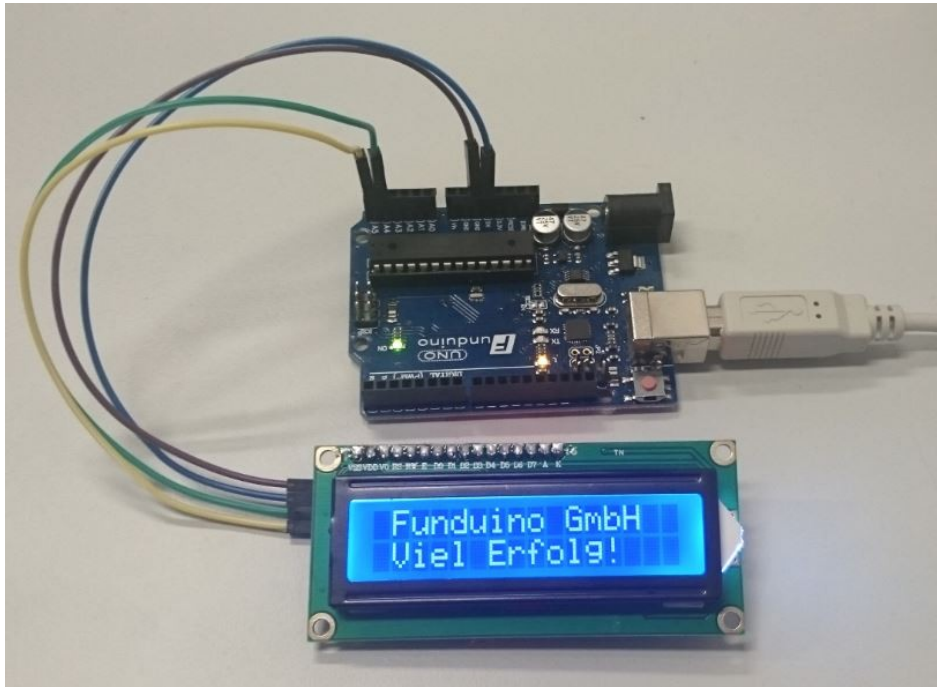
void loop()
{
  digitalWrite (2, HIGH);
  lcd.setCursor(9,1); // move cursor to second line "1" and 9 spaces over
  lcd.print(millis()/1000); // display seconds elapsed since power-up

  lcd.setCursor(0,1); // move to the begining of the second line
  lcd_key = read_LCD_buttons(); // read the buttons
```

```
switch (lcd_key) // depending on which button was pushed, we perform an action
{
case btnRIGHT:
{
lcd.print("RIGHT ");
digitalWrite (2, LOW);
break;
}
case btnLEFT:
{
lcd.print("LEFT ");
break;
}
case btnUP:
{
lcd.print("UP ");
break;
}
case btnDOWN:
{
lcd.print("DOWN ");
break;
}
case btnSELECT:
{
lcd.print("SELECT");
break;
}
case btnNONE:
{
lcd.print("NONE ");
break;
}
}
}
}
```

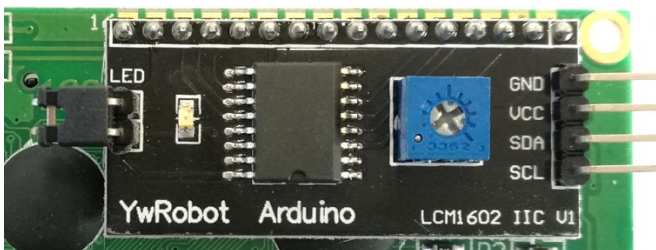
Anleitung zum LCD Display mit I2C Anschluss

Das LCD Modul mit angelötetem I2C Bus ermöglicht die Verwendung eines LCD Moduls mit einer einfachen Verkabelung. Dies ist bei komplexeren Projekten besonders vorteilhaft. Ein weiterer Unterschied zum normalen LCD Display besteht darin, dass sich auf der Rückseite des Displays ein Drehregler befindet, mit dem die Leuchtstärke des LCD reguliert werden kann.



Hinweis:

Diese Anleitung funktioniert nur mit einem I²C Modul auf der Rückseite des Displays ohne drei Lötstellen mit der Bezeichnung A0, A1 und A2. Das Modul muss folgendermaßen aussehen:

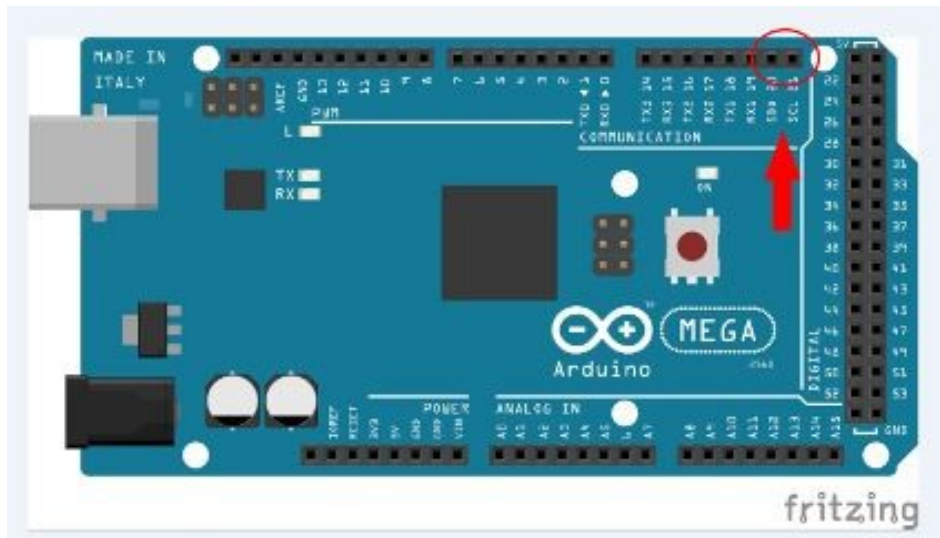


Bei LCDs mit den Lötstellen auf dem I²C Modul siehe Anleitung „Zwei I²C LCD Module gleichzeitig ansteuern“.

Material: Mikrocontroller (in diesem Beispiel UNO R3), LCD mit I2C Modul, Kabel

Verkabelung: Die Verkabelung ist sehr simpel. Am I2C LCD Modul sind nur vier Kontakte vorhanden. GND wird mit dem GND Kontakt am Mikrocontroller verbunden. VCC mit dem 5V Kontakt am Microcontroller, SDA mit dem Analogen Eingang A4 und SCL mit dem Analogen Eingang A5.

Achtung!: Bei dem MEGA2560 R3 Microcontroller gibt es für die SDA – und SCL- Kontakte eigene Eingänge auf dem Board unter 20 und 21.



Programmieren:

Um Mit dem I2C LCD Modul zu arbeiten benötigt man eine Library welche noch nicht im Arduino Programm vorinstalliert ist. Diese kann man zum Beispiel unter <https://github.com/fdebrabander/Arduino-LiquidCrystal-I2C-library> als .ZIP Datei herunterladen. Danach muss die Library im Arduino Programm hinzugefügt werden.

Das kann unter dem Punkt „Sketch“ dann „Include Library“ und „add .ZIP Library ..“ gemacht werden. Jetzt kann im Code auf die Library zurückgegriffen werden.

Sketch:

```
#include <Wire.h> // Wire Bibliothek hochladen

#include <LiquidCrystal_I2C.h> // Vorher hinzugefügte LiquidCrystal_I2C
//Bibliothek hochladen

LiquidCrystal_I2C lcd(0x27, 16, 2); //Hier wird festgelegt um was für
//einen Display es sich handelt. In diesem Fall einer mit 16 Zeichen in 2
//Zeilen.

void setup()
{
  lcd.begin(); //Im Setup wird der LCD gestartet (anders als beim einfachen
LCD Modul ohne 16,2 in den Klammern denn das wurde vorher festgelegt
}

void loop()
{
  lcd.setCursor(0,0); //Ab hier kann das I2C LCD Modul genau wie das einfache
LCD Modul programmiert werden.

  lcd.print("Funduino GmbH:");
  lcd.setCursor(0,1); // lcd.setCursor um Zeichen und Zeile anzugeben

  lcd.print("Viel Erfolg!"); //lcd.print um etwas auf dem Display anzeigen zu
//lassen.
}
```

Anwendungsbeispiel:

Mit dem I2C LCD Modul können wie mit dem einfachen LCD Modul, auch Messwerte angezeigt werden.

Hier ein Beispielcode, bei dem ein Feuchtigkeitssensor an Pin A0 angeschlossen wurde :

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

int messwert=0;

void setup()
{
  lcd.begin();
}

void loop()
{
  messwert=analogRead(A0); //Der Messwert vom Analogen Eingang A0 soll
  //ausgelesen, und unter der Variablen „messwert“ gespeichert werden.

  lcd.setCursor(0,0); //In der ersten Zeile soll der Text „Messwert:“
  //angezeigt werden.

  lcd.print("Messwert:");

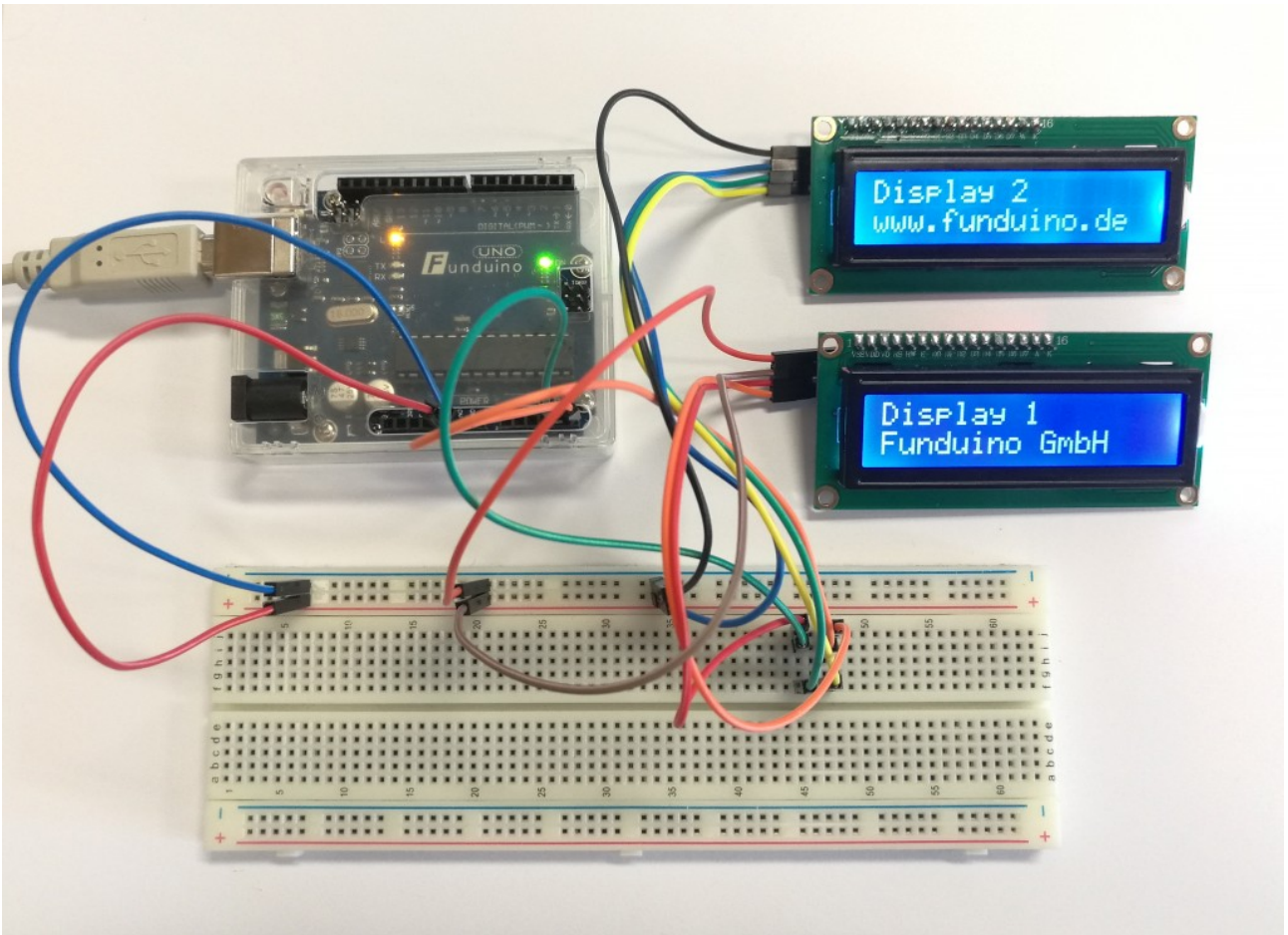
  lcd.setCursor(0,1); // In der zweiten Zeile soll der Messwert, der vom
  //Feuchtigkeitssensor bestimmt wurde, angezeigt werden.

  lcd.print(messwert);

  delay(500);
}
```

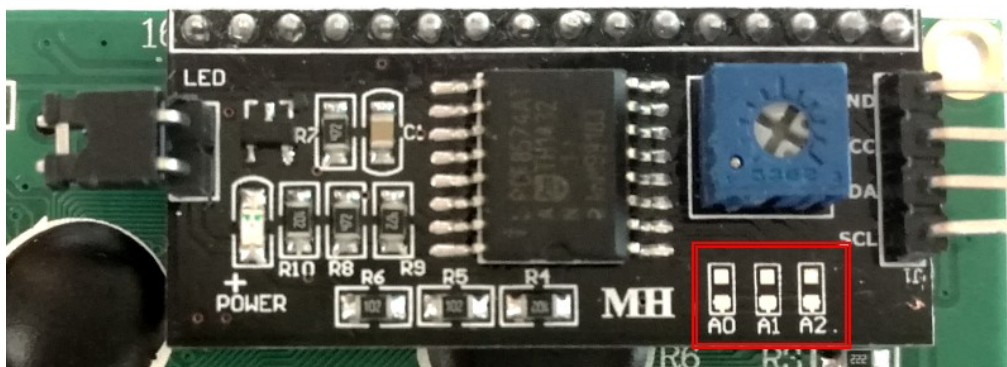
Zwei I²C Displays

Zwei I²C Displays am Arduino gleichzeitig verwenden.



Achtung:

Dieser Aufbau und die damit verbundene Änderung der I²C Adresse ist nur bei Displays möglich, die über eine Jumper-funktion verfügen. Die erkennt man auf dem folgenden Bild an dem rot markierten Bereich. Auf den Stellen A0,A1 und A2 kann eine Kontaktbrücke aufgelötet werden.



Außerdem wird für diese Anleitungen die NewliquidCrystal_1.3.4 Library benötigt, welche hier heruntergeladen werden kann: <https://bitbucket.org/fmalpartida/new-liquidcrystal/downloads>

Die Library muss in der Arduino Software hinzugefügt werden (siehe vorherige Anleitungen z.B. I²C Display).

Material: Arduino / 2 Displays mit I²C Modul / Breadboard / Kabel

Aufgabe: Zwei LCDs mit I²C Modul sollen gleichzeitig von einem Arduino Mikrocontroller angesteuert werden und zwei verschiedene Texte anzeigen. Dazu sollen die I²C Adressen der Displays vorher herausgefunden werden und die Adresse eines der Displays geändert werden.

Als erstes eine kurze Erläuterung zu der I²C Adresse:

Jedes I²C Modul hat eine sogenannte „HEX Adresse“. Über diese Adresse reagiert das I²C-Modul auf die Daten, die vom Arduino auf dem Datenbus an genau diese Adresse gesendet werden. Viele I²C-LCDs auch die gleiche HEX-Adresse. Das bedeutet, dass beim verwenden von zwei Displays beide Displays auf die gesendeten Daten vom Arduino-Board reagieren würden. Man könnte also auf zwei Displays keine unterschiedlichen Daten darstellen.

Die HEX-Adresse kann bei dem Display mit Hilfe der A0, A1 und A2 Lötstellen jedoch verändert werden. Im unveränderten Zustand sind alle drei Lötstellen nicht verbunden. Je nach Kombination, welche der Stellen man mit einer Lötstelle überbrückt, sind also 8 verschiedene Adressen möglich. Abhängig vom Display Typ kann diese Adresse anfangs 0x27 oder 0x3F sein (kann mit dem Adressen „Scanner“ herausgefunden werden, dazu später mehr).

Tabellen zu HEX Adressen je nach verlöteten Stellen(**I** = verbunden, **:** = nicht verbunden):

A0	A1	A2	HEX Adresse	HEX Adresse
:	:	:	0x27	0x3F
I	:	:	0x26	0x3E
:	I	:	0x25	0x3D
I	I	:	0x24	0x3C
:	:	I	0x23	0x3B
I	:	I	0x22	0x3A
:	I	I	0x21	0x39
I	I	I	0x20	0x38

Kommen wir zum I²C Adressen „Scanner“:

Der „Scanner“ ist im Prinzip nur ein Code der auf den Arduino hochgeladen wird an dem das LCD Modul angeschlossen ist und dann am seriellen Monitor die HEX Adresse anzeigt.

Schrittweise Anleitung zum I²C Adressen „Scanner“:

1. Entsprechendes LCD Modul mit dem Arduino verbinden:

I²C LCD Modul >> Arduino

VCC >> 5V

GND >> GND

SDA >> A4

SCL >> A5

2. Folgenden Sketch auf den Mikrocontroller laden:

```

// I2C Scanner
// Written by Nick Gammon
// Date: 20th April 2011

#include <Wire.h>

void setup() {

Serial.begin (115200); // Leonardo: wait for serial port to connect while (!
//Serial)
{
}

Serial.println ();

Serial.println („I2C scanner. Scanning ...");

byte count = 0;

Wire.begin();

for (byte i = 8; i < 120; i++)
{
Wire.beginTransmission (i);
if (Wire.endTransmission () == 0)
{
Serial.print („Found address: „);
Serial.print (i, DEC);
Serial.print (" (0x");
Serial.print (i, HEX);
Serial.println („)");
count++;
delay (1); // maybe unneeded?
} // end of good response
} // end of for loop

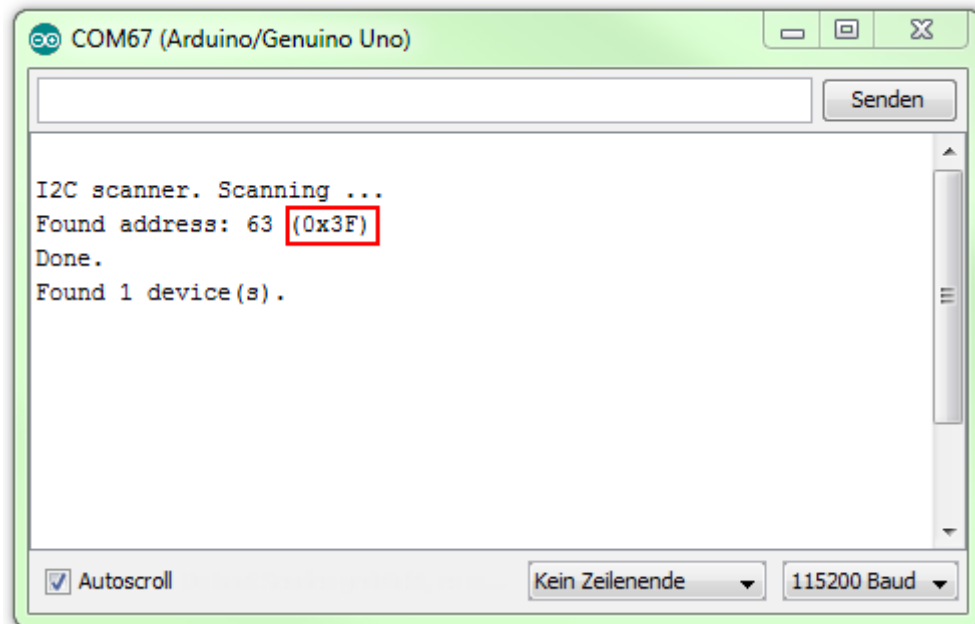
Serial.println („Done.");
Serial.print („Found „);
Serial.print (count, DEC);
Serial.println (" device(s).");
}

```

```
void loop() {}
```

3. Seriellen Monitor in der Arduino Software öffnen
4. Baudrate auf 115200 ändern
5. Ggf. Reset Knopf auf dem Arduino drücken

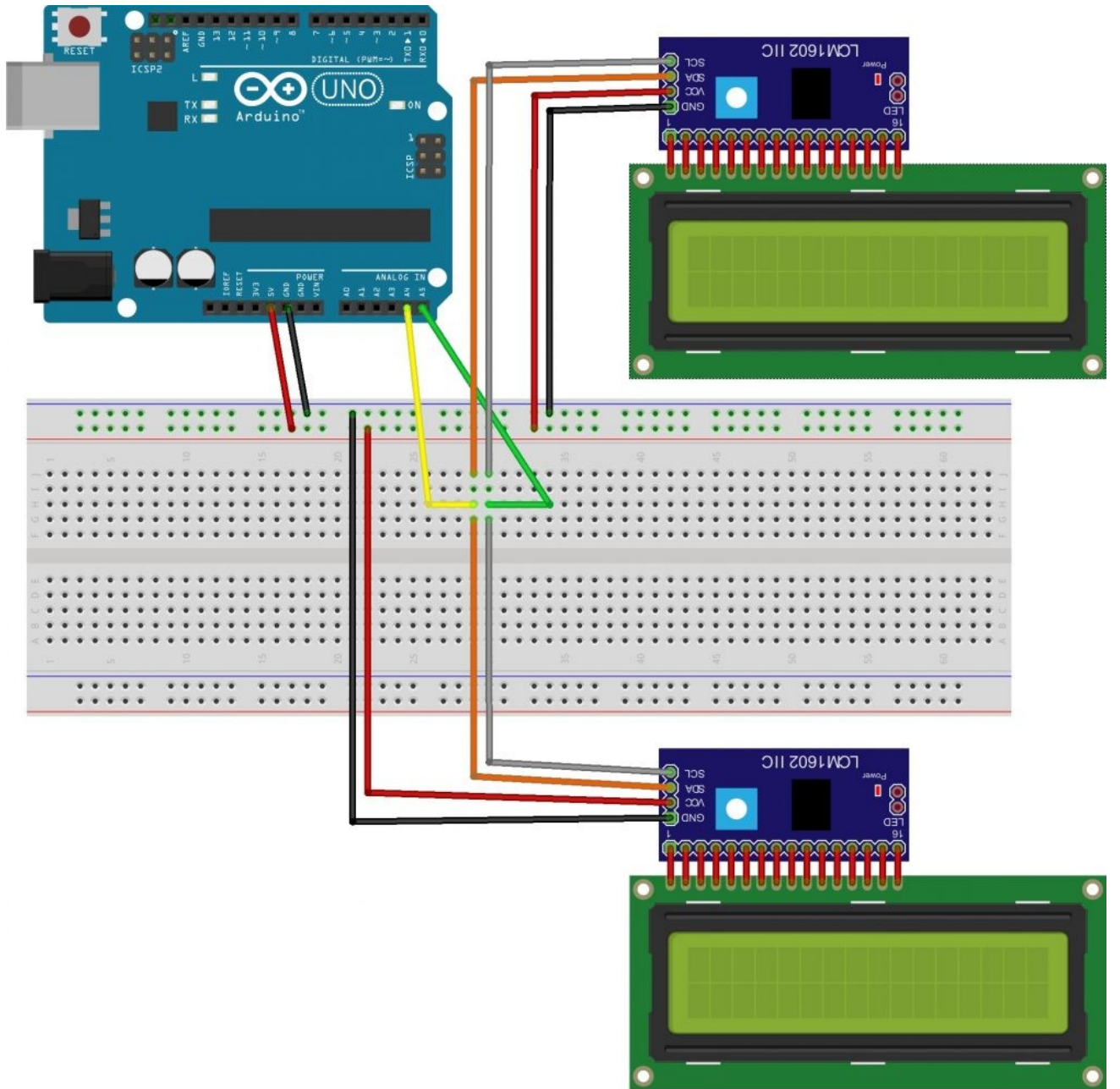
In dem seriellen Monitor sollte folgendes zu sehen sein (HEX Adresse ist rot markiert):



Wieder zur eigentlichen Anleitung:

Um nun auf zwei I²C Displays gleichzeitig zwei verschiedene Texte anzeigen lassen zu können müssen die Displays natürlich auch verschiedene HEX Adressen haben. Also verlöten wir in unserem Beispiel bei einem der Displays den A1 Kontakt, sodass dieser nun die Adresse 0x3D hat (kann mit dem Adressen Scanner nochmal geprüft werden). Wir nennen diesen Display ab jetzt Display 2 und den anderen Display 1.

Aufbau: Die Displays müssen nun mit dem Arduino verbunden werden:



fritzing

Code:

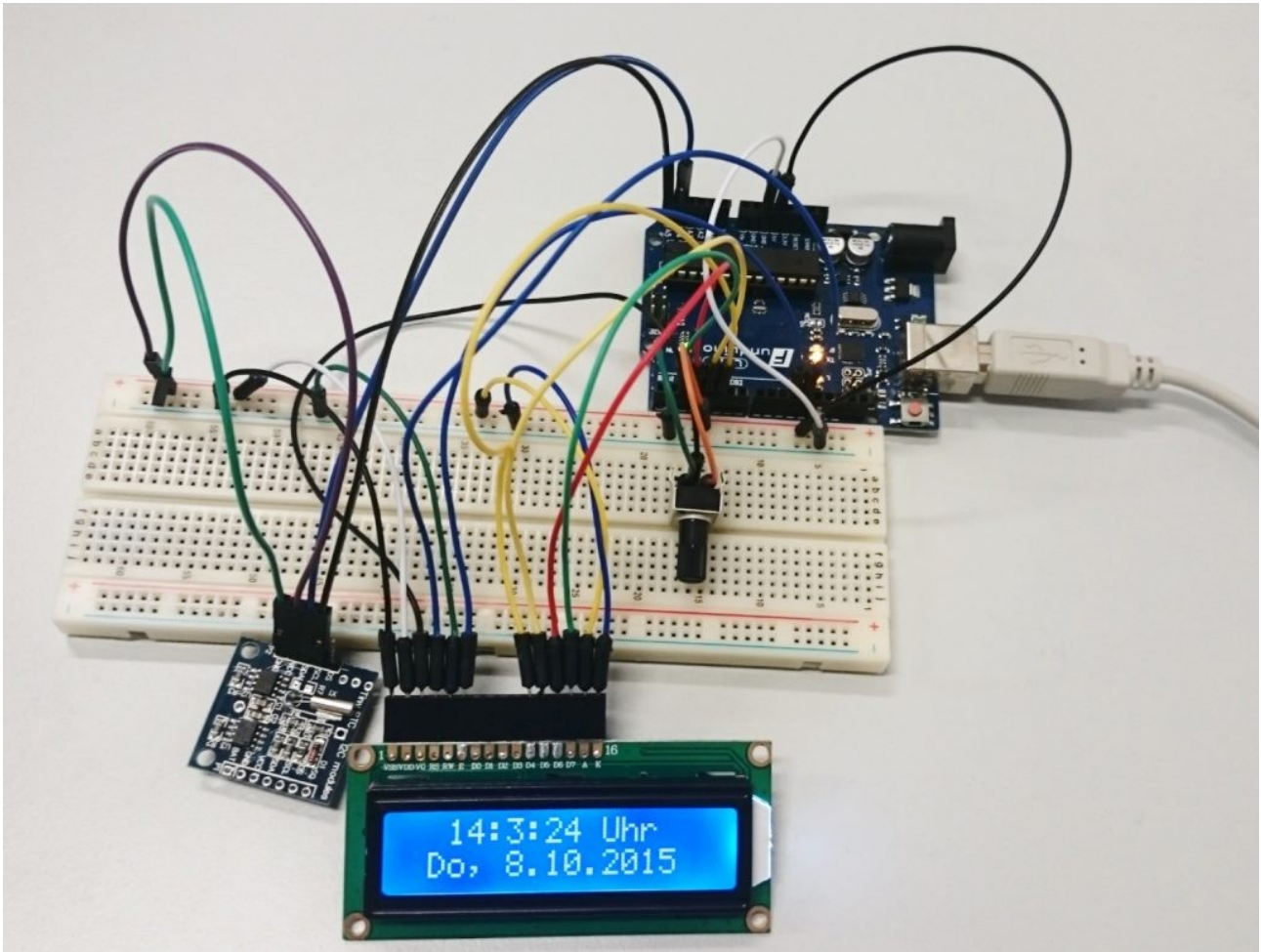
```
#include <Wire.h> //Wire.h Bibliothek einbinden
#include <LiquidCrystal_I2C.h> //LiquidCrystal_I2C.h Bibliothek einbinden
LiquidCrystal_I2C lcd1(0x3F, 2, 1, 0, 4, 5, 6, 7, 3, POSITIVE); //Hier wird

//Display 1 als lcd1 benannt, die Adresse angegeben „0x3F“ und die
//Pinnbelegung durch das I2C Modul angegeben
LiquidCrystal_I2C lcd2(0x3D, 2, 1, 0, 4, 5, 6, 7, 3, POSITIVE); //Hier wird
//Display 2 als lcd2 benannt, die Adresse angegeben „0x3D“ und die
//Pinnbelegung durch das I2C Modul angegeben
void setup()
{
  lcd1.begin(16,2); //Display 1 wird gestartet und die Größe des Displays
//wird angegeben(16 Zeichen, zwei Zeilen)
  lcd1.backlight(); //Beleuchtung Display 1 einschalten
  lcd2.begin(16,2); //Display 2 wird gestartet und die Größe des Displays
//wird angegeben(16 Zeichen, zwei Zeilen)
  lcd2.backlight(); //Beleuchtung Display 2 einschalten
}
void loop()
{
  lcd1.setCursor(0,0); //Text soll beim ersten Zeichen in der ersten Reihe
//bei Display 1 beginnen..
  lcd2.setCursor(0,0); //bei Display 2 auch..
  lcd1.print(„Display 1“); //Display 1 soll oben „Display 1“ anzeigen
  lcd2.print(„Display 2“); //Display 2 soll oben „Display 2“ anzeigen
  delay(1000); // Eine Sekunde warten
  lcd1.setCursor(0,1); //Text soll beim ersten Zeichen in der zweiten Reihe
//bei Display 1 beginnen..
  lcd2.setCursor(0,1); //bei Display 2 auch..
  lcd1.print(„Funduino GmbH“); //Display 1 soll unten „Funduino GmbH“
//anzeigen
```

```
lcd2.print(„www.funduino.de“); //Display 2 „www.funduino.de“  
}
```

Arduino-RTC

Uhrzeitmodul für Arduino: RTC – Real Time Clock



Mit der Arduino kompatiblen DS1307 I2C Real Time Clock kann man Uhrzeit und Datum nahezu in Echtzeit anzeigen lassen.

Material: Mikrocontrollerboard (In diesem Beispiel UNO R3), ein Drehregler (bzw. Potentiometer), Breadboard, LCD Display, DS1307 I2C Real Time Clock, Jumperkabel

Verkabelung:

Das LCD Display wird im Prinzip wie in der Anleitung Nr. 13 verkabelt. Dazu kommt dann noch die RTC, welche einmal mit 5V und GND verbunden wird, der SDA-Kontakt an den Analogen Eingang A4 und der SCL-Kontakt an den Analogen Eingang A5.

Wenn alles richtig verkabelt ist kann auch schon fast mit dem Programmieren begonnen werden. Vorher müssen, für die Programmierung notwendige, Libraries heruntergeladen und in der Arduino Software hinzugefügt werden.

Bei den benötigten Libraries handelt es sich einmal um die Time Library, welche z.B. Unter folgendem Link :http://www.pjrc.com/teensy/td_libs_Time.html als .ZIP Datei

heruntergeladen werden kann und um die DS1307RTC Library, welche zB.

Unter http://www.pjrc.com/teensy/td_libs_DS1307RTC.html als .ZIP Datei heruntergeladen werden kann.

Wenn man beide Libraries runtergeladen hat muss man diese noch in der Arduino Software zu den bereits vorhandenen Libraries hinzufügen. Dazu öffnet man die Arduino Software und wählt unter dem Punkt „Sketch“ das Feld „Include Libraby“ aus. Danach kann man „Add .ZIP Library“ auswählen und die vorher heruntergeladenen Time und DS1307RTC Library hinzufügen.

Auf diese kann jetzt in einem Code zurückgegriffen werden.

Als nächstes muss man den sogenannten „Set-Time“ Sketch hochladen um die Zeit und das Datum vom Gerät (Computer, Laptop etc.) an dem der Microcontroller angeschlossen ist, auf der Real-Time-Clock einzustellen.

Dies macht man, indem man unter „Datei“ auf „Beispiele“ dann „DS1307RTC“ den Punkt „Set Time“ auswählt. Es öffnet sich der entsprechende Sketch, welchen man nun auf den UNO R3 hochlädt.

Wenn man das alles erledigt hat kann man zum eigentlichen Code kommen.

Code:

```
#include <Time.h> Bibliotheken laden
#include <Wire.h>
#include <DS1307RTC.h>
#include <LiquidCrystal.h>
LiquidCrystal lcd(12, 11, 5, 4, 3, 2); //In dieser Zeile wird
festgelegt, welche Pins des Mikrocontrollerboards für das LCD
verwendet wird

void setup()
{
  lcd.begin(16, 2); //Im Setup wird angegeben, wie viele Zeichen und
Zeilen verwendet werden.
  Serial.begin(9600); //Öffnet den seriellen Port und legt die Baud
Rate (9600) für die serielle Übertragung fest.
  setSyncProvider(RTC.get); //Dies ist die Funktion um die Zeit- und
Datumsangabe von der RTC zu bekommen
}

void loop()
{
  Serial.print(hour()); //Serial.print ist der Befehl etwas im
seriellen Monitor anzuzeigen (Stunde, Minute, Sekunde,
Leerzeichen, Tag, Leerzeichen, usw.)
  printDigits(minute()); //bei den Minuten und Sekunden wird der
Befehl
  printDigits(second()); //printDigits angegeben welcher am Ende des
Codes noch festgelegt wird.
  Serial.print(" ");
  Serial.print(day());
  Serial.print(" ");
  Serial.print(month());
  Serial.print(" ");
  Serial.print(year());
  Serial.println();
}
```

```

delay(1000); //warte eine Sekunde
lcd.setCursor(2, 0); //setCursor gibt an wo der Text beginnen
soll. In diesem Fall beim dritten Zeichen in der ersten Reihe.
lcd.print(hour()); //Hier soll jetzt die Uhrzeit angezeigt werden,
also „hour“ „:“ „minute“ usw..
lcd.print(":");
lcd.print (minute());
lcd.print(":");
lcd.print(second());
lcd.print(" ");
lcd.print("Uhr"); //Hier soll nach der Uhrzeit noch das Wort „Uhr“
angezeigt werden, dazu müssen noch 3 Leerzeichen folgen, sonst
würde bei Zahlen <10 immer ein weiteres „r“ hinter Uhr angezeigt
werden. Das liegt daran, dass in der LCD Library der Befehl eine 0
vor Zahlen <10 anzuzeigen nicht vorhanden ist.
lcd.print(" ");
lcd.print(" ");
lcd.print(" ");
lcd.setCursor(1, 1); // Der nächste „Text“ soll nun beim zweiten
Zeichen in der zweiten Reihe beginnen.
lcd.print(day()); // Das Datum soll nun als
lcd.print("."); // „Tag“, „.“ „Monat“ usw. angegeben werden.
lcd.print(month());
lcd.print(".");
lcd.print(year());
}

void printDigits(int digits) //In diesem Abschnitt wird festgelegt,
dass bei Zahlen <10 im seriellen Monitor automatisch eine 0 vor
den Ziffern angezeigt wird. Das gilt nur für den seriellen Monitor
und nicht für LCD Display.
{
Serial.print(":");
if(digits < 10)
Serial.print('0');
Serial.print(digits);
}

```

Hinweis: Die DS1307 Real Time Clock ist nicht auf die Sekunde genau. Das liegt daran, dass zwischen Hochladen des Set Time Sketch und dem Hochladen des endgültigen Sketch eine gewisse Zeit von ca. 15 Sekunden, die die RTC nicht ausgleichen kann.

Zusätzlich kann man vor dem Datum noch den Wochentag anzeigen lassen.

Hierzu muss vor dem Datum im lcd.print Abschnitt der Befehl **printDay()**; gegeben werden und nach dem Loop folgender Code eingefügt werden:

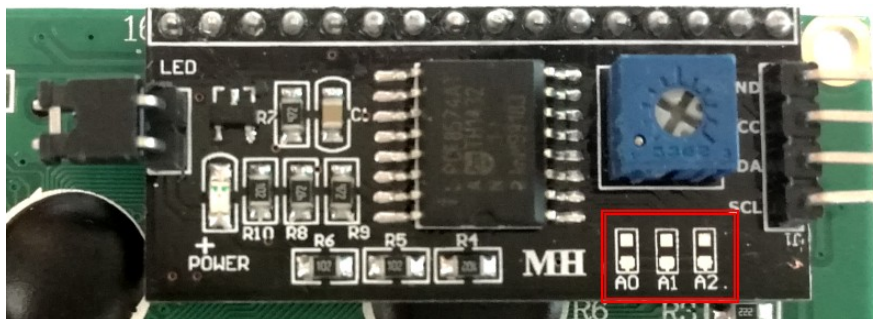
```
void printDay() // Hier wird für den vorher im Code schon  
verwendeten Befehl printDay eine Bedeutung festgelegt
```

```
{  
int day;  
day = weekday(); // Die Wochentage sollen abhängig vom Datum  
angezeigt werden.  
if(day == 1){lcd.print("So, ");} // Wenn es sich um Tag 1 handelt  
soll „So“ usw. angezeigt werden.  
if(day == 2){lcd.print("Mo, ");}  
if(day == 3){lcd.print("Di, ");}  
if(day == 4){lcd.print("Mi, ");}  
if(day == 5){lcd.print("Do, ");}  
if(day == 6){lcd.print("Fr, ");}  
if(day == 7){lcd.print("Sa, ");}  
}
```

Erweiterung:

Zwei I²C Bauteile gleichzeitig ansteuern: Uhrzeit und Datum mit einer Real Time Clock mit I²C Bus, auf einem I²C LCD anzeigen lassen.

Um zwei I²C Komponenten gleichzeitig ansteuern zu können, muss eins der Bauteile eine andere I²C Adresse als das andere. In unserem Fall können wir nur die Adresse des LCDs ändern. Sofern es die drei Lötstellen A0, A1 und A2 auf der Rückseite des I²C Moduls hat (s. Bild).



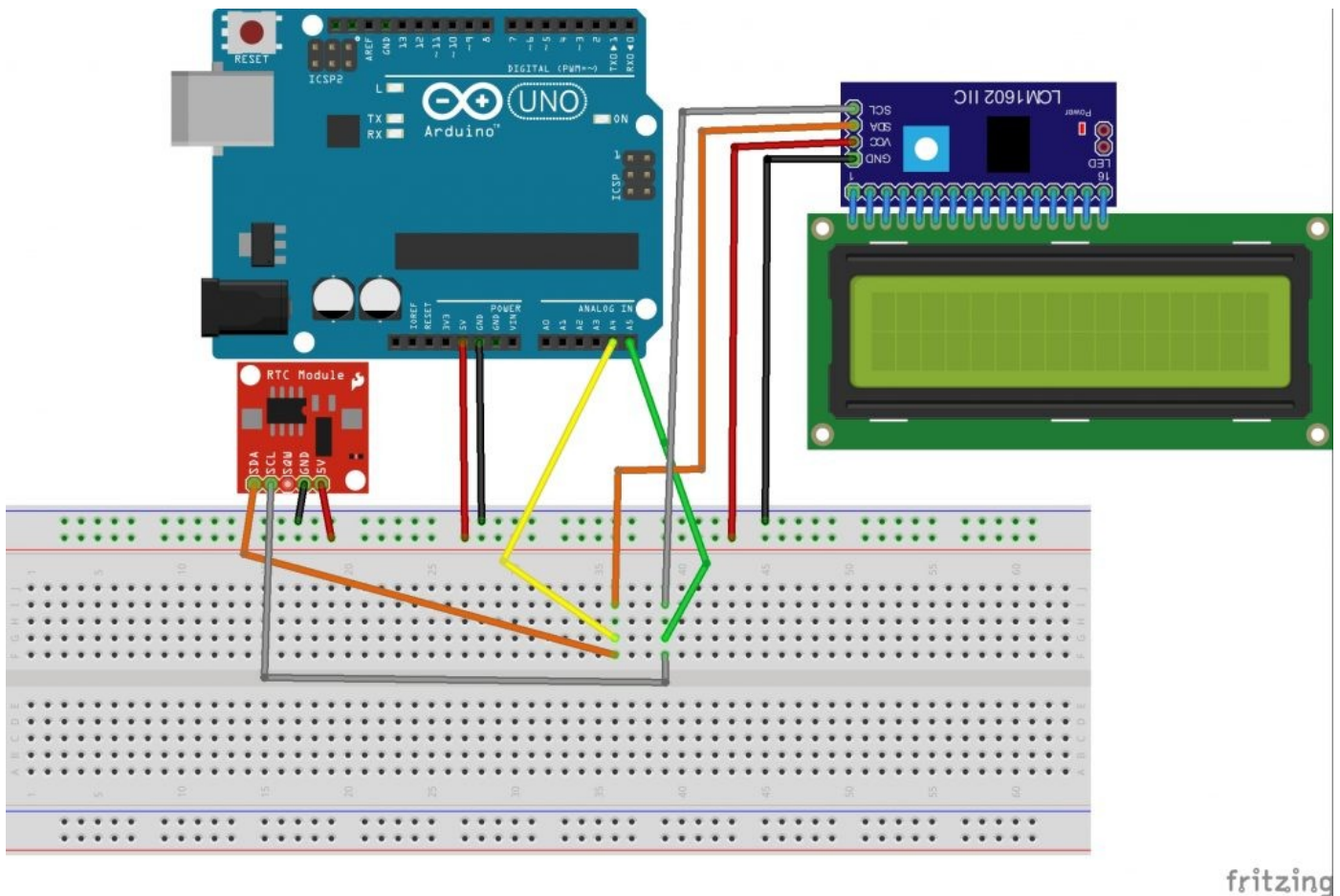
Die I²C Adresse der Real Time Clock ist festgelegt und kann nicht geändert werden.

Um die Adresse des LCDs zu ändern, müssen die Lötstellen anders verbunden sein als zu Beginn (alle drei Lötstellen getrennt). So ändert sich die HEX Adresse des LCDs (s. Tabelle) (**I** = verbunden, **:** = nicht verbunden):

A0	A1	A2	HEX Adresse	HEX Adresse
:	:	:	0x27	0x3F
I	:	:	0x26	0x3E
:	I	:	0x25	0x3D
I	I	:	0x24	0x3C
:	:	I	0x23	0x3B
I	:	I	0x22	0x3A
:	I	I	0x21	0x39
I	I	I	0x20	0x38

Kann ggf. mit dem „Scanner“ aus der Anleitung „Zwei Displays mit I²C Modul gleichzeitig ansteuern“ geprüft werden.

Wenn die Adresse des LCDs verändert wurde, können wir nun zur Verkabelung kommen:



Zum Programmieren werden wie oben schon erwähnt, die Time und die DS1307RTC Library benötigt. Außerdem wird für das I²C LCD die NewliquidCrystal_1.3.4 Library benötigt, welche hier heruntergeladen werden kann: <https://bitbucket.org/fmalpartida/new-liquidcrystal/downloads>

Alle Bibliotheken müssen der Arduino Software hinzugefügt werden:

Sketch > Include Library > Add ZIP library > vorher heruntergeladene Datei auswählen

Als nächstes muss man den sogenannten „Set-Time“ Sketch hochladen um die Zeit und das Datum vom Gerät (Computer, Laptop etc.) an dem der Microcontroller angeschlossen ist, auf der Real-Time-Clock einzustellen.

Dies macht man, indem man unter „Datei“ auf „Beispiele“ dann „DS1307RTC“ den Punkt „Set Time“ auswählt. Es öffnet sich der entsprechende Sketch, welchen man nun auf den UNO R3 hochlädt.

Wenn man das alles erledigt hat kann man zum eigentlichen Code kommen.

```
#include <Time.h>

#include <Wire.h>
#include <DS1307RTC.h>
#include <LiquidCrystal_I2C.h> //Bibliotheken laden
LiquidCrystal_I2C lcd(0x3D, 2, 1, 0, 4, 5, 6, 7, 3, POSITIVE); //Das I2C Display
bennen und die HEX Adresse //eingeben (bei uns 0x3D)
void setup() {
  lcd.begin(16, 2); //Das Display starten, festlegen dass es sich um ein Display
mit 16 Zeichen in 2 Zeilen //handelt
  lcd.backlight(); //Beleuchtung des Displays einschalten
  Serial.begin(9600); //Seriellen Verbindung mit Baudrate 9600 starten
  setSyncProvider(RTC.get); //Daten von der RTC abrufen
}
void loop() {
  Serial.print(hour()); //Serial.print ist der Befehl etwas im seriellen Monitor
anzuzeigen (Stunde, Minute, //Sekunde,Leerzeichen, Tag, Leerzeichen, usw.)
  printDigits(minute()); //bei den Minuten und Sekunden wird der Befehl
  printDigits(second()); //printDigits verwendet, welcher am Ende des Codes noch
festgelegt wird
  Serial.print(" ");
  Serial.print(day());
  Serial.print(" ");
  Serial.print(month());
  Serial.print(" ");
  Serial.print(year());
  Serial.println();
}
```

```

delay(1000); //eine Sekunde warten
lcd.setCursor(2, 0); // setCursor gibt an wo der Text beginnen soll. In diesem
Fall beim dritten Zeichen in //der ersten Reihe.
lcd.print(hour()); //Die Uhrzeit soll angezeigt werden im Format:
lcd.print(":"); //Stunden:minuten:sekunden
lcd.print (minute());
lcd.print(":");
lcd.print(second());
lcd.print(" ");
lcd.print("Uhr"); //Dahinter soll das Wort „Uhr“ angezeigt werden
lcd.print(" ");
lcd.print(" ");
lcd.print(" ");
lcd.setCursor(1, 1); //In der zweiten Zeile soll das Datum angezeigt //werden
lcd.print(day());
lcd.print(".");
lcd.print(month());
lcd.print(".");
lcd.print(year());
}
void printDigits(int digits){ //Der printDigits Befehl für den seriellen Monitor
Serial.print(":");
if(digits < 10)
Serial.print(,0');
Serial.print(digits);
}

```

Tastefeld

Ein Tastefeld am Arduino verwenden

by Simon Spettmann

Aufgabe: Mit dem Arduino soll ein Keypad (Tastefeld) ausgelesen und die gedrückte Taste am Serial Monitor angezeigt werden.

Bei dieser Anleitung wird ein **3×4 Tastefeld** benutzt. Alternativ kann auch ein **4×4 Tastefeld** verwendet werden, hierbei müssen die Pins und die definierten Tasten (COLS/hexaKeys) angepasst werden (s.u.).

Material: Arduino / Kabel (7x) / Tastefeld / Stromversorgung

Für die Verwendung des Tastefelds benötigt man die Keypad Library.

Die Library der Arduino IDE hinzufügen:

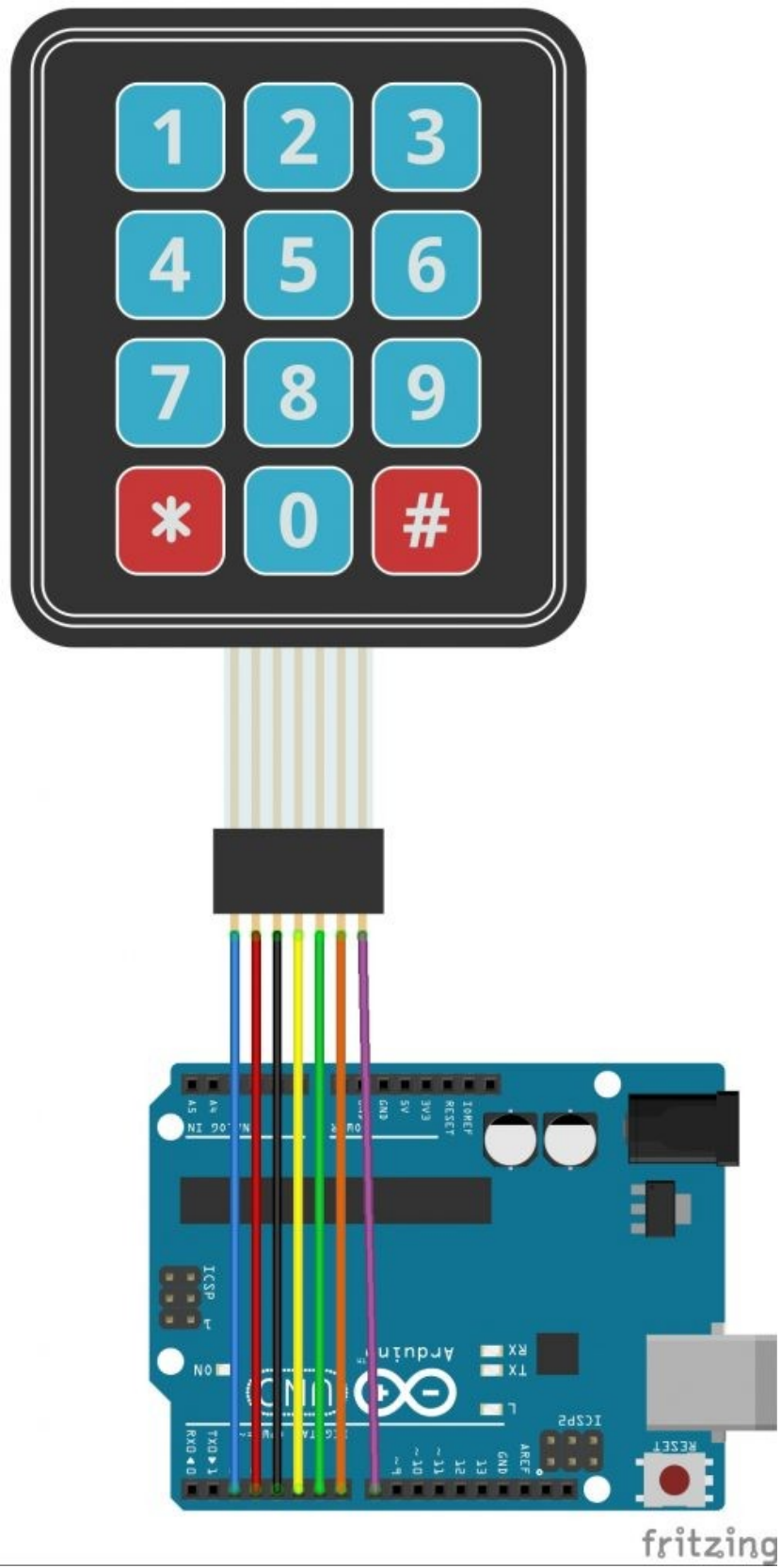
-Öffnen Sie die Arduino IDE

-Menüpunkt: Sketch -> Bibliothek einbinden -> Bibliothek verwalten

-mit der Suchzeile nach „Keypad“ suchen, finden und installieren (Version von Mark Stanly)

Das Tastefeld besteht aus einem 3×4 Feld mit 12 Tasten.

Um die Funktionsweise zu verstehen muss man sich das Keypad wie ein Koordinatensystem vorstellen. Jede Taste ist mit zwei Pins verbunden; ein Pin für die Spalten(COLS) und einer für die Zeilen (ROWS). *Wird eine Taste gedrückt, werden die entsprechenden Pins miteinander verbunden. Diese Pins kann der Arduino mit der `digitalRead()` Funktion auslesen.* (Das Tastefeld benötigt keine externe Stromversorgung.) Zur Anschlussbelegung bitte die Definitionen (s.u.) beachten.



Der Sketch:

```
#include <Keypad.h>

//Hier wird die grÖÙe des Keypads definiert
const byte COLS = 3; //3 Spalten
const byte ROWS = 4; //4 Zeilen
//Die Ziffern/Zeichen:
char hexaKeys[ROWS][COLS] = {
  { '#', '0', '*' },
  { '9', '8', '7' },
  { '6', '5', '4' },
  { '3', '2', '1' }
};
byte colPins[COLS] = { 8, 7, 6 }; //Definition der Pins für die 3 Spalten
byte rowPins[ROWS] = { 5, 4, 3, 2 }; //Definition der Pins für die 4 Zeilen
char pressedKey; //pressedKey entspricht in Zukunft den gedrückten Tasten
Keypad myKeypad = Keypad(makeKeymap(hexaKeys), rowPins, colPins, ROWS,
COLS); //Das Keypad kann ab sofort //mit myKeypad angesprochen werden
void setup() {
  Serial.begin(9600);
}
void loop() {
  pressedKey = myKeypad.getKey(); //pressedKey entspricht der gedrückten Taste
  if (pressedKey) { //Wenn eine Taste gedrückt wurde
    Serial.print("Die Taste ");
    Serial.print(pressedKey);
    Serial.print(" wurde gedrueckt");
    Serial.println(); //Teile uns am Serial Monitor die gedrückte Taste mit
  }}
```

Verwendung eines 4×4 Tastenfeldes (folgende Abschnitte werden geändert):

```
> const byte COLS = 4; //4 Spalten
> byte colPins[COLS] = { 9, 8, 7, 6 }; //Definition der Pins für die 4 Spalten

> char hexaKeys[ROWS][COLS] = {
  { 'D', '#', '0', '*' },
  { 'C', '9', '8', '7' },
  { 'B', '6', '5', '4' },
  { 'A', '3', '2', '1' }
};
```

So sollte es aussehen wenn alles gelungen ist:



Tastentfeld – Schloss

Mit dem Tastentfeld ein Zahlencode-Schloss programmieren

Aufgabe: Mit der Eingabe eines 3-stelligen Codes auf dem Tastentfeld soll eine LED aufleuchten und ein Servo eine bestimmte Position einnehmen. Am Servo könnte z.B. ein Riegel befestigt werden, der eine Tür entriegelt.

Material: Arduino / Breadboard / Tastentfeld (in diesem Beispiel 4x4) / Kabel / ein rote LED/ eine grüne LED / zwei 100 Ohm Widerstände / Servo

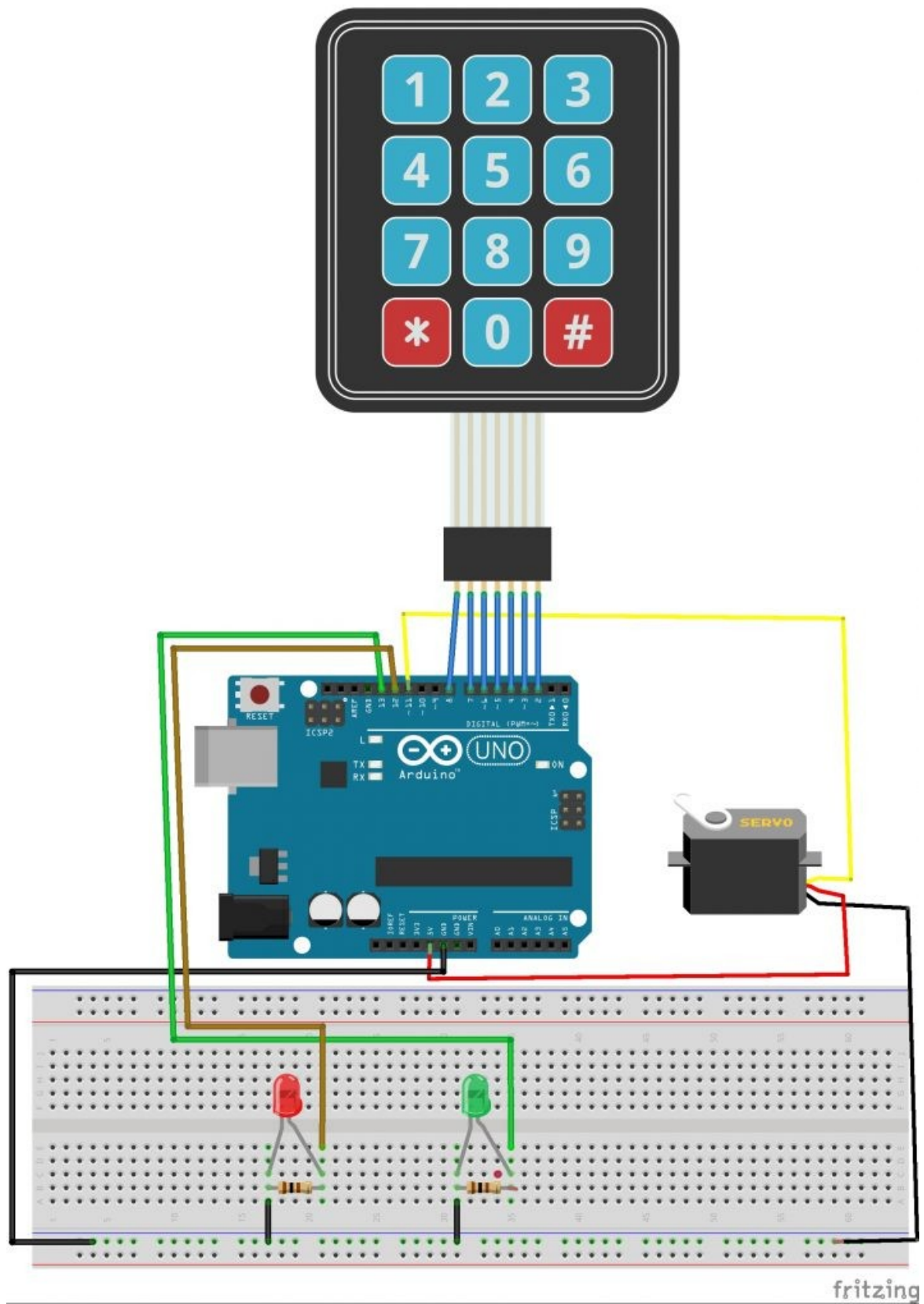
Wir wollen mit der Eingabe eines 3-stelligen Passworts auf dem Tastentfeld ein grüne LED aufleuchten lassen und einen Servo eine bestimmte Position einnehmen lassen. Wenn das „Schloss“ gesperrt ist soll eine rote LED leuchten und der Servo eine andere Position einnehmen. Dieses Tutorial dient als Anregung und Beispiel dazu, wie aus diesen einfachen Bauteilen z.B. ein „Safe“ gebaut werden kann.

Kommen wir nun zum Aufbau. Diese ist recht simpel und geht aus der nachfolgenden Fritzing Skizze hervor.

Zur besseren Orientierung bei dem Verkabeln des Tastentfelds:

An den äußersten Kontakten des Tastentfeld sind die Zahlen 1 und 7 zu sehen. Der Kontakt 1 am Tastentfeld wird mit dem Pin 2 am Arduino verbunden. Aufsteigend geht es dann weiter bis zu Kontakt 7 am Tastentfeld welcher mit dem Pin 8 am Arduino verbunden wird.

Skizze:



Für den Code wird die Keypad Library benötigt, welche der Arduino Software erst hinzugefügt werden muss.

Arduino Software öffnen > „Sketch“ auswählen > „Include Library“ auswählen > „Manage Libraries..“ wählen > In der Suchzeile des neuen Fensters „Keypad“ eingeben > die oberste Library von Mark Stanley auswählen und installieren.

Ab jetzt kann die Keypad Library im Code verwendet werden.

Code:

```
#include <Keypad.h> //Keypad und Servo Library wird eingebunden
#include <Servo.h>

Servo servoblau; //Servo wird ab jetzt mit „servoblau“ angesprochen
char* password = „123“; //Das Passwort wird festgelegt. In diesem Beispiel
// „123“
int position = 0;
const byte ROWS = 4; //Hier wird angegeben wie viele Zeilen und Spalten das
const byte COLS = 3; //Tastenfeld besitzt
char keys[ROWS][COLS] = { //Die Ziffern und Zeichen des Tastenfelds werden
{, #', , 0', , *'}, //angegeben
{, 9', , 8', , 7'},
{, 6', , 5', , 4'},
{, 3', , 2', , 1'}
};
byte rowPins[ROWS] = {5, 6, 7, 8}; //Die Verbindung mit dem Arduino wird
byte colPins[COLS] = {2, 3, 4}; //festgelegt
Keypad keypad = Keypad( makeKeymap(keys), rowPins, colPins, ROWS, COLS );
int roteLED = 12; //Die rote LED ist an Pin 12 angeschlossen
int grueneLED = 13; //Die grüne LED wird an Pin 13 angeschlossen
void setup()
{
pinMode(roteLED, OUTPUT); //Die LEDs werden als Ausgang festgelegt
pinMode(grueneLED, OUTPUT);
servoblau.attach(11); //Der Servo ist an Pin 11 angeschlossen
```

```

setLocked(true);
}
void loop()
{
char key = keypad.getKey();
if (key == ,*' || key == ,#') //wenn das Schloss entsperrt ist kann es mit
der
//Taste „*“ oder „#“ wieder gesperrt werden
{
position = 0;
setLocked(true); //Schloss sperren wenn * oder # gedrückt wurde
}
if (key == password[position])
{
position ++;
}
if (position == 3) //Diese Zeile gibt die Länge des Passwortes an. In
unserem
//Fall 3 Stellen.
{
setLocked(false);
}
delay(100);
}
void setLocked(int locked)
{
if (locked) // Wenn das Schloss gesperrt ist soll..
{
digitalWrite(roeteLED, HIGH); //..die rote LED leuchten..
digitalWrite(grueneLED, LOW); //..die grüner LED nicht leuchten..
servoblau.write(90); //und der Servo soll 0 Grad ansteuern.
}
else //wenn das Schloss entsperrt ist soll..

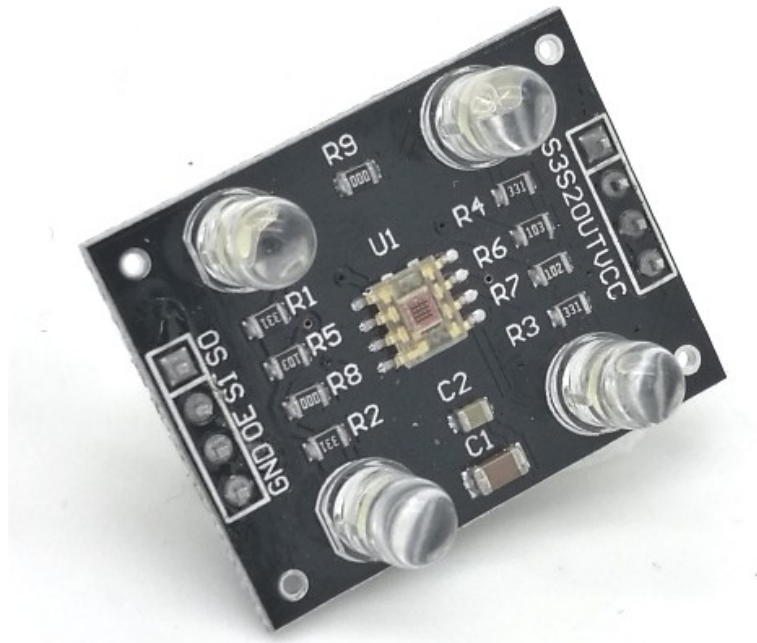
```

```
{  
digitalWrite(roeteLED, LOW); //..die rote LED nicht leuchten..  
digitalWrite(grueneLED, HIGH); //..die grüne LED leuchten..  
servoblau.write(0); //..und der Servo 90 Grad ansteuern.  
}  
}
```

Diese Anleitung dient als Beispiel und Grundstein wie man ein einfaches „Schloss“ mit Hilfe weniger Bauteile und dem Arduino konstruieren kann.

Farbsensor

Farbsensor TCS3200 mit Arduino Mikrocontrollern auslesen



Aufgabe: Mit Hilfe eines Farbsensor sollen die Farben Rot, Grün und Blau erkannt werden und eine entsprechende LED aufleuchten.

Material: Arduino / Kabel / [Farbsensor](#) / Rote LED / Grüne LED / Blaue LED / Widerstände für die LEDs

Auf dem Farbsensor befinden sich 4 weiße LEDs um genauere Werte zu erhalten. In der Mitte der LEDs befindet sich ein kleines Viereck aus Fotodioden, welche die Intensität einer Farbe filtern können. Es sind Fotodioden für grüne, rote und blaue Farbe, sowie Fotodioden ohne Farbfilter vorhanden. Diese „filtern“ heraus welche Farbe gerade vor den Farbsensor gehalten wird und wandeln diese in Werte um, die der Arduino Mikrocontroller verarbeiten kann.

Der Farbsensor kann zum Beispiel in dem Aufbau einer Sortiermaschine verwendet werden.

Für unseren Testaufbau haben wir folgenden Aufbau mit entsprechender Verkabelung vorgenommen

Verkabelung:

Farbsensor >>> Arduino
VCC >>> 5V
GND >>> GND
S0 >>> 8
S1 >>> 9
S2 >>> 12
S3 >>> 11
OUT >>> 10
OE >>> GND
LEDs >>> Arduino
Rote LED >>> 2
Grüne LED >>> 3
Blaue LED >>> 4

Dies ist ein Beispielcode, mit dem die drei Farben Rot, Grün und Blau vom Arduino ausgelesen werden. Gleichzeitig zeigen drei entsprechend farbige LEDs an, welche Farbe gerade vom Sensor erkannt wurde.

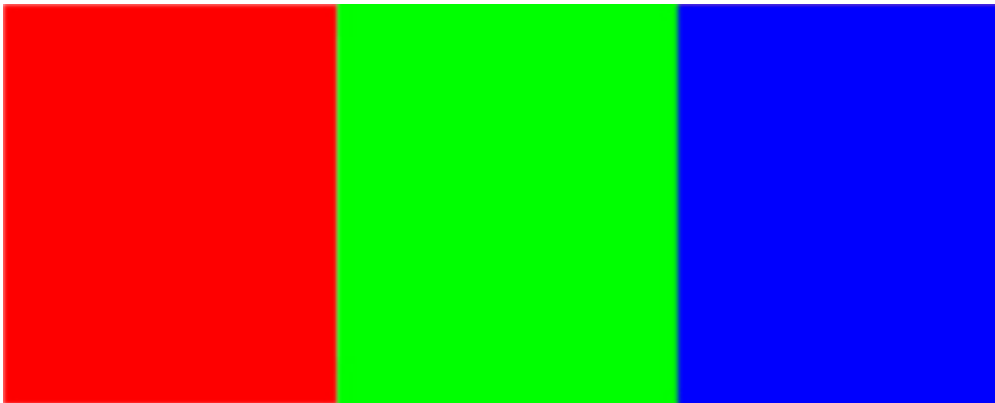
```
const int s0 = 8; //Verbindung der Farbsensorkontakte mit dem Arduino festlegen
const int s1 = 9;
const int s2 = 12;
const int s3 = 11;
const int out = 10;
int roteLED = 2; //Verbindung der LEDs mit dem Arduino festlegen
int grueneLED = 3;
int blaueLED = 4;
int rot = 0; //Variablen für LEDs benennen
int gruen = 0;
int blau = 0;
void setup()
{
  Serial.begin(9600); //Serielle Kommunikation starten
  pinMode(s0, OUTPUT); //Die Kontakte des Farbsensors werden als Output oder
  pinMode(s1, OUTPUT); // Input festgelegt
  pinMode(s2, OUTPUT);
  pinMode(s3, OUTPUT);
  pinMode(out, INPUT);
  pinMode(roteLED, OUTPUT); //Die LEDs werden als Output festgelegt
  pinMode(grueneLED, OUTPUT);
  pinMode(blaueLED, OUTPUT);
  digitalWrite(s0, HIGH); //Die vier weißen LEDs am Farbsensor sollen leuchten
  digitalWrite(s1, HIGH);
}
void loop()
{
```

```

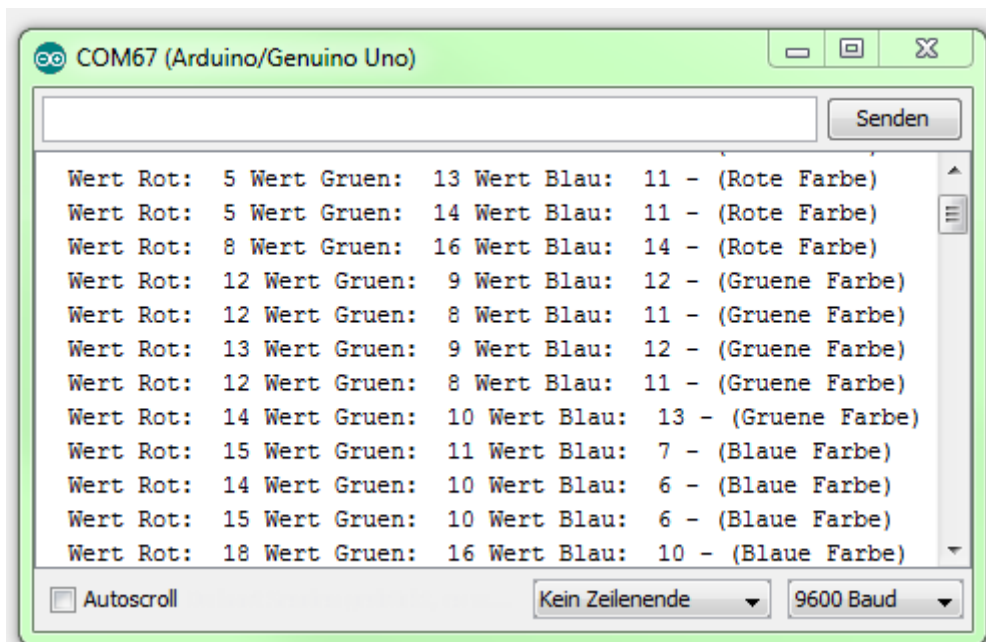
color();//Diese Funktion wird am Ende des Codes festgelegt (s.“void color();”)
Serial.print(" Wert Rot: "); //Auf dem seriellen Monitor soll jeweils „Wert“
Serial.print(rot, DEC); //mit der entsprechenden Farbe angezeigt
Serial.print(" Wert Gruen: "); //werden und dahinter der Wert, welcher in der
Serial.print(gruen, DEC); //void color(); Funktion ausgelesen wurde.
Serial.print(" Wert Blau: ");
Serial.print(blau, DEC);
//Hier folgen die Befehle für die LEDs
if (rot < blau && rot < gruen && rot < 20) //Wenn der Filterwert für rot
//kleiner ist als alle anderen Werte..
{
Serial.println(" - (Rote Farbe)"); //..soll "Rote Farbe" am seriellen //Monitor
angezeigt werden und..
digitalWrite(roteLED, HIGH); //...ie rote LED leuchtet auf, die anderen
digitalWrite(grueneLED, LOW); //bleiben aus
digitalWrite(blaueLED, LOW);
}
else if (blau < rot && blau < gruen) //Das gleiche bei Blau und Grün
{
Serial.println(" - (Blaue Farbe)");
digitalWrite(roteLED, LOW);
digitalWrite(grueneLED, LOW);
digitalWrite(blaueLED, HIGH);
}
else if (gruen < rot && gruen < blau)
{
Serial.println(" - (Gruene Farbe)");
digitalWrite(roteLED, LOW);
digitalWrite(grueneLED, HIGH);
digitalWrite(blaueLED, LOW);
}
else{ //Wenn keine Werte vorhanden sind..
Serial.println(); //..nichts auf dem seriellen Monitor anzeigen und..
}
delay(300);
digitalWrite(roteLED, LOW); //...alle LEDs ausschalten
digitalWrite(grueneLED, LOW);
digitalWrite(blaueLED, LOW);
}
void color() //Hier werden die Werte vom Farbsensor ausgelesen und unter den
//entsprechenden Variablen gespeichert
{
digitalWrite(s2, LOW);
digitalWrite(s3, LOW);
rot = pulseIn(out, digitalRead(out) == HIGH ? LOW : HIGH);
digitalWrite(s3, HIGH);
blau = pulseIn(out, digitalRead(out) == HIGH ? LOW : HIGH);
digitalWrite(s2, HIGH);
gruen = pulseIn(out, digitalRead(out) == HIGH ? LOW : HIGH);
}

```

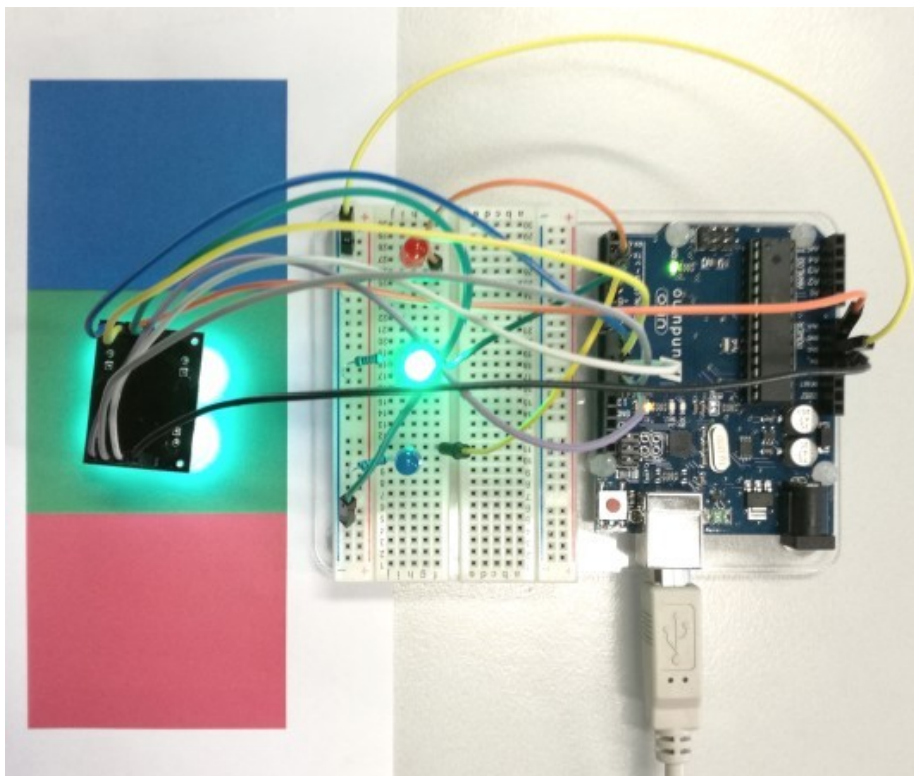
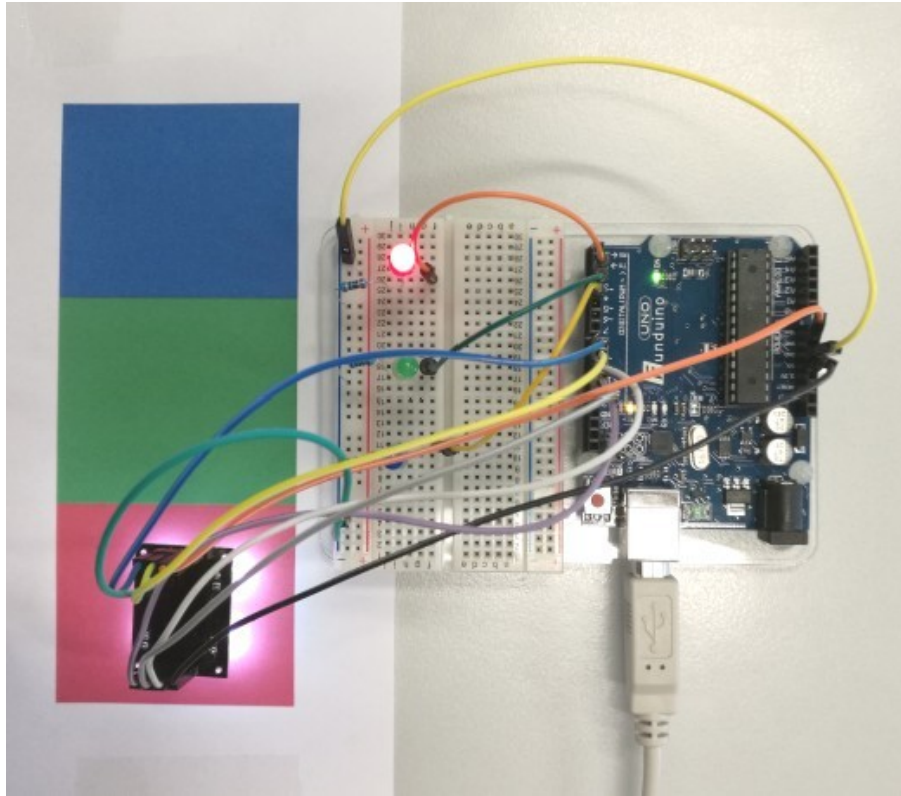

Um unseren Aufbau zu überprüfen haben wir uns einen Zettel mit einer roten, grünen und blauen Fläche ausgedruckt.

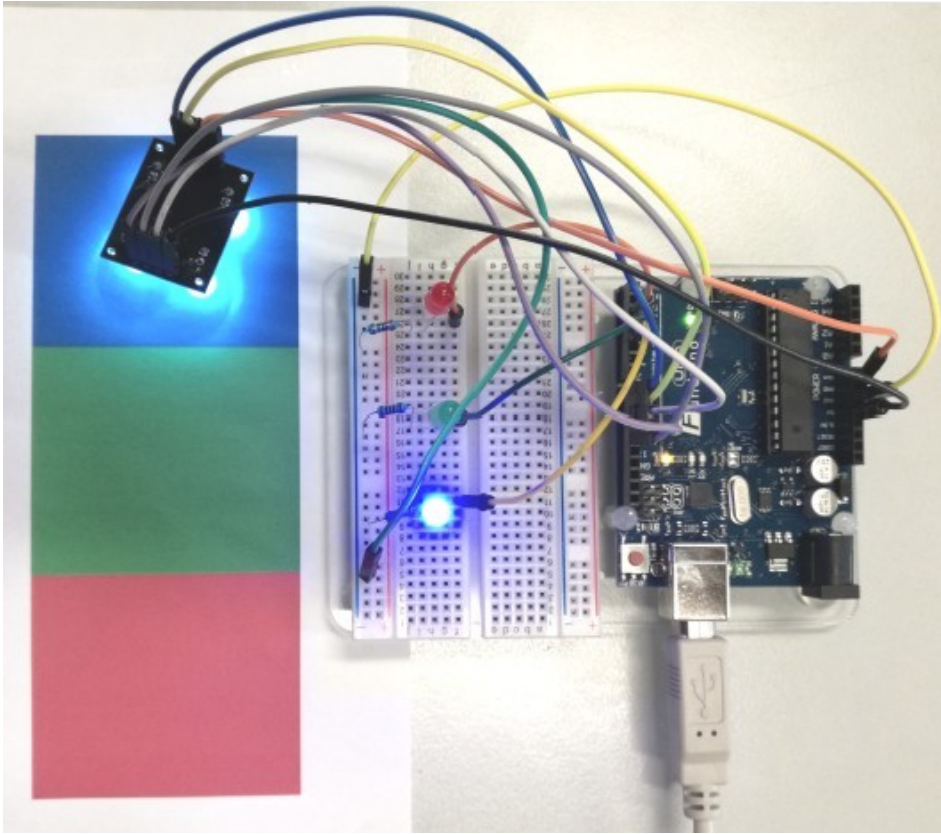


Der serielle Monitor sollte so aussehen:



Je nach dem über welche Farbe man den Farbsensor hält sollte jetzt die entsprechende LED aufleuchten.





Töne erzeugen

Mit dem Arduino und einem Lautsprecher Töne erzeugen

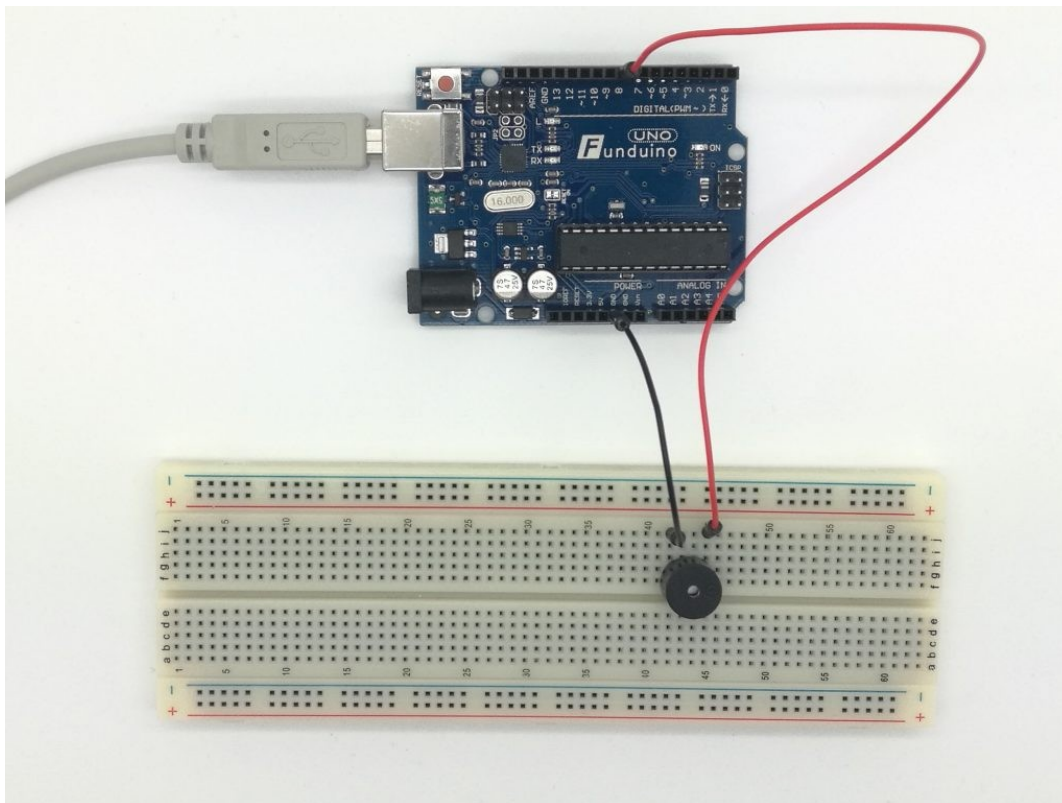
Mit dem Arduino lassen sich auf verschiedenen Art und Weisen Töne erzeugen. Die einfachste Möglichkeit ist die Tonerzeugung mit einem aktiven Lautsprecher (active Buzzer), der lediglich an die Spannung von 5V angeschlossen wird. Die Tonerzeugung übernimmt eine im Inneren eingebaute Elektronik. Der Nachteil liegt jedoch darin, dass ein „active buzzer“ nur einen einzigen Ton erzeugen kann – Melodien oder Sirengeräusche sind damit nicht möglich.

Mit einem passiven **Lautsprecher (passive buzzer)** hat man die Möglichkeit mit Hilfe des Arduino Mikrocontrollers verschiedene Töne, Melodien oder Sirensignale zu generieren, da im passive buzzer keine Elektronik vorhanden ist, die einen Ton vorgibt.

Aufgabe: Mit einem passiven Lautsprecher sollen unterschiedliche Töne und eine Melodie erzeugt werden.

Material: Arduino Mikrocontroller / Ein passiver Lautsprecher (passive buzzer) / Breadboard / Kabel

Aufbau:



Die Erzeugung des Tones basiert maßgeblich auf dem Befehl „tone(x, y)“, wobei der x-Wert den Pin angibt, an dem der Lautsprecher mit der positiven Seite angeschlossen ist und der y-Wert der die Tonhöhe angibt.

Ein Beispiel:

```
tone(8, 100); // Der Lautsprecher an Pin 8 wird mit der Tonhöhe „100“ aktiviert
```

```
delay(1000); // Pause mit 1000 Millisekunden, also 1 Sekunde – Der Lautsprecher bleibt für diese Zeit aktiv.
```

```
noTone(8); // Der Lautsprecher an Pin 8 wird deaktiviert
```

Code1: Einfache Tonerzeugung

```
void setup() // Im Setup werden keine Informationen benötigt.  
//Die Spannungsausgabe für den Piezo-Lautsprecher wird im Sketch durch den Ardui  
no-Befehl "tone" automatisch festgelegt.  
{  
}  
  
void loop()  
{  
tone(8, 100); // Im Hauptteil wird nun mit dem Befehl "tone ( x , y )" ein Ton  
abgegeben.  
delay(1000); // mit einer Dauer von 1 Sekunde  
noTone(8); // Der Ton wird abgeschaltet  
delay(1000); // Der Lautsprecher bleibt eine Sekunde aus  
}
```

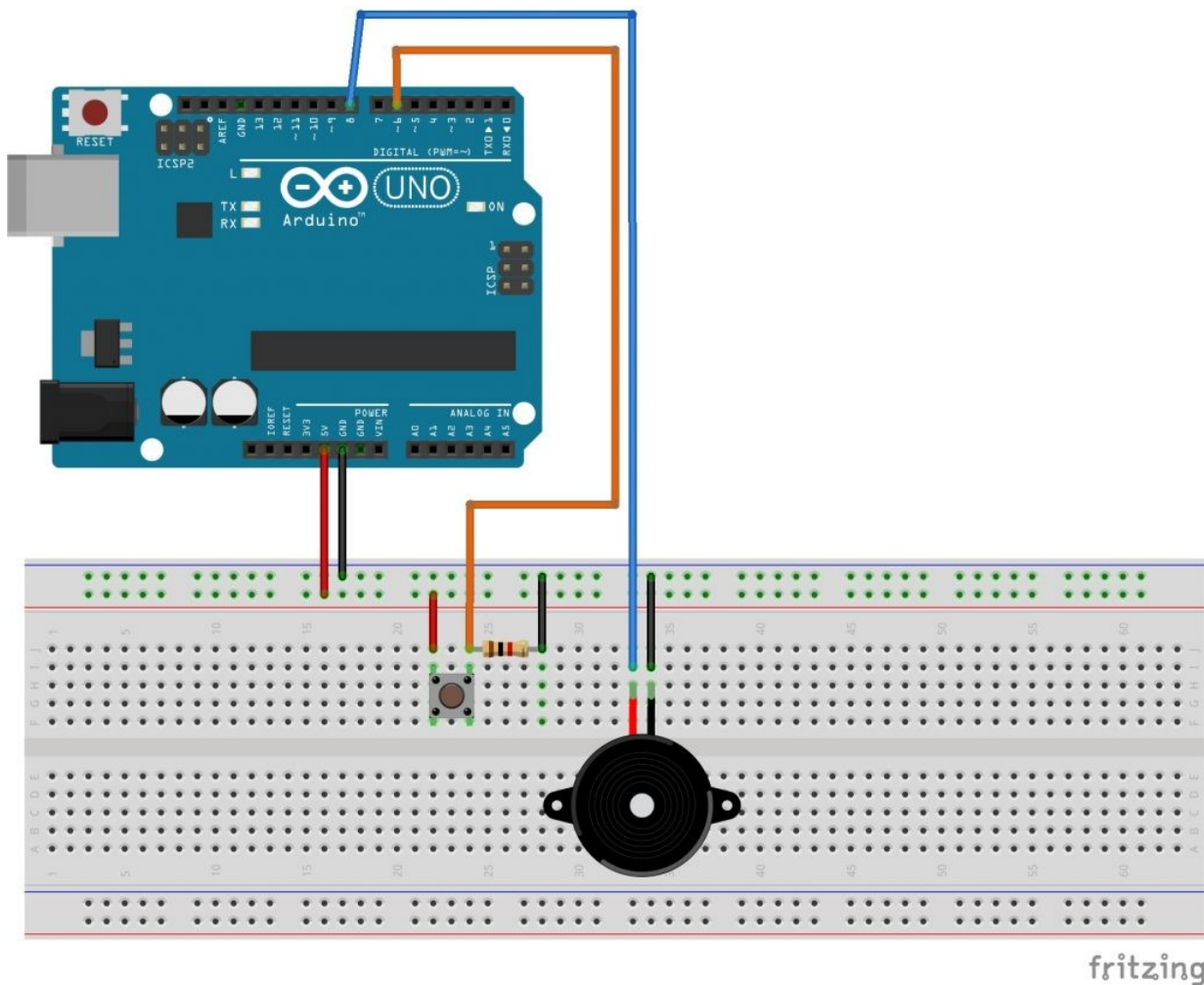
Code2: Abwechselnde Tonhöhen

Im zweiten Beispiel werden im Hauptteil (loop) nur ein paar Zeilen ergänzt. Dadurch ertönen zwei Töne abwechselnd mit der Tonhöhe “100” bzw. “200”, wie bei einer Sirene. Eine pause zwischen den Tönen gibt es nicht.

```
void setup()  
{  
}  
void loop()  
{  
tone(8, 100);  
delay(1000);  
noTone(8); // An dieser Stelle geht der erste Ton aus.  
tone(8, 200); // Der zweite Ton mit der neuen Tonhöhe “200” ertönt.  
delay(1000); //... und zwar für eine Sekunde...  
noTone(8); // An dieser Stelle geht der zweite Ton aus und der Loop-Teil des  
Sketches beginnt von vorn.  
}
```


Code3: Ton erzeugen durch Tastendruck

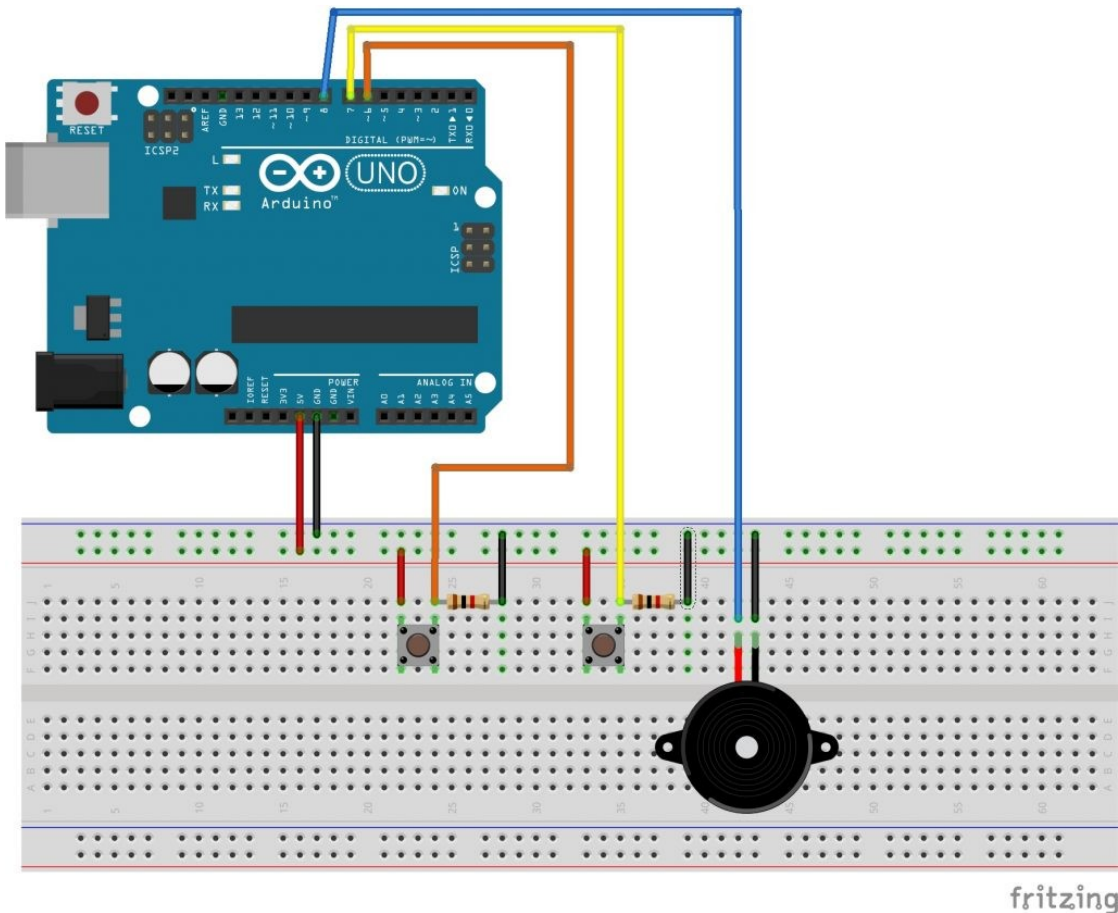
Im dritten Beispiel wird ein Taster an Pin6 zum Aufbau hinzugefügt. Im Hauptteil wird durch eine "IF-Abfrage" der Taster ausgelesen. Falls der Taster gedrückt ist, ertönt für eine Sekunde ein Ton mit der Tonhöhe "300".



```
int Taster1=6;
int Tasterstatus=0;
void setup()
{
  pinMode(Taster1, INPUT);
}
void loop()
{
  Tasterstatus = digitalRead(Taster);
  if (Tasterstatus == HIGH)
  {
    tone(8, 300);
    noTone(8);
  }
}
```

Code4: Töne in Abhängigkeit von verschiedenen Tasten

Im vierten Beispiel soll je nach Betätigung unterschiedlicher Tasten entschieden werden, welcher Ton vom Lautsprecher abgegeben wird. Dazu wird jeweils ein Taster an den Pins 6 und 7 angeschlossen. Im Hauptteil wird durch zwei "IF-Abfragen" entschieden, welcher Ton abgegeben wird.



Code:

```
int Taster1=6; // Taster1 an Pin6 angeschlossen
int Taster2=7; // Taster2 an Pin7 angeschlossen
int Tasterstatus1=0; // Variable um den Status des Tasters 1 zu speichern.
int Tasterstatus2=0; // Variable um den Status des Tasters 2 zu speichern.

void setup()
{
  pinMode(Taster1, INPUT); // Taster1 als Eingang festlegen
  pinMode(Taster2, INPUT); // Taster2 als Eingang festlegen
}

void loop()
{
  Tasterstatus1 = digitalRead(Taster1); // Status von Taster1 auslesen (HIGH oder LOW)
  Tasterstatus2 = digitalRead(Taster2); // Status von Taster2 auslesen (HIGH oder LOW)
```



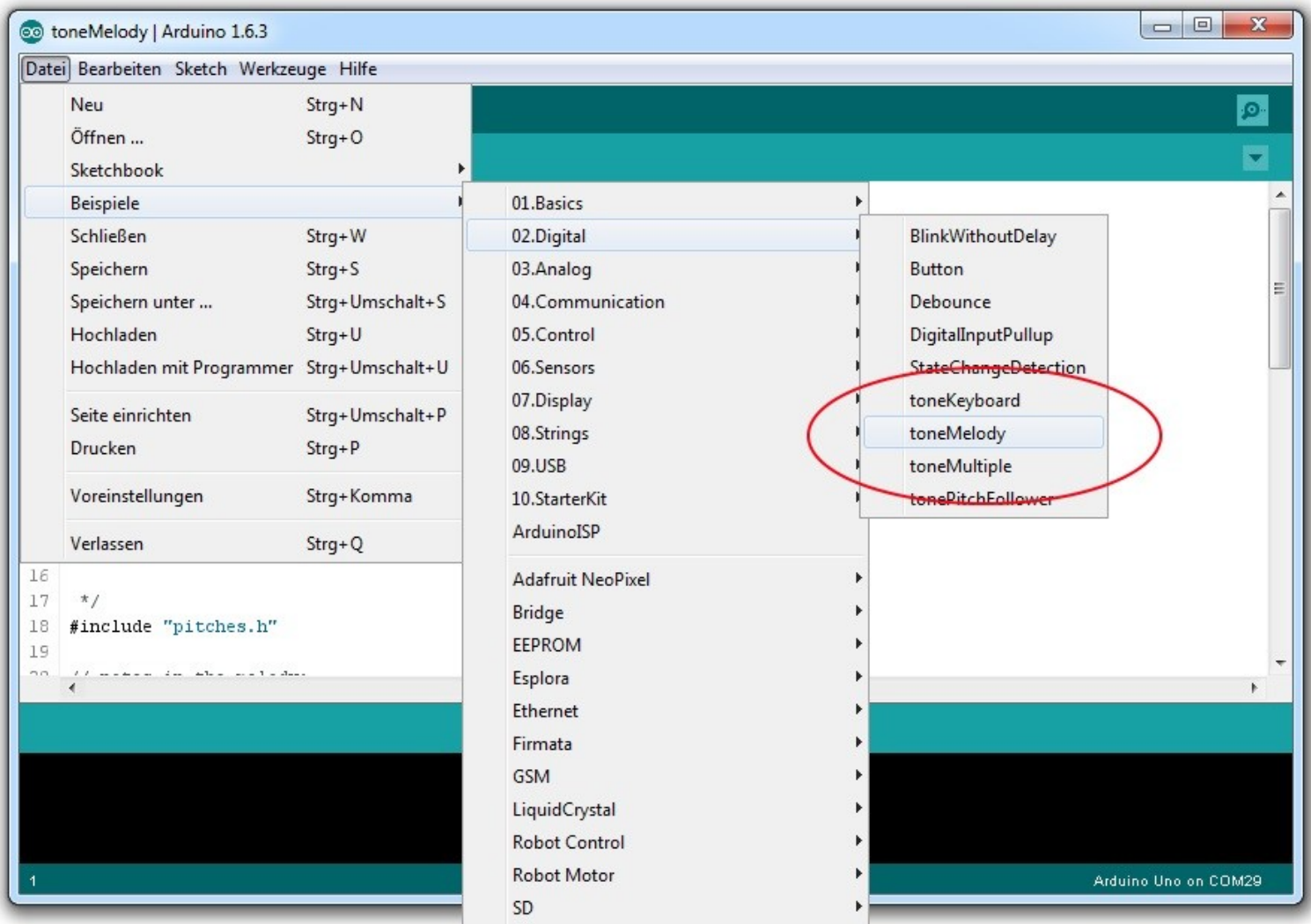
```
if (Tasterstatus1 == HIGH) // Wenn der Taster1 gedrückt ist, dann...
{
tone(8, 100); // Ausgabe eines Tons mit der Tonhöhe 100.
delay (1000); // mit der Dauer von einer Sekunde
noTone(8); // Ausschalten des Tons.
}

if (Tasterstatus2 == HIGH)
{
tone(8, 200);
delay (1000); // mit der Dauer von einer Sekunde
noTone(8); // Ausschalten des Tons.
}
}
```

Melodien erzeugen

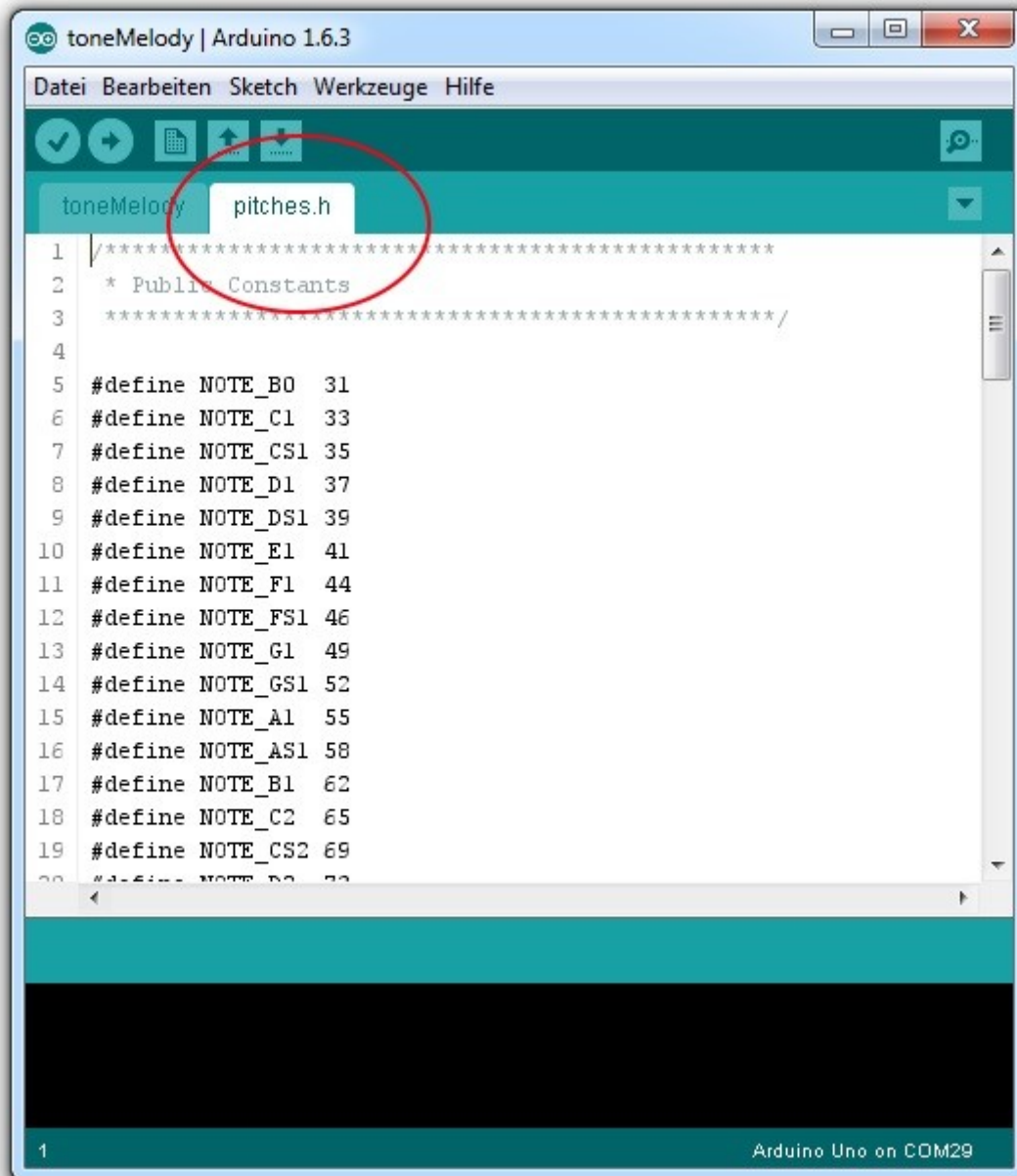
In der Arduino-Software ist eine Datei speziell zum generieren von Tönen mit Hilfe eines Lautsprechers (passive Buzzer) enthalten. Anstelle von Tonhöhen in Form von Zahlen wie bspw. "tone(8, 200)", können nun auch Töne aus der Tonleiter ausgewählt werden.

Als Beispiel gibt es dafür in der Arduino-Software den Sketch "toneMelody".



Zu finden unter “Beispiele” -> “02.Digital” -> “toneMelody”

In dem Beispiel ist bereits die Datei hinterlegt, in der sich die Zuordnung von Tonhöhe und Zahlenwert befindet. Sie hat den Namen “pitches.h”. Den Inhalt kann man sich ansehen, wenn man in dem geöffneten Beispielprogramm auf den zweiten Kartenreiter innerhalb des Sketches klickt, wie im folgenden Bild dargestellt.



Um auf die Datei zurückgreifen zu können, muss im Sketch lediglich der Befehl `#include „pitches.h“` eingebaut werden und die Datei selber muss als Tab geöffnet sein. Am besten geht das, indem man den Sketch “toneMelody” aus den Beispielen in der Arduino-Software öffnet und dann bearbeitet. Dadurch wird die Datei “pitches.h” automatisch als Tab aufgerufen. Das wird im folgenden Beispiel deutlich.

Dies ist ein ganz kleiner Sketch, der Abwechselnd zwei Töne aus der Datei "pitches.h" abspielt.

```
#include "pitches.h"

void setup()
{
  pinMode (8,OUTPUT); // Lautsprecher an Pin8
}

void loop()
{
  tone(8, NOTE_C4, 1000); // An Pin8 wird die Note C4 für 1000ms gespielt
  delay(3000); //Nachdem die Note ertönt, pausiert der Sketch für 3 Sekunden. Das
  bedeutet, dass nachdem der Ton zu ende gespielt wurde, noch zwei Sekunden Pause
  ohne Ton verbleiben.
  tone(8, NOTE_G3, 1000); // An Pin8 wird die Note G3 für 1000ms gespielt
  delay(3000); //Nachdem die Note ertönt, pausiert der Sketch für 3 Sekunden. Das
  bedeutet, dass nachdem der Ton zu ende gespielt wurde, noch zwei Sekunden Pause
  ohne Ton verbleiben.
}
```

In diesem Sketch wird durch den erweiterten "tone-" Befehl die Beendigung des Tones nicht mehr benötigt.

Befehl: "tone(x, y, z)" x= Pin des Lautsprechers, y= Tonhöhe, z= Dauer des Tons in Millisekunden.

Arduino Neigungssensor SW-520D

Sketch: Mit dem Arduino einen Neigungssensor SW-520D auslesen

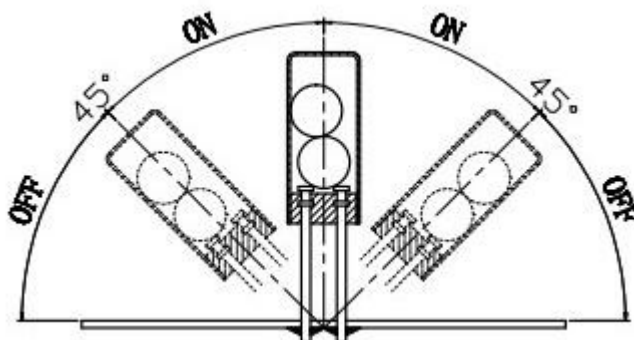


Mit dem Arduino lassen sich auf verschiedenen Art und Weisen Neigungen messen. Sehr exakte Messungen bekommt man zum Beispiel mit einem Beschleunigungssensor, bei dem man nahezu jeden Winkel genau messen kann. Es geht jedoch auch ganz einfach mit dem [Neigungssensor](#) (engl. Tilt sensor oder Tiltswitch).

Dieser Sensor erkennt jedoch nur zwei Stufen, nämlich "geneigt" oder "nicht geneigt". Dazu rollen im inneren des Sensors zwei frei bewegliche Metallkugeln umher. Sobald der Sensor so gekippt ist, dass eine der Kugeln im Inneren gegen die zwei Kontakte stößt, werden diese beiden Kontakte miteinander verbunden. Das ist dann so, als wäre ein Taster gedrückt, und der elektrische Strom kann von einem Kontakt zum anderen Kontakt fließen.

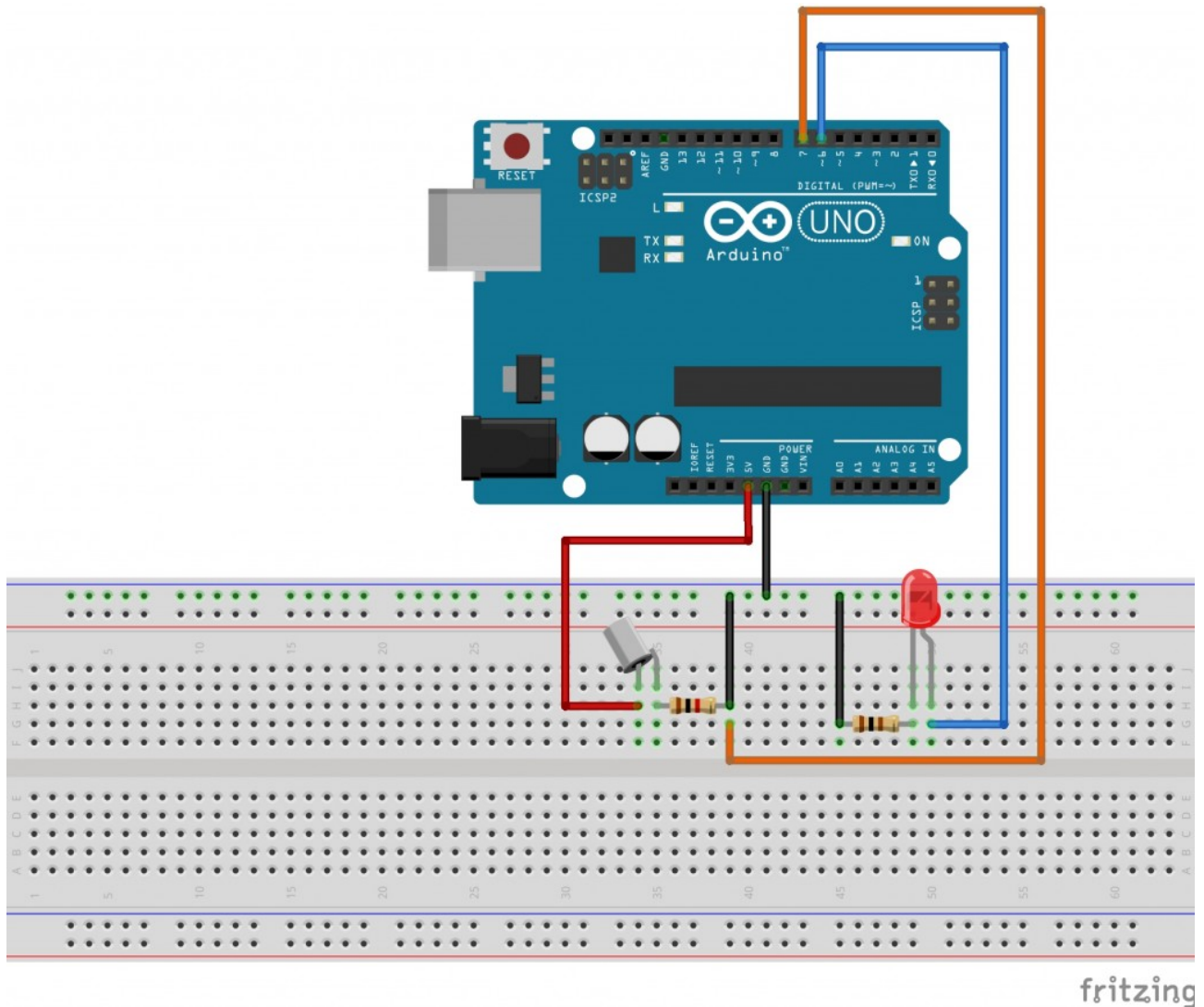
Daher ist der Sketch und der Aufbau sehr ähnlich, wie der Stetch mit dem Taster.

Im folgenden Bild erkennt man die Funktionsweise des Sensors mit den beiden Kugeln.



Wenn der Sensor wie auf diesem Bild senkrecht verwendet wird, beispielsweise wenn er einfach im Breadboard eingesteckt wird, ist der Stromkreis zwischen den beiden Kontakten geschlossen und der Strom kann von einem Kontakt zum anderen fließen. Sobald der Sensor jedoch um mehr als 45° geneigt wird, rollen die Kugeln von den beiden Kontakten weg und die Verbindung ist unterbrochen.

Aufbauskitze:



Aufgabe: Mit einem Neigungssensor soll eine LED eingeschaltet werden, sobald eine Neigung von mehr als 45° vorliegt

Material: Arduino / eine LED / Ein Widerstand mit 100 Ohm / Ein Widerstand mit 1K Ohm (1000 Ohm) / Breadboard / Kabel / Neigungssensor

Der Sketch:

```
int LED=6; //Das Wort „LED“ steht jetzt für den Wert 6.
int TILT=7; //Das Wort „TILT“ steht jetzt für den Wert 7 - Der Neigungssensor
wird also an Pin7 angeschlossen
int NEIGUNG=0; //Das Wort „NEIGUNG“ steht jetzt zunächst für den Wert 0. Später
wird unter dieser Variable gespeichert, ob eine Neigung von mehr als 45 Grad
vorliegt oder nicht.

void setup() //Hier beginnt das Setup.
{
  pinMode(LED, OUTPUT); //Der Pin mit der LED (Pin 6) ist jetzt ein Ausgang.
  pinMode(TILT, INPUT); //Der Pin mit dem Neigungssensor (Pin 7) ist jetzt ein
Eingang.
}

void loop()
{ //Mit dieser Klammer wird der Loop-Teil geöffnet.
  NEIGUNG=digitalRead(TILT); //Hier wird der Pin7, also der Neigungssensor
ausgelesen (Befehl:digitalRead). Das Ergebnis wird unter der Variable „NEIGUNG“
mit dem Wert „HIGH“ für 5Volt oder „LOW“ für 0Volt gespeichert. Wenn der Sensor
gerade steht, kann der Strom zwischen beiden Kontakten des Sensors fließen. Es
liegt dann am Pin7 eine Spannung an. Der Status wäre dann also „HIGH“. Wenn der
Sensor 45 oder mehr Grad geneigt ist, kann kein Strom fließen und der Status
wäre dann „LOW“.

  if (NEIGUNG == HIGH) //Verarbeitung: Wenn der Sensor weniger als 45 Grad geneigt
ist
  { //Programmabschnitt des IF-Befehls öffnen.
    digitalWrite(LED, LOW); //dann soll die LED nicht leuchten
  } //Programmabschnitt des IF-Befehls schließen.
  else //...ansonsten...
  { //Programmabschnitt des else-Befehls öffnen.
    digitalWrite(LED, HIGH); //...soll die LED leuchten.
  } //Programmabschnitt des else-Befehls schließen.
} //Mit dieser letzten Klammer wird der Loop-Teil geschlossen.
```


Temperatur messen mit dem LM35

Aufgabe: Mit dem Temperatursensor LM35 soll die Temperatur ausgelesen und mit dem serial-monitor angezeigt werden.

Material: Arduino / Kabel / [Temperatursensor LM35](#) / [Schraubklemmen Shield](#) oder Kabel zum anlöten an den Sensor

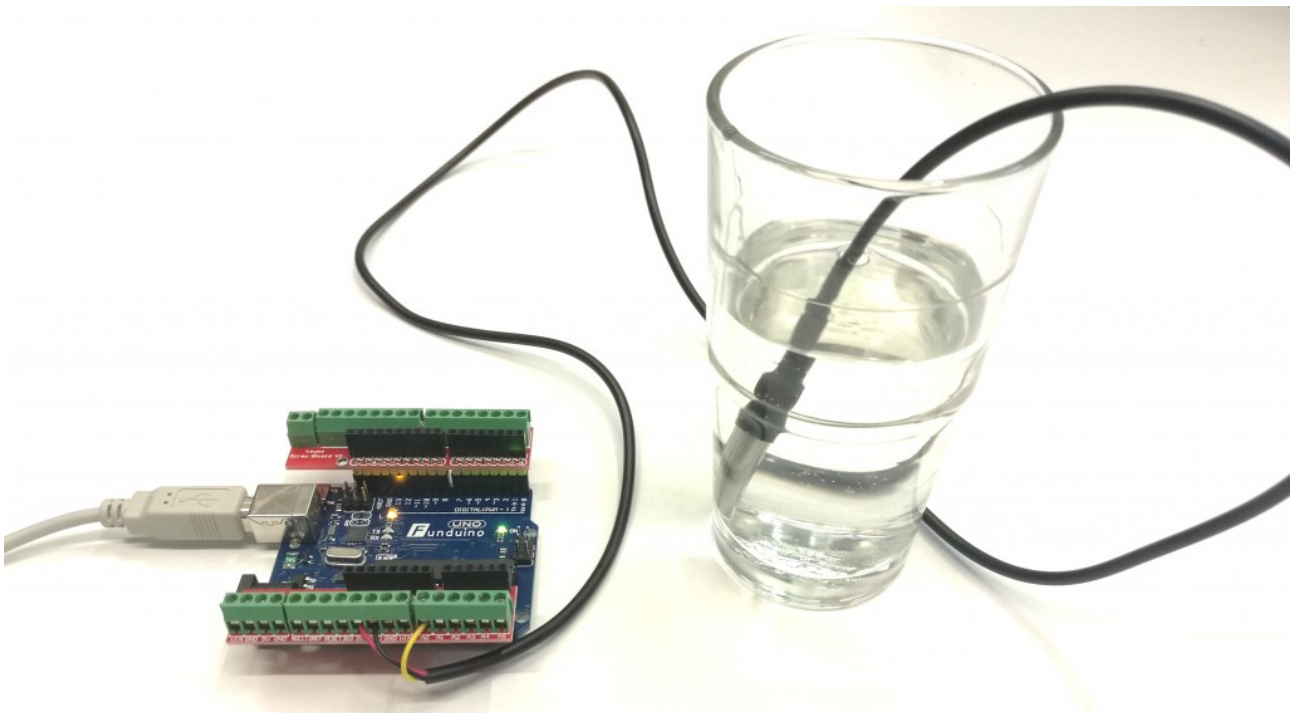
Der Sensor hat drei Anschlüsse. der rote Kontakt 5V, der schwarze GND und der gelbe ist der Kontakt für das Temperatursignal. An diesem Kontakt gibt der Sensor eine Spannung zwischen 0 und 1,5 Volt aus. Wobei 0V 0°C entsprechen und der Wert 1,5V entspricht 150°C. Die Spannung dieses Pins muss vom Mikrocontroller-Board ausgelesen und in einen Temperaturwert umgerechnet werden.

– Das besondere an diesem Temperatursensor ist seine wasserdichte Eigenschaft. Somit kann mit dem LM35 die Temperatur von Flüssigkeiten gemessen werden.

– ACHTUNG: Wenn der Sensor falsch angeschlossen wird, brennt er durch!

– Bei dem Aufbau sollte nach Möglichkeit eine externe Stromversorgung verwendet werden, da dies die Sensorgenauigkeit wesentlich verbessert (9V Netzteil oder 9V-Batterie).

Aufbau mit Schraubklemmen Shield:



Code:

```
int LM35 = A0; //Der Sensor soll am analogen Pin A0 angeschlossen werden. Wir nennen den Pin ab jetzt „LM35“
```

```
int temperatur = 0; //Unter der Variablen „temperatur“ wird später der Temperaturwert abgespeichert.
```

```
int temp[10]; //Damit man gute Werte erhält, liest man zunächst mehrere Werte aus und bildet dann den Mittelwert. Die eckige Klammer „[10]“ erzeugt hier gleich zehn Variablen mit den Namen „temp[0]“, „temp[2]“, „temp[3]“, ... bis ... „temp[9]“. Mit der Schreibweise [10] spart man also nur etwas Platz.
```

```
int time= 20; //Der Wert für „time“ gibt im Code die zeitlichen Abstände zwischen den einzelnen Messungen vor.
```

```
void setup()  
{  
  Serial.begin(9600); //Im Setup beginnen wir die serielle Kommunikation  
}
```

```
void loop()  
{  
  //genauere Erklärung unter dem Code *1  
  temp[0] = map(analogRead(LM35), 0, 410, -50, 150);  
  delay(time);  
  temp[1] = map(analogRead(LM35), 0, 410, -50, 150);  
  delay(time);  
  temp[2] = map(analogRead(LM35), 0, 410, -50, 150);  
  delay(time);  
  temp[3] = map(analogRead(LM35), 0, 410, -50, 150);  
  delay(time);  
  temp[4] = map(analogRead(LM35), 0, 410, -50, 150);  
  delay(time);  
  temp[5] = map(analogRead(LM35), 0, 410, -50, 150);  
  delay(time);  
  temp[6] = map(analogRead(LM35), 0, 410, -50, 150);  
  delay(time);  
  temp[7] = map(analogRead(LM35), 0, 410, -50, 150);
```

```

delay(time);
temp[8] = map(analogRead(LM35), 0, 410, -50, 150);
delay(time);
temp[9] = map(analogRead(LM35), 0, 410, -50, 150);
temperatur=(temp[0]+temp[1]+temp[2]+temp[3]+temp[4]+temp[5]+temp[6]+temp[7]+temp
[8]+temp[9])/10;
Serial.print(temperatur);
Serial.println(" Grad Celsius");
}

```

Genauere Erklärung zum Loop *1:

Hier wird zehnmal die Temperatur ausgelesen. Dazwischen ist je eine kleine Pause mit der Dauer „time“ in Millisekunden. Aber was passiert hier genau? Sehen wir uns den Befehl einmal genauer an.

```
temp[1] = map(analogRead(LM35), 0, 307, 0, 150);
```

Der Reihe nach:

temp[1] – ist der Name der ersten Variablen.

„map (a, b, c, d, e)“ – Dies ist der sogenannte „Map-Befehl“. Hiermit kann man einen eingelesenen Wert (a) bei dem die Werte zwischen (b) und (c) liegen, in einen anderen Zahlenbereich umwandeln, und zwar in den Bereich zwischen (d) und (e).

In unserem Befehl passiert folgendes:

Der Wert des Sensors wird unmittelbar im „Map-Befehl“ ausgelesen mit „analogRead(LM35)“. Die Messwerte sollten zwischen 0 und 307 liegen. Das entspricht am analogen Port den Werten zwischen 0 V und 1,5V. Die Spannung gibt der Sensor bei Temperaturen zwischen 0°C und +150°C aus. Diese Werte am analogen Port werden nun durch den „Map-Befehl“ direkt in die °Celsius Werte zwischen 0 und 150 umgewandelt.

Danach werden alle zehn ermittelten Temperaturwerte zusammengerechnet und durch zehn geteilt. Der durchschnittliche Wert wird unter der Variablen „temperatur“ gespeichert.

Zuletzt wird der Wert „temperatur“ über die serielle Kommunikation an den PC gesendet. Nun muss man den seriellen Monitor in der Arduino-Software öffnen um die Temperatur am Sensor abzulesen.

Erweiterung des Programms:

Sobald die Temperatur von 30°C erreicht ist, soll ein Warnsignal ertönen (Piepston eines Piezo-Lautsprechers).

Code:

```
int LM35 = A0;
int temperatur = 0;
int temp[10];
int time= 20;

int piezo=5; //Das Wort „piezo“ ist jetzt die Zahl 5, also wird an Port 5 der
Piezo angeschlossen.

void setup()
{
  Serial.begin(9600);
  pinMode (piezo, OUTPUT); //Der Piezo-Lautsprecher an Pin5 soll ein Ausgang sein
(Logisch, weil der ja vom Mikrokontroller-Board ja eine Spannung benötigt um zu
piepsen.
}

void loop() {

temp[0] = map(analogRead(LM35), 0, 307, 0, 150);
delay(time);
temp[1] = map(analogRead(LM35), 0, 307, 0, 150);
delay(time);
temp[2] = map(analogRead(LM35), 0, 307, 0, 150);
delay(time);
temp[3] = map(analogRead(LM35), 0, 307, 0, 150);
delay(time);
temp[4] = map(analogRead(LM35), 0, 307, 0, 150);
delay(time);
temp[5] = map(analogRead(LM35), 0, 307, 0, 150);
delay(time);
temp[6] = map(analogRead(LM35), 0, 307, 0, 150);
delay(time);
temp[7] = map(analogRead(LM35), 0, 307, 0, 150);
delay(time);
temp[8] = map(analogRead(LM35), 0, 307, 0, 150);
delay(time);
temp[9] = map(analogRead(LM35), 0, 307, 0, 150);

temperatur=(temp[0]+temp[1]+temp[2]+temp[3]+temp[4]+temp[5]
+temp[6]+temp[7]+temp[8]+temp[9])/10; // Alles in eine Zeile!

Serial.print(temperatur);

Serial.println(" Grad Celsius");

if (temperatur>=30) //Es wird eine IF-Bedingung erstellt:Wenn der Wert für die
Temperatur über oder gleich 30 ist, dann...

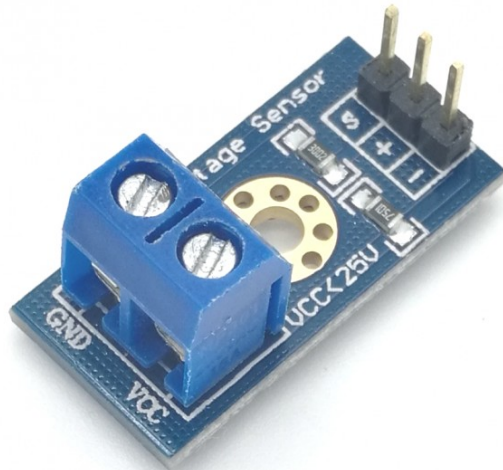
{
digitalWrite(piezo,HIGH);//...fange an zu piepsen.
}
}
```

```
else //Und wenn das nicht so ist...  
{  
digitalWrite(piezo,LOW); //...dann sein leise.  
}  
}
```

Spannung messen

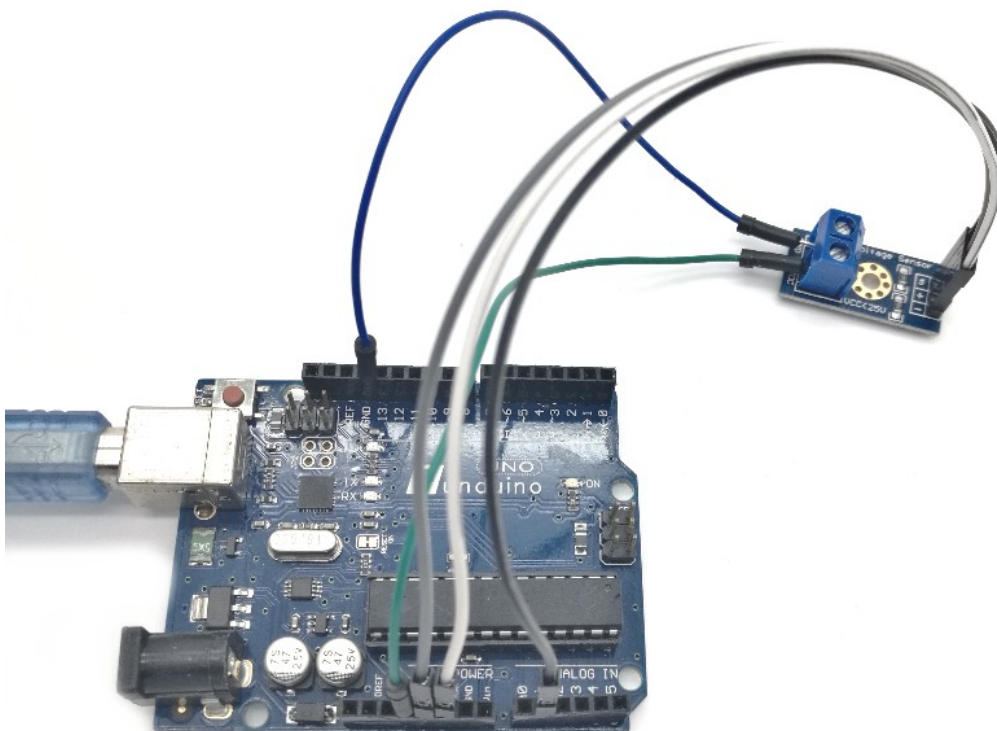
Spannungssensor

Mit dem Arduino Mikrocontrollerboard und einem [Spannungssensor](#) kann man die elektrische Spannung messen. Mit dem von uns verwendeten Sensor kann die zu Messende Spannung in einem Bereich zwischen 0 und 25V DC (Gleichstrom) liegen.



Aufgabe: Eine Spannung mit einem Spannungssensor (Voltage sensor) messen und auf dem seriellen Monitor anzeigen lassen.

Material: Arduino/ Kabel / Spannungssensor



Wir wollen mit einem Spannungssensor eine Spannung messen und diese am seriellen Monitor anzeigen lassen. So ein Spannungssensor macht Sinn, wenn man eine größere Spannung, als die maximale messbare Spannung (5V) vom Arduino messen möchte.

So ein Spannungssensor kann in einem Bereich von 0-25V Spannungen messen. Das Modul misst die Spannung und wandelt diese in für den Arduino messbare Werte um.

Durch mathematische Formeln in dem Code ist es dann möglich eine Volt-Angabe auf dem seriellen Monitor anzugeben.

Verkabelung: Diese gestaltet sich einfach, da der Sensor nur 3 Kontakte besitzt welche mit dem Arduino verbunden werden müssen.

Spannungssensor >>> Arduino

S >>> A1

+ >>> 5V

- >>> GND

Auf der anderen Seite befinden sich die mit GND und VCC beschrifteten Kontakte an welche die zu messende Spannung angeschlossen wird.

Kommen wir nun zum Code:

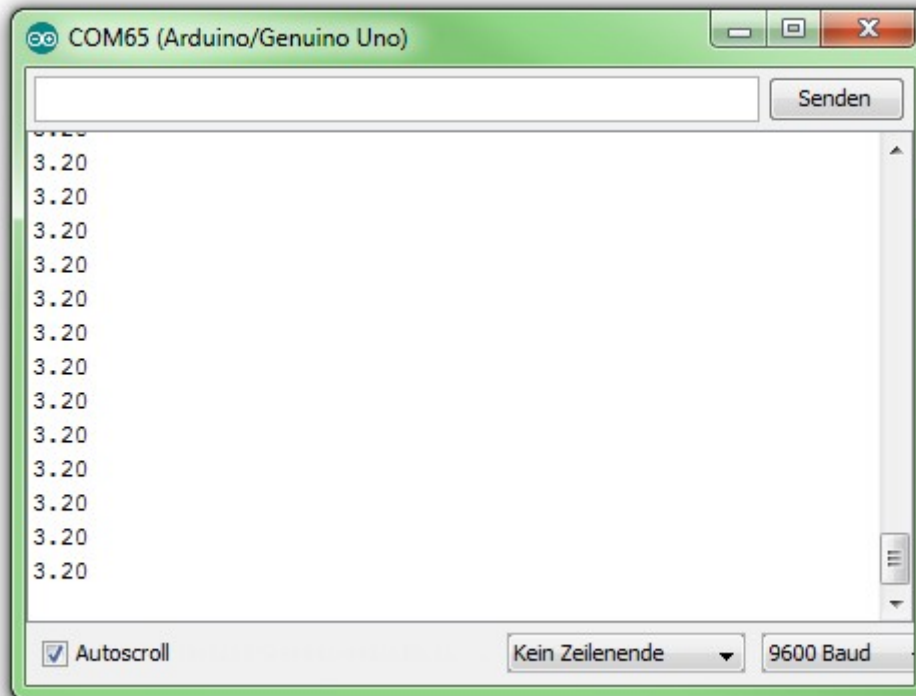
```
int wert1;
float wert2;

void setup()
{
  Serial.begin(9600); //serielle Verbindung starten
}

void loop()
{
  float temp;
  wert1=analogRead(1); //Spannungswert am analogen Eingang 1 auslesen
  temp=wert1/4.092; //Wert mathematisch umwandeln um den Spannungswert in Volt zu erhalten
  wert1=(int)temp;
  wert2=((wert1%100)/10.0);
  Serial.println(wert2); //Endgültigen Spannungswert im seriellen Monitor anzeigen
  delay(1000); //Eine Sekunde warten
}
```

Öffnet man nach erfolgreichem Hochladen des Codes nun den seriellen Monitor, sieht man nun im Sekundentakt einen Spannungswert. Da noch keine Spannungsquelle angeschlossen ist, sollte diese bei 0,00 liegen.

Um zu sehen ob alles funktioniert, kann man testweise den GND Kontakt vom Sensor mit dem GND Kontakt vom Arduino und den VCC Kontakt vom Sensor mit dem 3,3V Kontakt vom Arduino verbinden. Nun sollte ein Wert um 3,3V auf dem seriellen Monitor auftauchen. Der Sensor ist nicht ganz exakt. So waren es bei uns 3,2V.



Vierstellige 7 Segment Anzeige



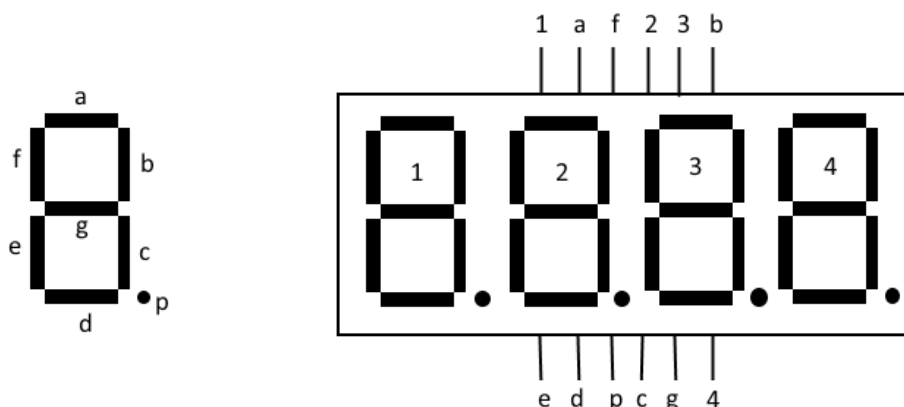
Aufgabe: Wir wollen eine beliebige Zahl auf dem vierstelligen 7 Segment Display anzeigen lassen.

Material: Arduino/ Kabel/ 4 Stelliges 7-Segment Display /1K Ohm Widerstände

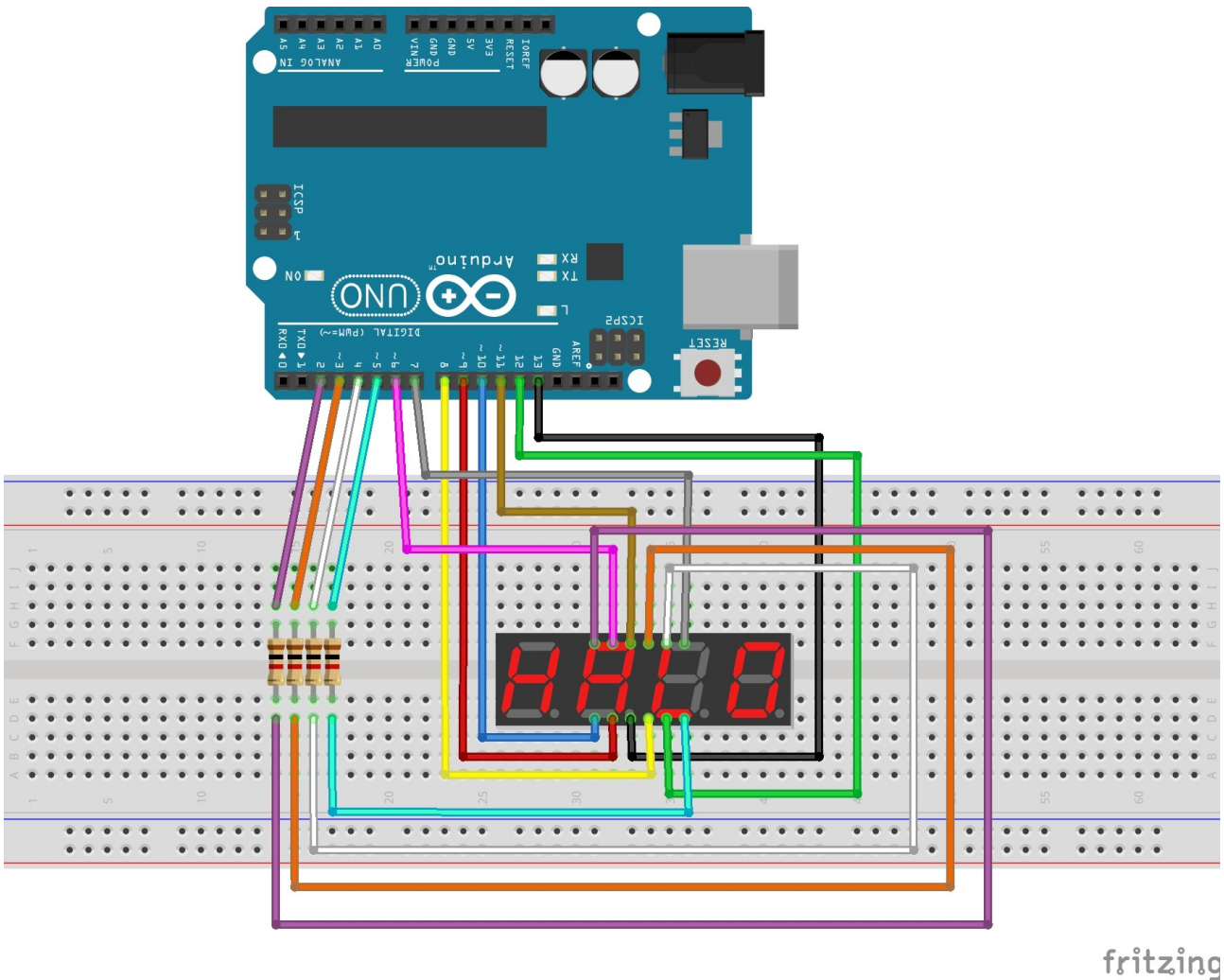
Das 7-Segment Display besitzt insgesamt 12 Kontakte auf der Rückseite, sechs oben und sechs unten. Vier dieser Kontakte gehören jeweils zu einer Ziffer. Diese können entweder von der Sorte „Common Cathode“ oder „Common Anode“ sein. Welche Art von Display man hat kann man mit dem Code einfach austesten, aber dazu später mehr. Die anderen acht Kontakte gehören jeweils zu einem Segment und zu dem Punkt neben einer Ziffer.

Bei 7-Segment Displays mit nur einer oder zwei Ziffern werden die Segmente jeder Ziffer einzeln angesteuert. Da das bei vier Ziffern aber ein noch größeres Kabeldurcheinander wäre als es schon ist, funktionieren diese Displays mit „Multiplexing“. Das bedeutet wenn beispielsweise alle vier Ziffern gleichzeitig angesteuert werden sollen, werden diese extrem schnell hintereinander angesteuert. Dies geschieht so schnell, dass es für das menschliche Auge aussieht als würden alle vier Ziffern gleichzeitig angezeigt werden.

Anschauliche Erklärung der Kontakte:



Aufbau:



Programmierung:

Um ein 7-Segment Display ohne endlos langen Code zu programmieren benötigt man eine Library, die noch nicht in der Arduino Software installiert ist. Diese „SevenSeg“ Library von Dean Reading kann hier heruntergeladen werden:

<https://github.com/DeanIsMe/SevSeg> . Die Library muss dann, wie schon aus vorherigen Anleitungen bekannt, zu Arduino Software hinzugefügt werden.

Dies geht am leichtesten in der Arduino Software unter: Sketch > Include Library > Add .ZIP Library..

Kommen wir zum Code:

```
#include "SevSeg.h" //Die vorher hinzugefügte Library laden
SevSeg sevseg; //Ein sieben Segment Objekt initialisieren

void setup() {

  byte numDigits = 4; //Hier wird die Anzahl der Ziffern angegeben

  byte digitPins[] = {2, 3, 4, 5}; //Die Pins zu den Ziffern werden festgelegt

  byte segmentPins[] = {6, 7, 8, 9, 10, 11, 12, 13}; //Die Pins zu den
//Segmenten werden festgelegt

  sevseg.begin(COMMON_CATHODE, numDigits, digitPins, segmentPins); //In diesem
//Abschnitt kann man nun entweder testen welche Art von Display man besitzt oder
//wenn man es schon weiß angeben ob es sich um ein COMMON_CATHODE oder
//COMMON_ANODE Display handelt. Das Display funktioniert nur wenn die richtige
//Art eingetragen ist, ansonsten werden alle Segmente gleichzeitig leuchten.

}

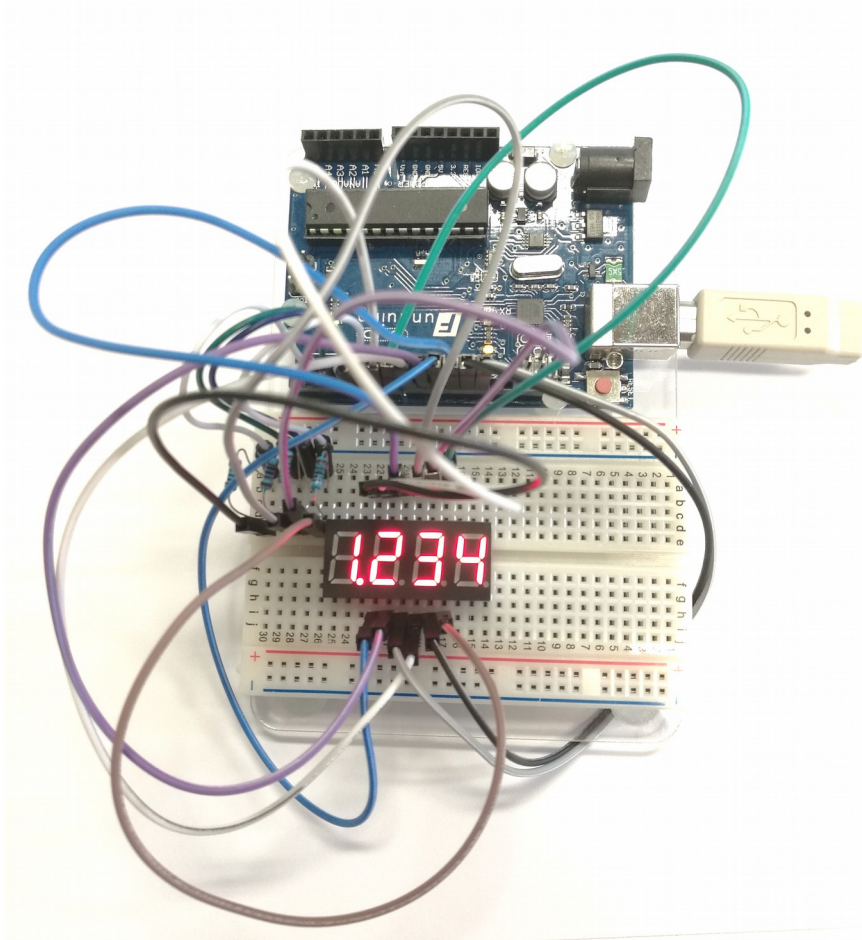
void loop() {

  sevseg.setNumber(1234,3); //Hier können wir nun die gewünschte Zahl eintragen.
//Wir haben als Beispiel 1234 angegeben. Die Zahl hinter dem Komma steht für den
//Punkt hinter einer Ziffer. Hierbei ist 3 der Punkt neben der ersten Ziffer und
//0 wäre der Punkt ganz rechts neben der letzten Ziffer. Wenn man keinen Punkt
//mit angezeigt haben möchte kann man z.B. 4 angeben.

  sevseg.refreshDisplay(); // Dieser Teil lässt die Nummer auf dem Display
//erscheinen.

  sevseg.setBrightness(90); //Hier kann die Helligkeit des Displays angepasst
//werden. In einem Bereich von 0-100 wobei 100 das Hellste ist. 0 bedeutet
//jedoch nicht dass das Display komplett dunkel ist. Für die Anzeige einer Zahl
//ist allein die "sevseg.refreshDisplay();" Zeile verantwortlich

}
```



In der sevseg library sind zudem noch interessante Beispiel Codes vorhanden. Diese können in der Arduino Software unter: Datei > Beispiele > SevSeg-master aufgerufen werden.