ARDUINO

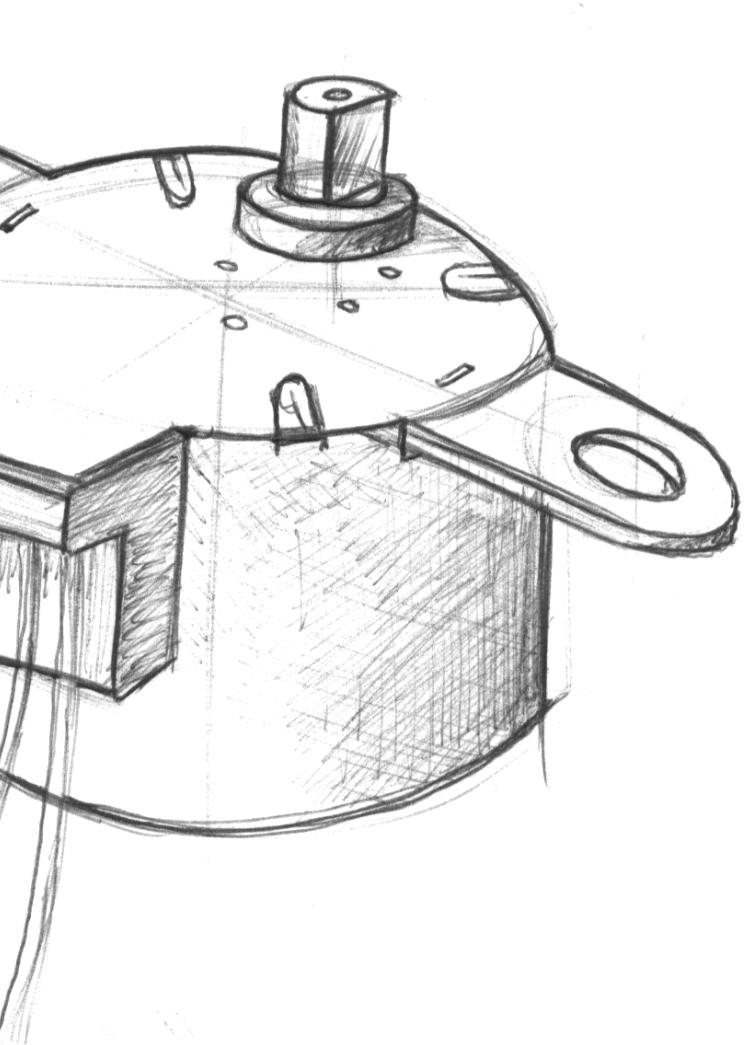
FÜR PRODUKTDESIGNER

INHALTSVERZEICHNIS

Autor Sascha Greilinger
Produktdesign-Student
Hochschule Coburg

sascha.greilinger@stud.hs-coburg.de

Version 1.6 5/2017



Diese Anleitung soll...

Produktdesign-Studenten den Einstieg in die Arduino-Welt ermöglichen

Die Grundlagen ausführlich, aber leicht verständlich beschreiben und darstellen

Ein Nachschlagewerk für die wichtigsten C-Befehle sein und diese mit Beispielen verständlich machen

Hilfe zu häufig auftretetenden Problemen liefern

Eine Übersicht über einsetzbare elektronische Bauteile liefern

Für Produktdesigner interessante elektronische Formeln erklären

Ziel dieses Kurses ist es...

Möglichkeiten kennenlernen - mit Mikrocontrollern in Bezug auf kreative Projekte

Verständnis aufbauen für einen Bereich von Ingenieuren, Informatikern und Technikern

Berührungsängste abbauen zu Programmierung und Elektronik

Interesse wecken, Neugierig machen

Grundlagen des Programmierens kennen

Anwendung von Elektronik vertiefen

Vorteile für Produkt-Designer:

Darstellung elektronischer Features in Mockups, Modellen und Prototypen

Usability testen (sinnvolle Bedienabläufe)

Intuitive Bedienung entwickeln (Einfachheit der Bedienung)

Ergonomie mit funktionierenden Modellen testen (Anordnung und Form von Taster, Anzeigen, Aktoren)

Gesamteindruck von elektronischen Produkten darstellen

Auch elektronische Funktionen von Designern gestaltbar

Flexible Änderungen des Programms, Unabhängigkeit von vorhandener Hardware

Komplexere Funktionen auch ohne tiefere Elektronik-Kenntnisse realisierbar

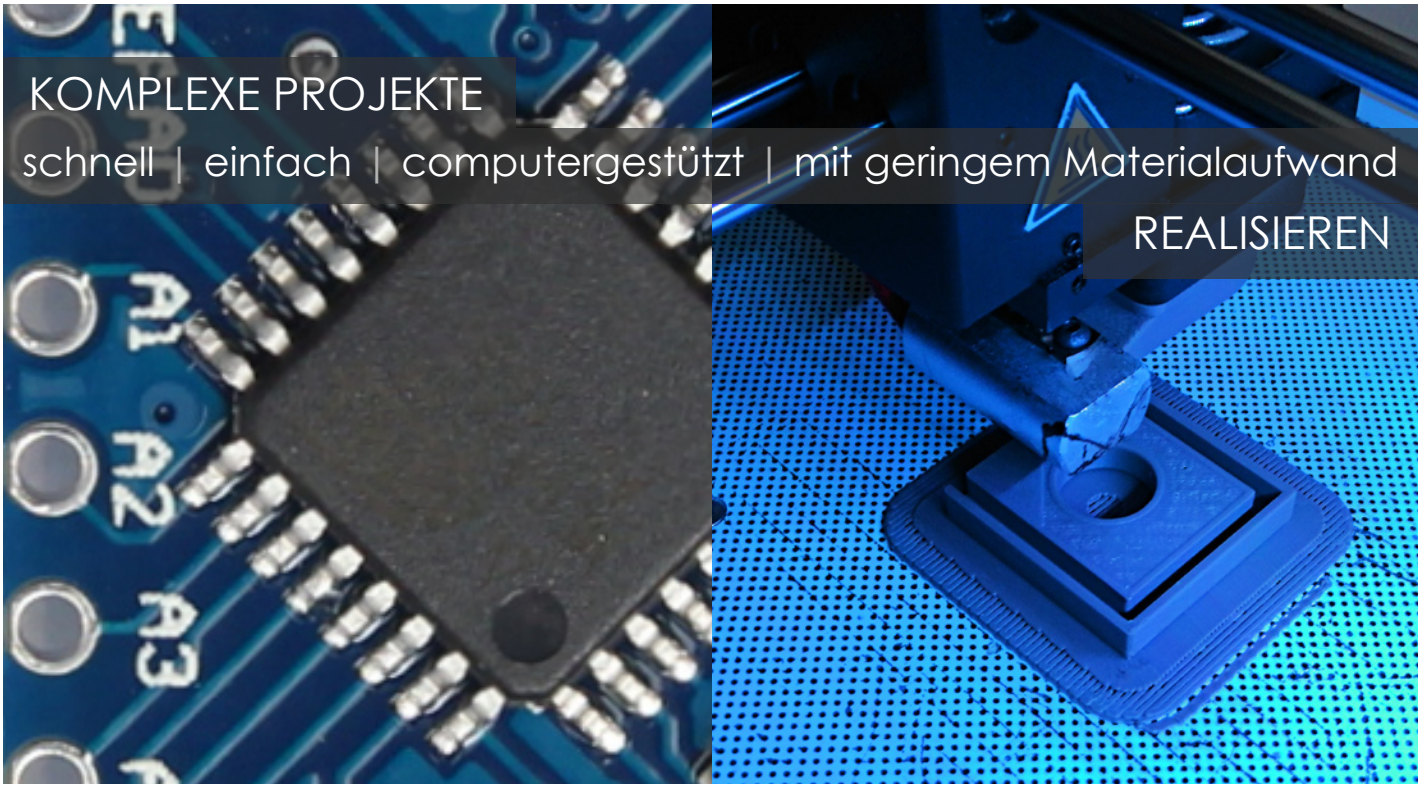
Mikrokontroller sind auch für andere Projekte wiederverwendbar

Entwickelte Programme können ggf. auch für die Fertigung der Produkte weiterverwendet werden

IT‘S NOT(ONLY) A NERD THING,

IT‘S A PRODUCT DESIGN TOOL!

Mikrocontroller sind für die Elektronik das, was 3D-Drucker für die Konstruktion und den Modellbau sind...



Es handelt sich um Chips, sog. **„ICs“ (integrated circuit)** in denen ein **miniaturisierter Computer** untergebracht ist. Das **Funktionsprinzip** ist dasselbe wie bei PCs, Laptops, Tablets oder Smartphones: Ein **Prozessor** lädt **Programme** von einem **Festspeicher** in einen schnellen **Arbeitsspeicher**, führt diese Programme aus und liest und schreibt Daten auf die beiden Speicher oder nutzt **Schnittstellen** zur Kommunikation mit anderen Geräten.

Mikrocontroller **(umgangssprachliche Abk. „µC“)** sind auf eine gewisse Anzahl **digitaler oder analoger Aus- und Eingänge** beschränkt und ihr Betriebssystem ist auf einfache Steuerfunktionen reduziert. Ihre Rechenleistung beträgt nur Millionstel der Leistung von modernen Computern.

Sie sind jedoch in der Lage, mit übergeordneten Systemen über **Schnittstellen** (z.B. WiFi, BT, LAN, CAN-Bus, Gebäudeautomationsnetze) Daten auszutauschen, was ihnen in einer immer vernetzteren Welt einen hohen Stellenwert einbringt.

In der Welt des **IoT (Internet of Things)** spielen sie eine zentrale Rolle. Sie können über **Sensoren** benötigte Informationen aufnehmen und in lokalen Netzwerken oder dem Internet bereitstellen, **Aktoren** steuern oder beides. Dabei können sie aber auch „offline“ Steuerungsaufgaben übernehmen und eine eigene, dezentrale „Intelligenz“ besitzen.

Mikrocontroller sind also ein mächtiges Werkzeug:

Flexibel, da das Programm anpassbar und erweiterbar ist.

Einfach, da wenig zusätzliche Elektronik für das Realisieren komplexer Aufgaben notwendig ist.

Vernetzbar mit weit verbreiteten Bussystemen.

Platzsparend, da ein wenige Millimeter großer Chip umfangreiche Schaltungen ersetzen kann.

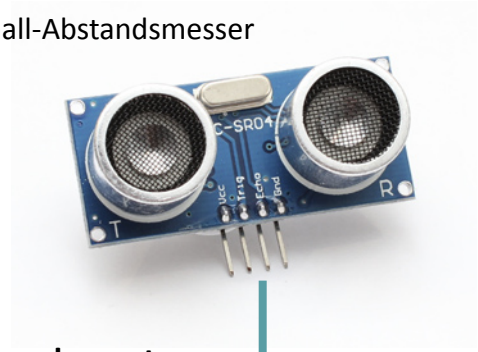
Mobil einsetzbar, da geringe Stromaufnahme.

Fertigungsnah, da in vielen industriell hergestellten, elektronischen Produkten auch Mikrocontroller verbaut sind.

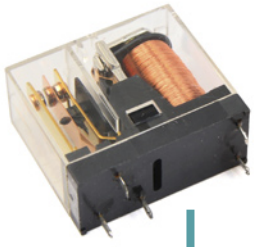
Günstig, da massenproduziertes Bauteil.

An einen Mikrocontroller können viele elektronische Bauteile direkt oder mit geringem Schaltungsaufwand angeschlossen werden:

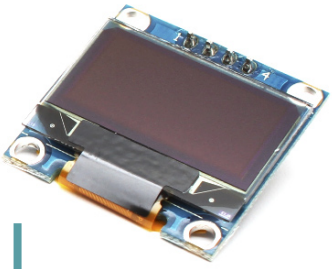
Sensoren
z.B. Ultraschall-Abstandsmesser



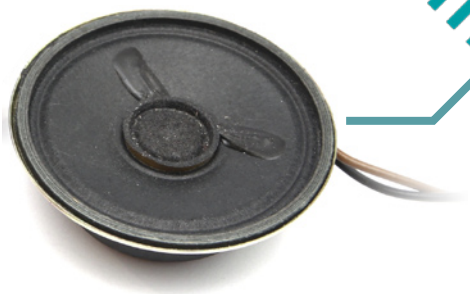
Leistungselektronik/Relais
zum Schalten großer Ströme und Spannungen



Digitale Bedienelemente
Taster, Schalter



Displays
Benutzerinterface, Anzeige von Werten

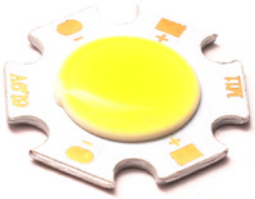


Elektroakustische Bauteile
Lautsprecher, Piezo-Signalgeber, Mikrofone

Leuchtdioden (LEDs)
Signalleuchten, Anzeigen

Analoge Bedienelemente
Dreh- und Schiebepotentiometer

Schnittstellenmodule
WiFi, BT, LAN, 2,4 GHz Funk
Web-Interface, App-Steuerung



Konstantstromquellen
Dimmen von Hochleistungs-LEDs

ARDUINO-BOARDS

Sogenannte **Entwicklungsboards** machen es den AnwenderInnen einfach ihre Projekte mit Mikrocontrollern umzusetzen, indem alle für den grundlegenden Betrieb notwendige Bauteile bereits auf einer Platine aufgebracht sind. Die Boards können entweder über Lötpunkte direkt mit Leitungen verbunden werden oder besitzen Stifte bzw. Buchsen zum Verbinden mit Steckboards oder Steckleitungen.

Nachfolgende Informationen beziehen sich speziell auf das **Arduino „Nano“** Board, andere Entwicklungsboards sind aber sehr ähnlich aufgebaut.

Programmiert werden kann das Board mit einem **USB-Anschluss**, der über einen Schnittstellen-Adapter mit der UART-Schnittstelle des Mikrocontrollers verbunden ist.

Der **Resettaster** dient zum manuellen Rebooten des Microcontrollers. Dies kann auch über den Pin „RST“ erfolgen.

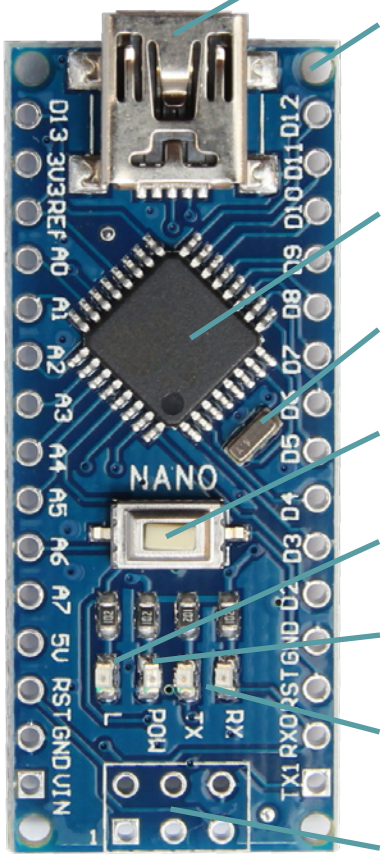
Pin 13 ist zudem mit einer **LED auf dem Board** verbunden und kann als Statusanzeige oder für einfache Programmierübungen genutzt werden.

Drei weitere **LEDs** zeigen **Spannungsversorgung**, sowie das **Senden und Empfangen** von Daten über die serielle Schnittstelle an.

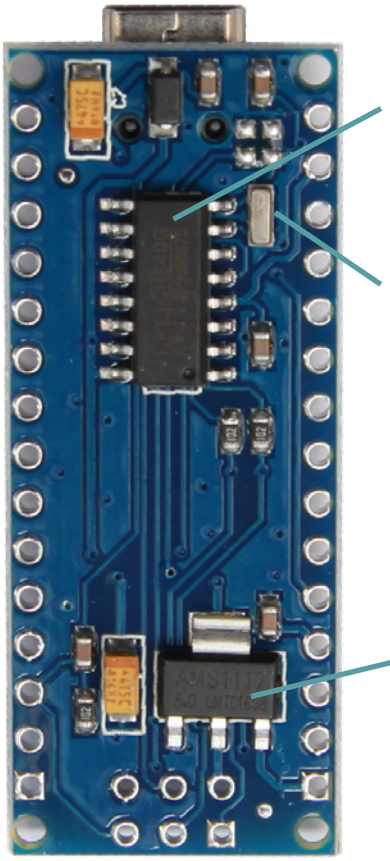
Die **SPI-Schnittstelle** dient zum direkten Programmieren des Controllers ohne Bootloader mit einem speziellen Programmiergerät und ist für unsere Zwecke vernachlässigbar.

Ein **Spannungsregler** auf der Rückseite stellt eine stabile Spannung für das Board und angeschlossene Bauteile her. Zudem steht eine **3,3 Volt** Quelle sowie eine konfigurierbare **Referenzspannung** für die Analogeingänge zur Verfügung.

Vorderseite



Rückseite



Technische Daten:

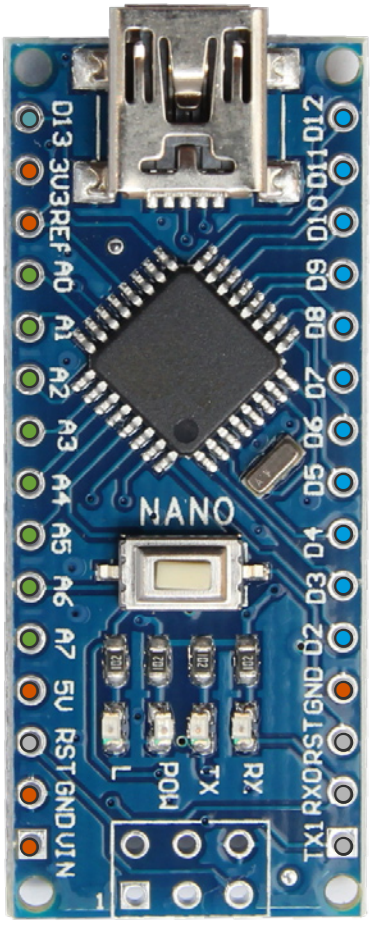
Controller	ATmega328
Spannung	direkt: 5V DC
	über Spannungsregler: 7-12V DC
Stromaufnahme	19 mA
Flash-Speicher (Programmspeicher)	32 kByte
RAM (Arbeitsspeicher)	2 kByte
EEPROM (nichtflüchtiger Speicher)	1 kByte

Taktrate	16 MHz
Digitale IO-Pins	12
...davon PWM-fähig	6
Belastbarkeit der Ausgänge	max. 40 mA
Analogeingänge	8
Auflösung Analogeingänge	10 Bit
Abmessungen:	18 x 45 mm

Pinout-Übersicht:

- Digitale IO Pins
- Analogeingänge
- Spannungsversorgung
- Sonstiges

Digitaler IO Pin D13	D12	Digitaler IO Pin
3,3 Volt Spannungsquelle	D11	(PWM)
Referenzspannung für Analogeingänge	D10	(PWM)
Analogeingang A0	D9	(PWM)
A1	D8	
A2	D7	
A3	D6	(PWM)
A4	D5	(PWM)
A5	D4	
A6	D3	(PWM)
A7	D2	
5 Volt Spannungsquelle	Minus	
Reset	Reset	
Minus	Schnittstelle	
Anschluss Spannungsversorgung 7-12V	Schnittstelle	



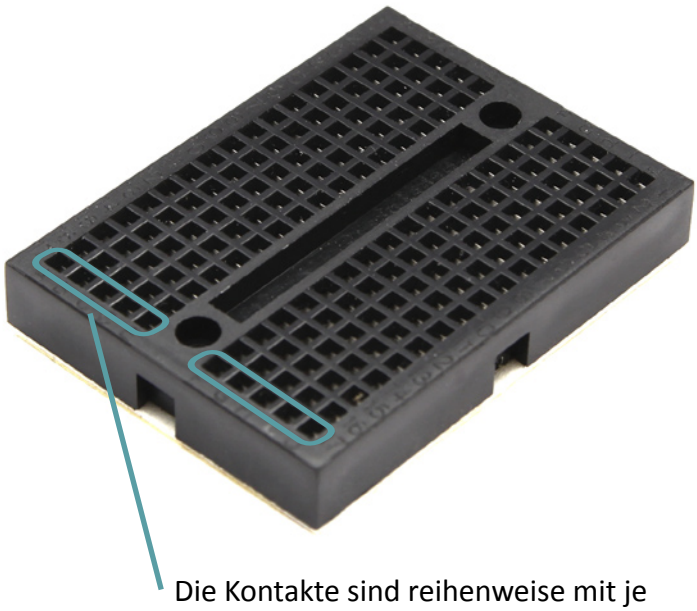
ARBEITEN MIT STECKBOARDS

Steckboards dienen in der Elektronik zum **schnellen Aufbau von Schaltungen** zu Test- und Entwicklungszwecken. Fast alle Bauteile können darauf eingesteckt und durch Steckbrücken oder einfache Drähte miteinander verbunden werden.

Unter dem Kunststoffgehäuse mit Lochraster befinden sich Kämme aus Blech, die eingesteckte Drähte mit einem Federmechanismus festhalten und den elektrischen Kontakt herstellen. Nebeneinanderliegende Löcher **(Reihen)** sind gebrückt, übereinanderliegende **(Spalten)** isoliert.

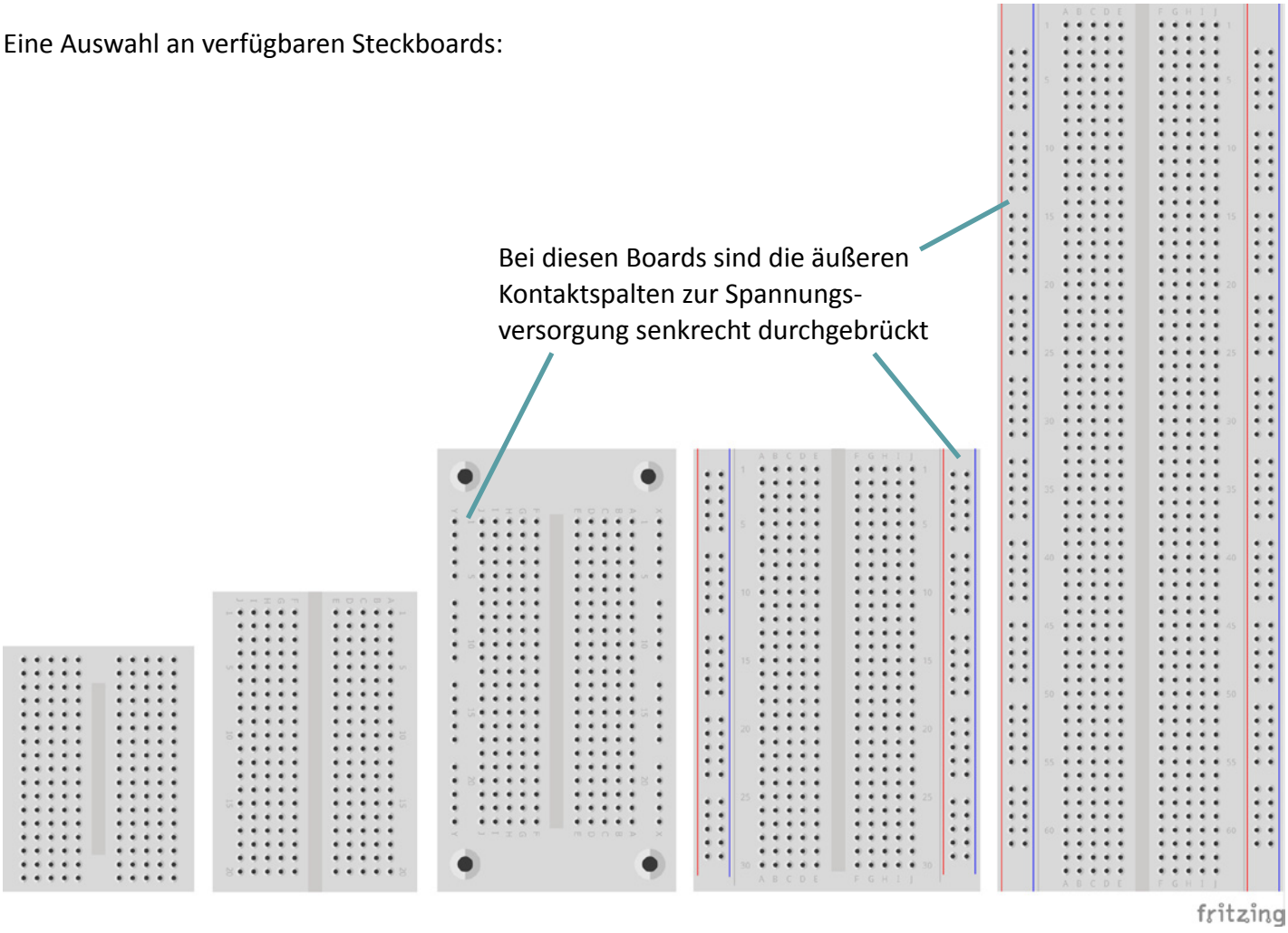
Die Boards gibt es in verschiedenen Größen und Ausführungen. Manche besitzen auch senkrecht verlaufende **Kontaktbahnen** für die **Spannungsversorgung**, welche farblich markiert sind.

Viele Steckboards sind untereinander **erweiterbar** oder mit **Montagelöchern** und **Klebeflächen** auf der Rückseite versehen.



Die Kontakte sind reihenweise mit je 5 Löchern auf jeder Hälfte gebrückt

Eine Auswahl an verfügbaren Steckboards:



Der Arduino Nano ist speziell für den Einsatz auf Steckboards konzipiert. Mit angelöteten **Stiftleisten** an den Pins kann er direkt auf die Boards gesteckt und seitlich kontaktiert werden.

Leitungsfarben

Um sich im Kabelsalat auf den Steckboards besser zurechtzufinden, macht es Sinn, verschiedenfarbige Drähte zu verwenden, denen bestimmte Potentiale bzw. Aufgaben zugeordnet sind:

	Rot	Plus
	Blau	Minus
	Grau	Steuer- und Messsignale

Achtung!

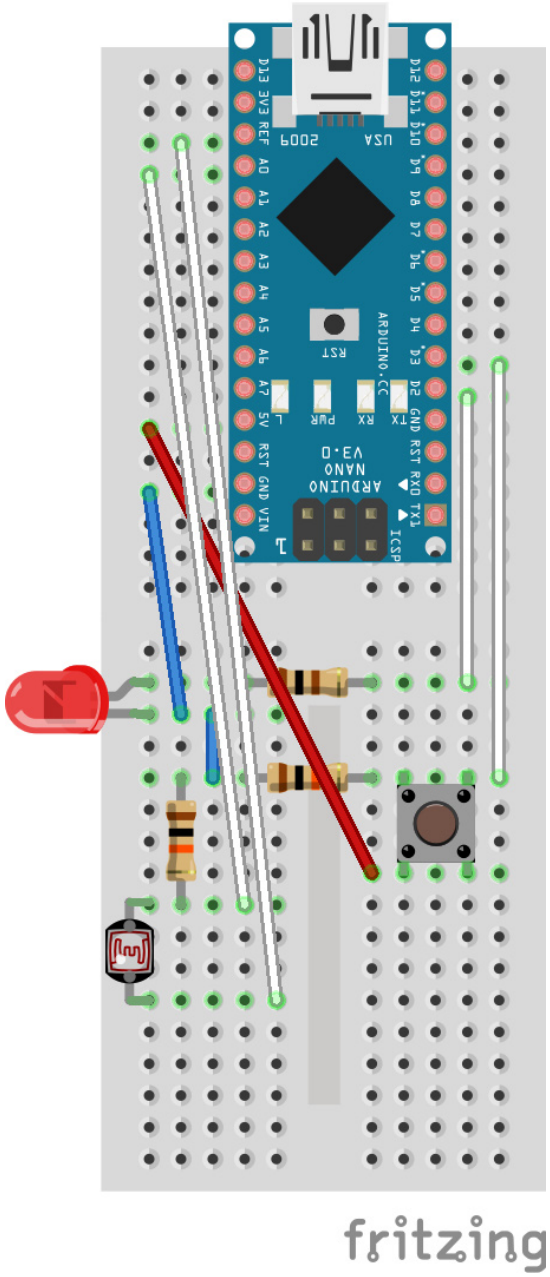
Kurzschlüsse zwischen Plus und Minus können im Batteriebetrieb zu hohen Strömen, starker Erwärmung und Zerstörung der Bauteile sowie zu Explosion der Batterien führen. Bei Nutzung des USB-Ports als Spannungsquelle kann sich der angeschlossene Computer abschalten oder schlimmstenfalls Schäden davontragen.

Vorsichtsmaßnahmen:

- Nicht unter Spannung verdrahten!
- Aufbau vor Inbetriebnahme überprüfen!
- Aderfarben beachten!
- Drähte mit Plus und Minus nicht direkt nebeneinander stecken!

Bei der Positionierung der Bauteile ist man relativ frei, man sollte jedoch auf eine **platzsparende Anordnung** achten, die keine zu langen Drahtstücke erfordert. Die Bauteile müssen immer so gesteckt sein, dass ihre Kontakte mit Drähten erreichbar sind und nicht von der Brückung der Steckboard-Kontakte kurzgeschlossen werden.

Taster und Potentiometer sollten gut bedienbar sein. Sensoren sollten nicht durch Drähte oder



Aufbau zur Programmierung eines Dämmerungsschalters mit Taster, LED und Fotowiderstand (LDR).

Aktoren gestört werden. Ein Helligkeitssensor sollte beispielsweise nicht direkt neben einer LED platziert werden.

Es ist einfacher, zuerst die Bauteile einzustecken und diese danach erst untereinander und mit dem Mikrocontroller zu verbinden.

Die Board-Pläne wurden mit der Freeware **„Fritzing“** erstellt, welche die Arbeit mit den Steckboards sehr vereinfacht. Aus den Board-Plänen können damit auch direkt Schaltpläne und Platinenlayouts erstellt werden.

Aufgrund der übersichtlicheren Darstellung wird in diesem Dokument der elektronische Schaltplan bevorzugt verwendet.

Die Programmiersoftware, auch **IDE (Integrated Development Environment)** genannt, wurde möglichst übersichtlich und einfach aufgebaut, so dass sich auch Menschen **ohne Programmierkenntnisse** schnell zurecht finden. Kompliziertes Konfigurieren und Anpassen des Codes an den Microcontroller werden durch ein auf dem Chip vorinstalliertes „Betriebssystem“, dem sogenannten **Bootloader**, umgangen. Da die Boards **standardisiert** und in der IDE vorkonfiguriert sind, ist die Inbetriebnahme und das Übertragen des Programms mit wenigen Klicks möglich. Außerdem werden zahlreiche interessante und gut dokumentierte **Beispielprogramme** mitgeliefert, die als Vorlage für eigene Projekte verwendet werden können. Nutzer der Software können also sofort loslegen und sich ganz und gar auf das Programmieren konzentrieren.

Programmiert wird in **C**, bzw. **C++**, einer Sprache, die relativ **einfach zu erlernen** und eine gute Basis für das Erlernen weiterer Programmiersprachen ist.

Ein weiterer Punkt, der Arduino bei Einsteigern so beliebt macht, sind die vielfältigen Möglichkeiten sich die Programmiersprache **selbst beizubringen** und auf Hilfe und Dokumentationen einer riesigen, **weltweiten Community** zurückgreifen zu können. Oft genügt eine kurze Websuche und man findet Beispiele zur Lösung seines Problems.

Es gibt selten etwas, das noch nicht jemand davor schon in einer ähnlichen Weise umgesetzt hat. Man muss also nur nach **vergleichbaren Projekten** suchen.

Diese Seiten sollte jeder Arduino-Nutzer kennen:

www.arduino.cc

Download-Quelle, Projektvorstellungen, News

Reference
Die Code-Referenz für C und spezielle Arduino-Befehle. Quasi das Wörterbuch und die Rechtschreibung zum Programmieren.

Forum
Man muss nicht angemeldet sein, um die Forenbeiträge sehen zu können und oft reicht auch schon eine gezielte Suche (auf Englisch) um die Lösung für sein Projekt zu finden.

Playground
Eine Arduino-Wikipedia, gefüllt mit Wissen zahlreicher erfahrener Programmierer.

www.funduino.de

Eine gute deutschsprachige Seite mit vielen nützlichen Anleitungen für Einsteiger.

www.youtube.de

Wer sich lieber mit Videos helfen lassen will, der kann einfach auf YouTube nach ähnlichen Projekten suchen. Außerdem findet man dort viele interessante Projekte mit Mikrocontrollern, von denen man sich inspirieren lassen kann.

www.fritzing.org

Fritzing ist eine freie Layoutsoftware zum einfachen Erstellen von Steckboard-Plänen, Schaltplänen und Platinen-Layouts.

Die Freeware beinhaltet auch eine Bauteil-Datenbank mit vielen Entwicklerboards, Erweiterungen und häufig genutzten elektronischen Bauteilen.

www.embrio.io

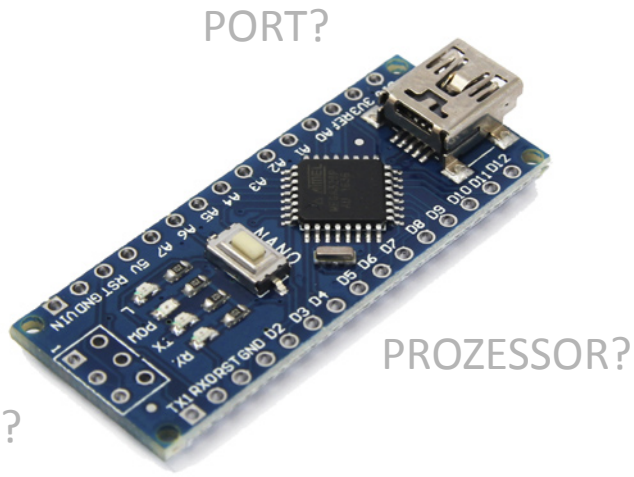
Eine grafische Echtzeit-Programmierumgebung für Arduino. Empfehlenswert für alle, die mit Code-Programmierung Probleme haben.

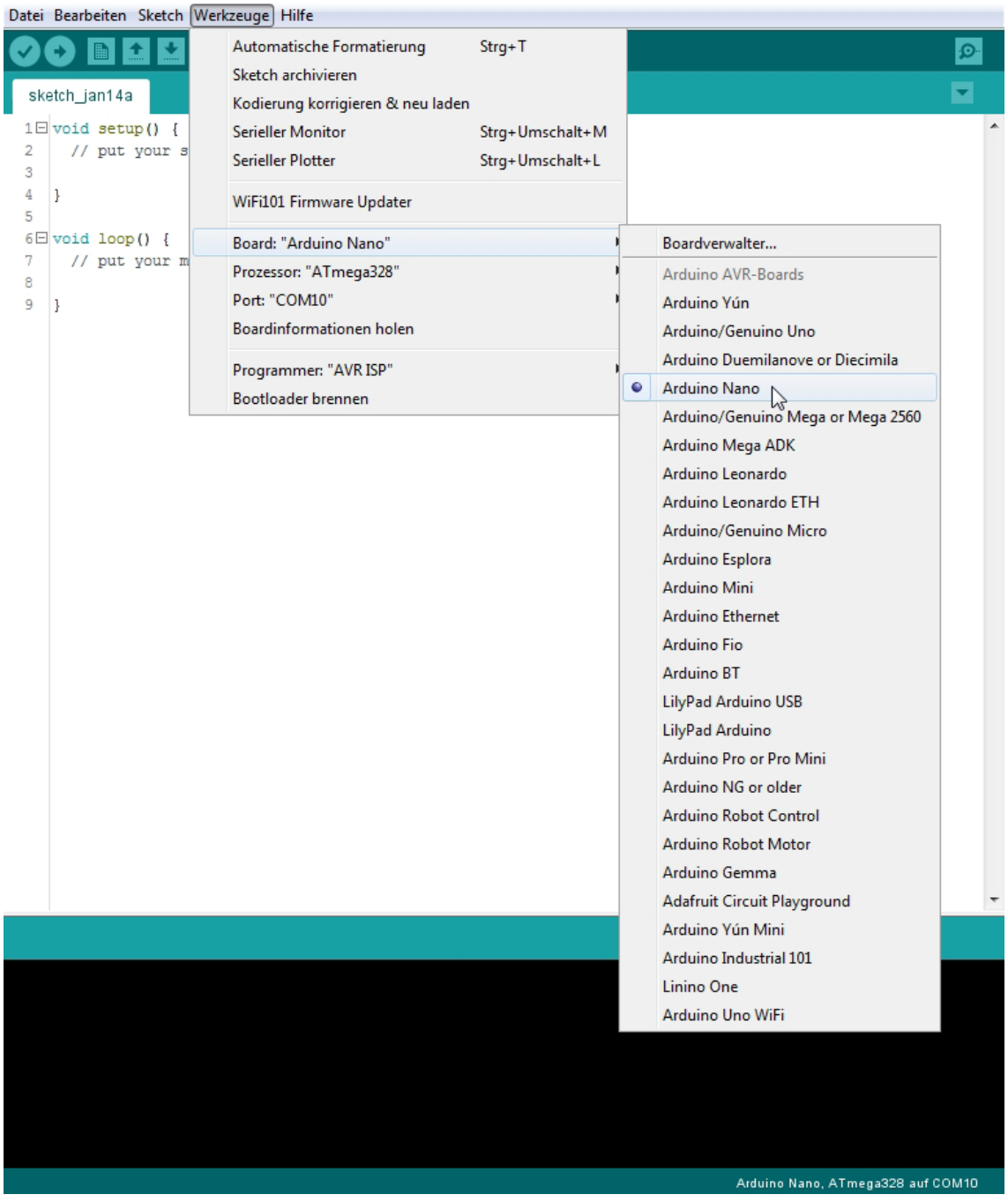
www.elektronik-kompendium.de

Hier findet man leicht verständliche Erklärungen und Beispiele rund um die Elektronik.

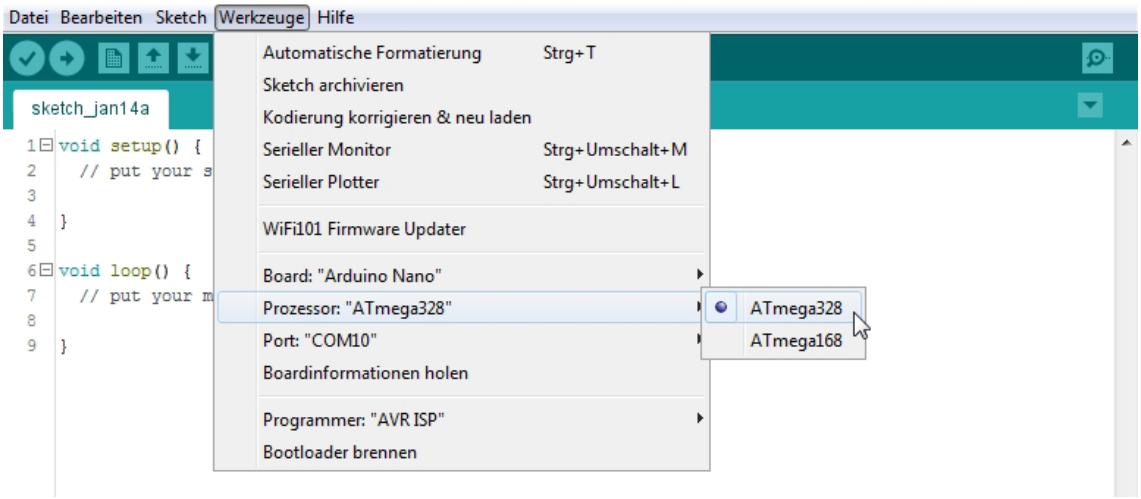
EINRICHTEN DES BOARDS

Eines der wenigen Dinge, die bei Arduino konfiguriert werden müssen, sind die Einstellungen zur verwendeten Hardware, also das Board, der Mikrocontroller und die verwendete Schnittstelle zur Übertragung des Programms. Dies ist aber schnell erledigt, hier am Beispiel eines Arduino Nano:

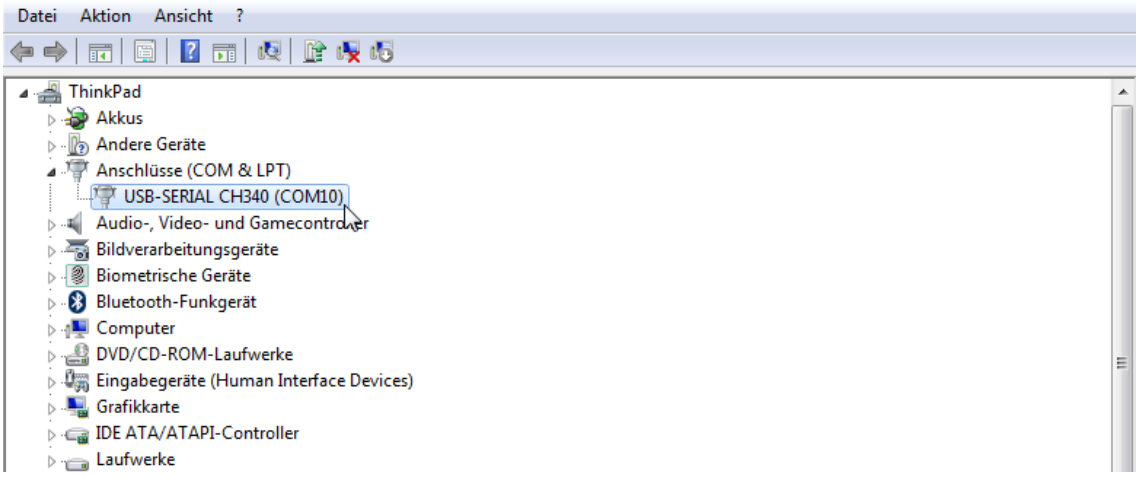




Schritt 1: Auswahl des angeschlossenen Boards

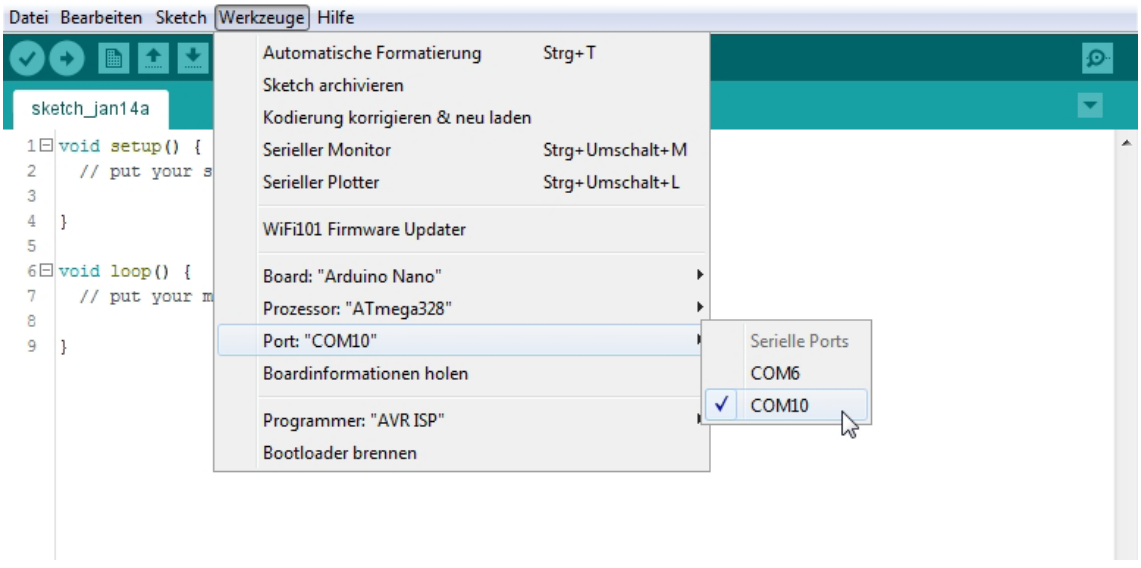


Schritt 2: Auswahl des verwendeten Mikrocontrollers



Schritt 3: Herausfinden des richtigen COM-Ports

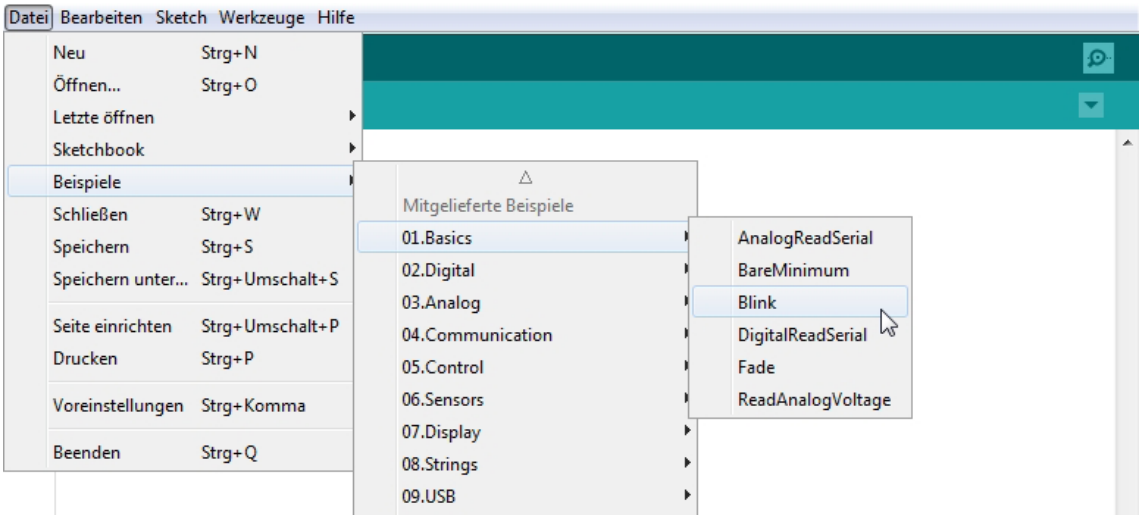
Auf der Unterseite der Platine des Arduino Nano befindet sich ein **Schnittstellenwandler-Chip**, der die Kommunikation zwischen Computer und Mikrocontroller über USB herstellt. Im **Gerätemanager (Windows)** wird dieser Chip als USB-Adapter für serielle Schnittstellen erkannt. Dahinter wird die **Port-Adresse** des Adapters angezeigt, hier COM10.



Schritt 4: Auswahl der richtigen Port-Adresse. Fertig!

Jetzt gehts los! Nachdem die Verbindung zum Mikrocontroller konfiguriert wurde, kann ein **Beispielprogramm** hochgeladen werden.

Auf dem Nano-Board befindet sich eine kleine LED, die am IO-Pin 13 des µC angeschlossen ist und angesteuert werden kann. Unter den Beispielprogrammen befindet sich ein Sketch, welcher diese LED blinken lässt. Dazu ist keine weitere Verdrahtung notwendig.



Schritt 1: Öffnen des Beispielprogramms „Blink“

Schritt 2: Kompilieren und Hochladen des Programms
Wenn alles funktioniert hat, erscheint unten „Hochladen abgeschlossen“ und nach einem kurzen, selbständigen Reboot des µC sollte die LED auf dem Board blinken.

Bevor mit dem Programmieren begonnen werden kann, sollten einige grundsätzliche Begriffe bekannt sein:

Firmware

Damit bezeichnet man die Gesamtheit eines Programms auf einem Mikrocontroller. Es besteht im Normalfall aus drei Komponenten:

Bootloader

Ein - in unserem Fall vorinstalliertes - Programm, welches die Kommunikation mit einem Computer über die UART- oder Ethernet-Schnittstelle ermöglicht und die Installation der hochgeladenen Firmware in den Flash-Speicher übernimmt.

Betriebssystem (OS)

Ähnlich wie bei einem PC sorgt das µC-Betriebssystem für das Ausführen von Prozessen, die Verwaltung des Speichers und das Bereitstellen von Diensten für aufgespielte Programme.

Applikationssoftware (App)

Dieser Firmwareteil ist das eigentliche Programm, welches von Arduino-ProgrammiererInnen hochgeladen wird.

Bibliotheken (library, lib)

Sie enthalten vorgefertigte Unterprogramme oder abrufbaren Programmcode, der normalerweise nicht standartmäßig in einer Programmiersprache vorhanden ist, wie z.B. höhere mathematische Funktionen, Textverarbeitung oder Ansteuerung von Bus-Schnittstellen.

Die verwendeten Bibliotheken werden in C am Anfang des Programmcodes definiert. Die Arduino-Programmierumgebung bindet die meisten Bibliotheken aber selbständig ein.

Compiler

Compiler übersetzen eine Programmiersprache wie C in einen von Computern lesbaren Maschinencode. Dieser Maschinencode ist für einen Prozessor zwar direkt verarbeitbar, für ProgrammiererInnen ist dieser Code aber sehr unhandlich.

Der Übersetzungsvorgang wird als „Kompilieren“ bezeichnet. Beim Kompilieren wird auch der Programmcode auf Fehler überprüft. Ebenso werden verwendete Bibliotheken in den Maschinencode integriert.

Synthax

Sie ist die Rechtschreibung einer Programmiersprache und beschreibt welche Zeichen wie wo verwendet werden dürfen, z.B. die Klammern nach einem Befehl oder das Semikolon (;) bei Befehlsende.

Kommentare

Kommentare dienen dazu, dem Code Erklärungen und Beschriftungen hinzuzufügen. Sie werden beim Kompilieren ignoriert und haben damit keine Auswirkungen auf das Programm auf dem Mikrokontroller.

Man kann damit auch sehr gut Programmcode gezielt unwirksam machen um z.B. eine Funktion schnell zu deaktivieren.

Kommentare in C können wie folgt gesetzt werden:

```
// Einzeiliger Kommentar

/* mehrzeiliger Kommentar mit Anfang und

Endmarkierung */
```

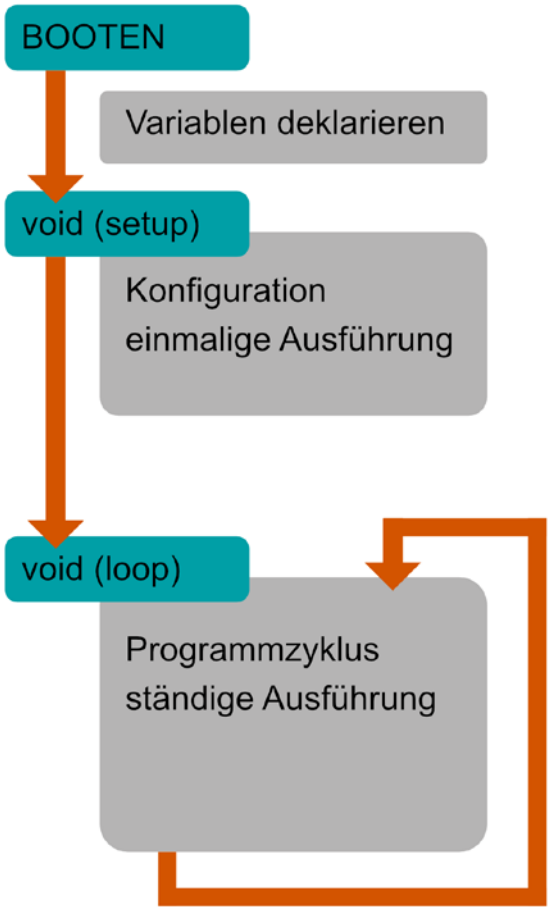
AUFBAU EINES PROGRAMMS

Nachdem der µC hochgefahren (gebootet) ist, arbeitet der Prozessor das Programm ab.

Im oberen Teil des Programms werden die **Variablen deklariert**, die für die Ausführung notwendig sind.

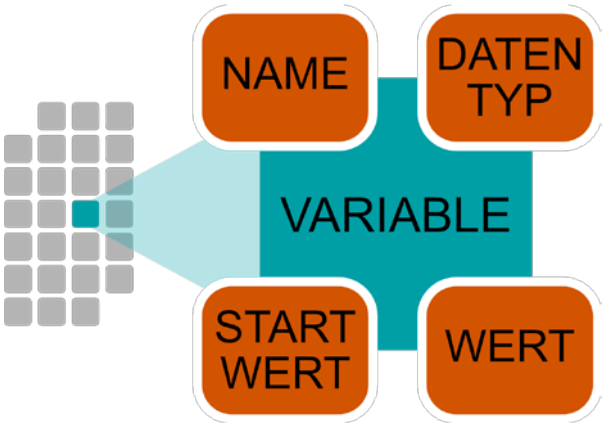
Danach erfolgt der **einmalige Durchlauf der Setup-Schleife**. Hier werden Einstellungen und alle Aktionen durchgeführt, die vor dem eigentlichen Programm abgearbeitet werden müssen.

Nach dem Durchlauf der Setup-Schleife erfolgt die **ständige Ausführung der Programm-Schleife**. Innerhalb eines Programmzyklus werden alle Befehle innerhalb der Schleife ausgeführt. Nach der Abarbeitung der Schleife startet die Programm-Schleife erneut.



VARIABLEN

Unter einer Variable versteht man eine Speicherzelle mit einem **Namen**, einem **Datentyp** und ggf. einem **Startwert**. Der Datentyp der Variable bestimmt, wie viel Speicher sie auf dem Mikrocontroller benötigt und wozu die Variable verwendet werden kann.



Die wichtigsten Datentypen

Bezeichnung	Inhalt	Speicher	Verwendungsbeispiel
bool	Wahrheitswert true/false	1 Bit	Abfrage Taster
char	Einzelnes Zeichen/Ziffer Es wird dabei nicht direkt das Zeichen gespeichert, sondern eine Zahl, die sich auf ein ASCII-Zeichen bezieht.	8 Bit	Textausgabe auf Displays
int	Ganzzahl 0 bis 65.535	16 Bit	Zählwert
double	Ganzzahl 0 bis 4.294.967.295	32 Bit	Positionsangabe, Zeitwert
float	Kommazahl von ca. 10^{-38} bis 10^{38}	32 Bit	Analogwerte, Rechnungen

Datentypen können zusätzlich noch mit Eigenschaften ausgezeichnet werden:

const DATENTYP	Datentyp mit **Schreibschutz** (constant), z.B. Festlegen eines IO-Pins
Bei Zahlen:	
signed DATENTYP	Datentyp **mit Vorzeichen** (positive und negative Werte)
unsigned DATENTYP	Datentyp **ohne Vorzeichen** (nur positive Werte)

Erlaubte Namen für Variablen

Es sind Kleinbuchstaben **a-z**, Großbuchstaben **A-Z**, Zahlen **0-9** und der **Unterstrich _** erlaubt.

Umlaute und andere **Sonderzeichen** sind nicht erlaubt und verursachen Fehler beim Kompilieren.

Die Variablennamen dürfen nicht mit einer **Zahl beginnen**.

Sogenannte **Schlüsselwörter**, die in der Programmiersprache vorkommen, dürfen ebenfalls nicht als Variablenname genutzt werden, z.B. *if*, *else*, *auto*, usw…

Zudem soll darauf geachtet werden, dass die Variablen **nachvollziehbare Bezeichnungen** haben. Dies macht das gesamte Programm für einen selbst als auch für andere leichter lesbar.

Deklaration

Vor der ersten Verwendung einer Variablen muss diese **deklariert** werden.

Dabei wird festgelegt, welche **Variable** (z.B. temperatur) welchen **Datentyp** (z.B. float) und ggf. welchen **Startwert** (meist 0) die Variable haben soll. Es macht deshalb Sinn, sich vor dem Programmieren bereits Gedanken zu machen, welche Variablen benötigt werden.

Wurde eine Variable verwendet, die nicht deklariert wurde, wird beim Kompilieren ein entsprechender Hinweis ausgegeben und das Programm kann nicht hochgeladen werden.

Die Deklaration kann theoretisch überall im Programm erfolgen, auch im setup-Bereich oder außerhalb der beiden Bereiche. Auf alle Fälle muss die Deklaration einer Variablen vor oder mit ihrer ersten Verwendung im Code erfolgen.

Datentyp Variable = Startwert ;

> *Beispiel:*
>
> ***bool*** *LED1 =* ***false****;* *// Variable für LED1*

Wertzuweisungen:

Die Wertzuweisung ist wohl der häufigste Befehl im Programm. Es können sowohl Zahlenwerte, als auch Zustände oder Zeichen, bzw. Zeichenketten sein.

Der zuzuweisende Wert kann entweder direkt eingegeben (Konstante) oder von einer anderen Variablen kommen. Wichtig ist, dass die Variablen den gleichen Datentyp haben. Bei Zahlen sind hier Typumwandlungen möglich, durch die aber evtl. Informationen verloren gehen, z.B. durch Kürzen, wenn eine float-Variable in eine int-Variable umgewandelt werden soll.

Variable, der ein Wert zugewiesen wird = zuzuweisender Wert ;

> *Beispiel:*
>
> *LED1 =* ***true****;* *// Einschalten von LED1*

Typumwandlung

Schreibt man den gewünschten Datentyp in Klammern vor die Variable, wird für die weitere Verarbeitung dieser Datentyp verwendet.

Damit können auch unterschiedliche Datentypen miteinander verrechnet werden.

Beispiel:

```
float messwert ;
int abweichung ;
int gewicht ;

gewicht = (int)messwert + abweichung ;
```

VARIABLEN VERKNÜPFEN

Logische Verknüpfungen:

Verknüpfungsart	Zeichen	Beispiel
UND-Verknüpfung	&&	LED1 = (taster1 && sensor2);
ODER-Verknüpfung	||	LED2 = (taster1 ||taster2);
NICHT-Verknüpfung	!	LED3 = !(taster1 && !sensor4);

Mathematische Operationen:

Verknüpfungsart	Zeichen	Beispiel
größer oder gleich	>=	Vergleichfunktion: if (temperatur_ist >= temperatur_soll)
kleiner oder gleich	<=	
genau gleich	==	Nicht verwechseln mit Wertzuweisung durch = !
ungleich	!=	
inkrementieren	++	Wert um 1 erhöhen (pro Durchlauf)
dekrementieren	--	Wert um 1 verringern (pro Durchlauf)
addieren	+	
subtrahieren	-	
multiplizieren	*	
dividieren	/	
Modulo	%	rest = (9 % 5); >> rest = 4

Die Modulo-Funktion nutzt man zur Bestimmung des Restes einer Division.

Auf dem Arduino Nano Board befinden sich zwei Arten von IO-Pins:

- **Digitale IO-Pins**, die als Ausgang, Eingang oder Kommunikationsschnittstelle genutzt werden können
- **Analoge Eingänge**, die über einen integrierten A/D-Wandler Spannungen messen können

Digitale IO-Pins

Diese Anschlüsse können als **Ausgang** oder **Eingang** konfiguriert sein. Dies erfolgt im **Setup-Bereich** mit den Befehlen:

pinMode(PIN, OUTPUT) ; oder **pinMode(PIN, INPUT) ;**

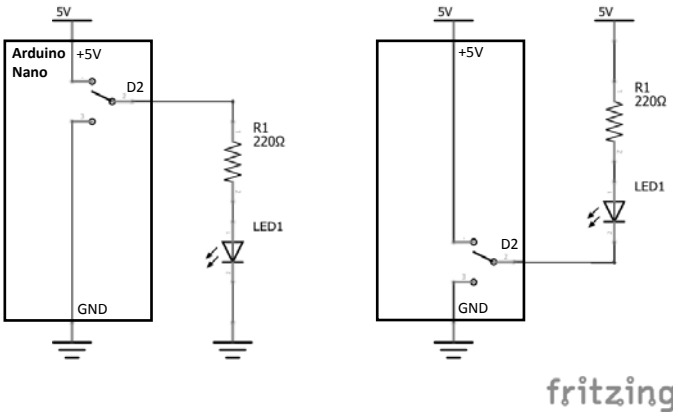
Im Programmzyklus können als **Ausgang** konfigurierte Pins mit folgenden Befehlen angesteuert werden:

digitalWrite(PIN, HIGH) ;

Am Ausgang liegen **5 Volt** an.

digitalWrite(PIN, LOW) ;

Am Ausgang liegen **0 Volt** an, bzw. der Pin wird mit **Minus** verbunden. Es kann also sowohl Plus als auch Minus geschalten werden:

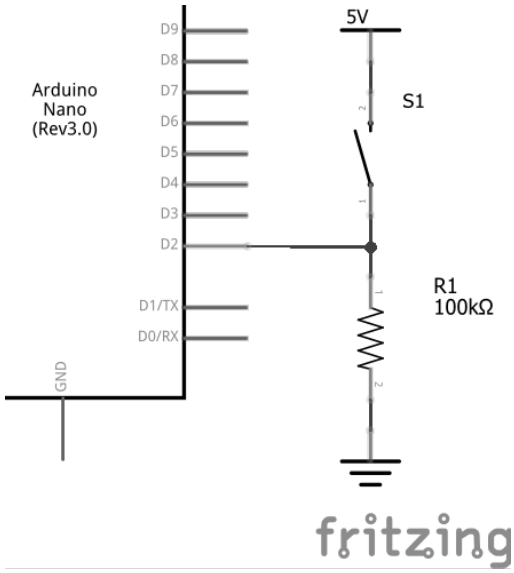


Die als **Eingang** festgelegten Pins können mit einem Befehl, der wie eine Variable vom Datentyp ***bool*** benutzt werden kann, abgefragt werden:

Variable = digitalRead(PIN) ;

Eine Spannung **unter 1,5 Volt** am Eingang ergibt den Status ***false***, eine Spannung **über 3,0 Volt** den Status ***true***.

Damit an einem „in der Luft hängenden“ Eingang keine undefinierten Spannungen anliegen, z.B. bei einem nicht gedrückten Taster, sollte der Eingang mit einem hochohmigen Widerstand (**Pull-Down-Widerstand**) mit z.B. 100 kOhm auf Minus geschalten werden:



Der Eingang wird dadurch auf ein **definiertes Potential** geschalten und bleibt ***false***.

Interne Pullup-Widerstände nutzen

Eine einfachere Variante, Taster und Schalter anzuschließen, wird durch **interne Pullup-Widerstände** ermöglicht.

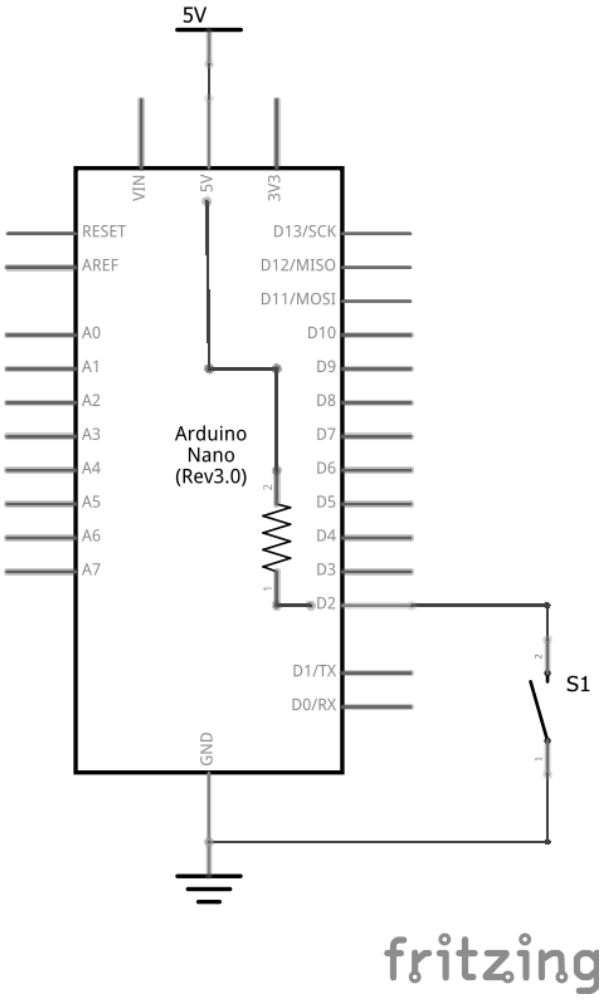
Diese internen Widerstände verbinden **im Mikrocontroller selbst** einen IO-Pin mit **Plus**. Der Eingang ist also standardmäßig ***true***.

Schaltet ein Taster nun diesen Eingang auf **Minus**, liefert er ***false***. Die Abfrage des Tasters ist also **invertiert**, was im Programm aber einfach wieder umgekehrt werden kann, z.B. durch eine **NICHT-Verknüpfung** mit einem „!“ vor der Abfrage des Eingangs.

Um den Pullup-Widerstand für einen digitalen Eingang zu aktivieren, muss **im Setup-Bereich** der Befehl wie folgt geändert werden:

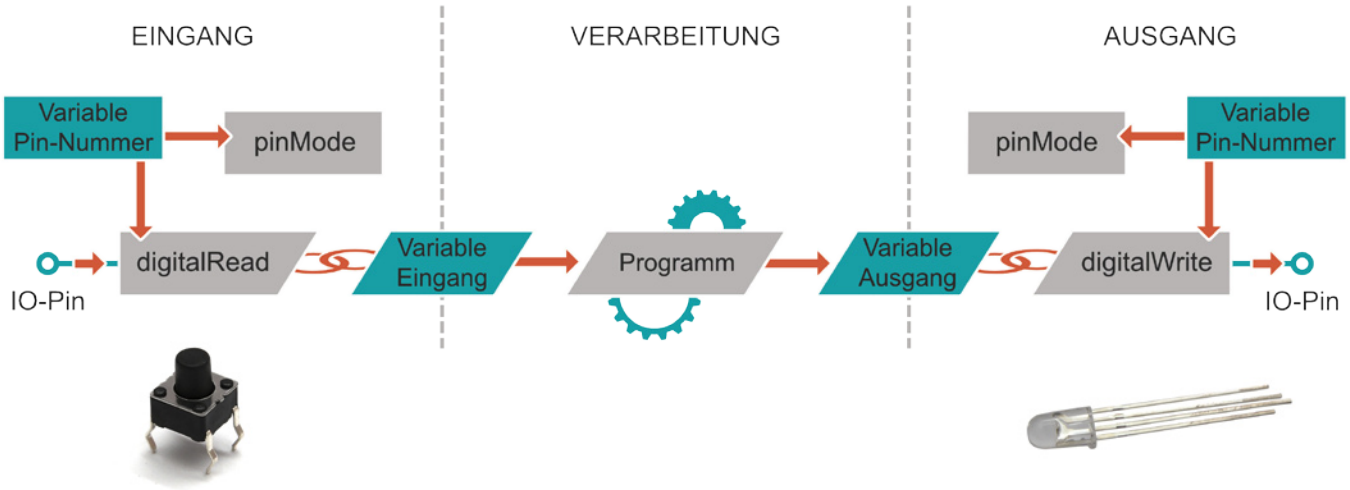
```
pinMode(PIN, INPUT_PULLUP) ;
```

Die Beschaltung von Tastern wird so wesentlich vereinfacht, da kein externer Widerstand notwendig ist.



IO-Pins mit Variablen konfigurieren, abfragen und steuern

Für den Zustand und der Pin-Nummer der Ein- und Ausgänge werden Variablen deklariert, die dann im weiteren Programm immer wieder abgefragt bzw. angesteuert werden können:



Beispiel: Mit Taster eine LED ansteuern

```
int led_pin = 13;                    // Variable für Pin der LED
int taster_pin = 12;                 // Variable für Pin des Tasters

bool led = false;                    // Variable für Zustand der LED
bool taster = false;                 // Variable für Zustand des Tasters

void setup() {
  pinMode (led_pin, OUTPUT)          // Konfigurieren des LED-Pins als Ausgang
  pinMode (taster_pin, INPUT_PULLUP) // Konfigurieren des Taster-Pins als Eingang
}

void loop () {
  taster = !digitalRead(taster_pin); // Verknüpfen der taster-Variable mit dem Eingang
                                     // das Ausrufezeichen invertiert das Signal (wegen Pullup)
  if (taster == true) {              // Steuern der led-Variable abhängig vom Tasterzustand
    led = true;
  }

  digitalWrite(led_pin, led);        // Steuern des Ausgangs mit der led-Variable
}
```

Pulsweitenmodulation (PWM)

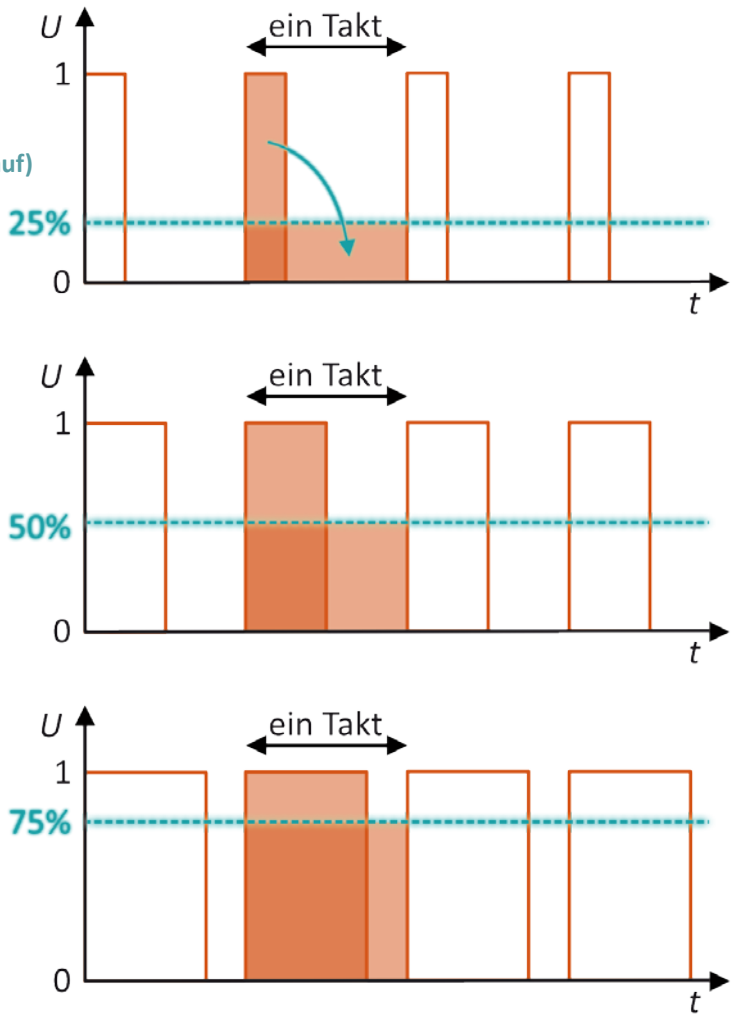
Nicht immer ist es möglich oder sinnvoll die **Leistung** eines elektronischen Verbrauchers über das Verändern der Betriebsspannung zu steuern. Oft ist dazu eine komplexe Schaltung nötig oder überschüssige Spannung wird mit Widerständen „verheizt“, was hohe Verluste und unnötige Wärmeentwicklung bedeutet.

Darum werden viele Verbraucher durch das **Puls-Pausen-Verhältnis** einer **Rechteck-Spannung** gesteuert. Je länger die Pause pro Takt, desto geringer die Leistung. PWM-fähige IO-Pins (siehe Pinout-Übersicht), die als **Ausgang** konfiguriert sind, können Signale zur **Pulsweitenmodulation** ausgeben.

analogWrite(PIN, WERT) ;

Der übergebene Wert zwischen **0 und 255** stellt das **Puls-Pausen-Verhältnis** des PWM-Signals ein:

Mittelwert
(Integral aus Spannungsverlauf)



Da das menschliche Auge ab einer Frequenz von ca. 60 Hz kein Flimmern mehr erkennt, kann PWM auch zum **Dimmen von LEDs** genutzt werden. Gedimmtes LED-Licht besteht eigentlich aus Lichtblitzen, die aber so schnell hintereinander erfolgen, dass die chemischen Vorgänge auf der Netzhaut nicht schnell genug ablaufen um dieses Blitzgewitter wahrzunehmen.

Analoge Eingänge

Die analoge Eingänge müssen nicht im Setup-Bereich konfiguriert werden, da sie nur als Eingangs-Pins genutzt werden können. Sie können mit einem Befehl ausgelesen werden, der einen Integer-Wert zwischen 0 und 1023 ausgibt.

Variable = analogRead(PIN) ;

Damit analoge Signale digital verarbeitet werden können, muss die am Eingang anliegende Spannung in einzelne Bitwerte **abgestuft** werden. Die **Analog-Digital-Wandler** unseres Mikrocontrollers haben dazu eine **Auflösung von 10 Bit**. Mit 10 Bit lassen sich **2^{10}-1 Werte**, also Zahlen bis 1023 darstellen.

Der A/D-Wandler teilt eine Referenzspannung in **1024 Stufen** (Null eingeschlossen) ein.

Die **Referenzspannung** beträgt standardmäßig **5 Volt**, sie kann aber über einen Befehl im Setup-Bereich auf **1,1 Volt** geändert werden.

Standartwert 5 Volt: **analogReference(DEFAULT) ;**

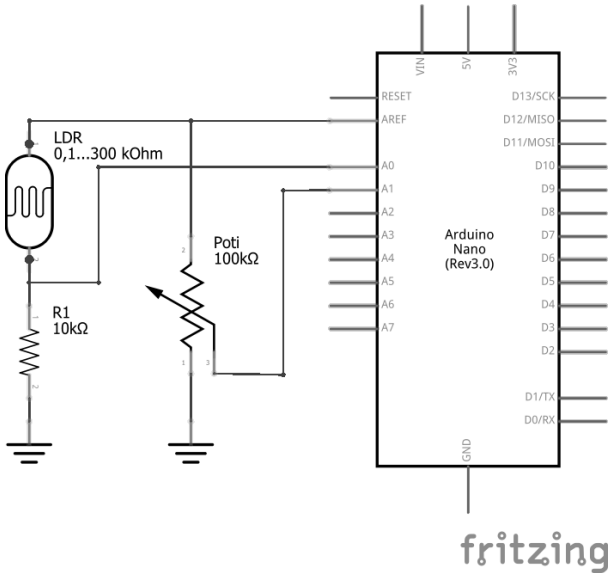
Interner Wert 1,1 Volt: **analogReference(INTERNAL) ;**

Bei 5 Volt beträgt eine Spannungsstufe also 5 V/1024 = **4,88 mV ca. 5 mV**. Die Spannungsstufen sind also in etwa so verteilt:

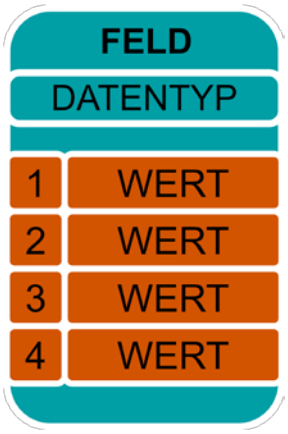
Messspannung	Bitmuster	Wert
4,995 ... 5,000 Volt	1111111111	1023
4,990 ... 4,995 Volt	1111111110	1022
4,985 ... 4,990 Volt	1111111101	1021
⋮	⋮	⋮
0,010 ... 0,015 Volt	0000000010	2
0,005 ... 0,010 Volt	0000000001	1
0,000 ... 0,005 Volt	0000000000	0

Analoge Sensoren und **Potentiometer** nutzen die Referenzspannung am **Pin „REF“** um genaue Ergebnisse zu erhalten. Sie bilden zusammen mit einem zusätzlichen Widerstand, der auf Minus geschalten ist, einen **Spannungsteiler**. Der mittlere Abgang des Spannungsteilers liefert eine Spannung, welche Abhängig vom Widerstand des Sensors ist und mit einem Analogeingang verbunden ist.

Der Schaltplan zeigt die Beschaltung der Analogeingänge eines Arduino Nano mit einem Lichtsensor (LDR) und einem Potentiometer.



Felder (array)



Will man viele Werte vom **gleichen Datentyp** speichern, dann bieten sich Arrays an. Es ist wie ein Regal für Daten und sorgt für mehr Übersicht und Ordnung. Außerdem können die Daten in einem Array leichter von Funktionen addressiert werden als einzelne Variablen.

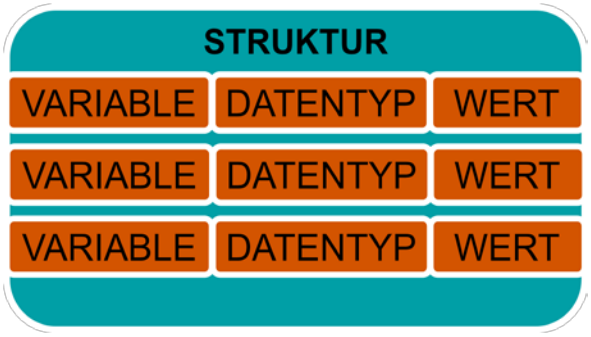
Beispiel: RGB-Farbwert

```
int innenbeleuchtung [3] ;          /* Deklariere ein Feld vom Datentyp Integer mit 3 Einträgen */
innenbeleuchtung [0] = 128 ;        /* Rot wird auf den Wert 128 gesetzt */
innenbeleuchtung [1] = 39 ;         /* Grün wird auf den Wert 39 gesetzt */
innenbeleuchtung [2] = 0 ;          /* Blau wird auf den Wert 0 gesetzt */
```

Denselben Effekt hätte eine Deklarierung des Feldes mit den gewünschten Werten:

```
int innenbeleuchtung [] = {128, 39, 0} ;
```

Strukturen (struct)



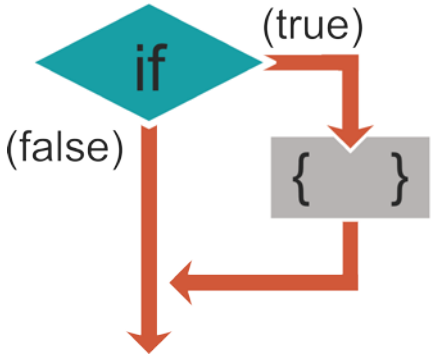
Sollen mehrere Daten **unterschiedlichen Datentyps** zusammengefasst werden, nutzt man Strukturen. Mit dem Namen der Struktur, einem Punkt und dem Namen der enthaltenen Variablen kann das „Paket“ geöffnet und auf einzelne Inhalte zugegriffen werden.

Beispiel: Schimmelwarner
Speicherung von Luftfeuchte- und Temperatur mit einem Zeitwert

```
struct messung {
double int time;
float temperature;
float humidity;
}

// Abfrage:
if (messung.humidity >= 70.0) {
alert = true;
}
```

if-Abfrage

Die wohl häufigste Abfrage, die eine **Entscheidung** zwischen **zwei Fällen** durch eine **Bedingung** trifft. Eine oder mehrere Bedingungen werden in Klammern hinter dem if festgelegt. Das können logische Verknüpfungen wie UND (&&) und ODER (||) sein, aber auch Zahlenoperationen oder Vergleiche. Wird die Bedingung erfüllt, wird das, was in den geschweiften Klammern steht, ausgeführt. Ansonsten **überspringt** der Microcontroller das Programm in der if-Abfrage. Auch Verschachtelungen sind möglich, wobei auf das richtige Setzen der Klammern geachtet werden muss.



Beispiel:

```
if (taster1 == true) {
programm_nummer++;
}
```

switch-case Abfrage

Gibt es nicht nur zwei Fälle zu unterscheiden, sondern gleich **mehrere Fälle**, kann man dies elegant mit dieser Abfrage realisieren. Jeder Fall bekommt den **Wert einer Variablen** zugewiesen, den diese annehmen kann. Nimmt die Variable einen Wert an, für den kein Fall definiert ist, wird der **default-Fall** aufgerufen. Jeder Fall muss mit dem **break-Befehl** beendet werden.

Die switch-case Abfrage eignet sich besonders gut für die Auswahl von Werten anhand einer Nummer oder für getaktete Abläufe.

Beispiel: Auswahl verschiedener Lichtfarben durch eine Integer-Variable

```
switch (farbnummer) {

default:    red = 0;      green = 0;      blue = 0;      break ;
case(0):    red = 255;    green = 0;      blue = 0;      break ;
case(1):    red = 255;    green = 255;    blue = 0;      break ;
case(2):    red = 0;      green = 255;    blue = 0;      break ;
case(3):    red = 0;      green = 255;    blue = 255;    break ;
case(4):    red = 0;      green = 0;      blue = 255;    break ;
case(5):    red = 255;    green = 0;      blue = 255;    break ;
case(6):    red = 255;    green = 255;    blue = 255;    break ;

}
```

Durch Schleifen im Programm wird es möglich einen Ablauf zu **wiederholen**, bis eine bestimmte **Bedingung** erfüllt ist. Ein einfaches Beispiel ist die Steuerung einer Waschmaschine. Die Vorgänge „Trommel rechts drehen“ und „Trommel links drehen“ sollen so oft wiederholt werden, bis ein Zähler den Zählerstand erreicht, den das gewählte Waschprogramm vorgibt um die Wäsche sauber zu bekommen.

for-Schleife

for (Zählervariable, Schleifenbedingung, Änderung der Variable) {Schleifeninhalt} ;

Bei dieser Schleife wird ein Zähler genutzt, um zu bestimmen, wie oft eine Aktion ausgeführt werden soll. Die Syntax der for-Schleife besteht aus drei Teilen: Initialisieren einer **Zählvariable**, Abfrage der **Schleifenbedingung** und **Änderung der Zählvariable** nach einem Durchlauf. Der Schleifeninhalt wird von geschweiften Klammern umschlossen.

Beispiel: Waschmaschine

```
for (int n = 0; n<10; n+1) {

trommel _r = true;
trommel_l = false;

delay(20s);

trommel_r = false;
trommel_l = true;

delay(20s);

}
```

Was passiert? Zu Beginn wird die Zählvariable auf 0 gesetzt. Sie ist damit kleiner als 10, die Schleifenbedingung ist daher *true*. Die Schleife kann durchlaufen. Der Trommelmotor dreht sich jeweils für 20 Sekunden in beide Richtungen.

Nach einem Schleifendurchlauf – das Programm in der Schleife ist abgearbeitet – wird die Zählvariable n um 1 erhöht. Erneut wird die Schleifenbedingung geprüft. 1 ist kleiner als 10, also wird die Schleife erneut abgearbeitet.

Nach dem zehnten Durchlauf ist die Schleifenbedingung nicht mehr *true*, da n den Wert 10 erreicht hat. Die Schleife wird nicht mehr ausgeführt und das Programm fährt mit der Abarbeitung des Codes unterhalb der Schleife fort.

while-Schleife

Ist die Schleifenbedingung erfüllt, wird der Inhalt der Schleife wiederholt ausgeführt. Es wird dabei immer vor Abarbeitung des Schleifeninhalts geprüft, ob die Bedingung noch erfüllt ist. Wenn nicht, wird die Schleife übersprungen und der Code unterhalb der Schleife abgearbeitet.

while (Bedingung) {Schleifeninhalt} ;

do while-Schleife

Während for- und while- Schleifen zuerst prüfen, ob die Schleifenbedingung erfüllt ist bevor die Schleife gestartet wird, erfolgt bei der do while-Schleife die **Prüfung erst nach einem Durchlauf.** Die Schleife wird eingesetzt wenn zuerst eine Aktion gestartet werden muss, bevor entschieden werden kann, ob die Schleife abgebrochen wird.

do {Schleifeninhalt} while (Bedingung) ;

Am Beispiel unserer Waschmaschine würde das die Wasserpumpe betreffen, welche das Schmutzwasser nach der Wäsche abpumpt:

```
do {

wasserpumpe = true;
delay(10s);

} while (wasserstand = true);
```

Was passiert? Erreicht das Programm die Schleife, wird diese sofort gestartet. Die Wasserpumpe wird eingeschaltet. Nach 10 Sekunden wird die Schleife beendet und es wird geprüft, ob noch Wasser in der Maschine ist. Erkennt der Wasserstandsensor noch einen Rest, bleibt dieser auf true und die Schleife startet erneut. Ist nach weiteren 10 Sekunden abpumpen kein Wasser mehr am Sensor, wird die Schleife beendet.

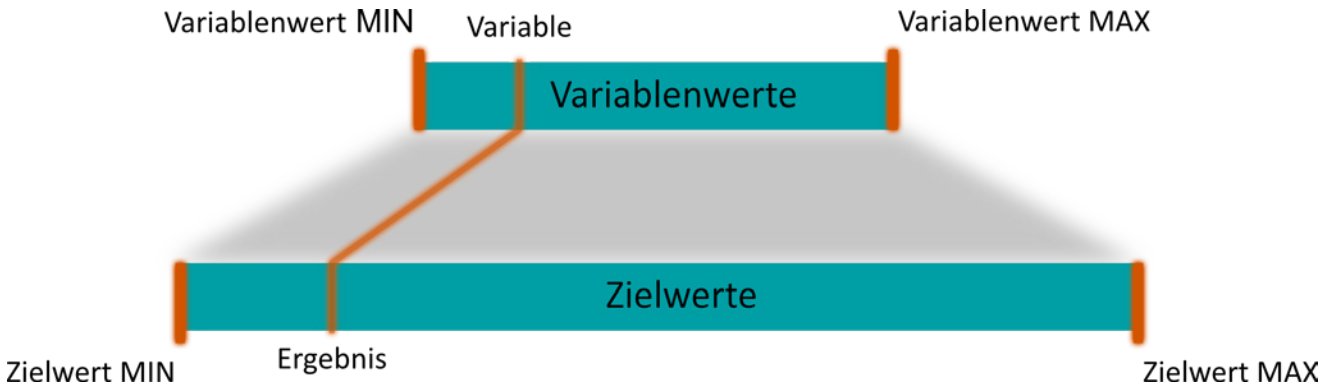
Abbruch einer Schleife (break)

Soll eine Schleife sofort beendet werden, unabhängig von der Schleifenbedingung, dann kann innerhalb der Schleife der break-Befehl genutzt werden. Angenommen, die Wasserpumpe würde trocken laufen und sich dadurch erhitzen, sollte ein Temperaturfühler im Motor das Abpumpen vorzeitig beenden um den Motor zu schützen:

```
do {
        wasserpumpe = true;
        if (wasserpumpe_temp >= 70) {
        break;
        }

delay(10s);

} while (wasserstand = true);
```

Nutzt man statt ***break*** den Befehl ***continue***, wird die Schleife nicht abgebrochen, sondern neu gestartet.

map-Funktion



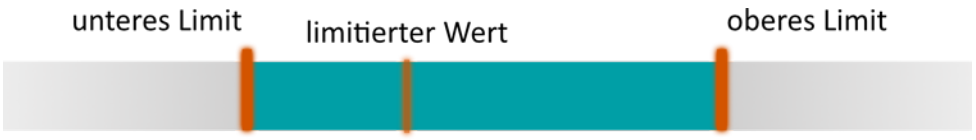
Wert = *map*(Variable, VariablenwertMIN, VariablenwertMAX, ZielwertMIN, ZielwertMAX)

Diese Funktion macht es sehr einfach, einen Wert innerhalb eines bestimmten Wertebereiches abhängig von einem anderen Wert mit einem anderen Wertebereich abzubilden. Das macht die map-Funktion sehr nützlich bei Messaufgaben, wenn eine Spannung von einem Sensor wieder in die ursprüngliche Messgröße umgerechnet werden soll. Auch Prozentrechnungen lassen sich damit einfach umsetzen.

> *Beispiel mit Werten:* *50 = **map**(512, 0, 1024, 0, 100)*

Wichtig: Die Umrechnung erfolgt streng linear. Viele Sensoren, vor allem Temperaturfühler, bilden die Messgröße aber nicht linear auf das abgegebene Spannungssignal über den Messbereich ab. Hier ist diese Methode nicht ganz genau.

constrain-Funktion



limitierter Wert = *constrain*(Wert, unteres Limit, oberes Limit)

Um zu verhindern, dass ein Wert zu groß oder zu klein wird, kann man diese Funktion nutzen. Überschreitet der Wert das obere oder untere Limit, wird das Ergebnis auf das entsprechende Limit gesetzt.

Als serielle Ausgabe bezeichnet man die Kommunikation zwischen dem Mikrocontroller und dem Computer um Werte zu kontrollieren oder Funktionen zu überprüfen. Um die serielle Ausgabe zu ermöglichen, muss im setup-Bereich die **Baudrate** (Übertragungstakt) festgelegt werden, z.B. 19200 Baud.

Die Baudrate beschreibt die Anzahl an Symbolen, die während einer Sekunde von einer Kommunikationsschnittstelle übertragen werden. Ein Symbol kann aus mehreren Bit bestehen, die Bitübertragungsrate kann also wesentlich höher sein.

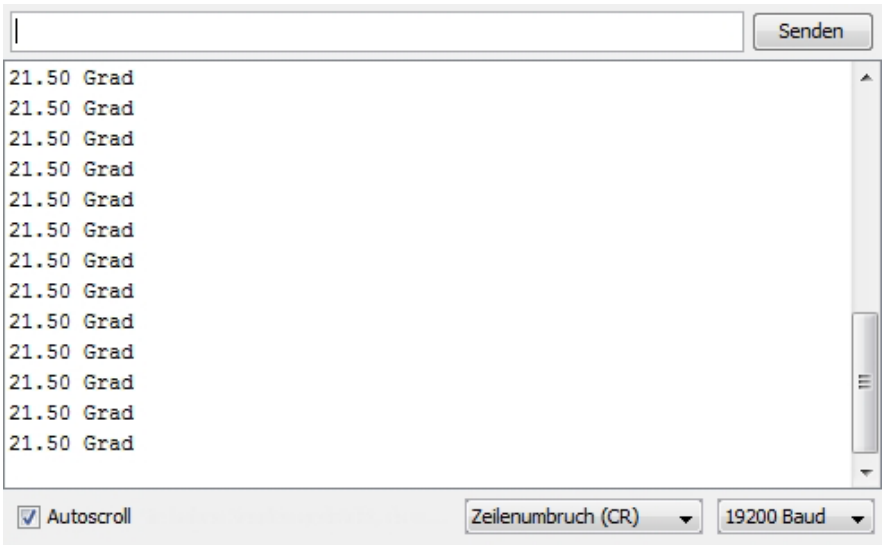
Einstellen der Baudrate:

```
void setup() {

Serial.begin(19200);

}
```

Im Programmzyklus kann an beliebiger Stelle (Empfehlung: am Programmanfang und zusammenhängend) die Ausgabe der seriellen Daten erfolgen:

Abfrage von Variablen	`Serial.print(temp);`
Einfügen von Text	`Serial.print(„ Grad “);`
Neue Zeile beginnen	`Serial.println();`

Die Anzeige der Daten erfolgt als Text auf dem **seriellen Monitor** der Arduino-Programmierumgebung. Dort muss die **gleiche Baudrate** wie auf dem Mikrocontroller eingestellt sein:



In jedem Programmzyklus wird die serielle Übertragung einmal ausgeführt, ggf. wird pro Sekunde also mehrmals der Text übertragen. Hier kann die delay-Funktion z.B. `delay`(1000); helfen, mit der man den Programmablauf für eine bestimmte Zeit in Millisekunden pausieren kann.

millis()

Mit dieser Funktion kann man die **Systemlaufzeit** des Mikrocontrollers seit dem letzten Neustart auslesen. Die Zeit wird in **Millisekunden** als **unsigned long** int ausgegeben. Nach etwa 49 Tagen kommt es zum Overflow – der maximale Wert innerhalb der Variablen wird erreicht und der Zähler beginnt bei 0 erneut. Interessant ist bei der Verwendung von **millis**() oft nicht die Laufzeit selbst, sondern die Möglichkeit mithilfe dieser Zahl **eigene Takte und Impulse** zu generieren, ohne mit dem **delay**-Befehl das ganze Programm anhalten zu müssen. Neben **millis**() gibt es auch noch **micros**(), welche **Mikrosekunden** hochzählt und noch kürzere Zeitfunktionen ermöglicht. Hier ein Anwendungsbeispiel zur Benutzung der millis-Funktion:

Erzeugen eines einfachen Blinktaktes von 1 Hz:

```
unsigned long previousMillis = 0;     // Variable, welche die Zeit des letzten Durchlaufs speichert
bool blink = false;                   // Variable für das Blinksignal

void setup() {
pinMode (12, OUTPUT);                 // Konfigurieren der IO-Pins für die LEDs
pinMode (13, OUTPUT);
}

void loop() {
if (millis() - previousMillis > 500) {  // Wenn seit dem letzten Durchlauf 500ms vergangen sind,
                                        // ist die Bedingung erfüllt
previousMillis = millis();              // Der Zeitwert des letzten Durchlaufs wird mit
                                        // aktuellem Zeitwert überschrieben
blink = !blink;                         // Invertieren des aktuellen Signalzustandes, dadurch ist nur
}                                       // eine if-Bedingung für das Ein- und Ausschalten notwendig

digitalWrite(12, blink);              // Ansteuern einer LED mit der Blinktakt-Variablen
digitalWrite(13, !blink);             // Ansteuern einer LED, die invertiert blinken soll (Wechselblinker)

}
```

Unproblematisches Abfragen von Tastern

Bei der Verwendung von Tastern trifft man beim Programmieren meist auf zwei Probleme:

- Auch wenn der Taster dauerhaft gedrückt bleibt, wird die mit ihm verknüpfte Funktion **in jedem Programmzyklus** einmal ausgeführt.

- Ungewolltes, mehrfaches Schalten des Tasters durch **Vibrationen beim Drücken und Loslassen**, sogenanntes „Prellen“ stört den gewünschten Programmablauf.

Beides wird im folgendem Code mit zwei If-Abfragen, einer zusätzlichen Variablen sowie einer Zeitfunktion realisiert, welche durch Prellen hervorgerufene Signale 100 ms nach Tastendruck ausblendet. Zusätzlich wird gezeigt, wie das „Toggeln“ einer Variable, z.B. für das Ein-/Ausschalten damit realisiert werden kann.

Die Variable *button_state* speichert dabei den letzen Zustand des Tasters. Wird *button_state* in der ersten If-Abfrage true, kann die Abfrage beim nächsten Programmzyklus nicht erneut durchlaufen werden. Dafür wird die Bedingung der zweiten If-Abfrage erfüllt, wenn der Taster wieder losgelassen wird. Auch die Abfrage für den nicht gedrückten Taster wird nur für einen Zyklus aktiv.

In der ersten If-Abfrage wird die Variable *power* invertiert. Jedesmal, wenn der Taster gedrückt wird, ändert diese ihren Zustand. Sie steuert am Ende des Programms die LED an.

Werden mehrere Taster verwendet, ist es notwendig für jeden Taster eine eigene Zustandsvariable sowie ggf. eine eigene ***unsigned long*** Variable für den Zeitwert anzulegen.

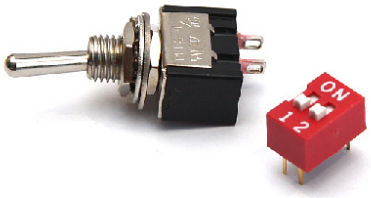
```
// Variables
bool button = false;                    // Abfrage Eingang Taster
bool button_state = false;              // Hilfsvariable für Taster-Zustand
bool power = false;                     // Zu toggelnde Variable (Ein-/Ausschalten)
bool LED = false;                       // Variable für LED
unsigned long previousMillis = 0;       // Letzter Millis-Wert

// Pin configuration
byte button_pin = 12;
byte LED_pin = 10;

void setup() {
pinMode(LED_pin, OUTPUT);
pinMode(button_pin, INPUT);
}

void loop() {
button = digitalRead(button_pin);
digitalWrite(LED_pin, LED);

if (button==true && button_state==false && millis()-previousMillis > 100){
  button_state = true;
  power = !power;
  previousMillis = millis();
  }

if (button==false && button_state==true && millis()-previousMillis > 100) {
  button_state = false;
  previousMillis = millis();
}

if (power==true){
  LED=true;
  } else LED=false;

}
```

Taster

Bedienelemente oder Kontakte zur Positionsabfrage von Abdeckungen, Gehäuseteilen, Mechanik, usw. Sie schließen den Stromkreis nur solange eine Betätigung erfolgt.

Beispiel: Computertaste

Schalter



Bedienelemente, die nach dem Betätigen in ihrer Schaltstellung bleiben. Durch erneutes Betätigen oder ändern der Betätigungsrichtung kann die Schaltstellung wieder geändert werden. Der Stromkreis bleibt bis zum nächsten Ändern der Schaltstellung geschlossen.

Beispiel: Schalter an Taschenlampe

Bei Tastern und Schaltern unterscheidet man zwischen Druck-, Schiebe-, Kipp-, Zug-, Hebel- und Drehschalter bzw. -taster. Sie können Einfach-, Mehrfach- oder Wechselkontakte (Umschalter) haben. Auch mehrere Schaltstellungen sind möglich. Für viele Modelle gibt es eine große Auswahl von aufsteckbaren Knöpfen und weiteres Zubehör. Auch beleuchtete Varianten sind erhältlich.

Potentiometer

Ein durch Drehen oder Schieben **veränderbarer Widerstand**. Ein beweglicher Schleifer kann auf einer Widerstandsschicht aus Metall, Kohle oder leitfähigem Plastik einen **variablen Widerstands- bzw. Spannungswert** abgreifen. Da Potentiometer meist nicht sehr belastbar sind, steuern sie selten direkt den Verbraucher sondern geben einen Widerstand für eine angeschlossene elektronische Schaltung oder den Analogeingang eines Mikrocontrollers vor.

Potentiometer können direkt auf einer Platine (Print) oder in das Gehäuse (Zentral) mit Schrauben montiert werden. Die Wellen sind genormt und es besteht eine große Auswahl an aufsteckbaren Drehknöpfen. Auch eigene Knöpfe lassen sich relativ einfach montieren.

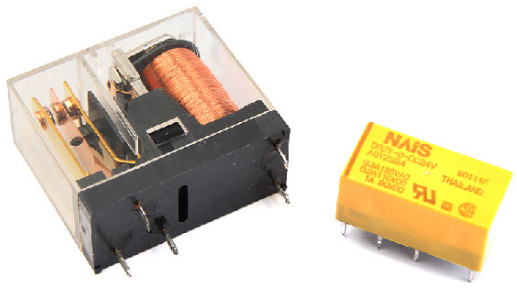
Beispiele: Lautstärkeregler, Dimmer, Joysticks

Reed-Kontakt

Ein kleines **Glasröhrchen mit zwei Kontakten** an den Enden, welche mit **magnetisierbaren Plättchen** im Inneren verbunden sind, die nahe aneinander liegen. Nähert man ein **Magnetfeld**, magnetisieren sich die Plättchen und ziehen sich gegenseitig an, wodurch Strom über die Kontakte fließen kann. Im Glaskörpers befindet sich ein Gas, welches Funkenbildung und Oxidation der Kontakte vermeidet.

Beispiel: Türkontakt von Alarmanlage, Sensor von Fahrradtacho

Relais



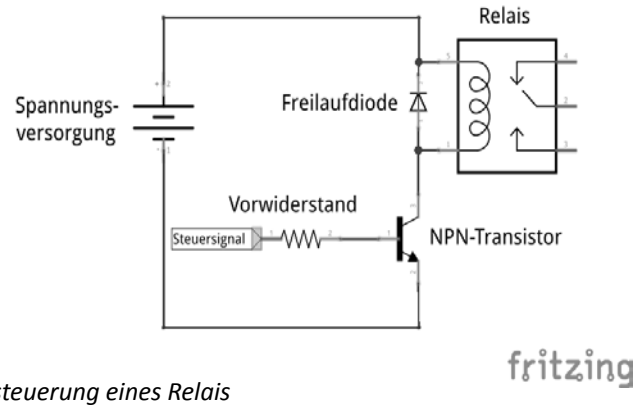
Mit Relais kann man mit einer kleinen Spannung eine große Spannung schalten oder auch umgekehrt. Sie bestehen aus einer **elektromagnetischen Spule** und einer Mechanik, welche bei bestromter Spule Kontakte schließt. Da die beiden angeschlossenen Stromkreise **keine elektrische Verbindung** haben, spricht man von einer **galvanischen Trennung**.

> *Beispiel: Schaltung der Heizung von einem Wäschetrockner durch die Steuerelektronik*

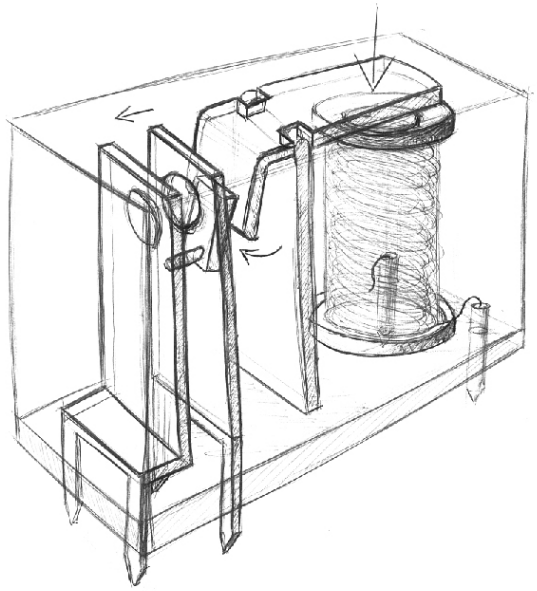
Der große Bruder des Relais ist das **Schütz**, welches im industriellen Bereich eingesetzt wird und große Ströme und Spannungen schalten kann.

Solid-State-Relais funktionieren im Gegensatz zu mechanischen Relais **rein elektronisch**, das heißt **Halbleiter** übernehmen die Schaltfunktion. Sie sind schneller, wartungsfrei, geräuschlos und meist kompakter, aber auch anfälliger für Spannungsspitzen z.B. beim Schalten von Motoren.

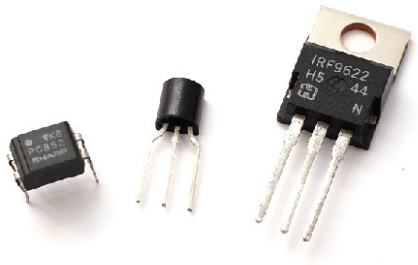
Mechanische Relais dürfen nicht direkt am Ausgang eines Mikrocontrollers angeschlossen werden, da die Spulen **Spannungsspitzen** erzeugen, welche den Chip beschädigen können. Meist werden **Transistoren** zur Steuerung von Relais dazwischen geschaltet. Zusätzlich werden sogenannte „Freilaufdioden“ in Sperrrichtung zur Relaisspule parallel geschaltet:



Ansteuerung eines Relais



LEISTUNGSELEKTRONIK



Optokoppler, Transistor und MOSFET

Mit einem Mikrocontroller können nur kleine Verbraucher mit **max. 20...40 mA** wie z.B. eine Standard-LED direkt am Ausgang betrieben werden. Sollen Verbraucher mit **höheren Leistungen** oder Spannungen geschaltet werden, sind Bauteile nötig, welche mit dem Signal eines Controllers den elektrischen Energiefluss eines Laststromkreises steuern.

Optokoppler

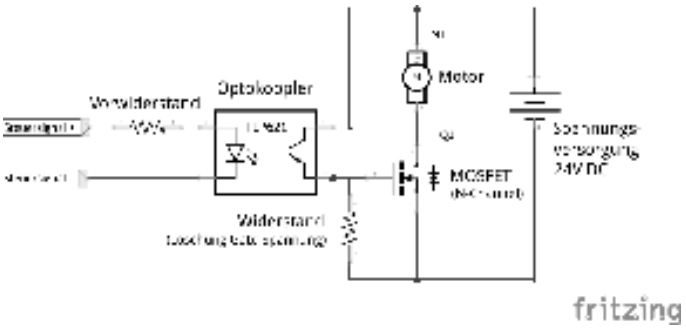
Das Bauteil bestehen aus einer **LED** und einem **lichtempfindlichen Halbleiterbauteil**, welche elektrisch voneinander isoliert, d.h. galvanisch getrennt sind. Nur durch das Licht wird das **Steuersignal** übertragen. Sie werden genutzt, um den **Steuerstromkreis** z.B. von einem Microkontroller von dem **Laststromkreis** zu **trennen**. So kann ein 5V Signal eine Schaltung mit 230V steuern ohne dass die Spannungen miteinander in Berührung kommen. Die Eingangsseite des Optokopplers mit der LED benötigt einen **Vorwiderstand**. Die Ausgangsseite kann oft nur schwach belastet werden, weshalb weitere Bauteile der Leistungselektronik nötig sind.

Transistor

Ein Transistor ist ein **elektronischer Schalter** (digital) bzw. auch **Verstärker** (analog) mit drei Anschlüssen. Die Basis (B) steuert bei einem NPN-Transistor über einen schwachen Strom die Leitfähigkeit zwischen Collector (C) und Emitter (E), wobei der Basis-Strom über den Emitter abfließt. Der Emitter wird mit Minus verbunden, die Basis über einen geeigneten **Vorwiderstand** (z.B. 10 kOhm) mit einem digitalen Ausgang verbunden und der Collector mit dem Minuspol eines Verbrauchers. Transistoren sind für kleine bis mittlere Lasten (mehrere Ampere) geeignet und sind in verschiedenen Belastbarkeiten und Bauformen erhältlich. Bei Lasten ab ca. 1 A sind jedoch meist Kühlkörper nötig um die Verlustwärme der Collector-Emitter-Verbindung abzuführen. Der Transistor ist **nur für Gleichspannung** geeignet.

MOSFET

Im Gegensatz zum Transistor ist der **Übergangs-widerstand** bei diesem Bauteil nur sehr gering und es können auch Lasten von zehn Ampere geschalten werden ohne das Bauteil kühlen zu müssen. Das Steuersignal kann bei den meisten Typen direkt auf das sog. „Gate“ gegeben werden. Die Verbindung zwischen „Source“ und „Drain“ wird dann leitfähig. Der MOSFET ist **nur für Gleichspannung** geeignet.



Schaltung zur Ansteuerung eines Motors mit einer anderen Betriebsspannung als der Microcontroller

TRIAC

Mit einem TRIAC lassen sich **Wechselspannungen fast verlustfrei schalten**, z.B. ein 230V Motor oder die Raumbeleuchtung. Sie werden z.B. in Bewegungsmeldern ohne Relais eingesetzt.

ACHTUNG: Schaltungen mit 230V sollten nur von einer Elektrofachkraft aufgebaut werden! Bitte wendet euch bei einem solchen Vorhaben an eine fachkundige Person!

LED – Light Emitting Diode

Durch gezielte Verunreinigung (Dotierung) von Halbleitern wie Silizium kann eine **Diode mit lichterzeugenden Eigenschaften** hergestellt werden. Je nach dotiertem Stoff kann Licht mit einer bestimmten Wellenlänge erzeugt werden. Weißes Licht wird durch bestrahlen einer **Phosphorschicht** mit blauem Licht erzeugt.

Für die Lichterzeugung wird **weniger als 20%** der elektrischen Energie benötigt, wie für die gleiche Lichtleistung bei einer Glühlampe nötig wäre.

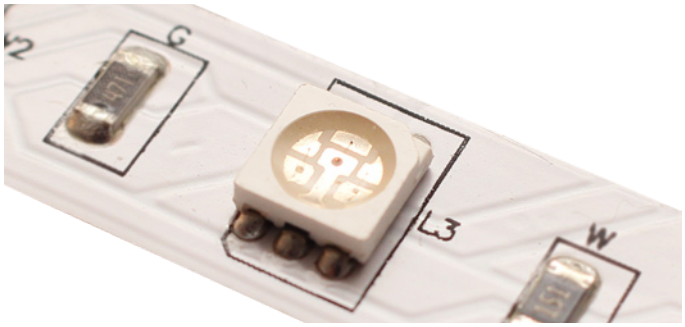
LEDs werden mit **Gleichspannung** und einer vorgegebenen **maximalen Spannung und Strom** betrieben. Sie reagieren sehr empfindlich auf Spannungsänderungen und müssen deshalb immer mit einem **Vorwiderstand**, bei höheren Leistungen mit einer **Konstantstromquelle** (KSQ) oder **LED-Treiber**, betrieben werden.

Der Pluspol einer LED wird **Anode** genannt, der Minuspol **Kathode**. Da ein Verpolen der LED zu ihrer Zerstörung führen kann, sind die Gehäuse oder Anschlussdrähte entsprechend gekennzeichnet.

LEDs sind in verschiedenen Farben (auch Ultraviolett und Infrarot), Gehäuseformen, Abstrahlwinkel und Helligkeiten erhältlich und können gezielt nach Verwendungszweck ausgewählt werden.

Mehrere LEDs können in Reihe geschalten werden, wodurch sich die Betriebsspannung entsprechend erhöht und der (begrenzte!) Strom gleich bleibt.

RGB-LEDs



In ihnen befinden sich gleich **drei LED-Chips** mit den Grundfarben **Rot, Grün und Blau**. Durch Mischung der drei Farben durch unterschiedliche Ansteuerung der LED-Chips kann fast jede beliebige sichtbare Lichtfarbe erzeugt werden, auch weiß. Die Mischung erfolgt entweder durch **Spannungsänderung** an den LED-Chips oder durch **Pulsweitenmodulation** (PWM).

Die LED-Chips können entweder mit gemeinsamer Anode (+), gemeinsamer Kathode (-) oder mit offenen Kontakten ausgeführt sein.

In der Innenraumbeleuchtung werden zunehmend **RGBW-LEDs** eingesetzt, die neben den drei Grundfarben auch noch warm-, kalt- oder neutralweiße LED–Chips besitzen. Dadurch lassen sich angenehmere Farbtemperaturen und Pastellfarben erzeugen.

Beispiele: LED-Streifen, Großbildschirme für Veranstaltungen, individualisierbare Lichtdekore, Stimmungslichter

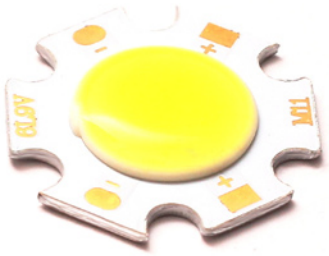
Konstantstromquellen (KSQ)

Bei LEDs höherer Leistung ist ein **gleichbleibender Betriebsstrom** erforderlich um eine maximale Lebensdauer und Helligkeit zu erreichen.

Dies ermöglichen Konstantstromquellen, kleine Module, welche die Spannung immer so regeln, dass ein **festgelegter Strom** fließt.

Viele KSQs lassen sich auch mit einem **PWM-Signal** an einem zusätzlichen Anschluss direkt dimmen.

Hochleistungs-LEDs



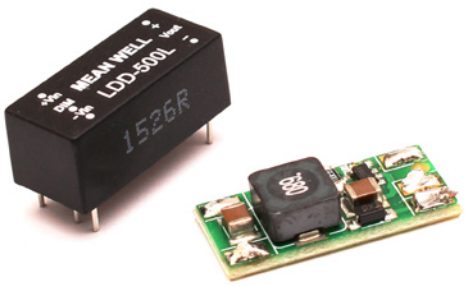
Für die Beleuchtung von Räumen und Außenbereichen werden LEDs höherer Leistung benötigt. Diese bestehen meist aus einer Vielzahl in einem Gehäuse untergebrachten LED-Chips, die in Summe eine hohe Lichtleistung erbringen.

Um die **Wärme** der eng beieinander liegenden Chips abführen zu können, werden die Halbleiter auf eine **Aluminium- oder Kupferplatte** aufgebracht, die über eine **Wärmeleitpaste, - kleber oder –pad** mit einem **Kühlkörper** aus Aluminium thermisch verbunden ist. Je nach abgegebener Wärmeleistung der LED wird ein entsprechend großer Kühlkörper, ggf. auch eine aktive Kühlung mit Lüftern (z.B. bei LED-Beamer) notwendig.

Eine Überhitzung der LED kann zu Änderungen der Lichtfarbe, Verkürzung der Lebensdauer oder zur Zerstörung führen. Das **Wärmemanagement** spielt bei Hochleistungs-LEDs deshalb eine große Rolle.

Aufgrund der hohen Leistungen werden diese LEDs fast ausschließlich über Pulsweitenmodulation (PWM) gedimmt.

Beispiele: Taschenlampen, Glühbirnen-Ersatz, Fahrzeugbeleuchtung, LED-Blitz in Smartphones, Bühnen- und Eventbeleuchtung



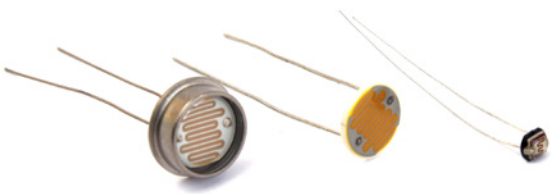
Akustische Signalgeber

Es gibt zwei häufig verwendete Techniken mit Strom Töne zu erzeugen: Entweder als Lautsprecher mittels an **Elektromagneten** gekoppelte Membrane oder mithilfe von **Piezo-Kristallen**, die sich bei anlegen einer Spannung minimal ausdehnen und so Töne erzeugen können. Im Gegensatz zu Lautsprechern sind Piezo-Signalgeber auf den Hoch- bis Ultraschallbereich beschränkt und eigenen sich daher weniger zur Klangwiedergabe. Sie werden aber aufgrund ihrer kompakten Bauform oft in elektronischen Geräten eingesetzt, die den Benutzer akustisch auf sich aufmerksam machen wollen.

Beispiele: Akustische Fertigmeldung von Geschirrspülern und Waschmaschinen, Rauchmelder

SENSOREN

Lichtabhängige Widerstände (LDR)



Halbleiter-Bauteile, die ihren Widerstandswert stark verringern, wenn Licht auf sie fällt. Sie sind vor allem für sichtbares Licht geeignet.

Beispiel: Dämmerungsschalter

Temperaturabhängige Widerstände

Man unterscheidet:

NTC	negativer Temperaturkoeffizient Heißleiter, steigt die Temperatur, sinkt der Widerstandswert
PTC	positiver Temperaturkoeffizient Kaltleiter, steigt die Temperatur, steigt der Widerstandswert

Digitaler Temperatursensor

Ein Sensor mit **integrierter Schaltung**, welcher die Temperatur misst und den Wert als **Bitfolge** oder Bus-Telegramm (z.B. I2C) ausgibt.

Sein Vorteil liegt darin, dass die Leitung zum Sensor keinen Einfluss auf das Messergebnis hat.

Infrarot-Messung

Ein optischer Sensor, der auf **Infrarot- bzw. Wärmestrahlung** reagiert. Er wird dort eingesetzt, wo **berührungslos** gemessen werden muss. Da der Sensor nicht direkt mit der Messstelle in Berührung kommt, sind auch sehr hohe Temperaturen erfassbar.

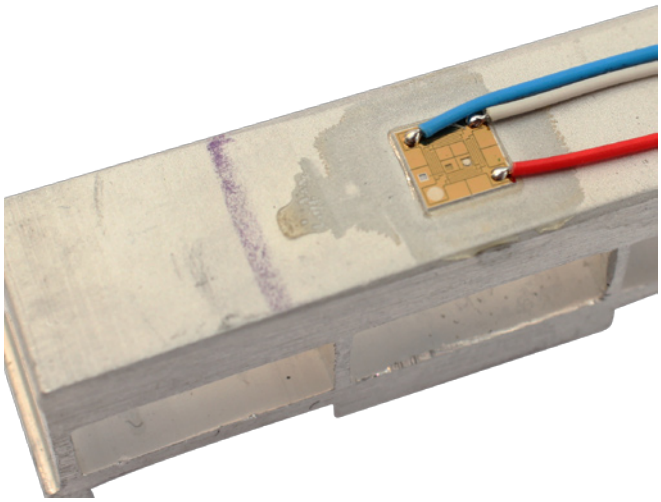
Beispiel: Temperaturkontrolle von Lebensmitteln oder unter Spannung stehender Teile

Beschleunigungssensor

Der Sensor selbst arbeitet meist mit **Piezo-Kristallen**, auf denen eine bestimmte Masse bei Beschleunigungen Druck ausübt. Die Kristalle liefern dann eine kleine Spannung, die aber ohne weitere Verstärkung nicht ausgewertet werden kann. Meist ist in den Sensoren deshalb gleich direkt die **Verstärkerschaltung** mit integriert. Diese gibt dann entweder zur Beschleunigung proportionale Spannungen aus oder stellt die Messwerte digital über ein Bussystem (z.B. I2C) dem Mikrocontroller zur Verfügung.

Beispiele: Gyroskop von Drohnen, Smartphones und Navigationsgeräten, Alarmanlagen von Motorrädern, Auslösung von Airbags

Dehnungsmessstreifen (DMS)



Auf eine **Kunststofffolie** sind **Metallbahnen** aufgebracht. Wird die Folie mitsamt den Metallbahnen gedehnt, wird deren **Leitungsquerschnitt geringer** und ihre **Leitungslänge länger**. Ihr elektrischer **Widerstand steigt** dadurch. Der Effekt tritt hauptsächlich bei Dehnung in Längsrichtung zu den Metallbahnen auf. DMS werden mit speziellen Klebstoffen auf die Bauteile geklebt, deren Dehnung oder Biegung erfasst werden soll. Sie sind selbst relativ kompakt, reagieren aber auf kleinste Längenänderungen von Oberflächen. Die Widerstandsänderung ist aber nur sehr gering und können von einem Mikrokontroller nicht erfasst werden, was eine Verstärkerschaltung notwendig macht.

Beispiele: Personenwaage, Kranwaage, Schwerlastwaage, Belastung von Stahlträgern, Verwindung von Achsen bei Drehmomenten

Lichtschranken

Eine **LED oder ein Laser** senden einen Lichtstrahl aus, der von einem **Lichtsensor** erfasst wird. Wird der Strahl unterbrochen weil z.B. eine Person hindurch geht, erhält der Lichtsensor kein Signal mehr und gibt ein Steuersignal aus.

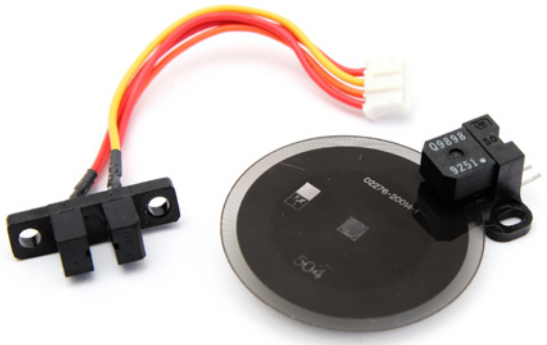
Es gibt zwei prinzipielle Arten von Lichtschranken:

<u>Einweglichtschranke</u>
Der Sensor besteht aus einem Sender, der den Lichtstrahl aussendet und einen Empfänger, der den Strahl überwacht und das Steuersignal liefert. Es sind **zwei getrennte Einheiten**, die sich gegenüber stehen und mehrere Meter voneinander entfernt sein können.

Beispiel: Fahrzeugzähler am Parkhaus

<u>Reflexlichtschranke</u>
Hier sind Sender und Empfänger **in einer Einheit untergebracht** und zeigen in die gleiche Richtung. Entweder reflektiert ein Spiegel den Lichtstrahl oder es werden reflektierende Gegenstände erkannt, die den Strahl zurückwerfen. Die Reichweite ist bei Reflexionslichtschranken geringer, da das Licht den doppelten Weg zurücklegen muss.

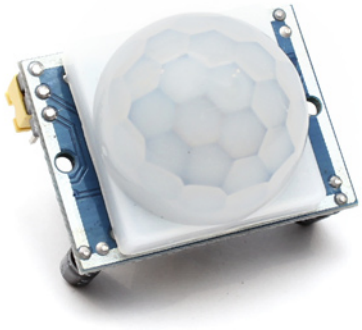
Beispiel: Berührungsloser Schalter an öffentlichen Wasserhähnen



Gabellichtschranken sind Einweglichtschranken, die aber nur eine sehr kurze Distanz überwachen und deshalb **in einem Gehäuse** untergebracht sind. Sie dienen als optische **Positionsschalter** bei Klappen, zur **Drehzahlmessung** oder zur **Positionserkennung** bei linearen oder rotorischen Bewegungen mithilfe von geschlitzten Streifen oder Scheiben.

Beispiel: Scrollrad bei Maus

Bewegungsmelder (PIR)

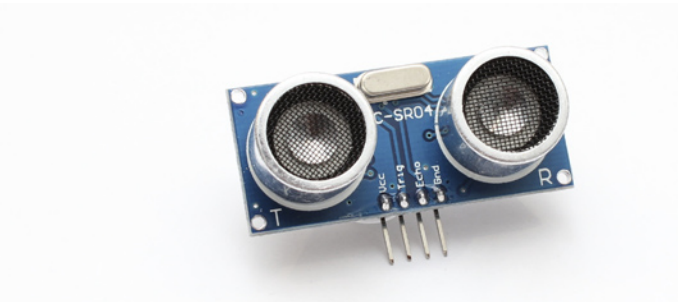


In den meisten Bewegungsmeldern ist ein **Infrarotsensor** verbaut, der üblicherweise aus zwei eng nebeneinanderliegenden Halbleiterelementen besteht, die auf **Wärmestrahlung** reagieren. Durch eine spezielle **Waben- oder Lamellenoptik** vor dem Sensor werden Änderungen im Einfallswinkel der Infrarotstrahlung, z.B. von einer vorbeigehenden Person, von den beiden Halbleitern erkannt.

Zum Sensor gehört auch immer eine **Auswerte-schaltung** mit einstellbarer Empfindlichkeit. Im Vergleich zu Meldern der Installationstechnik haben Bewegungsmelder für Mikrocontroller kein Netzteil und keinen Relaisausgang für 230V sondern geben nur ein Steuersignal an den Controller weiter.

Beispiel: Alarmanlagen, Lichtsteuerung

Ultraschall-Abstandssensor



Dieser Sensor beruht auf dem Echolot-Prinzip, wie es Fledermäuse nutzen oder es in der Seefahrt oder der Medizin zum Einsatz kommt. Ein Hochfrequenz-Schallgeber sendet **Ultraschall-Impulse** aus, die von einem davor befindlichem Objekt **reflektiert** und von einem „Mikrofon“ wieder empfangen werden. Aus der Zeit zwischen Senden und Empfangen ermittelt der Sensor den Abstand zum Objekt.

Beispiel: Kollisionserkennung, Anwesenheits-erkennung, Entfernungsmessung

Zudem gibt es **Sensoren** für viele weitere physikalische Größen, auf die aber nicht weiter eingegangen wird:

- Luftfeuchtigkeit
- Luftdruck
- Drucksensor (Druck/Unterdruck)
- Durchflussmesser
- Feuchtigkeit in Stoffen
- Gasmelder
- Kompass-Module
- Kapazitiver Näherungsschalter
- Induktiver Näherungsschalter
- Geigerzähler (Radioaktivität)
- Textiler Dehnungsmessstreifen
- Optischer Maussensor
- uvm...

AKTOREN

DC-Motoren

Gleichstrommotoren bestehen prinzipiell aus **mindestens zwei Spulen** um einen **drehbar gelagerten Magneten**. Ein auf der Achse montierter Kommutator polt die Spulen immer so um, dass ihr erzeugtes Magnetfeld den Magneten und die Achse in eine **Drehbewegung** versetzt.

Durch **Umpolen** der Anschlüsse kann die **Drehrichtung umgekehrt** werden. Über Pulsweitenmodulation kann die **Drehgeschwindigkeit** gesteuert werden.

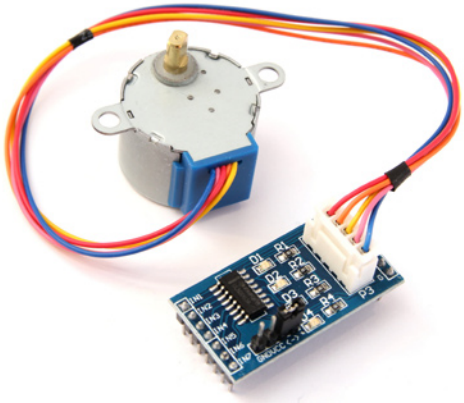
Der Motor eignet sich für kontinuierliche, schnelle Drehbewegungen, bei denen eine genaue Positionierung der Achse nicht notwendig ist.

Beispiele: Lüfter, Pumpen, allgemeine Antriebe, in Kombination mit Getrieben auch für größere Kräfte

Servomotoren

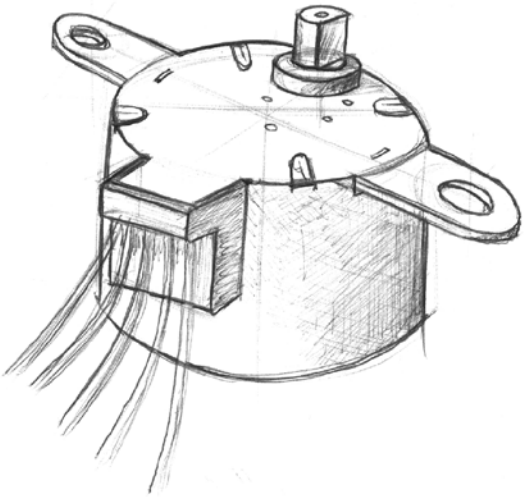
Die häufig im RC-Modellbau eingesetzten Aktoren besitzen ein **integriertes Getriebe** und eine eigene **Reglerelektronik**, finden aber aufgrund ihrer einfachen Nutzung auch häufigen Einsatz bei Arduino-Projekten. Servos können über einen digitalen IO-Pin mit **Pulsweitenmodulation** vom Mikrocontroller direkt am PWM-Eingang angesteuert werden. Das Puls-Pausenverhältnis des Signals gibt den Soll-Wert für den Winkel der Servo-Achse vor.

Schrittmotoren

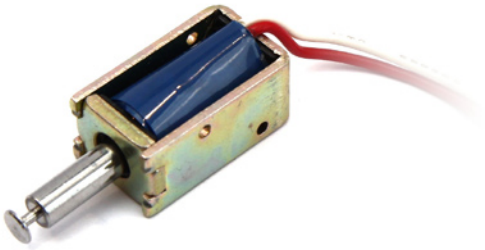


Durch ein spezielle Beschaltung der verbauten Spulen kann die Achse dieses Motors um einen **kleinen Winkel** verstellt werden. In Verbindung mit einem Getriebe sind damit sehr **genaue Positionieraufgaben** realisierbar. Für den Betrieb an Mikrocontrollern sind fertig aufgebaute **Treiberplatinen** verfügbar.

Beispiele: Drucker, 3D-Drucker, kleine Positionierantriebe, Neigung von Spiegeln in optischen Geräten, Kleinroboter



Elektromagnete



Wird ein elektrischer Leiter von Strom durchflossen, entsteht um ihn herum ein **Magnetfeld**. Diesen Effekt machen sich Elektromagnete zunutze, indem sie mit Spulen, die aus mehreren hundert oder tausend Wicklungen Draht bestehen, ein relativ starkes Magnetfeld erzeugen. Dieses kann eine ferromagnetische (magnetisierbare) Metallplatte oder einen Metallstift bewegen, der eine mechanische Funktion ausführt.

Beispiele: Türöffner, Gong/Klingel, Verriegelungen, Ventile, Relais und Schütze, Schrottkran

Konstantan/Widerstandsdraht

Ein Draht aus einer Kupfer/Nickel/Mangan Legierung mit relativ hohem elektrischem Widerstand, der über einen weiten Temperaturbereich gleich bleibt. Dadurch eignet sich das Material für die Anfertigung von Widerständen oder für Heizwicklungen.

Beispiel: Haartrockner, elektrische Fußboden-heizung, Bügeleisen, Öfen

Nitinol®

Eine spezielle **Nickel-Titan Legierung**, die dazu instande ist, sich seine Form zu „merken“ und diese nach einer Verformung durch Erwärmen wieder anzunehmen.

Als Draht kann das Material als **elektronischer Muskel** eingesetzt werden, der sich bei Anlegen einer Spannung zusammenzieht. Kleine lineare Bewegungen sind ohne weitere Mechanik realisierbar.

Beispiel: Medizinische Geräte, Modellbau

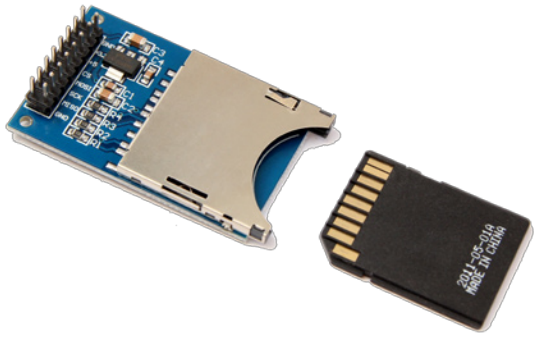
ERWEITERUNGEN FÜR MICROCONTROLLER

DCF-Empfänger

Mit diesem kleinen Modul und dieser großen Antenne können die Funksignale der Atomuhr bei Frankfurt empfangen und ausgelesen werden. Damit steht ein **hochpräziser Zeitwert** zur Verfügung, der sich sogar automatisch auf Sommer- und Winterzeit einstellt.

Beispiele: Zeitschaltprogramme, selbstgebaute Funkuhren

SD-Kartenleser



Ein SD-Kartenslot mit dem ein Mikrocontroller größere Datenmengen auf eine Speicherkarte schreiben oder lesen kann.

Beispiele: Aufzeichnung von Messwerten (Datenlogger), von Computer aufgespielte, individuelle Muster für Lichteffekte oder Melodien

Relaiskarte

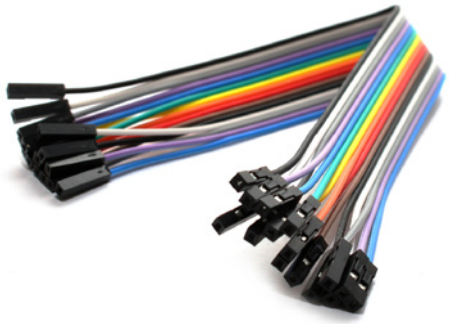
Als Bausatz oder als fertige Platine erhältlich, enthalten Relaiskarten alle notwendigen Bauteile um mit den schwachen Arduino-Signalen **größere Lasten** wie Motoren, Ventile, etc. zu steuern. Ebenso gibt es MOSFET-Platinen für die Anteuerung von Power-LEDs oder LED-Streifen.

Schnittstellen-Boards

Diese Boards erweiteren den Arduino um eine **LAN, WLAN** oder **BT Schnittstelle** um den Mikrocontroller in ein bestehendes Netzwerk einzubinden.

SONSTIGES

Steckboard-Verbinder



Diese Leitungen eignen sich durch ihre **Stift- bzw. Buchsenstecker** sehr gut für die schnelle Verdrahtung von Bauteilen auf dem Steckboard oder bei Mockups. Es gibt sie in unterschiedlichen Längen und verschiedenen Steckerkombinationen.

Batteriehalter

Sie besitzen entweder Fahnen zum Anlöten von Leitungen, Clips von 9V-Blockbatterien oder sind für die Bestückung auf Platinen vorgesehen. Die Zellen sind dabei immer in Reihe geschaltet, d.h. die Spannung wird erhöht. Sie lassen sich relativ einfach durch Schraublöcher an Gehäusen montieren.

Häufig benutzte Batteriegrößen:

Bezeichnung	Typ	Spannung (Batterie)	Spannung (Akku)
Micro	AAA	1,5V	1,2V
Mignon	AA	1,5V	1,2V
Baby	C	1,5V	1,2V
Mono	D	1,5V	1,2V
9V Block	9V	9V	8,4V

Ohmsches Gesetz:

Das ohmsche Gesetz beschreibt den **Zusammenhang zwischen Strom, Spannung und Widerstand** und ist damit Grundlage fast aller elektrotechnischen Berechnungen.

$$I = U / R$$

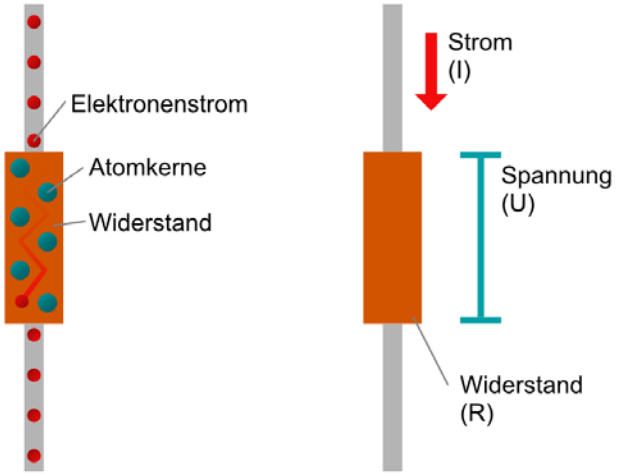
„Wenn ein Widerstand R an eine Spannung U angelegt wird, dann fließt durch ihn ein Strom I."

$$U = R \times I$$

„Ein Widerstand R ruft bei einem Strom I einen Spannungsabfall U hervor."

$$R = U / I$$

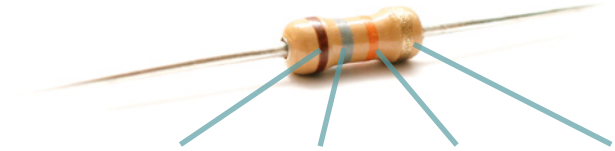
„Um bei einer Spannung U den Stromfluss auf einen Wert I zu begrenzen, muss ein Widerstand R verschaltet werden."



Rechenbeispiel:
Eine Heizwicklung von einem Motorradgriff hat im Betrieb einen Widerstand von 6 Ohm. Die Batteriespannung beträgt 12 Volt. Der zu erwartende Strom beträgt:

$$I = 12\ V / 6\ Ohm = 2\ A$$

Zeichen	Größe	Einheit	Erklärung
U	Spannung	V (Volt)	Spannung am Widerstand
I	Strom	A (Ampere)	Strom, der durch Widerstand fließt
R	Widerstand	Ω (Ohm)	Größe des Widerstandes



	Ring 1	Ring 2	Ring 3	Ring 4
Farbe	**Ziffer 1**	**Ziffer 2**	**Exponent**	**Toleranz**
Silber	-	-	-	+/- 10 %
Gold	-	-	-	+/- 5 %
Schwarz	-	0	x 1	-
Braun	1	1	x 10	+/- 1 %
Rot	2	2	x 100	+/- 2 %
Orange	3	3	x 1.000	-
Gelb	4	4	x 10.000	-
Grün	5	5	x 100.000	+/- 0,5 %
Blau	6	6	x 10^6	+/- 0,25 %
Violett	7	7	x 10^7	+/- 0,1 %
Grau	8	8	x 10^8	+/- 0,05 %
Weiß	9	9	x 10^9	

Widerstand-Farbcode Tabelle

Übliche Kohleschichtwiderstände werden mit **Farbringen** gekennzeichnet. Ring 1 bestimmt die erste Ziffer des Widerstandswertes, Ring 2 die zweite Ziffer und Ring 3 den Zehner-Exponent.

Der vierte Ring sagt etwas über die **Genauigkeit** des Bauteils aus, also wie weit der tatsächliche Wert maximal vom aufgedruckten Wert abweichen kann. Je nach Anwendung ist eine gewisse Präzision des Wertes erforderlich, z.B. bei Messwiderständen.

Bei SMD-Widerständen sind die ersten beiden Ziffern und der Exponent auf das Gehäuse aufgedruckt.

Beispiel:
Der über der Tabelle abgebildete Widerstand hat einen Wert von 18.000 Ω bzw. 18 kΩ

Berechnung von Vorwiderständen:

Viele elektronische Bauelemente dürfen nur mit einer gewissen Spannung betrieben werden, da sie ansonsten beschädigt oder überlastet werden. Kleine Spannungsunterschiede bei kleinen Strömen können mit Widerständen erzeugt werden. Man spricht dann von einem Vorwiderstand, an dem der zu hohe Teil der Spannung abfällt. Dazu wird das ohmsche Gesetz genutzt:

$$R = U/I$$

Beispiel: Bei einer USB-Leuchte soll eine weiße LED mit den Nennwerten 3,8V / 20mA an der genormten USB-Spannung von 5V betrieben werden.

Als am Widerstand abfallende Spannung ergibt sich:

$$5V - 3,8V = 1,2V$$

Daraus kann zusammen mit dem durch Widerstand und LED fließende Strom der Widerstand berechnet werden:

$$R = 1,2V / 0,020A = 60\ Ohm$$

Widerstände gibt es nur leider nicht in allen Werten! Übliche Widerstandswerte sind:

1 - 2,2 - 3,3 - 4,7 - 6,8 - 10 - 22 - 33 - .... Ω

Man wählt den **nächsthöheren Wert**, damit die LED nicht überlastet wird. Für die LED muss also ein Vorwiderstand mit 68 Ohm verwendet werden. Feinere Abstufungen gibt es bei Widerständen mit niedrigerer Toleranz.

Widerstände lassen sich auch **kombinieren**: Mit sechs 10-Ohm-Widerständen hintereinander (in Reihe) könnte man beispielsweise 60 Ohm realisieren.

Verlustleistungsberechnung:

Widerstände „fackeln“ die überschüssige Spannung ab, indem sie elektrische Energie in **Wärme** umwandeln.

Dem sind natürlich Grenzen gesetzt. Übliche Kohleschicht-Widerstände können maximal 0,25 Watt „verbraten“.

Man sollte deshalb kontrollieren, ob der Widerstand auch die Verlustleistung aushält:

$$P = U \times I$$

Am Beispiel unserer USB-Leuchte:

$$P = 1,2V \times 0,020A = 0,024W$$
$$0,024W < 0,25W$$

>> Mit Widerstand möglich!

Gleiche LED mit 1 kOhm Vorwiderstand an 24V:

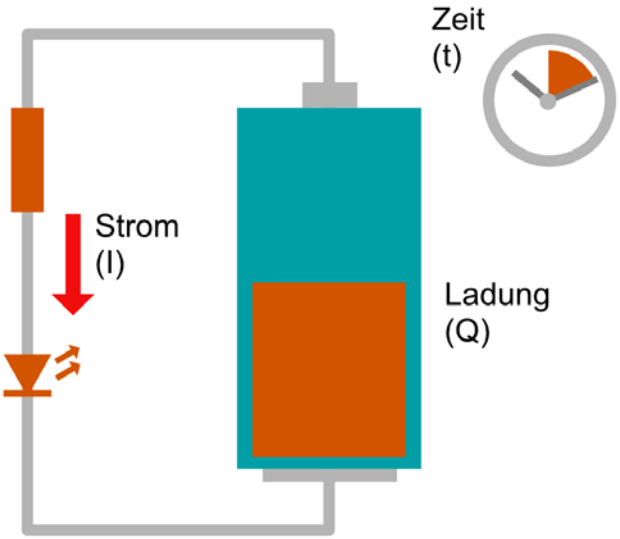
$$P = 20,2V \times 0,020A = 0,404W$$
$$0,404\ W > 0,25W$$

>> Widerstand wird durchbrennen!

Laufdauer im Batteriebetrieb:

Möchte man berechnen, wie lange ein Gerät mit einem Satz Batterien oder einem verbauten Akku betrieben werden kann, verwendet man die Formel für die elektrische Ladung:

$$t = Q/I \qquad Q = I*t \qquad I = Q/t$$



Beispiel: Unsere USB-Leuchte soll mit einem Akku auch unabhängig vom PC funktionieren und mit einer Klammer an einem Buch befestigt als Leselicht dienen. Wir wollen, dass die Leuchte 4 Stunden Licht zum Lesen liefert, bis sie wieder am USB-Port geladen werden muss.

Da wir die nötige Kapazität des Akkus wissen wollen, verwenden wir die Formel

$$Q = I * t$$

$$Q = 0{,}020\ A * 4\ h = 0{,}080\ Ah$$

$$0{,}080\ Ah = \underline{80mAh}$$

Jetzt können wir noch berechnen, wie lange es dauert, bis der Akku vom USB-Port wieder voll geladen werden kann. Angenommen, der Akku hält den bei USB-Ports üblichen Ladestrom von 500mA aus, kann man die Ladezeit wie folgt berechnen:

$$t = Q / I$$

$$t = 0{,}080\ Ah / 0{,}500A = 0{,}16h$$

$$0{,}16h / 60 = \underline{9{,}6\ Minuten}$$

Wegen Ladeverlusten - der Akku erwärmt sich beim Laden - wird die reale Ladezeit etwas länger dauern.

Zeichen	Größe	Einheit	Erklärung
Q	Ladung	Ah (Amperestunden)	Ladung im Akku/in Batterie
I	Strom	A (Ampere)	Strom, der den Akku lädt/entlädt
t	Zeit	h (Stunden)	Zeit zum Laden/Entladen des Akkus

Berechnung des Stromverbrauchs:

…oder besser der elektrischen Arbeit. Sie ist das Produkt aus Leistung und der Zeit, in der die Leistung erbracht wird:

$$W = P \times t$$

Beispiel: Berechnung des Jahresverbrauchs durch die Standby-Leistung eines Netzteils von 0,3 Watt:

$$W = 0{,}3W \times 24h \times 365d = 2628\ Wh$$

$$2628\ Wh = 2{,}628\ kWh$$

Zeichen	Größe	Einheit	Erklärung
W	Arbeit	Wh (Wattstunden)	Stromverbrauch
P	Leistung	W (Watt)	Leistungsaufnahme des Verbrauchers
t	Zeit	h (Stunden)	Betriebszeit des Verbrauchers

Mindestquerschnitte für Leitungen

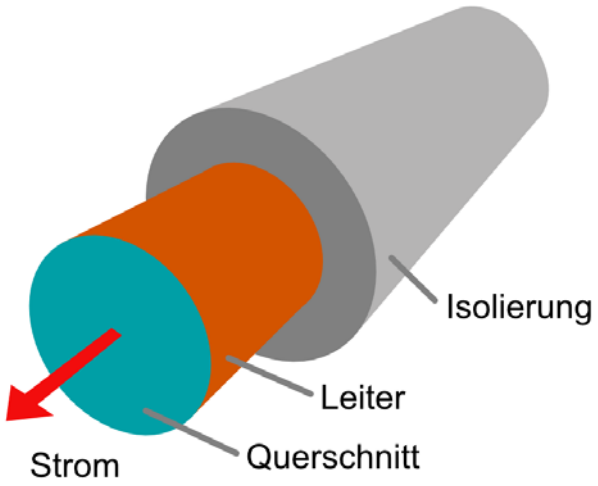
Sobald Strom durch einen Leiter fließt - ausgenommen Supraleiter - erwärmt dieser den Leiter und das umgebene Isolationsmaterial. Außerdem liegt durch den mit der Temperatur steigenden Widerstand der Leitung eine geringere Spannung am anderen Ende an.

Bei zu hohen Strömen kann dies zum Schmelzen der Isolierung, Kurzschlüssen, Lösen von elektrischen Verbindungen und zum Durchbrennen des Drahtes bis zu gefährlichen Bränden führen. Darum muss der Leitungsquerschnitt an den zu erwartenden Betriebsstrom angepasst sein.

Eine richtige Dimensionierung der Leitungen spart zudem Kupfer, Gewicht und Platz in Gehäusen.

Die Tabelle rechts ist eine kleine Hilfe zur Bestimmung des Leiterquerschnittes. Besondere Umstände wie eine höhere Umgebungstemperatur oder hohe mechanische Belastung der Leitung kann auch größere Querschnitte notwendig machen.

Ebenfalls einzuhalten sind die maximalen Spannungswerte einer Leitung, die von Material und Stärke der Isolation der Adern abhängig sind.



Maximaler Strom	Querschnitt
1 A	0,05 mm²
2 A	0,14 mm²
4 A	0,25 mm²
9 A	0,5 mm²
12 A	0,75 mm²
15 A	1,0 mm²
18 A	1,5 mm²
26 A	2,5 mm²
34 A	4 mm²

Allgemeine Richtwerte für Mindestquerschnitte von Einzeladern bei Raumtemperatur (30°C) nach DIN VDE 0100 Teil 430