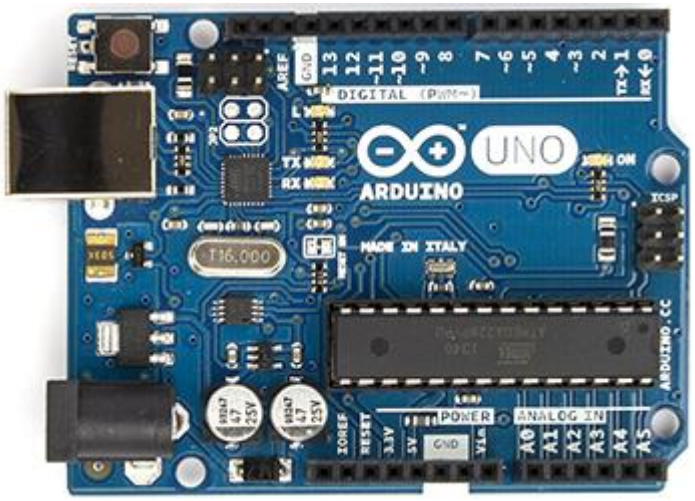


Reinhard Atzbach

Vom Bit zum Objekt – vom Transistor zum Arduino

Ein Streifzug durch die Digitaltechnik
mit mehr als 100 Versuchen und Aufgaben
als Kursunterlage oder zum Selbststudium

Was Sie erwartet

Die vorliegende Aufgabenreihe umfasst Versuche, die sich mit verschiedenen Zwiebelschalen der Computertechnik beschäftigen. Die Aufgaben wurden mit einer Informatik-AG an einem Gymnasium Klasse 10 (E-Phase) praktisch erprobt, dabei wurde das E-Book ins Schulnetz eingestellt und die Schülerinnen und Schüler arbeiteten mit dem von mir zusammengestellten Materialset in individuellem Tempo.

Die notwendige Hardware umfasst einen Arduino-Mikrocontroller am USB-Port, ein Breadboard und diverse Elektronik-Bauteile. Der Arduino kommt am Anfang nur als Spannungsquelle zum Einsatz, später wird er auch programmiert. Vom Transistor als Schalterelement führt der Kurs über Gatter-ICs bis zum Mikroprozessor, zur Assemblerprogrammierung und Hochsprachen-Compiler. Zum Einsatz kommen dabei mehrere kostenlose Programme: ein Logiksimulator, ein CPU-Simulator und schließlich der reale Arduino und der zugehörige C-Compiler.

Dies Buch ist weder ein Arduino-Experimentierbuch noch eine systematische Einführung in die Informatik. Es versucht nur, in möglichst praxisorientierter Herangehensweise die Zwiebelschalen der Digitaltechnik etwas transparenter zu machen.

Februar 2015

Reinhard Atzbach

Kommentare und Korrekturen an

r.atzbach@web.de

Verwendete Software

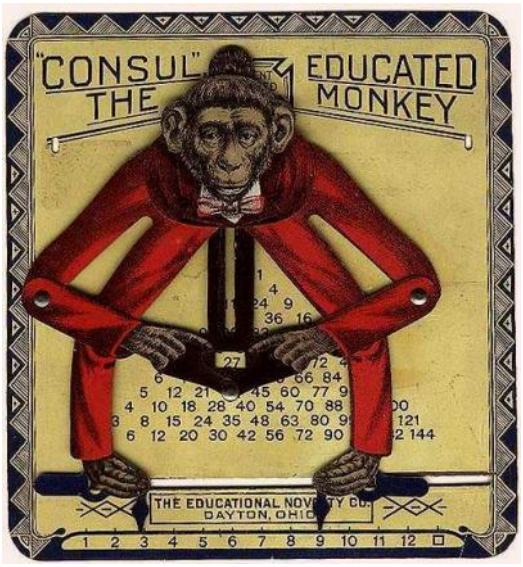
Entpacken Sie jeweils die ZIP-Datei in einen Ordner auf dem Desktop.
LogikSim bei http://logiksim.dbclan.de/
WinBreadboard-Demo bei http://www.yoeric.com/breadboard.htm
Codecruncher bei http://atzbach.net/arduino/codecruncher.zip
CPUSim (deutsch) bei http://atzbach.net/arduino/cpusim.zip
CPUSim (englisch) bei http://softwareforeducation.com/
Arduino-Software bei http://arduino.cc
Materialpaket zu diesem Heft bei http://atzbach.net/arduino/Material.zip
Außerdem interessant:
Simulator for Arduino bei http://www.virtronics.com.au/Simulator-for-Arduino.html
Fritzing bei http://fritzing.org (wurde für viele Abbildungen verwendet)

Verwendete Hardware

ab S. 10: Arduino UNO oder kompatibles Board
ab S. 10: Kurzes Breadboard und Kabel–Satz („Dupont Kabel“)
ab. S.10: Kleine Kombizange
ab S. 10: ggf. 2 Zenerdioden
ab S. 10: 2 Transistoren NPN und 1 Transistor PNP
ab S. 10: je 20 Widerstände 220 Ω, 10 kΩ, Anschlussdrähte möglichst Ø 1 mm
ab S. 10: je 3 rote, gelbe und grüne LEDs
ab S. 18: ggf. Logikgatter 7408, 7402, 7432, 7486
ab S. 42: ggf. 9V-Block mit DC-Hohlstecker außen Ø 5,5 mm / innen Ø 2,1 mm
ab S. 43: ggf. 2 Prototyping Shields für Arduino mit Mini-Breadboard
ab S. 10: Draht Ø 1 mm isoliert für kurze Steckbrücken auf Breadboard
ab S. 43: 3 Mikrotaster für Breadboard
ab S. 43: 1 Fotowiderstand ca. 10KΩ
ab S. 49: 1 Potentiometer oder besser Trimmpotentiometer für Breadboard 10 kΩ
ab S. 51: ggf. Kondensator ca. 22µF
ab S. 54: ggf. 1 AA-Batterie 1,5 V
ab S. 57: 1 Piezo-Beeper (aktiv) oder kleiner Computerlautsprecher
ab S. 61: 1 7Segmentanzeige 3361AS 12 mm hoch 3 Stellen, gemeins. Kathode
ab S. 63: ggf. 1 Tilt-Switch
ab S. 63: ggf. 1 Heißleiter 10KΩ
ab S. 64: 1 Ultraschall-Entfernungsmessser HC-SR04
ab S. 65: 1 8x8-Punktmatrix mit MAX 7219
ab S. 67: ggf. 1 Breakout-Baustein mit 2 Relais
ab S. 70: 1 Arduino Smart-Robot Car Bausatz
ab S. 70: 1 Mini-Breadboard
ab S. 68: 1 L9110S Dual-DC Motor-Treiber Board
ab S. 73: 1 Infrarot-Fernbedienung NEC-kompatibel, Kit aus Sender und Empfänger

In gängigen Arduino-Sets ab 30 € sind viele der aufgeführten Teile enthalten. Die Hardware ist überwiegend auch einzeln bei Ebay auffindbar, wenn man nach „Arduino“ sucht. Mit „ggf.“ gekennzeichnete Teile sind zur Not verzichtbar.

Direkte Lieferungen aus Hongkong oder China über Ebay oder direkt von tayda.com (Bauteile), dx.com oder banggood.com (Arduino-Zubehör) sind preiswert, benötigen in der Regel aber drei Wochen Lieferzeit. Zollbefreit sind Lieferungen bis 22 €.



Level 0:
Automaten

In diesem Kapitel erfahren Sie, ...

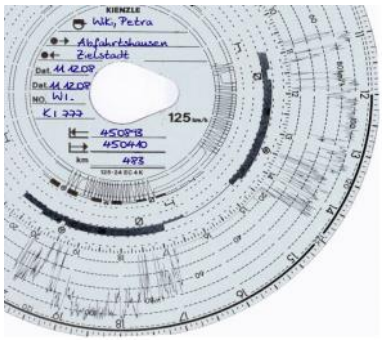
- dass die Grundfunktion einer Maschine im Regeln oder Schalten besteht
- was der Unterschied zwischen analogen und digitalen Steuerungssystemen ist
- welche mechanischen, elektromechanischen und elektronischen Arten von Schaltern es gibt

Regeln (analog) und schalten (digital)

Bei vielen Zuständen reicht eine ungefähre Beschreibung („Heute ist es etwas kühl“), und viele Vorgänge kann man stufenlos regeln („Dreh die Heizung etwas höher“). Auch Geräte und Maschinen können so funktionieren: Die Temperaturkurve in einem Museum wird analog aufgezeichnet, Der Fliehkraftregler einer Dampfmaschine sorgt stufenlos für eine gleichmäßige Laufgeschwindigkeit. Ein Plattenspieler setzt die Kurven der Plattenrille zahlenfrei in akustische Schwingungen um. Und das Telefon machte aus Schallwellen analoge elektrische Schwingungen. Ohne Zahlen geht‘s auch - eben **analog**.

*Ein Fliehkraftregler arbeitet **analog**, d.h. er öffnet oder schließt stufenlos ein Ventil.*

*Edisons Phonograph setzte Schwingungen von Tönen in **analog** geformte Wellen um.*



*Die Kurve des Fahrtenschreibers ist **analog** zur Geschwindigkeit eines Lastwagens.*

Analog- und Digitalrechner

Sogar zum Rechnen setzte man früher häufig analoge Skalen ein. Durch Nebeneinanderlegen von Skalenabschnitten konnte man mit passabler Genauigkeit dividieren und sogar Wurzeln ziehen.

Doch das Wesen von Zahlen ist **digital**: stufenlos, exakt und ohne Zwischenwerte. Schon 1623 hatte der Theologe und Mathematiker Schickard die erste Addiermaschine konstruiert. Wenn man in der abgebildeten Addiermaschine das Rad der Einerstelle immer weiterdreht, dann dreht sich die Zehnerstelle nicht gleichmäßig mit, sondern sie springt irgendwann um eine Position weiter.

So benutzte man Zahnräder, Staffelwalzen und Sprossenräder, um die zehn Ziffern des Dezimalsystems mechanisch abzubilden. Gottfried Wilhelm Leibniz (1646 - 1716) konstruierte bereits eine DIvidiermaschine, und zweihundert Jahre später hatten die Urenkel dieser Maschine gelernt, ohne menschliche Eingriffe automatisch zu dividieren.

***Analogrechner** funktionierten mit verschiebbaren Skalen.*

***Digitalrechner** benutzten früher Zahnräder und Hebel. Diese Addiermaschine funktioniert ähnlich wie die 1623 von Schickard konstruierte.*

Mit Löchern kann man schalten

Für viele Anwendungen der Digitaltechnik genügten auch früher schon zwei Zustände: ein und aus.

Automaten, etwa Spieluhren, deren Klangzungen von Stiftwalzen angerissen wurden, gab es schon sehr lange. Um 1900 aber wurden auf Lochbänder ganze Orchesterstücke für Kirmesorgeln aufgezeichnet. Die Lochbandsteuerung hatte sich der französische Textilunternehmer Joseph-Marie Jacquard ausgedacht, um Muster in seine Stoffe zu weben. Aber bald hatten die Löcher sich noch weiter ausgebreitet: Und im Jahre 1870 wertete Hermann Hollerith die auf Karten gelochten Datensätze der amerikanischen Volkszählung maschinell aus: Die Daten jedes Einwohners der USA wurde auf einer Lochkarte kodiert, und Hollerith lieferte die Maschinen, die einen Stapel dieser Karten nach beliebigen Kriterien sortieren und die Anzahl der ausgeworfenen Karten auszählen konnte.

Eine Spieluhr mit Stiftwalze

Lochband eines Webstuhls

Hollerithmaschine

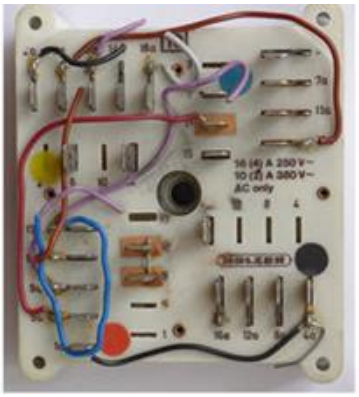
Schalter kann man programmieren

Mit dem elektrischen Strom kam im 20. Jahrhundert neuer Schwung in die Digitaltechnik, da in der Elektromechanik die Logik von der Kraftübertragung entkoppelt werden konnte. Kombiniert man viele Schalter, so lassen sich komplizierte Zähl– und Steuerungssysteme aufbauen. Dazu braucht man keinen kompletten Computer, es reicht eine Steuerungseinheit, die Motoren und Elektromagnete schaltet. Vollautomatische Waschmaschinen der 1950er Jahre wurden von Programmscheiben gesteuert, die sich langsam drehten und dabei elektromechanische Schalter für Motoren, Ventile und Pumpen betätigten.

Der Programmschalter einer Waschmaschine: Ein Motor dreht langsam eine Scheibe.

Betrachtet man die Unterseite der Scheibe, so entdeckt man Rillen und Nocken.

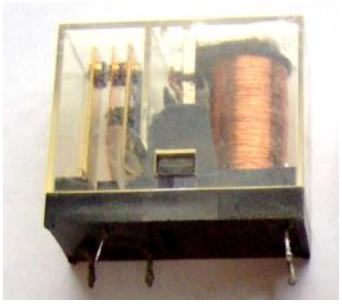
Im Gehäuse darunter befinden sich Taster, die von den Nocken herabgedrückt werden.



Auf der anderen Seite sind die Kabel der Motoren, Pumpen und Ventile angeschlossen.

Schalter werden immer kleiner

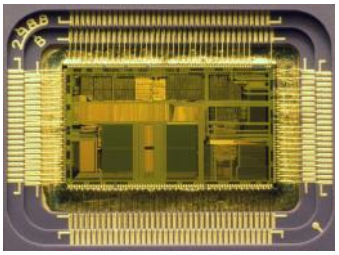
Auch Computer könnten (elektro-)mechanisch funktionieren, wie die auf S. 30 abgebildete Zuse Z3 beweist. Aber die Lochbleche von Konrad Zuses erster Maschine Z1 verhakten sich häufig. Mechanische Schalter erfordern Kraftanwendung, sie arbeiten vergleichsweise langsam, sie sind groß und beim Hintereinanderschalten störanfällig. Schon Leibniz hatte seine Dividiermaschine nicht zum Laufen bekommen und Charles Babbage war mit seiner Analytical Engine nicht an der Logik, wohl aber an der Mechanik gescheitert. Selbst die Relais und Elektronenröhren, durch die man die Zahnräder später ersetzte, reagierten langsam, sie nahmen Platz weg und brauchten viel Energie. Richtig in Schwung kam die Entwicklung der Digitaltechnik erst mit der Erfindung des Transistors (patentiert 1948) und später der ICs (integrierten Schaltkreise). Heutige CPUs (Mikroprozessoren) vereinigen Milliarden (Intel Core i7: 2.270.000.000) von Schaltern auf einem Chip: Schalter, die Zahlen speichern und unvorstellbar schnell verknüpfen können.



Relais: Mit einem Elektromagneten wird elektromechanisch ein Schalter betätigt.

Röhren: Verstärker für Radios, können auch als Schalter im Computer verwendet werden.

Transistoren: Der mittleren Pin schaltet den Stromdurchfluss vom linken zum rechten.



IC: Millionen Transistorschaltungen en auf einem Chip.

Analog-Digital-Analog-Wandler

Ein Problem blieb: Datenverarbeitung geschieht heute so gut wie ausschließlich digital. Messwerte wie Zeit oder Temperatur oder Wasserfluss ändern sich aber stufenlos. Also brauchen wir Vorrichtungen, die analoge Messwerte in digitale umwandeln und digitale Signale in analoge Aktionen.

Das Prinzip ist nicht neu: In jedem Wasserzähler sitzt ein kleines Schaufelrad. Das Wasser fließt gleichmäßig durch die Leitung, aber die Anzeige erfolgt digital. Wenn man an das Schaufelrad eine Lichtschranke anbaut, kann ein Computer dessen Umdrehungen zählen. Umgekehrt könnte der Computer das abgegebene Wasser dosieren, indem er das Schaufelrad blockiert oder freigibt. Das Wasser fließt analog, aber die Dosierung erfolgt digital.

Ein mechanischer Wasserzähler

Die digitale Variante

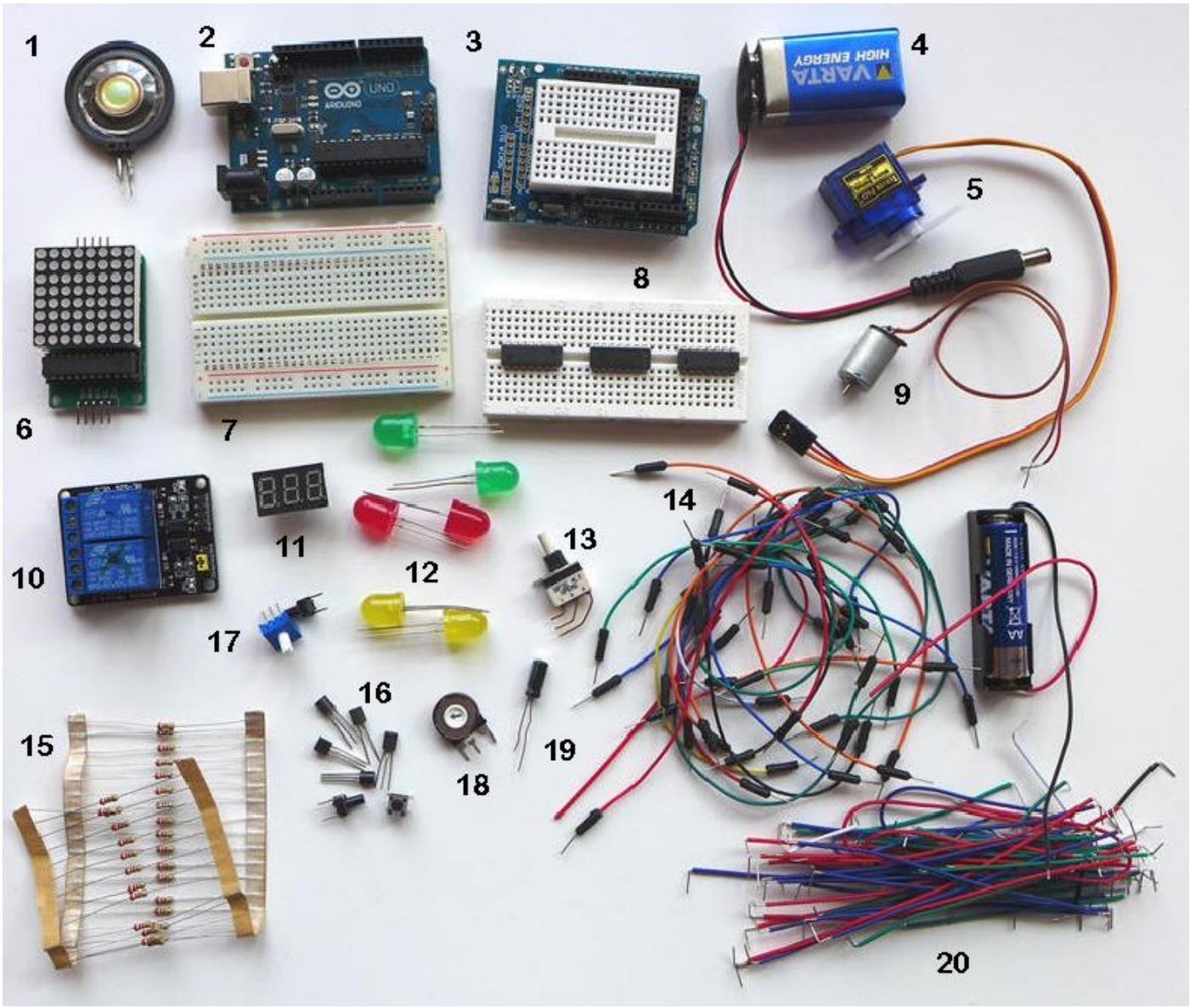
Level 1:
Stromkreise und Schalter

In diesem Kapitel erfahren Sie, ...

- welche Funktion verschiedene elektronische Bauteile haben
- wie man die Ringe auf den Widerständen liest
- dass Dioden den Strom nur in einer Richtung durchlassen
- dass man in einem Transistor mit dem mittleren Pin (Basis) den Durchfluss zwischen den anderen beiden Leitungen (Kollektor und Emitter) steuern kann
- dass Transistoren in Stromkreisen als Verstärker oder Schalter fungieren
- wie man Transistoren zu UND-, ODER- und NICHT-Schaltungen kombiniert

Bauelemente

Schauen wir uns nun einmal einige der Teile an, mit denen wir die Welt der Digitalelektronik erforschen wollen:



In der Abbildung finden Sie: Siebensegmentanzeige, Batterie, Breadboard, Kabel, Lautsprecher, Leuchtdioden (LED), LED-Matrix, Mikrocontroller, Gleichstrommotor, ICs, Potentiometer, Relais, Schalter, Servo, Steckbrücken, Transistor, Verbindungskabel, Widerstände

Aufgaben

1. Finden Sie für möglichst viele der in der Abbildung mit Nummern bezeichneten Bauteile heraus,
   - wie das Bauteil heißt,
   - wozu es eingesetzt wird und wie es sich verhält,
   - durch welches Symbol es in Schaltplänen dargestellt wird,
   - ob es analog oder digital arbeitet,
   - was man bei seinem Einsatz beachten muss.

Widerstände

Einige der auf der vorigen Seite abgebildeten Bauteile sind Widerstände. Den Widerstand eines Bauteils gibt man in Ohm (Ω) an. Man unterscheidet

- feste Widerstände, die den Stromfluss an einer bestimmten Stelle des Schaltkreises begrenzen, damit kein Bauteil beschädigt wird,
- variable Widerstände, die durch Licht oder Temperatur beeinflusst werden und als Sensoren eingesetzt werden,
- variable Widerstände, die eingesetzt werden, um einen Aktor zu beeinflussen, z.B. um die Lautstärke eines Lautsprechers zu steuern.

Feste Widerstände sind durch vier öder fünf farbige Ringe kodiert. Der goldene (5%) oder silberne (10%) Ring gibt die Toleranz des Bauteils an.

Halten Sie den Widerstand so, dass der goldene oder silberne Ring nach rechts zeigt, so geben die anderen drei Ringe von links nach rechts den Widerstandswert an. Dabei ist bei einem beige gefärbten Widerstand mit vier Ringen der erste Ring und der zweite Ring ein Code für eine Ziffer, der dritte ein Code für die Anzahl der Nullen, die an diese beiden Ziffern noch angehängt werden müssen.

Wenn Sie einen Widerstand mit grünem (4), violettem (7), orangefarbenem (3) und goldenem (5%) Ring ausmessen, dann müssten Sie zwischen 44650 und 49350 Ohm erhalten (47000 Ohm ±5%).

Ein orange (3), schwarz (0), braun (1) und silbern (10%) beringter Widerstand müsste zwischen 270 und 330 Ohm liefern.

Bei einem blau eingefärbten Widerstand mit fünf Ringen stehen drei Ringe für Ziffern, der vierte für die Anzahl der Nullen und der fünfte Ring für die Toleranz.

Farbkodierung von Widerständen mit 4 Ringen

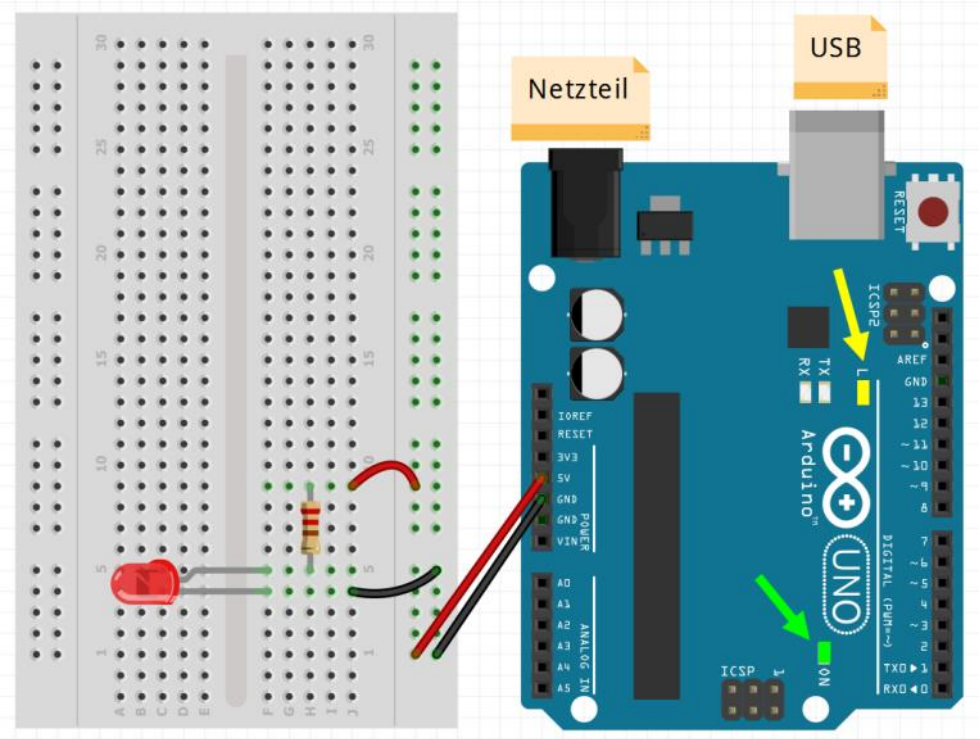
Farbe		Widerstandswert in Ω			Toleranz
		1. Ring (Zehner)	2. Ring (Einer)	3. Ring (Multiplikator)	4. Ring
„keine“	×	—	—	—	±20 %
silber		—	—	10^{-2} = 0,01	±10 %
gold		—	—	10^{-1} = 0,1	±5 %
schwarz		—	0	10^{0} = 1	—
braun		1	1	10^{1} = 10	±1 %
rot		2	2	10^{2} = 100	±2 %
orange		3	3	10^{3} = 1.000	—
gelb		4	4	10^{4} = 10.000	—
grün		5	5	10^{5} = 100.000	±0,5 %
blau		6	6	10^{6} = 1.000.000	±0,25 %
violett		7	7	10^{7} = 10.000.000	±0,1 %
grau		8	8	10^{8} = 100.000.000	±0,05 %
weiß		9	9	10^{9} = 1.000.000.000	—

Aufgaben

1. Bestimmen Sie die in Ihrem Set enthaltenen festen Widerstände.
2. Messen Sie, sofern vorhanden, die Widerstände mit einem Messgerät aus. Führen Sie auch Messungen an den variablen Widerständen (Potentiometer, Fotozelle, ...) durch und bestimmen Sie ihren Widerstand unter verschiedenen Bedingungen.

Schaltglieder

Höchste Zeit, dass etwas passiert:

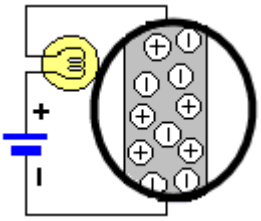


Aufgaben

1. Legen Sie Ihren Arduino und Ihr Breadboard nebeneinander wie abgebildet.
2. Schließen Sie den Arduino über das USB-Kabel an den Computer an. Dann sollte die grüne ON-LED leuchten und die gelbe L-LED blinken (Pfeile).
3. Der Arduino wird während des Programmierens vom Computer mit 5V Strom versorgt. Diese Spannung stellt er dem 5V-Anschluss in der linken Steckerleiste auch für Ihre Schaltung zur Verfügung. Einfache Schaltungen wie diese arbeiten mit dem Strom aus dem USB-Anschluss. Nur wenn Sie den Arduino unabhängig vom Computer betreiben wollen oder eine höhere Spannung benötigen, brauchen Sie eine Batterie oder ein zusätzliches Netzteil am anderen Stecker des Arduino.
4. Trennen Sie den Arduino vom Computer und bauen Sie die Schaltung auf.
5. Die Löcher Ihres Breadboards sind unterhalb der Plastikdecke mit Metallschienen verbunden. Diese verlaufen auf den beiden Seiten in einer Leiste von ganz oben nach ganz unten. In den beiden mittleren Bereichen sind jeweils die fünf nebeneinander liegenden Löcher verbunden. In der gezeigten Verkabelung ist also der obere Anschluss des Widerstands mit +5V auf dem Arduino verbunden, das untere Bein der LED mit GND (Minuspol, Masse).
6. Die LED muss mit dem längeren Bein (Anode) zur Plus-Seite, mit dem kürzeren Bein (Kathode) zur Minusseite hin eingesteckt werden.
7. Damit die LED nicht durchbrennt, muss sie immer über einen Vorwiderstand in den Stromkreis eingebaut werden. Der richtige Widerstand ist hier 220 Ω.
8. Schließen Sie den Arduino an einen USB-Port an. Die LED sollte leuchten.

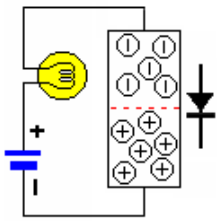
Was bitte ist ein Halbleiter?

Zwischendurch ein wenig Theorie: Ein Stromkreis besteht aus einer Spannungsquelle, einem Verbraucher (Widerstand) und Verbindungsdrähten aus einem Strom leitenden Material. Stellen wir uns die Verbindung vor wie ein Geflecht von Atomen. Einige davon sind **negativ geladene Ionen**, sie haben ein oder mehrere Elektronen zu viel, Andere haben ein Elektron zu wenig, das sind die **positiv geladenen Ionen**. Legt man nun an die Verbindung eine Spannung an, so herrscht an der Seite des Pluspols ein Mangel an Elektronen und am Minuspol ein Überschuss. Bei ausreichender Spannung wird irgendwann ein Ion an der Minuspol-Seite des Kabels ein Elektron an seinen Nachbarn in Richtung Pluspol abgeben. Dieses Ion braucht nun wieder ein Elektron, und das bekommt es von seinem anderen Nachbarn. Schließlich hüpfen immer mehr Elektronen von der Minuspol-Seite zur Pluspol-Seite. Strom fließt, die Lampe brennt!

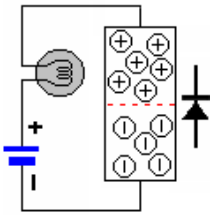


Die Diode

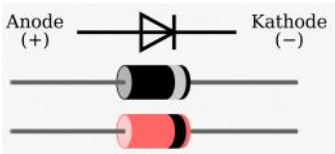
Irgendwann gelang es Wissenschaftlern, durch fertigungstechnische Tricks, zwei neue Materialien herzustellen: Eins, das nur positiv geladene, und ein anderes, das nur negativ geladene Ionen enthält. Diese beiden Materialien kombinieren wir und bauen sie in den Stromkreis ein. Nehmen wir an, der Pluspol der Batterie liegt an der Seite mit den negativen Ionen. Dann werden wir feststellen: Die Elektronen der negativen Ionen können bequem auf die positiven Ionen überspringen, und der Strom kann fließen.



Wenn wir jedoch die beiden Hälften unseres Materials vertauschen, werden wir feststellen: Es fließt kein Strom, egal wie hoch wir die Spannung drehen! Der negative Pol liegt an der negativen Seite unseres Materials an, deshalb können die Elektronen nicht eindringen, da für sie dort kein Platz ist.



Elektronische Bauelemente, die den Strom nur in einer Richtung durchlassen, in der anderen dagegen nicht, nennt man **Halbleiter**. Die einfachste Form eines Halbleiters ist die **Diode**. Sie lässt nur dann Strom durch, wenn ihre **Anode** mit dem Pluspol der Stromquelle verbunden ist und ihre **Kathode** mit dem Minuspol. In Elektronikschaltungen werden Dioden oft eingesetzt, um Wechselstrom in Gleichstrom zu verwandeln. Die Kathode ist auf dem Bauteil durch einen Ring markiert und im Schaltbild durch einen Balken.

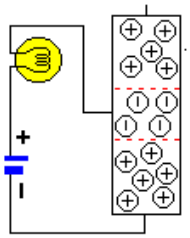


Aufgaben

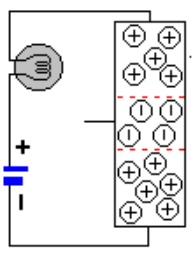
1. Sehen Sie sich im Wikipedia- Artikel http://de.wikipedia.org/wiki/Diode die Animation zur Diode an.

Der Transistor

Eine besonders raffinierte Variante der Diode ist der **Transistor**. Er hat nicht nur zwei, sondern drei Schichten mit Ionen und entsprechend drei Anschlüsse.



Schließen wir den Transistor mit dem mittleren und einem äußeren Bein an eine Stromquelle an, so leuchtet die Lampe. Das war auch nicht anders zu erwarten, da die Schichten, die in unserem Stromkreis liegen, wie die Diode aufgebaut ist.



Schließen wir aber die beiden äußeren Beine des Transistors an Plus- und Minuspol an und lassen das mittlere Bein unverbunden, so wird die Glühbirne nicht leuchten. Mindestens zwei Schichten sind wie eine sperrende Diode angeordnet.

Wir bauen ein Tor ein

Den Stromkreis der vorigen Seite wandeln wir nun ab, indem wir zusätzliche Bauteile hinzufügen, mit denen man den Stromkreis öffnen oder schließen kann.

Aufgaben

1. Zwischen die 5V-Leitung und den Widerstand fügen wir zunächst eine Diode ein. Wir können sie mit dem Markierungsring in Richtung zur Minusseite (Abb. 1) oder zur Plusseite des Stromkreises (Abb. 2) einbauen. Was macht die LED?
2. Kombinieren Sie zwei Dioden. (Abb. 3) Es gibt vier Möglichkeiten, die beiden Dioden hintereinander zu schalten. In welchen Fällen brennt die LED?
3. Setzen Sie nun statt der beiden Dioden einen Transistor ein. Verwenden Sie einen NPN-Transistor (z.B. Typ BC547). Setzen Sie den Transistor mit der abgeflachten Seite nach links ein (Abb. 4). Was macht die LED?
4. Das mittlere Bein des Transistors (die **Basis**) ist noch unbeschaltet. Wir ergänzen nun die Schaltung, indem wir vor die Basis einen 10 kΩ-Widerstand setzen und dessen andere Seite mit einem weiteren Kabel zuerst an +5V (Abb.5) und dann auch an GND (Abb.6) anschließen. Was macht dabei jeweils die LED?
5. Ersetzen Sie zum Schluss den NPN-Transistor durch einen PNP-Transistor (z. B. Typ 2N3906) und wiederholen Sie den vorigen Versuch.

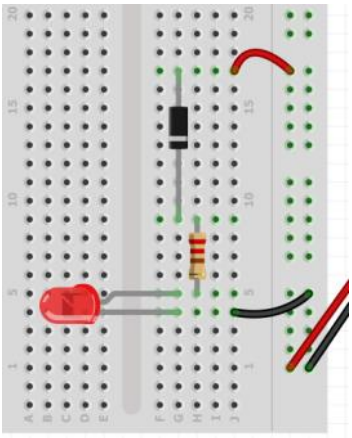


Abb. 1: Schaltung mit Diode Markierung an Minusseite

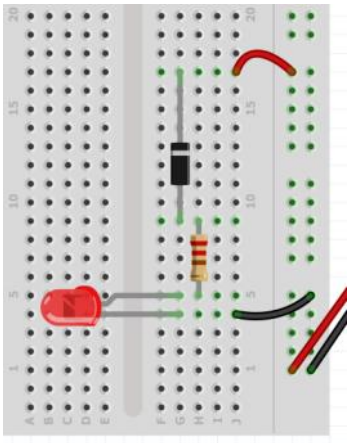


Abb. 2: Schaltung mit Diode Markierung an Plusseite

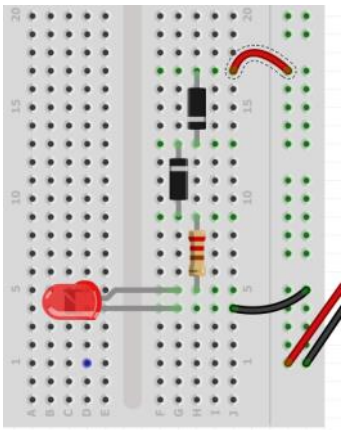


Abb. 3: Schaltung mit zwei Dioden. Markierungen innen.

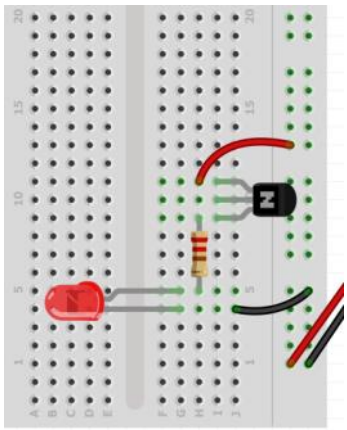


Abb. 4: Schaltung mit NPN-Transistor.

Schalter (NPN)

Basis an	LED
GND	
+5V	

Schalter (PNP)

Basis an	LED
GND	
+5V	

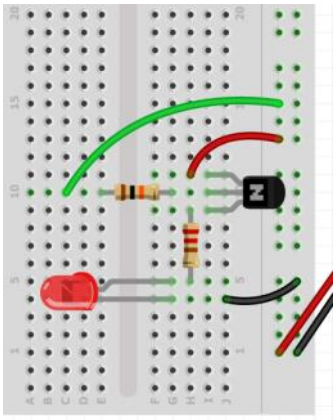


Abb. 5: Schaltung mit NPN-Transistor. Basis an +5V.

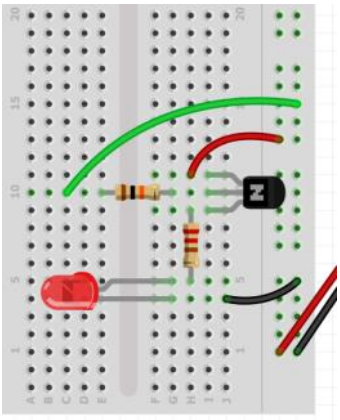
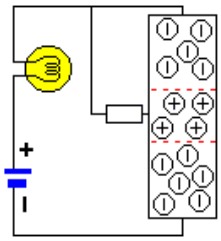
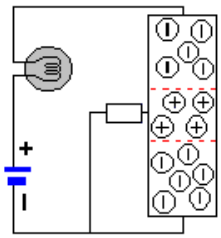


Abb. 6: Schaltung mit NPN-Transistor. Basis an GND

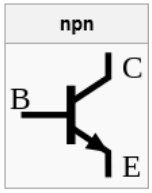
Die Basis wirkt als Schalter



Wunderbarerweise gibt es noch eine weitere Anschlussvariante: Wenn wir die mittlere (Sperr-)Schicht mit einem Anschluss versehen und - über einen Widerstand - an Plus anschließen, können wir damit den Stromkreis im Transistor durchlässig machen. Schließen wir die mittlere Schicht an Minus an, bleibt der Stromkries gesperrt. Ein Schalter ist entstanden.



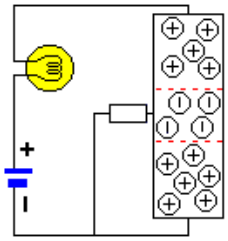
Was ist passiert? Der Pluspol unserer roten Stromquelle liegt an der negativen Schicht unseres Transistors. Dadurch werden die Elektronen aus der Schicht "herausgesaugt"; die negative Schicht wird immer dünner, bis Strom durch den Transistor fließen kann. Der Transistor verhält sich in dem Stromkreis mit der blauen Stromquelle so wie der Halbleiter im allerersten Beispiel. Weil bei dieser Bauweise des Transistors eine positive Schicht zwischen zwei negativen Schichten liegt, nennt man dies Bauteile einen **NPN-Transistor**.



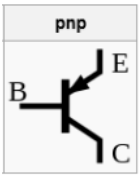
Der mittlere Anschluss im Transistor heißt **Basis**, die beiden anderen **Kollektor** (in Englisch mit C geschrieben) und **Emitter**. Je nachdem, ob man Spannung auf die Basis gibt, schaltet der Transistor den Durchgang zwischen Kollektor und Emitter frei oder sperrt ihn. Weil der Basisstrom dabei wesentlich geringer sein kann als der Strom zwischen Kollektor und Emitter, werden Transistoren in der Analogtechnik auch gern als Verstärker eingesetzt. In der Digitaltechnik gibt es dagegen nur zwei Möglichkeiten: Der Transistor öffnet den Durchgang zwischen Kollektor und Emitter oder er schließt ihn.

Aufgaben

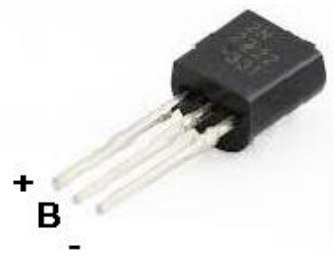
1. Sehen Sie sich im Artikel http://de.wikipedia.org/wiki/Bipolartransistor die Animation zum Transistor an.



PNP statt NPN

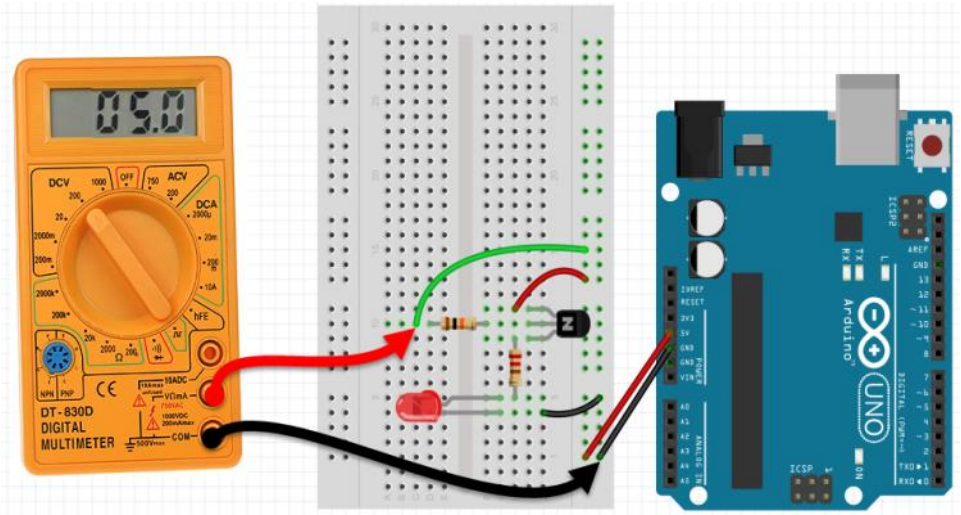


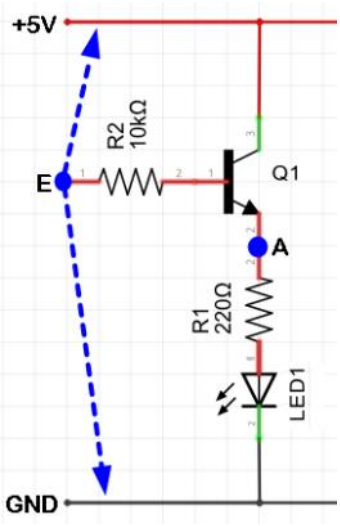
Für manche Zwecke ist es günstiger, dass der Transistor die Strecke zwischen Kollektor und Emitter freigibt, wenn die Basis an Minus liegt. Um das zu erreichen, muss man den Transistor anders konzipieren: Man legt hier die negative Schicht zwischen zwei positive und baut einen **PNP-Transistor**. Er öffnet den Stromkreis, wenn die mittlere Schicht an Minus angeschlossen wird.



Im Prinzip ist ein Transistor zwar symmetrisch aufgebaut. Bei realen Bauteilen ist dies nicht der Fall. Deshalb sind Transistorgehäuse auf der einen Seite rund, auf der anderen dagegen flach. Wenn Sie als NPN Transistor den Typ BC547 und als PNP-Transistor den Type 2N3906 verwenden, korrespondiert der Balken, an dem die Basisleitung im Symbol endet, mit der abgeflachten Seite des Transistorgehäuses. Beim NPN-Transistor fließt dann der Strom vom Kollektor zum Emitter, beim PNP-Transistor vom Emitter zum Kollektor. Sollten Sie andere Transistortypen verwenden, mag es sinnvoll sein, den Transistor versuchsweise andersherum einzubauen. Dadurch ändert sich nicht das Verhalten des Transistors, wohl aber der Grad der Verstärkerwirkung.

Schaltung, Schaltplan und Logik





Um den **Aufbau einer elektronische Schaltung** zu beschreiben, braucht man keine fotorealistische Abbildung. Es genügt ein Schaltplan, in dem die enthaltenen Bauelemente mit Symbolen eingezeichnet und richtig verbunden sind. Links finden Sie den Schaltplan unseres oben noch einmal abgebildeten Transistorschalters. Den Arduino lassen wir dabei weg, er dient bisher lediglich als Stromquelle.

Auch das **Verhalten der Schaltung** lässt sich einfacher angeben. Während in der Analogtechnik die Feinheiten wichtig sind, geht es in er Digitaltechnik meist nur um die Frage, ob die Schaltung an einer bestimmten Stelle Strom führt oder nicht.

Das können wir nachprüfen, indem wir das schwarze Kabel eines Voltmeters mit GND (Masse) verbinden und das rote Kabel an die interessante Stelle anlegen. Das Voltmeter zeigt dann entweder eine Spannung von etwa 5 Volt an oder eine Spannung nahe Null.

Oder wir bauen—wie wir es gemacht haben—eine LED in den Stromkreis ein und verbinden die eine Seite mit der zu prüfenden Stelle und die andere mit Masse. Dann leuchtet die LED entweder, oder sie tut es nicht.

Ein Transistor als Schalter

Rein logisch gesehen besitzt unsere Schaltung zwei interessante Stellen: den **Eingang E**, an den die Basis des Transistors angeschlossen ist, und den **Ausgang A**, der am Emitter des Transistors bzw. vor der LED sitzt. Und das Verhalten der Schaltung kann man mit einer der beiden folgenden Tabellen beschreiben. Die obere Tabelle beschreibt dabei die technische Seite der Angelegenheit. Um das logische Verhalten der Schaltung (untere Tabelle) zu beschreiben, genügen die beiden Ziffern 0 und 1.

Unser elektronischer Schalter besitzt einen Eingang (E) und einen Ausgang (A). An jeder der beiden Stellen kann man die Spannung gegen Masse feststellen. Das Ergebnis dieser Messungen wird in einer "Wahrheitstafel" notiert. Unsere untere Tabelle sagt also: Wenn ich den Eingang mit 0 (GND) verbinde, messe ich auch am Ausgang 0. Wenn ich den Eingang mit 1 verbinde (+5V), dann messe ich auch am Ausgang 1. Klingt logisch, oder?

Schalter (NPN)

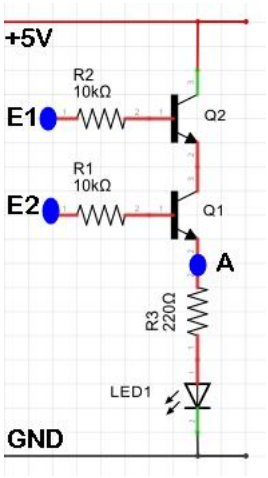
Basis an	LED
GND	leuchtet nicht
+5V	leuchtet

Schalter (NPN)

E	A
0	0
1	1

Die AND-Schaltung

In manchen Fällen möchte man Schaltungen bauen, deren Ausgang von zwei Eingängen abhängt. Nur wenn zwei Knöpfe gedrückt sind, wenn Bedingungen erfüllt sind, wenn zwei Eingänge gleichzeitig eine 1 liefern, soll der Ausgang aktiviert werden. Beide Eingänge können unabhängig voneinander an 0 oder 1 angeschlossen werden. Die Kombinationswirkung erreicht man dadurch, dass man zwei Transistoren in Reihe hintereinander anordnet. Es nützt nichts, dass der eine der beiden den Strom passieren lässt, wenn der andere ihn sperrt. Nur wenn beide Transistoren offen sind, fließt Strom zur LED. Die Gleichung lautet **A = E1 AND E2**.

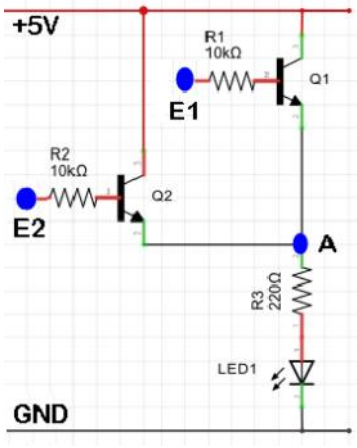


AND-Schaltung
Zwei NPN in Reihe

E1	E2	A
0	0	
1	0	
0	1	
1	1	

Die OR-Schaltung

In anderen Fällen mag es vorkommen, dass nur eine von zwei Bedingungen erfüllt sein muss. Eine Entschuldigung muss von Vater oder Mutter unterschrieben sein. Wenn beide sie unterschreiben, gilt sie zwar auch, aber nötig ist nur eine Unterschrift. Diesen Fall kann man durch einen Stromkreis abbilden, in dem die beiden Transistoren parallel geschaltet sind. Deren Gleichung lautet A = E1 OR E2.

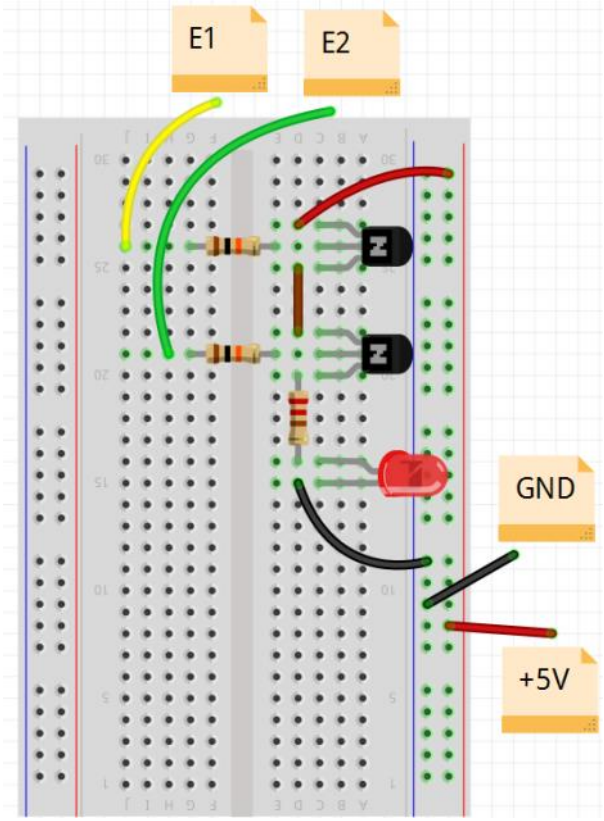


OR-Schaltung
Zwei NPN parallel

E1	E2	A
0	0	
1	0	
0	1	
1	1	

Aufgaben

1. Welche Schaltung ist rechts abgebildet?
2. Bauen Sie die beiden Schaltungen auf dem Breadboard nach. Schließen Sie die Eingänge nacheinander so an, wie in den vier Zeilen der Tabelle angegeben. Füllen Sie für jede Schaltung eine Wahrheitstafel aus. Notieren Sie in der Spalte A eine 1, wenn die LED leuchtet, und eine 0, wenn sie dunkel bleibt.
3. Sowohl die fotorealistischen Abbildungen als auch die Schaltpläne dieser Doppelseite sind mit der kostenlosen Platinenlayoutsoftware Fritzing entworfen worden. Laden Sie bei http://www.fritzing.org das Programm herunter und zeichnen Sie, wie Sie die hier nicht abgebildete Schaltung auf dem Breadboard realisiert haben. Speichern Sie Ihre Schaltung ab.

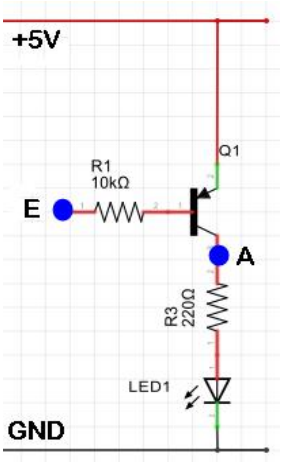


NOT: Wir stellen alles auf den Kopf

Warnleuchten sollen oft genau dann leuchten, wenn die eigentliche Schaltung nicht funktioniert, wenn also ein bestimmter Eingang keinen Strom liefert.

Auch das geht: Nehmen wir den einfachen Schalter von der vorletzten Seite. Wenn Sie dort versuchsweise den NPN-Transistor gegen einen PNP-Transistor austauschen, so werden Sie feststellen, dass das die Schalterwirkung umkehrt: Wenn Sie E an 0 (also an GND) anschließen, liefert der Ausgang 1, die LED brennt. Wenn Sie dagegen E an 1 anschließen, geht die LED aus.

Man spricht hier von einer NOT-Schaltung (das hat nichts mit einer Notlage, aber mit Nein zu tun). Die Gleichung A = NOT E bedeutet: Der Ausgang führt das umgekehrte Signal wie der Eingang.

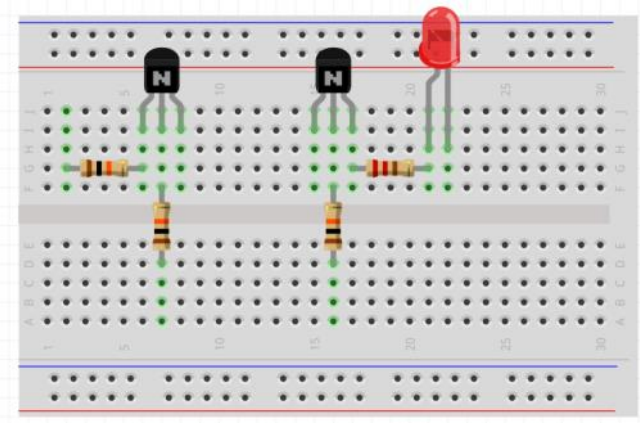


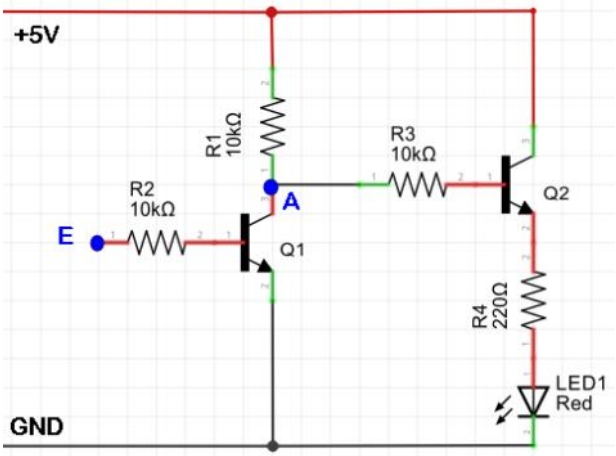
NOT-Schaltung
(PNP)

E	A
0	
1	

Kein PNP? Kein Problem!

Wenn Ihr Bausatz keinen PNP-Transistor enthält, können Sie trotzdem eine NOT-Schaltung realisieren. Bauen Sie die abgebildete Schaltung mit zwei NPN-Transistoren. Stecken Sie die Bauteile wie abgebildet in das Breadboard und verbinden Sie die Teile entsprechend dem Schaltplan. Schließen Sie dann E an GND an. Die LED sollte leuchten. Wenn Sie E dagegen an +5V anschließen, sollte sie dunkel bleiben. Für die eigentliche Schaltung brauchen wir auch hier nur einen Transistor. Die rechte Spalte in der Wahrheitstafel bezieht sich auf den Wert des Ausgangs A. Wenn wir zwischen A und GND ein Voltmeter anschließen würden, dann würden wir 5V Spannung messen. Allerdings ist die Stromstärke bei A so gering, dass die LED zwischen A und GND nicht aufleuchten würde. Der rechte Transistor dient also hier nur als Verstärker, nicht als Schalter.



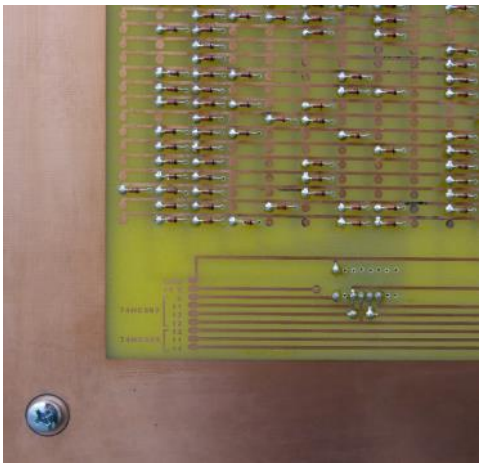


NOT-Schaltung
(NPN)

E	A
0	
1	

Aufgaben

1. Zeichnen Sie mit Fritzing die fehlenden Kabel ein und bauen Sie die Schaltung auf dem Breadboard nach.
2. Versuchen Sie, anhand der Eigenschaften des Transistors und des Widerstands die Wirkung der Schaltung zu erklären.



Level 2:
Logische Gatter

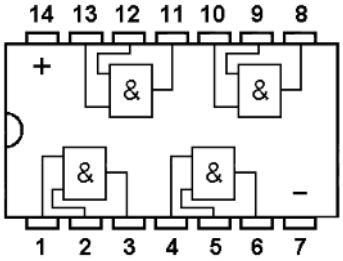
In diesem Kapitel erfahren Sie, ...

- dass man viele Transistoren in einen Chip verpacken kann
- dass man Zahlen, Buchstaben, Befehle und mehr in Form von Nullen und Einsen kodieren kann
- wie man aus UND-, ODER- und NICHT-Giedern Addierer, Vergleicher und Speicherbausteine konstruieren kann

Gatterbausteine

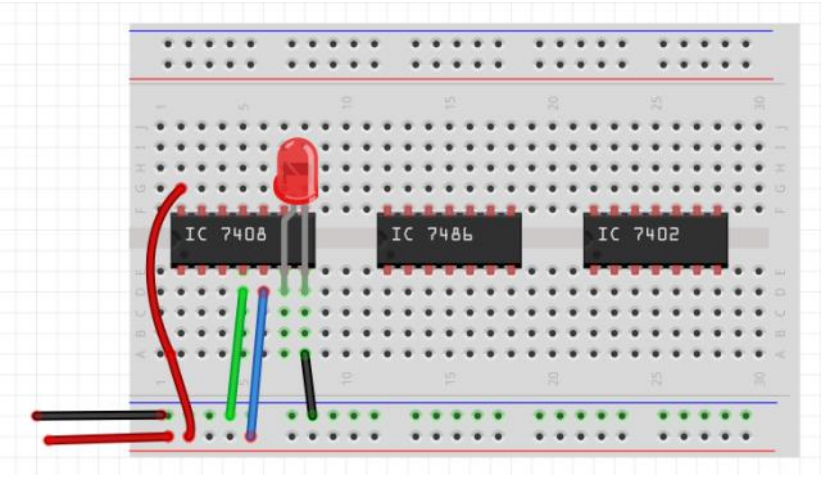
Im vorigen Abschnitt haben Sie gelernt, dass man mithilfe von Transistoren Schaltungen bauen kann, bei denen ein Zustand eines Ausgangssignals von einem oder mehreren Eingangssignalen gesteuert wird. Mithilfe einer UND-Schaltung oder einer ODER-Schaltung kann ich zwei Werte kombinieren, mithilfe einer NOT-Schaltung kann ich ein Signal umkehren.

Einige Jahre nach der Erfindung des Transistors begann man, die Funktionen mehrerer Transistoren in ICs (Integrated Circuits, **Integrierten Schaltkreisen**) zusammenzufassen. Wir vergessen also jetzt die ganze Technik mit Widerständen und Transistoren und gehen davon aus, dass man logische Schaltungen fertig kaufen kann. Solche Bausteine heißen **logische Gatter** und sind in ICs verpackt.



Eine verbreitete Reihe von ICs ist die Serie 74xx. Das sind Bausteine mit 14 Kontakten, die in zwei Reihen angeordnet sind. An Kontakt 14 wird die Versorgungsspannung 5V angelegt, an Kontakt 7 kommt Masse GND. Die übrigen Kontakte bestehen aus Ein- und Ausgangsleitungen. Auf einem 74xx-Baustein sind meist mehrere Gatter enthalten.

Dies ist ein Breadboard mit drei solchen Gatter-ICs. Wir testen zuerst den linken Schaltkreis. Er trägt die Bezeichnung 7408.



Gatter (engl. Gate): Denken Sie an das Tor zu einer Weide, das geöffnet oder geschlossen sein kann. Entsprechend lässt unser Gatter die Elektronen passieren oder nicht.

Aufgaben

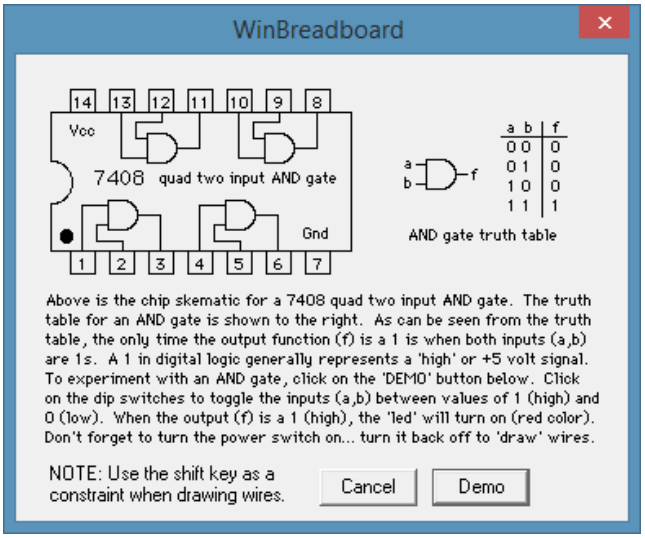
1. Verbinden Sie die Stromversorgungsleiste des Breadboards mit +5V und GND des Arduino. Um das Gatter mit Strom zu versorgen, verbinden Sie Kontakt 7 des Gatterchips mit GND und Kontakt 14 mit +5V.
2. Kontakt 6 ist ein Ausgang des Gatters. Stecken Sie hier die Anode (das längere Bein) und in Kontakt 7 die Kathode (das kürzere Bein) einer LED ein.
3. Zum Ausgang an Kontakt 6 gehören die beiden Eingänge an Kontakt 4 und 5. Versehen Sie sie mit Kabeln und stecken Sie das andere Ende des Kabels nacheinander in die GND-Leiste (0) oder in die +5V-Leiste (1). Wann brennt die LED? Füllen Sie die Tabelle aus und bestätigen Sie, dass es sich um eine AND-Schaltung handelt.
4. Drei weitere AND-Schaltungen finden sich an Kontakt 1.2.3, an Kontakt 8,9,10 und an Kontakt 11,12,13. Testen Sie mindestens eine davon aus.
5. Wie verhalten sich die anderen beiden ICs? Testen Sie wieder mit Kontakt 4,5 und 6 und stellen Sie eine Wahrheitstafel auf.

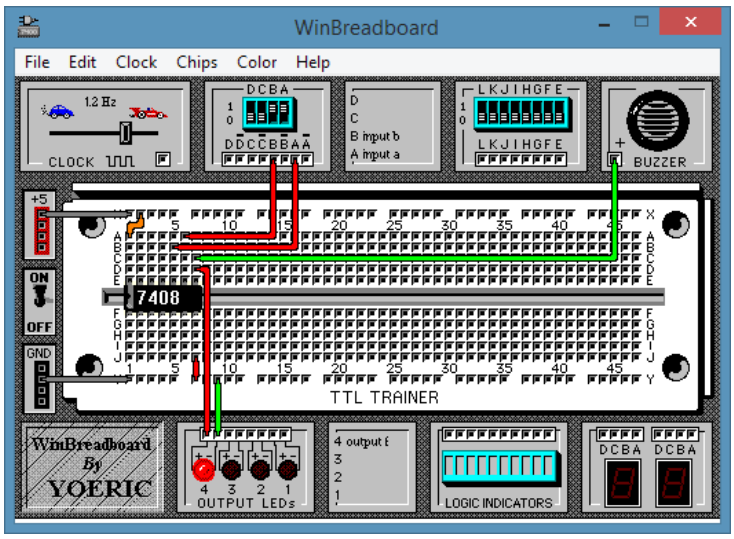
IC 7408 Pin 4,5,6

Pin4	Pin5	Pin6
0	0	
1	0	
0	1	
1	1	

Gatter kann man auch simulieren

Wenn Sie keine Gatterbausteine in Ihrem Experimentierset finden, macht das fast nichts. Auch für Gatterbausteine der 74-Serie gibt es Simulationsprogramme, z.B. WinBreadboard. Sie können sich bei http://yoeric.com/breadboard.htm eine Demo herunterladen.





Starten Sie das Programm, klicken Sie in den beiden Startdialogen auf „Demo“. Sie laden damit einen 7408-Chip mit 4 AND-Gattern, von denen eins fertig verdrahtet ist:

- Der Pin 7 des Chips ist mit der GND-Leiste des Breadboards verbunden, der Pin 14 mit der 5V-Leiste.
- Die beiden Eingangspins 9 und 10 sind mit zwei Schiebeschaltern in der oberen Leiste verbunden. Der Ausgangspin 8 ist mit einer LED in der unteren Leiste verbunden, außerdem mit einem Beeper.
- Wenn Sie den ON/OFF-Schalter links umlegen, dann können Sie durch Ein– und Ausschalten der Eingänge den Chip testen.

Aufgaben

1. Schalten Sie den Hauptschalter dann wieder aus. Bei Schalterstellung OFF können Sie durch einen Doppelklick auf den Chip dessen Pinbelegung anzeigen lassen. Die amerikanischen Symbole für AND, OR, XOR usw. sehen etwas anders aus als die europäischen.
2. Ersetzen Sie den 7408-AND-Chip der Reihe nach durch einen 7432 OR-Chip, durch einen 7404 Inverter (NOT) und durch einen 74386 ExOR. Und testen Sie jeweils einen Satz von Pins. Die Demo ist voll funktionstüchtig, Sie können allerdings immer nur einen Chip auf das Breadboard legen.
3. Testen Sie ein anderes Gatter eines Chips. Löschen Sie dafür Kabelverbindungen und setzen Sie sie neu.

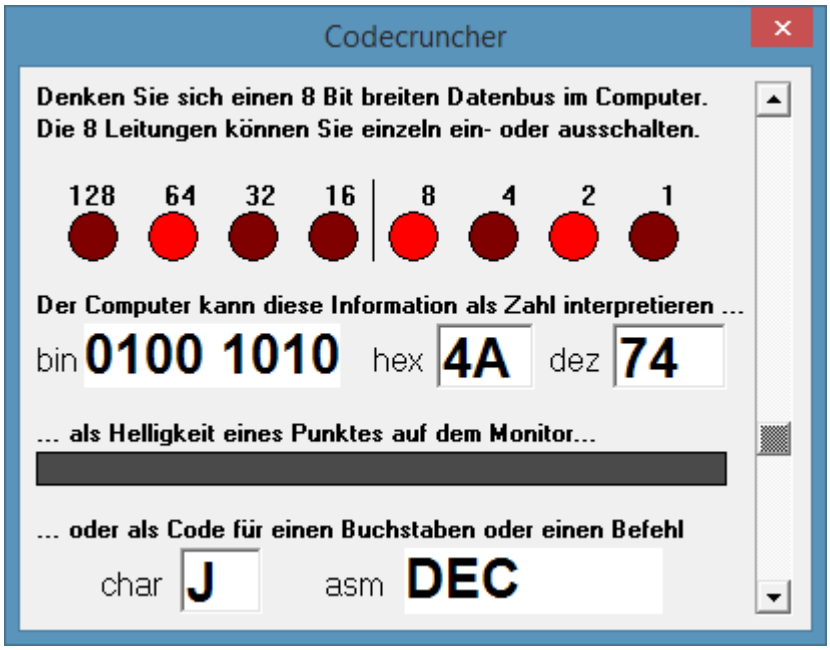
Was bedeuten Null und Eins?

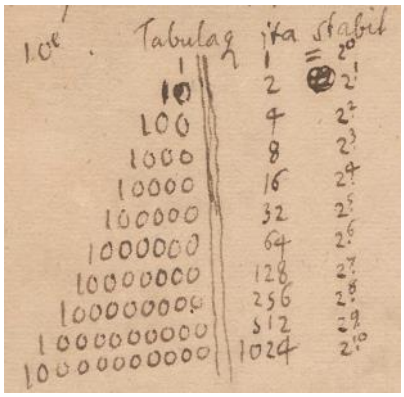
Wir haben vereinbart, den Umstand, ob eine Leitung Spannung führt, in „Wahrheitstafeln“ mit 1 oder 0 zu beschreiben. Mit 1 oder 0 lassen sich also Zustände und Eigenschaften wie wahr/falsch, an/aus oder ja/nein in kürzestmöglicher Form beschreiben.

Nun besteht die Welt nicht nur aus Schwarz und Weiß. Aber in vielen Kulturen haben sich Philosophen und Mathematiker Gedanken darüber gemacht, wie man eine Vielzahl von Zuständen beschreiben kann, indem man sie auf eine Kombination aus Alternativen zurückführt. Sie kennen sicher Ratespiele, bei denen auf Fragen nur mit ja oder nein geantwortet werden darf, um einen Beruf, eine Person oder ein Ding zu erraten.

Dualzahlen

Der deutsche Wissenschaftler Gottfried Wilhelm Leibniz hat vor 300 Jahren versucht, beliebige Zahlen nur mit Nullen oder Einsen darzustellen. Er benutzte dazu ein binäres Zahlensystem, auch Dualsystem genannt, bei dem eine Zahl von rechts beginnend nicht mit Einern, Zehnern, Hundertern, Tausendern usw., sondern mit Einern, Zweiern, Vierern, Achtern usw. kodiert wird. Die überraschende Folge: Zwei Einer geben bereits einen Zweier, zwei Zweier einen Vierer, ... Man braucht also für dieses Zahlensystem nicht mehr zehn Ziffern, sondern nur zwei, die Null und die Eins. Laden Sie sich den Codecruncher herunter. Knipsen Sie die roten Punkte an und aus. Lesen Sie bei „bin“ die Kombination in Form von Einsen und Nullen ab und bei „dez“ den Wert der Zahl im Dezimalsystem.





Schon Leibniz experimentierte zur Zahldarstellung mit Nullen und Einsen.

Bit und Byte

Die Informationsmenge, die man mithilfe einer einzelnen Null oder Eins kodieren kann, nennt man ein Bit. Acht Bit bilden ein Byte. Das reicht, um 256 verschiedene Zustände zwischen 0 und 255 zu kodieren. Das könnte zum Beispiel ein Satz von 256 verschiedenen Buchstaben oder Zeichen sein oder auch die 256 Helligkeitsstufen eines Pixels.

Kilo - Mega - Giga -Tera

1 KB (Kilobyte) sind 1000 Byte
(2 KB = eine kleine Buchseite)
1 MB (Megabyte) sind 1 Million Byte
(5 MB = Alle Buchstaben der Bibel
6 MB = Punkte eines Bildschirms)
1 GB (Gigabyte) sind 1 Milliarde Byte
(5 GB = Ganzer Spielfilm auf DVD)
1 TB (Terabyte) sind 1 Billion Bytes
(2 TB = Alle Daten einer Festplatte)

Aufgaben

1. Unser „Codecruncher“ arbeitet mit 8 Stellen. Was ist die kleinste, was ist die größte Zahl, die Sie damit darstellen können?
2. Wie viele Zustände könnte ein 16 Bit breiter Bus annehmen?
3. Wie viele Stellen wären mindestens erforderlich, um Zahlen bis tausend (bis zur Million) binär codieren zu können?

Datenbus

Einen Gruppe paralleler Leitungen, aus denen der Computer gleichzeitig Daten liest, nennt man einen Bus.

4. Verwandeln Sie Zahlen aus dem Dualsystem ins Dezimalsystem und umgekehrt. Versuchen Sie es erst mit dem Codecruncher, dann auch im Kopf.

dual	dez
1101 0001	
0011 0100	
1011 0101	
0000 1111	
1111 0000	

dez	dual
19	
240	
127	
68	
200	

dual	hex
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	A
1011	B
1100	C
1101	D
1110	E
1111	F

Hexadezimalzahlen

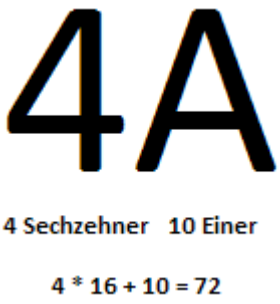
Dualzahlen sind für den Alltagsbetrieb zu lang und schlecht überschaubar. Deshalb benutzen Programmierer häufig Hexadezimalzahlen als Kurzform für Dualzahlen. Bei dieser Darstellung werden immer vier Zweierstellen zu einer Hexadezimalstelle zusammengefasst. Allerdings braucht man für jede Stelle einer Hexadezimalzahl 16 verschiedene Ziffern. So verwendet man als Ziffern für 10 bis 15 die Buchstaben A bis F. Die Hexadezimalzahl 4A liest sich so: Vier Sechzehner und 10 Einer. Das macht 72.

5. Übersetzen Sie die Zahlen zwischen Dualsystem und Hexadezimalsystem

dual	hex
1101 0001	
0011 0100	
1011 0101	
0000 1111	
1111 0000	

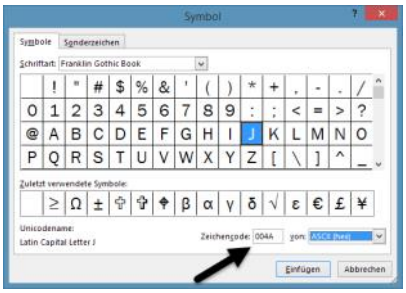
hex	dual
AF	
39	
B8	
8B	
FF	

hex	dez
AF	
39	
B8	
8B	
FF	



Codes

Bereits 1833 erfand Samuel Morse einen binären Code aus kurzen und langen Piepsern, um Buchstaben zu kodieren. Wie alle anderen Informationen (Helligkeit, Farbe, Befehlscodes, Adressen, ...) sind Im Computer auch Zeichen mit Zahlen kodiert. Sie können sich in der Symboltabelle von Word (Einfügen -> Symbol) den dezimalen oder hexadezimalen Code jedes Zeichens anzeigen lassen.



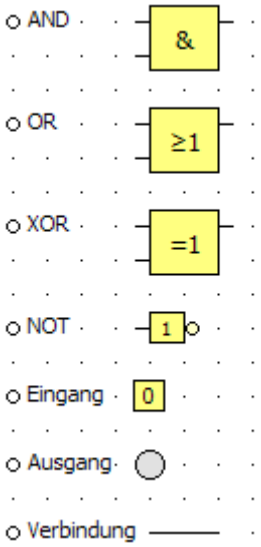
6. Ermitteln Sie den dezimalen / den hexadezimalen Code der folgenden Zeichen:

char	dez	hex
A		
Z		
m		
x		
*		

char	dez	hex
ü		
@		
Ø		
µ		
¥		

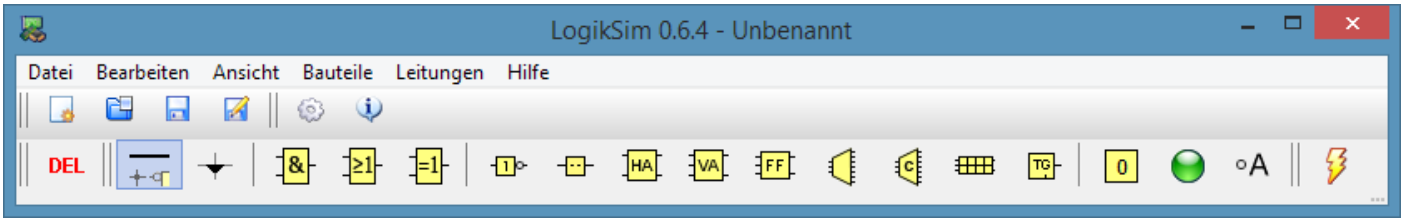
Schaltkreise mit Gattern

Nachdem wir nun festgestellt haben, dass man durch geschicktes Kombinieren von Nullen und Einsen praktisch jede Information speichern und übermitteln kann, betrachten wir nun die Methoden, mit denen man aus vorhandenen Informationen neue gewinnen kann.

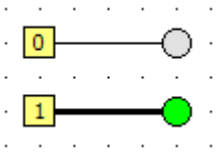


Dazu besorgen wir uns einen **Logisimulator**. Er ermöglicht es uns, das Verhalten logischer Schaltungen zu simulieren. Die Symbole erscheinen dabei als „black box“ Wir kümmern uns auf dieser Ebene nicht mehr um Stromversorgung und technischen Aufbau der Schaltung, sondern betrachten nur ihre Wirkung. Ein Quadrat mit dem Zeichen &. („kaufmännisches Und“) deutet eine AND-Schaltung an. Ähnliche Symbole gibt es für OR (≥1 bedeutet: mindestens 1 Kontakt muss an 1) und für NOT (ein kleiner Kringel am Eingang oder Ausgang eines Symbols.

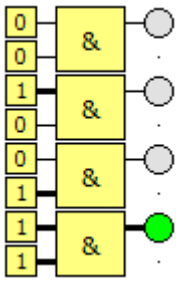
Ein kostenloser Logiksimulator ist hier oder bei http://logiksim.dbclan.de erhältlich. Entpacken Sie die ZIP-Datei auf Ihren Desktop und starten Sie das Programm: Sie benötigen zunächst nur die sieben rechts abgebildeten Symbole.



- Wählen Sie in der Symbolleiste den Eingang aus und klicken Sie auf den Schaltplan. Setzen Sie dann eine LED daneben. Sie erscheint zunächst grau. Sie können die Position von Symbolen nachträglich nicht mehr ändern. Sie müssen sie löschen (DEL einschalten und Symbol anklicken) und neu zeichnen.

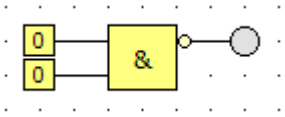


- Wählen Sie das Liniensymbol (2. Knopf von links) und ziehen Sie mit der Maus eine Linie vom Eingang zum Ausgang.
- Klicken Sie auf den Blitz und schalten Sie damit die Simulation ein.
- Ändern Sie den Wert des Eingangs durch Anklicken von 0 auf 1. Auf der Verbindung fließt Strom, die LED am Ausgang leuchtet.
- Beginnen Sie nun mit einem AND-Symbol. Setzen Sie die Eingänge und den Ausgang daran und zeichnen Sie die Verbindungen. Starten Sie dann die Simulation. Nur wenn Sie beide Eingänge auf 1 setzen, brennt die LED.

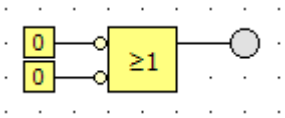


Aufgaben

1. Klicken Sie sich aus Schaltern, Gattern, LEDs und Verbindungslinien auch eine OR-und eine NOT-Schaltung zusammen.
2. Durch Hinzufügen eines kleinen Kringels am Ausgang (Verbindungssymbol auswählen, dann Ausgang anklicken) können Sie aus einem AND-Glied eine NAND-Glied machen. Die beiden Eingänge werden hier mit AND verknüpft und das Ergebnis dann mit NOT umgekehrt. Stellen Sie für NAND eine Wahrheitstabelle auf.
3. Nicht nur hinter die Ausgänge, auch vor die Eingänge eines Gatters können Sie einen NOT-Kringel setzen. Stellen Sie eine Wahrheitstafel für NOT E1 OR NOT E2 auf. Vergleichen Sie die Wahrheitstabelle mit der von NAND.
4. Bauen Sie NOT E1 AND NOT E2 und finden Sie eine Abkürzung.



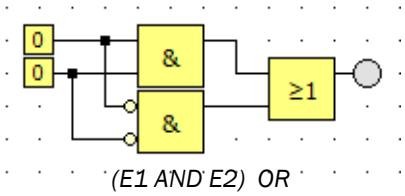
NOT (E1 AND E2)
E1 NAND E2



NOT E1 OR NOT E2

Logikpuzzle

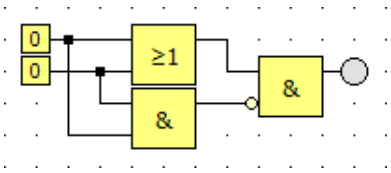
Die Gatter eines Logiksimulators kann man ohne Angst vor Kabelwirrwarr miteinander kombinieren. Während man bei realen Gatterbausteinen auf einer Platine viel Mühe darauf verwenden muss, die Leiterbahnen kreuzungsfrei zu halten, können Sie im Logiksimulator die Verbindungen auch übereinander führen Nur da, wo ein Punkt sitzt, sind die Leitungen verbunden. Haben Sie schon gemerkt, dass man Verbindungen auch um eine Ecke ziehen kann?



(E1 AND E2) OR (NOT E1 AND NOT E2)

Aufgaben

1. Bauen Sie die abgebildete Schaltungen *(E1 AND E2) OR (NOT E1 AND NOT E2)* und (E1 OR E2) AND NOT (E1 AND E2) nach. Stellen Sie für beide Schaltungen eine Wertetabelle auf. Was stellen Sie fest?

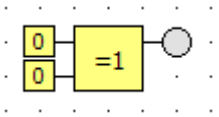


(E1 OR E2) AND NOT (E1 AND E2)

Einschließend oder exklusiv?

Wenn wir sagen „Vielleicht kommt Franz oder Fritz zum Geburtstag“ dann freuen wir uns vermutlich auch über den Besuch von Franz und Fritz. Sagen wir aber „Das Päckchen kommt morgen oder übermorgen“, so schließt die eine Möglichkeit die andere aus.

Rechts finden Sie eine häufig eingesetzte Elementarschaltung. Sie lautet XOR und bezeichnet das „exklusive ODER“. Ihr Ausgang ist nur dann 1, wenn entweder der eine oder der andere Einfang 1 meldet, *aber nicht beide*.



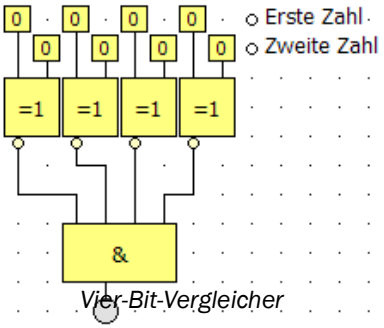
E1 XOR E2

XOR		
E1	**E2**	**A**
0	0	0
1	0	1
0	1	1
1	1	0

Vergleicherschaltungen

In der Computertechnik kommt es häufig vor, dass man testen muss, ob zwei Signale gleich oder ungleich sind. Das geht mit den beiden Schaltungen von Aufgabe 1 oder viel kürzer mit NOT XOR. Am Ausgang steht also eine 1, wo bei XOR eine 0 steht und umgekehrt.

Und was ist, wenn der Computer größere Zahlen vergleichen soll? Kein Problem. Wir vergleichen jede Stelle der ersten Zahl mit jeder Stelle der zweiten. (Zur besseren Übersicht wurden im Logiksimulator unter *Ansicht -> Standardbauteile* die Symbole auf die Baurichtung von oben nach unten umgestellt.)



Vier-Bit-Vergleicher

Aufgaben

2. Bauen Sie die Vier-Bit-Vergleicherschaltung nach.
3. Finden Sie eine Schaltung, die nur dann 1 liefert, wenn bei E1>E2 ist. Stellen Sie zunächst eine Wahrheitstabelle auf.
4. Finden Sie eine Schaltung, die 1 liefert, wenn E1 ≥ E2 ist, sonst nicht. Stellen Sie zunächst eine Wahrheitstabelle auf.
5. Konstruieren Sie eine eigene Schaltung und beschreiben Sie sie mit Worten.

Mit Gattern Zahlen addieren

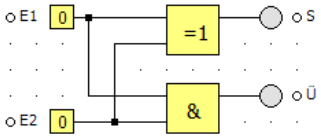
Noch alltäglicher als das Vergleichen von Zahlen ist für Computer das Rechnen. Die einfachste Rechenoperation für den Computer ist die Multiplikation. Das kleine Einmaleins des Dualsystems lautet nämlich: 0 mal 0 ist 0; 0 mal 1 ist 0; 1 mal 0 ist 0; 1 mal 1 ist 1. Merken Sie etwas? Das ist die Wahrheitstabelle einer ganz gewöhnlichen AND-Schaltung. Bei mehrstelligen Dualzahlen wird die Sache natürlich etwas anspruchsvoller.

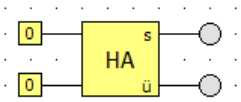
Schwieriger ist das Addieren zweier Zahlen: 0+0 = 0; 0+1 = 1; 1+0 = 1; 1+1 = 10.

Wieso 10? Die 10 des Dualsystems ist die 2 des Dezimalsystems. Wenn Sie im Dezimalsystem 5 + 7 = 12 rechnen, erhalten Sie einen Übertrag aus der Einerstelle in die Zehnerstelle. Im Dualsystem entsteht dieser Übertrag schon bei der Rechnung 1 + 1. Das Ergebnis dieser Rechnung ist 10, eben ein Zweier und null Einer.

Wenn Sie die Tabelle genau ansehen, stellen Sie fest: Die Spalte Übertrag entspricht einer AND-Schaltung, die Spalte Summe einer XOR-Schaltung. Eine solche Schaltung nennt man einen **Halbaddierer**.

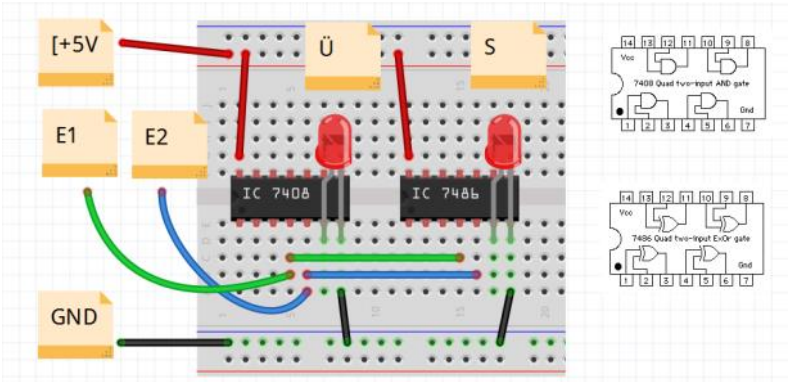
Halbaddierer			
S1	**S2**	**Ü**	**S**
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0



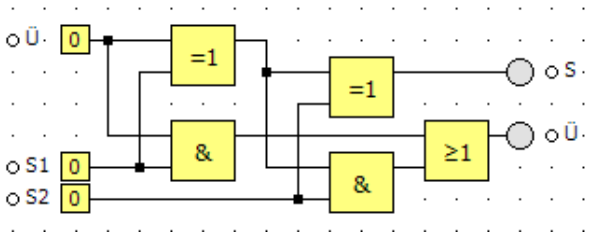


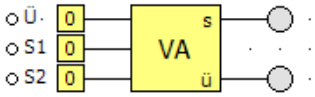
Aufgaben

1. Bauen Sie die Schaltung im Logiksimulator nach. Vergleichen Sie mit dem fertigen Halbaddierer des Logiksimulators.
2. Stecken Sie auf Ihr Breadboard einen Baustein mit vier XOR-Gattern (7408) und einen mit vier AND-Gattern (7486). Bauen Sie daraus eine Schaltung, die zwei Eingänge verknüpft und auf zwei LEDs die Summe und den Übertrag anzeigt.



Der Halbaddierer heißt übrigens "Halb"addierer, weil er zwar zwei binäre Zahlen addieren kann, aber noch nicht in der Lage ist, auch den Übertrag aus der vorangegangenen Stelle zu berücksichtigen. Eine Schaltung, die dies leistet, nennt man **Volladdierer**. Sie besteht aus zwei Halbaddierern und einer ODER-Schaltung:



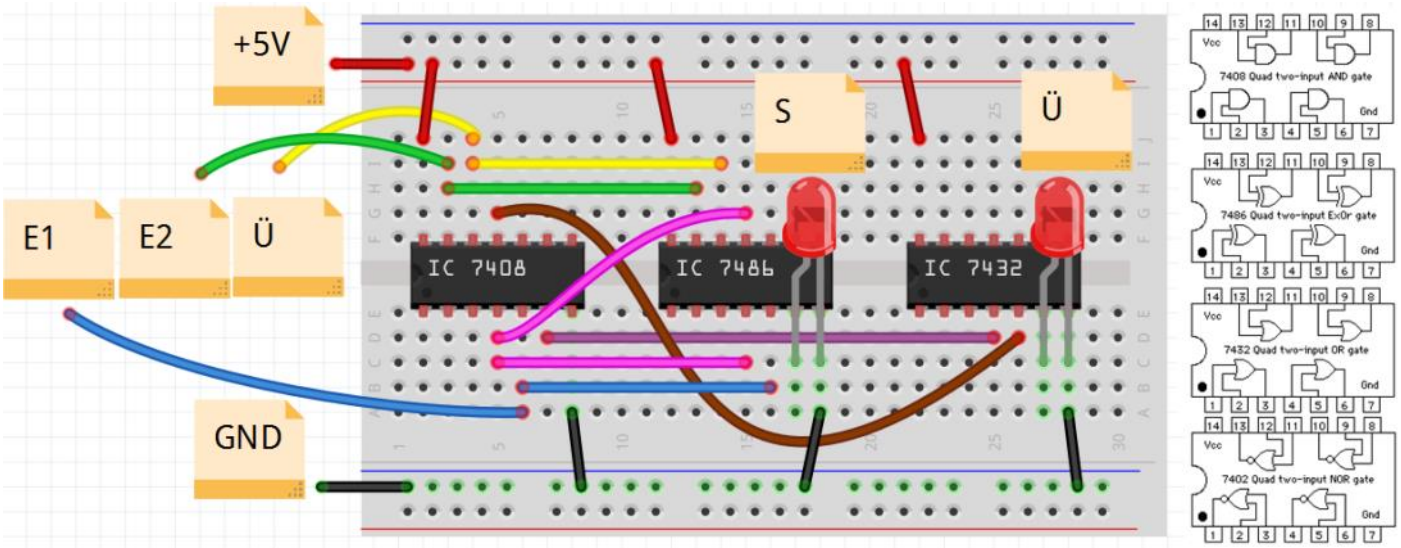


Volladdierer				
Ü	**S1**	**S2**	**Ü**	**S**
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Aufgaben

1. Bauen Sie die Volladdierer-Schaltung von Seite 24 im Logiksimulator nach. Vergleichen Sie mit der bei LogikSim als Beispieldatei mitgelieferten Volladdierer-Schaltung und mit dem fertigen Volladdierer-Symbol des Logiksimulators.
2. Stecken Sie auf Ihr Breadboard einen Baustein mit vier XOR-Gattern (7408), einen Baustein mit vier AND-Gattern (7486) und einen Baustein mit vier OR-Gattern (7432). Bauen Sie daraus eine Schaltung, die den vorhergehenden Übertrag Ü mit zwei neuen Summanden E1 und E2 verknüpft und auf zwei LEDs die Summe S und den neuen Übertrag anzeigt. Testen Sie die Schaltung, indem Sie Ü, E1 und E2 in verschiedener Kombination an +5V und GND anschließen.

Anmerkung: Wenn Sie keinen 7432-Chip zur Hand haben, können Sie die Schaltung auch mit einem NOR-Glied (7402) aufbauen. Im Gegenzug müssen Sie dann die LED mit der Kathode (-) an den NOR-Ausgang und mit der Anode (+) an +5V anschließen.
Achtung: Beim 7402 sind die Anschlüsse spiegelverkehrt angeordnet

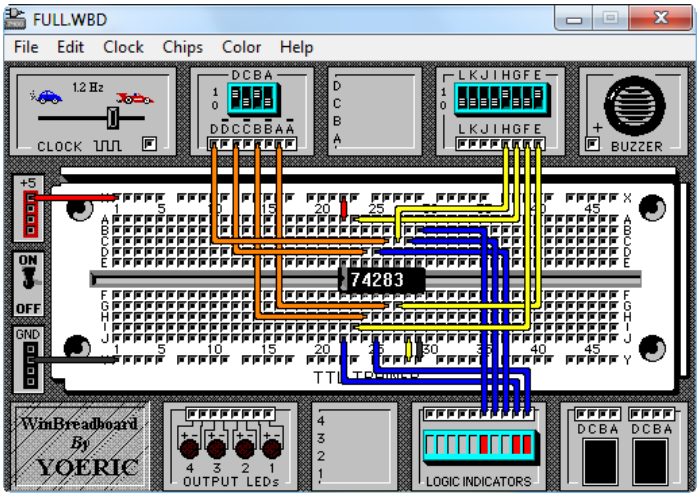


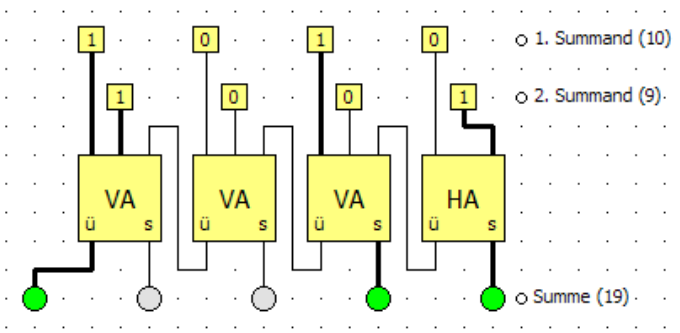
Aufgaben

3. Verdrahten Sie in WinBreadboard einen Vier-Bit-Volladdierer.
4. Bauen Sie in LogikSim einen Vier-Bit-Volladdierer.

In der Demoversion von Win-Breadboard können Sie weder Halbaddierer noch Volladdierer aus Elementen aufbauen. Sie können aber einen 74283-Chip verdrahten und ausprobieren. Das ist ein fertiger Volladdierer für Vier-Bit-Zahlen.

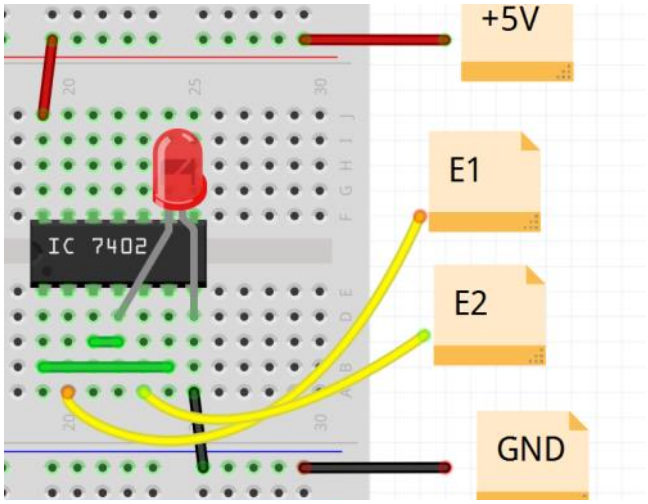
Unten ist ein Vier-Bit–Volladdierer aus einem Halbaddierer und drei Volladdierern in LogikSim aufgebaut.
Damit die vier Stellen der beiden Dualzahlen nebeneinander liegen, drehen Sie die Bauteile mit *Ansicht* > *Standardbauteile* in die Vertikale.

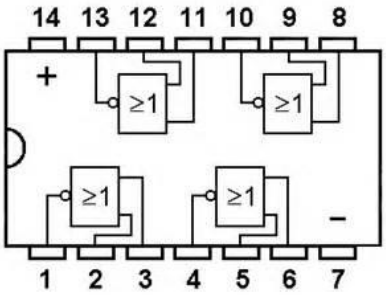




Aus zwei NOR-Gattern wird ein Flipflop

Mindestens genauso wichtig wie die Rechenfertigkeit des Computers ist seine Fähigkeit, Daten zu speichern und bei Bedarf wieder hervorzuholen. Auch das lässt sich mit einigen Transistoren realisieren. Mit zwei NOR-Bausteinen (ein ODER-Gatter, dessen Ergebnis mit NICHT umgekehrt wird), kann man ein **Flipflop** aufbauen. Das ist eine Vorrichtung, die sich ein einzelnes Bit merken kann. Sowohl zum Setzen als auch zum Löschen braucht es nur einen kurzen Impuls.

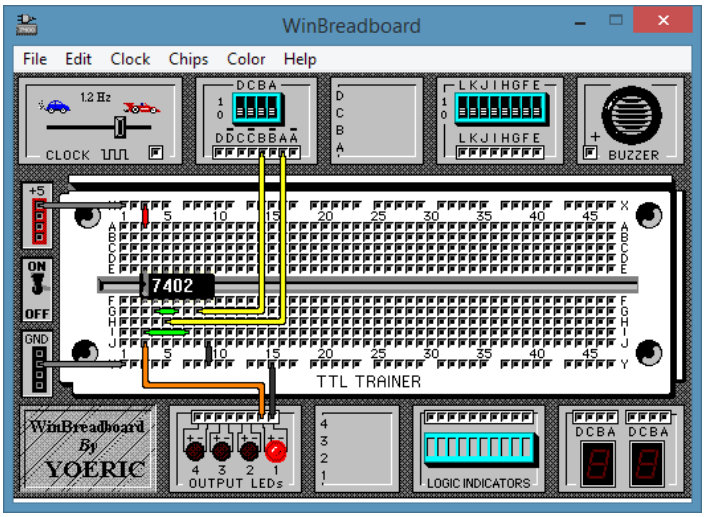




- Bauen Sie zunächst die Schaltung auf. Stecken Sie einen 7402-Gatterbaustein mit vier NOR-Gliedern auf Ihr Breadboard. Verbinden Sie zunächst Kontakt 7 mit GND und Kontakt 14 mit +5V.
- Setzen Sie eine LED zwischen den Ausgang 4 des rechten unteren Gatters und GND. Verbinden Sie den Ausgang 4 außerdem mit dem Eingang 3 des linken unteren Gatters und den Ausgang 1 des linken unteren Gatters mit Eingang 6 des rechten unteren Gatters. Die beiden noch offenen Gattereingänge E1 und E2 verbinden Sie mit GND.
- Die LED sollte jetzt brennen. Wenn Sie nun eine der beiden Eingangsleitungen herausziehen und wieder einstecken, geht sie kurzfristig von 0 auf 1 (offene Eingänge bei Gattern werden mit 1 beschaltet). Sie stellen fest: wenn Sie E1 kurz herausziehen und wieder einstecken, geht die LED an, wenn Sie E2 kurz herausziehen und wieder einstecken, geht die LED aus.

Ein Flipflop ist eine Schaltung, die sich merken kann, ob sie zuletzt auf 1 oder auf 0 gesetzt wurde.

Ein Flipflop ist also ein Speicher mit einer Kapazität von einem Bit.



Sie können das Flipflop auch im Breadboard-Simulator aufbauen. Kurzzeitiges Ein- und wieder Ausschalten von Schalter B lässt die LED aufleuchten, kurzzeitiges Ein- und wieder Ausschalten des Schalters A schaltet sie aus.

Ein Flipflop im Logiksimulator

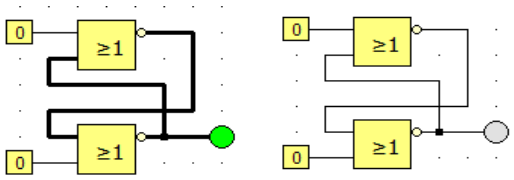
Im Logiksimulator benötigen wir für ein Flipflop zwei NOR-Gatter. NOR heißt NOT (E1 OR E2) und ist die Umkehrung von OR. Der Ausgang einer NOR-Schaltung wird nur dann 1, wenn beide Eingänge 0 sind.

Die Funktionsweise des Flipflops ergibt sich daraus, dass der Ausgang jedes der beiden NOR-Gatter mit dem Eingang des jeweils anderen NOR-Glieds rückgekoppelt ist. Zum Setzen oder Rücksetzen des Flipflops genügt jeweils ein kurzer Impuls.

NOR		
E1	E2	A
0	0	1
0	1	0
1	0	0
1	1	0

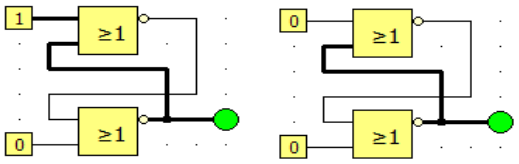
Aufgaben

1. Starten Sie die Simulation. Unmittelbar nach dem Einschalten flimmert die Schaltung, die Lampe blinkt.



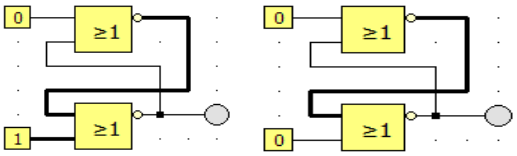
Nach dem Start flimmert das Flipflop.

2. Setzen Sie den oberen Eingang der Schaltung auf 1, so wird der Ausgang des oberen NOR-Gatters 0. Da der untere Eingang der Schaltung auf 0 steht, liefert das untere Gatter 1. Diese 1 ist rückgekoppelt zum oberen Gatter. Dadurch bleibt dessen Ausgang auf 0, auch dann, wenn der erste Eingang auf 0 zurückgesetzt wird.



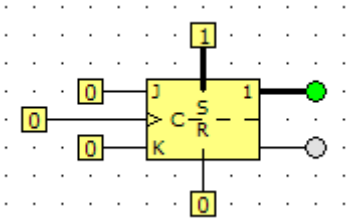
Ein Impuls auf E1 setzt den Ausgang auf 1.

3. Nun setzen wir den unteren Eingang der Schaltung auf 1. Weil jetzt beide Eingänge des unteren Gatters 1 führen, geht der Ausgang des zweiten Gatters auf 0, und er bleibt auch dann auf 0, wenn der untere Eingang der Schaltung auf 0 zurückgesetzt wird. Die 0 vom Ausgang des unteren Gatters wird zum Eingang des oberen Gatters transportiert und sorgt, solange der obere Schalter nicht gedrückt ist, dafür, dass das obere Gatter 1 ausspuckt.



Ein Impuls auf E2 setzt den Ausgang auf 0.

4. In der LogikSim-Werkzeugleiste gibt es ein fertiges Flipflop (Symbol beschriftet mit FF), das noch einige zusätzliche Eigenschaften besitzt. Beschalten Sie die Ein- und Ausgänge und untersuchen Sie die Wirkung der verschiedenen Eingangszustände.

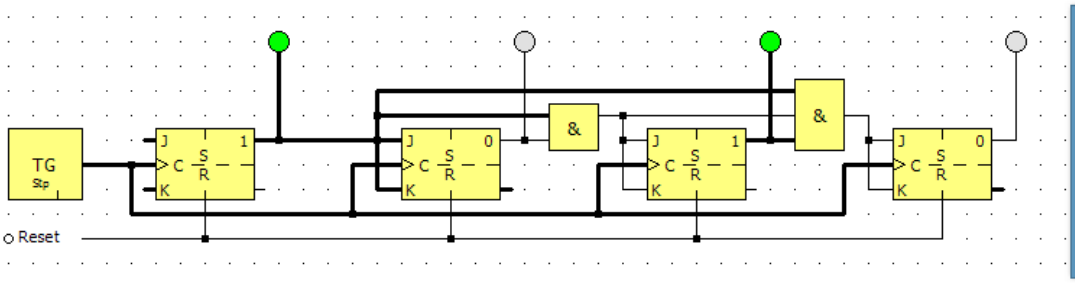


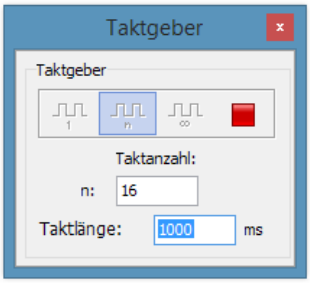
Noch zwei praktische Schaltungen

Ein typischer 74xx-Chip verfügt über drei Arten von Kontakten Zwei Pins versorgen den IC mit Strom, die anderen Kontakte dienen entweder zur Eingabe oder zur Ausgabe von Datensignalen. Die Eingänge und Ausgänge des Chips sind immer an die gleichen Leitungen angeschlossen. Alle Funktionen sind „fest verdrahtet“, das Verhalten der Schaltungen lässt sich nachträglich nicht mehr ändern. Ein Volladdierer kann nicht nachträglich zu einem „Vollsubtrahierer“ umgebaut werden. Ein Computer ist das noch nicht. Der Chip verhält sich im Wesentlichen passiv: Er reagiert zwar auf die Signale an seinen Eingängen, aber er agiert nicht selbst.

Ein Taktgeber sorgt für Action

Fügt man aber einen Taktgeber als aktives Teil hinzu, dann kann man aus solchen Bausteinen durchaus anspruchsvolle Steuerungen zusammenbauen. Laden Sie aus den Beispieldateien von LogikSim die Datei *sync-16-Zähler.sim*. Das Bauteil ganz links ist der Taktgeber, er lässt sich im Taktgeber-Dialog auf verschiedene Geschwindigkeiten einstellen. Ansonsten besteht die Schaltung aus vier Flipflops, zwei AND-Gliedern und vier LEDs.

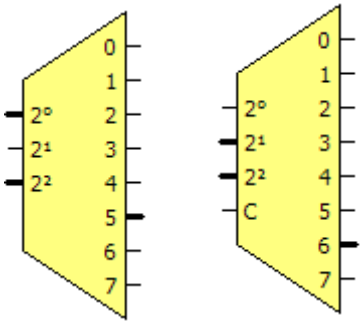




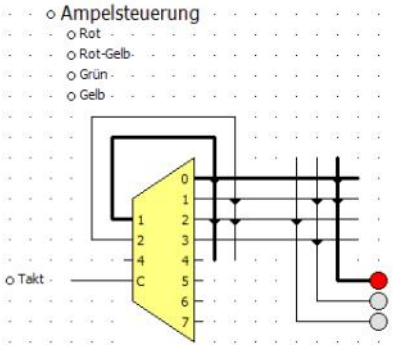
Stellen Sie den Taktgeber auf einen Takt von 1000 Millisekunden und die Anzahl n auf 16 Takte ein und starten Sie die Simulation. Die Schaltung zählt von 1 bis 16 (Im Gegensatz zur üblichen Anordnung liegen hier die Einer links).

Wenn Sie rechts noch etwas anbauen, können Sie Sekunden bis 64 hochzählen. Dann die gleiche Schaltung noch zweimal, einmal mit Überlauf bei 60, einmal bei 24, und fertig ist die Digitaluhr.

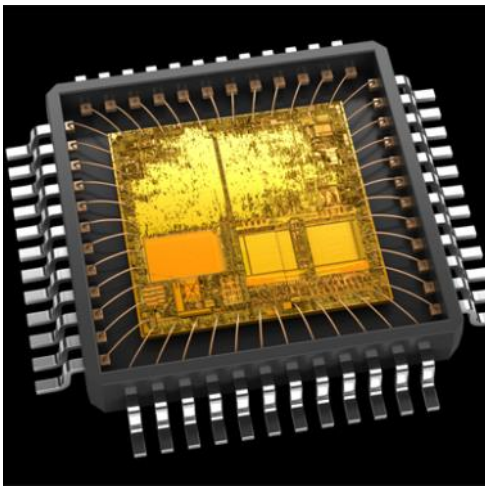
Ein Decoder sorgt für Abwechslung



Ein Bauteil, das Sie bisher noch nicht ausprobiert haben, ist der Decoder, den es in zwei Ausführungen gibt: einmal ohne und einmal mit Takteingang. Der Decoder verwandelt Dualzahlen an den Eingängen in Dezimalzahlen an den Ausgängen. Probieren Sie den Chip aus. Sie brauchen nur die Simulation zu starten und die Anschlüsse links (Eingänge) anzuklicken. Der Decoder mit Takteingang funktioniert genauso, aber er wartet auf einen Impuls an C, bevor er die Ausgänge neu beschaltet,



Eine andere LogikSim-Beispieldatei *Taktfolge.sim setzt diesen Decoderchip* als Steuerung für eine Ampel ein. Hier sind die Ausgänge zu den Eingängen rückverbunden. Nach dem Start steht der Ausgang auf 0 und die rote LED brennt. Der Ausgang 0 ist aber zum Eingang 1 rückgekoppelt. Klicken Sie zweimal auf die Taktleitung, und Sie sehen, wie dadurch der Ausgang 1 aktiviert wird. Das schaltet zusätzlich die gelbe LED ein und aktiviert gleichzeitig Eingang 1 den Eingang 2. Beim nächsten Impuls auf C wird dadurch Ausgang 2 aktiviert und das schaltet die grüne LED ein. Und so weiter.



Level 3:
Maschinensprache

In diesem Kapitel erfahren Sie, ...

- was ein Computer können muss: Befehlsfolgen, Schleifen und Verzweigungen
- dass das Herz eines Computers die CPU (Central Processing Unit) heißt
- dass ein Befehl für einen Computer in Form von Nullen und Einsen kodiert ist
- dass der Mensch sich diese Befehle als Kurzwörter merkt
- dass ein Assemblerprogramm die Kurzwörter übersetzt
- dass die CPU Transportbefehle, Rechenbefehle und Vergleichsbefehle kennt
- dass die CPU über Ports mit der Außenwelt in Verbindung tritt

Was ist eigentlich ein Computer?

Ein Computer ist eine **frei programmierbare digital arbeitende elektronische Datenverarbeitungsanlage**.

- Er besitzt Vorrichtungen zur **Eingabe**, **Verarbeitung**, **Ausgabe** und **Speicherung** von **Daten**. Eingegeben werden können Zahlen, Buchstaben, Messwerte, ausgegeben werden elektrische, akustische oder optische Signale.
- Er ist nicht auf einen Ablauf festgelegt, sondern **frei programmierbar**. Seine **Hardware** benötigt **Software**, um zu funktionieren. Dabei muss nicht jeder Benutzer gleichzeitig Programmierer sein.
- Die Programme lassen **Schleifen** (Wiederholungen) und, abhängig von zur Laufzeit auftretenden Bedingungen, **Verzweigungen** zu, die den Programmablauf ohne Benutzereingriff verändern.
- Er arbeitet **digital**, d.h. analoge Größen (Temperatur, Zeit, Spannung …) müssen zur Verarbeitung digitalisiert und zur Anwendung ggf. wieder in analoge Signale umgewandelt werden.
- Er arbeitet auf unterster Ebene **binär** (d.h. nur mit Null/Eins, aus/ein, wahr/falsch). Dezimalzahlen, Zeichen, Farben, Töne, Befehle usw. werden zur Verarbeitung binär codiert.
- Er arbeitet heutzutage immer **elektronisch**. Mechanische (Zahnräder, Lochbleche) und elektromechanische (Relais) Schalter erwiesen sich schnell als zu groß, zu langsam, zu teuer und zu energiehungrig.

Der 1938 von Konrad Zuse gebaute Rechenautomat Z1 erfüllte die meisten dieser Bedingungen. Er wurde zwar elektrisch angetrieben und er arbeitete mechanisch, aber das tat er binär und digital: mit Lochblechen als Schaltgliedern und Programmen auf Lochstreifen. Allerdings beherrschte er noch keine bedingte Verzweigung. So bezeichnet man heute erst den Nachfolger Zuse Z3 von 1943 als den ersten Computer der Welt.

Der englische Begriff *computer*, abgeleitet vom Verb (to) *compute*, bezeichnete ursprünglich Menschen, die zumeist langwierige Berechnungen vornahmen. In der New York Times tauchte das Wort erstmals am 2. Mai 1892 in einer Kleinanzeige der US-Marine mit dem Titel „A Computer Wanted“ (Ein Rechenspezialist gesucht) auf. In der Namensgebung des 1946 der Öffentlichkeit vorgestellten Electronic Numerical Integrator and *Computer* (ENIAC) taucht erstmals das Wort als Namensbestandteil auf. In der Folge etablierte sich *Computer* als Gattungsbegriff für diese neuartigen Maschinen.

(Wikipedia: ‚Computer‘, gekürzt)

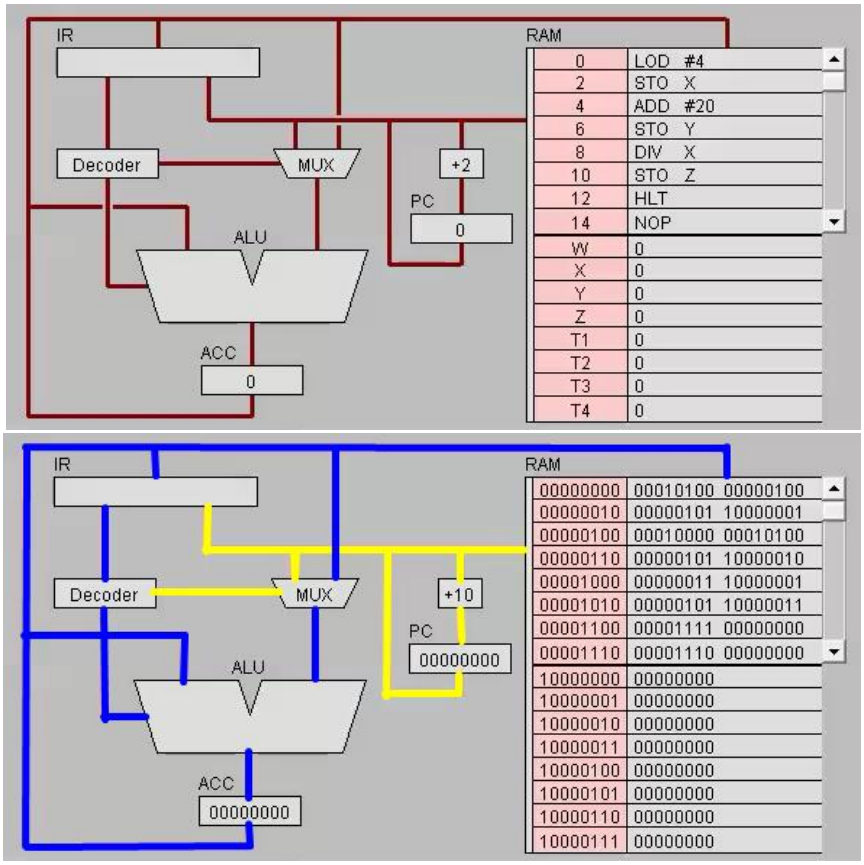
Konrad Zuse mit dem Nachbau seiner Zuse Z1, ausgestellt im Deutschen Technikmuseum Berlin.

Was tut ein Mikroprozessor?

Den Schaltkreis einer Digitaluhr oder einer Ampelschaltung mit festem Ablauf wird man noch nicht als Computer bezeichnen können, denn ein Computer ist— siehe oben— frei programmierbar. Der Chip, der diese Eigenschaft sicherstellt, ist der **Mikroprozessor**, kurz **CPU** (**Central Processing Unit**) . Man kann ihn sich im Prinzip vorstellen als einen Chip, dessen Eingänge und Ausgänge nicht fest verdrahtet sind, sondern dessen Funktion gesteuert durch Befehle verändert werden kann.

Angetrieben wird die CPU von einem externen **Taktgeber** (z. B. einem Quarz). Nach jedem Impuls des Taktgebers erhöht die CPU die Speicheradresse im **Befehlszähler** (PC: Program Counter) und holt sich den nächsten Befehl. Sie kann diese Speicheradresse aber auch programmgesteuert verändern. Für die Verarbeitung des Befehls benötigt die CPU

- ein **Rechenwerk** (ALU: Arithmetiic-Logical Unit) mit Logikschaltkreisen.
- den **Akkumulator** (ACC), in dem Zahlen addiert oder sonstwie manipuliert werden können, und weitere interne **Register** (rechts unten) zur internen Zwischenspeicherung von Daten.
- einen **Decoder**, der den Befehl im Befehlsregister in seine Befehls- und Adressbestandteile zerlegt, und einen **Multiplexer** (MUX), der Fluss der Daten und Adressen koordiniert.



Die CPU ist durch den **Datenbus** (in der Abbildung blau markiert) mit dem **Arbeitsspeicher** (RAM, rechts oben) verbunden, in dem Programme und Daten abgelegt werden. Sie kann über einen **Adressbus** (gelb markiert) jede Position dieses Arbeitsspeichers beliebig adressieren und die dort abgelegten Befehle und Daten in das **Befehlsregister** (IR, Instruction Register) laden.

Die Speicheradressen und ihre Inhalte sind in der oberen Abbildung so enthalten, wie der Prozessor sie sieht: als binäre Zahlencodes. In der Abbildung unten erkennen wir, dass es sich dabei um Zahlen und Befehle handelt. In den beiden Filmen Sequenz.mp4 und Binaer.mp4 können Sie einen Programmlauf verfolgen.

Aufgaben

1. Führen Sie zum Thema CPU eine Internetrecherche durch. Nennen Sie Unterschiede zwischen CISC und RISC-CPUs.
2. Es gibt viele verschiedene CPUs, mit unterschiedlichen Eigenschaften und jeweils verschiedenen Befehlscodes. Unter http://softwareforeducation.com oder http://wikis.zum.de/zum/Johnny finden Sie Simulatoren zum Üben.

Wenn Sie auf der folgenden Doppelseite mit CPUsim weiterarbeiten wollen, können Sie diese Doppelseite überspringen.

Abarbeiten einer Befehlsfolge

0	LOD #4
2	STO X
4	ADD #20
6	STO Y
8	DIV X
10	STO Z
12	HLT
14	NOP

Sehen wir uns nun das auf der vorigen Seite abgebildete Programm an.
Hier die Bedeutung der Befehle:

- LOD #2: Schreibe die Zahl 2 in den Akkumulator. Das Zeichen # signalisiert, dass es sich bei der folgenden Zahl nicht um eine Adresse, sondern um Daten handelt.
- ADD Y: Addiere zum Akkumulator die Zahl, die im Register Y steht.
- MUL X: Multipliziere die Zahl im Akkumulator mit der Zahl, die in X steht.
- SUB Z: Subtrahiere vom Akkumulator die Zahl, die in Z steht,
- DIV W: Dividiere die Zahl im Akkumulator durch die Zahl, die in W steht.
- STO X Überschreibe die Zahl in X mit dem Inhalt des Akkumulators.
- HLT: Halte hier an und beende das Programm
- NOP: Diese Speicherstelle enthält keinen Befehl (No Operation)

Im Film Sequenz.mp4 können Sie sich den Ablauf des abgebildeten Programms noch einmal ansehen. Versuchen Sie aber vorher durch Überlegung herauszufinden, welche Zahl am Schluss im X-Register steht.

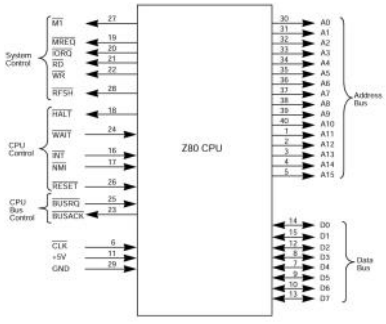
Unsere CPU kann im Akkumulator mit 8 Bit-Codes rechnen und in den Prozessorregistern 8-Bit-Codes speichern. Man spricht deshalb auch von einem 8-Bit-Prozessor. Der Zahlbereich von 8-Bit-Registern ist jedoch auf 0 bis 255 bzw. auf –128 bis +127 beschränkt und die Länge der Programme auf 256 Bytes. Aktuelle Prozessoren verarbeiten 64 Bit breite Daten und Adressen.

Unsere CPU benötigt etwa 20 Sekunden zur Ausführung eines Befehls. Sie arbeitet also mit einem Takt von 0,05 Hertz. Moderne Prozessoren arbeiten oft mit 3 Gigahertz, d.h. sie führen bis zu 3 Milliarden Befehlszyklen pro Sekunde aus.

Unsere CPU arbeitet ausschließlich mit ganzen Zahlen. Das Ergebnis der nebenstehenden Rechnung ist also 15. Moderne Prozessoren besitzen eine integrierte FPU (Floating Point Unit) und können intern auch mit Kommazahlen umgehen.

Aufgaben

1. Führen Sie zum Thema CPU eine Internetrecherche durch. Ermitteln Sie die Namen einiger älterer und aktueller CPUs und finden Sie heraus,
   - wann die CPU auf den Markt kam,
   - wie breit (in Bit) der Datenbus und wie breit der Adressbus ist,
   - mit welchem Takt (MHz, GHz: Taktzyklen pro Sekunde) die CPU arbeitet,
   - über wie viele Kerne (parallel arbeitende Unterprozessoren) die CPU verfügt.
2. Laden Sie die Kantenlänge einer quadratischen Pyramide in das X-Register und ihre Höhe in das Y-Register. Berechnen Sie das Volumen und geben Sie es in das Z-Register aus.
3. Schreiben Sie ein Programm, das den Rest der Division von X durch Y ermittelt.



Schleifen und Sprünge

Mit einer einfachen linearen Befehlsfolge ist ein Prozessor schnell fertig. Interessanter wird es bei wiederkehrenden Aufgaben. Gutmütig, wie Computer sind, genügt ein Befehl, um eine Befehlssequenz dauernd zu wiederholen. Im Befehl „JMP 2“ steht die 2 für eine Speicheradresse. Der Prozessor lädt sie nicht in den Akkumulator, sondern in den Program Counter (PC), wo die Adresse des als Nächstes auszuführenden Befehls aufbewahrt wird. Mit anderen Worten, der Prozessor kreist, bis ihm der Strom entzogen wird, zwischen den Programmzeilen 2 und 6.

0	LOD #32
2	ADD #16
4	DIV #2
6	JMP 2
8	NOP

Mit einem Programm wie dem in der Abbildung Schleife.mp4 kann man Näherungsrechnungen durchführen.

Aufgaben

1. Das abgebildete Programm läuft eigentlich unendlich lang, auch der Film dauert etwa 7 Minuten. Überlegen Sie, ob der Wert im X-Register in Richtung einer bestimmten Zahl konvergiert. Welche ist es?

Bedingte Verzweigungen

Mit Wiederholungsbefehlen sollte man vorsichtig sein, das lehrt uns Goethes Zauberlehrling. Deshalb gehören zum Befehlssatz aller Prozessoren auch Befehle, mit denen man Abbruchbedingungen formulieren kann. Im nebenstehend abgebildeten Programm sind zwei neue Befehle enthalten:

0	LOD #3
2	STO X
4	LOD X
6	SUB #1
8	STO X
10	CPZ X
12	JMZ 4
14	HLT

Zwei Befehle sind neu:

- -CPZ X bedeutet "Compare X to Zero" oder "Vergleiche X mit 0". Wenn das X-Register zum Zeitpunkt der Ausführung dieses Befehls 0 enthält, dann wird der Akkumulator auf 1 gesetzt, ansonsten auf 0.
- JMZ 4 bedeutet "Jump on Zero to address 4" oder „Springe zu Programmzeile 4, wenn der Akkumulator gerade 0 enthält. Ist das nicht der Fall, dann setze das Programm mit dem nächsten Befehl fort.

Starten Sie den Film unter Verzweigung.mp4 und beobachten Sie das Register X. Es wird zu Beginn mit der Zahl 3 geladen, dann auf 2, auf 1 und schließlich auf 0 heruntergezählt und wenn die 0 erreicht ist, hält das Programm an.

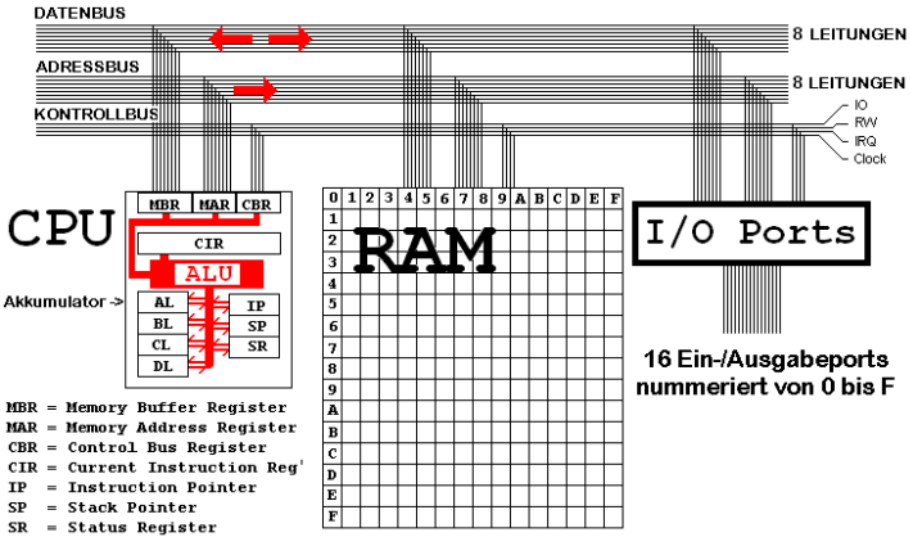
Aufgaben

2. Prozessorbefehle lassen sich als arithmetische Operationen, logische Operationen und Transportoperationen klassifizieren. Ordnen Sie die Befehle der abgebildeten Programme einer dieser drei Kategorien zu.
3. Schreiben Sie ein Programm, das im X-Register von 0 bis 10 zählt und anhält.
4. Speichern Sie eine beliebige einstellige Zahl n im X-Register. Schreiben Sie ein Programm, das die Summe aller Zahlen von 1 bis n berechnet und in Y ablegt. (Das Programm wird kürzer und damit schneller, wenn Sie mit der in X gegebenen Zahl anfangen und rückwärts bis zur 1 gehen.)

Das verwendete CPU-Modell wurde 1998 von Rick Decker und Stuart Hirshfield als Experimentierumgebung zu dem Buch „The Analytical Engine: An Introduction to Computer Science Using the Internet“ entwickelt. Leider genügen die Java Applets heutigen Sicherheitsanforderungen nicht mehr, deshalb wurden sie hier „abgefilmt“.

Ein CPU-Simulator

Der englische Lehrer Neil Bauers hat einen CPU-Simulator für ein vereinfachtes Modell der in Desktop-PCs üblicherweise verwendeten x86-CPUs geschrieben, mit dem Sie selbst experimentieren können. Außer einer CPU-Logik und einem kleinen Arbeitsspeicher (RAM) simuliert es Ein- und Ausgänge (Ports) mitsamt einiger Geräte. Zwar können Sie hier nicht die Daten über den Bus flitzen sehen, wohl aber können Sie Registerinhalte wahlweise in binärer, in hexadezimaler, in Zeichen– oder Befehlscodeform beobachten. Sie finden das (englische) Original unter http://softwareforeducation.com .



In größeren Systemen können dabei CPU, RAM und Ports getrennt untergebracht sein, in kleinen spezialisierten Steuerungssystemen sind sie auf einem Chip vereint.

Jede Speicherzelle im RAM, jeder Port zum Anschluss von Geräten besitzt eine eigene Nummer als Adresse. Indem die CPU diese Nummer auf den Adressbus legt, kann sie die RAM oder Ports auslesen oder beschreiben. Eine 0 auf der RW-Leitung signalisiert dabei einen Lesevorgang, eine 1 einen Schreibvorgang. Ist die IO-Leitung auf 0, so redet die CPU mit dem RAM, ist sie 1, so spricht sie einen Port an.

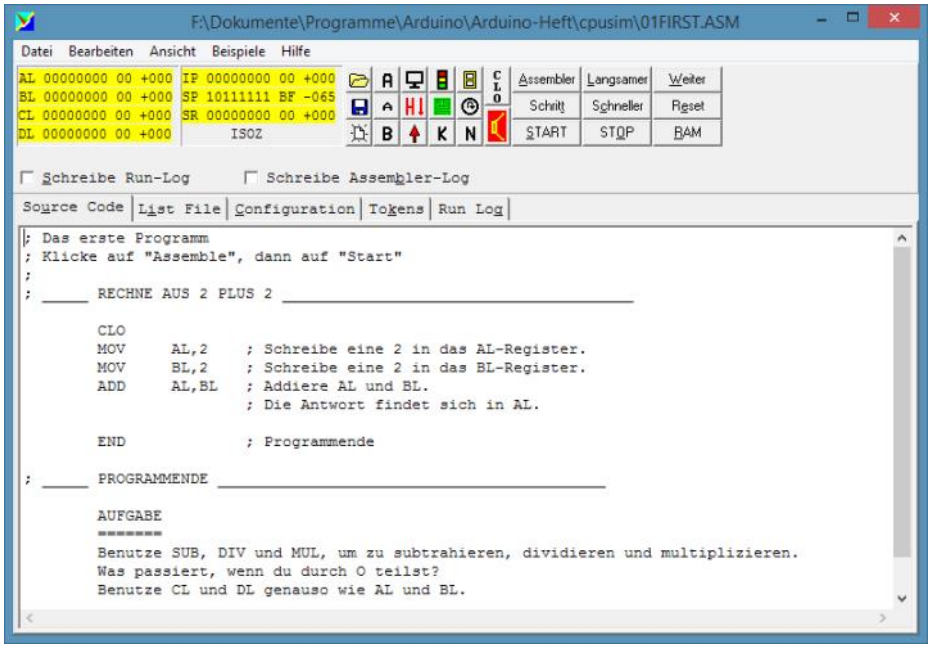
Starten Sie das Programm. Im Bereich links oben, nach dem Start ist er gelb hervorgehoben, sehen Sie die Register der CPU. Die **Datenregister** AL, BL, CL und DL können 8-Bit-Zahlen speichern. Sie werden hier sowohl binär als auch hexadezimal und dezimal angezeigt. Da das erste Bit als Vorzeichen interpretiert wird, reicht der **Zahlenbereich** dieser CPU von -127 bis +128. (In einer realen CPU sind die Register meist 32 Bit breit und fassen Zahlen bis zu ±2 Milliarden.)

Dabei fungiert das Register **AL** als **Akkumulator**. Die Ergebnisse von Rechenoperationen finden sich immer in diesem Register.

Der **Befehlszeiger IP** (Instruction Pointer) gibt die Adresse im RAM an, wo der nächste Befehl zu finden ist.

Der **Stapelzeiger SP** zeigt auf einen Speicherbereich, in dem die CPU kurzfristig Zwischenergebnisse ablegen kann.

Das **Statusregister SR** enthält die vier **Flags** I, N, Z und O. In jeweils einem Flag merkt sich die CPU, ob das letzte Ergebnis Null (**Z**ero), negativ (**S**ign) oder größer als 128 (**O**verflow) war. Mit dem I-Flag kann ein Programm Unterbrechungen verbieten.



Assemblerprogramme

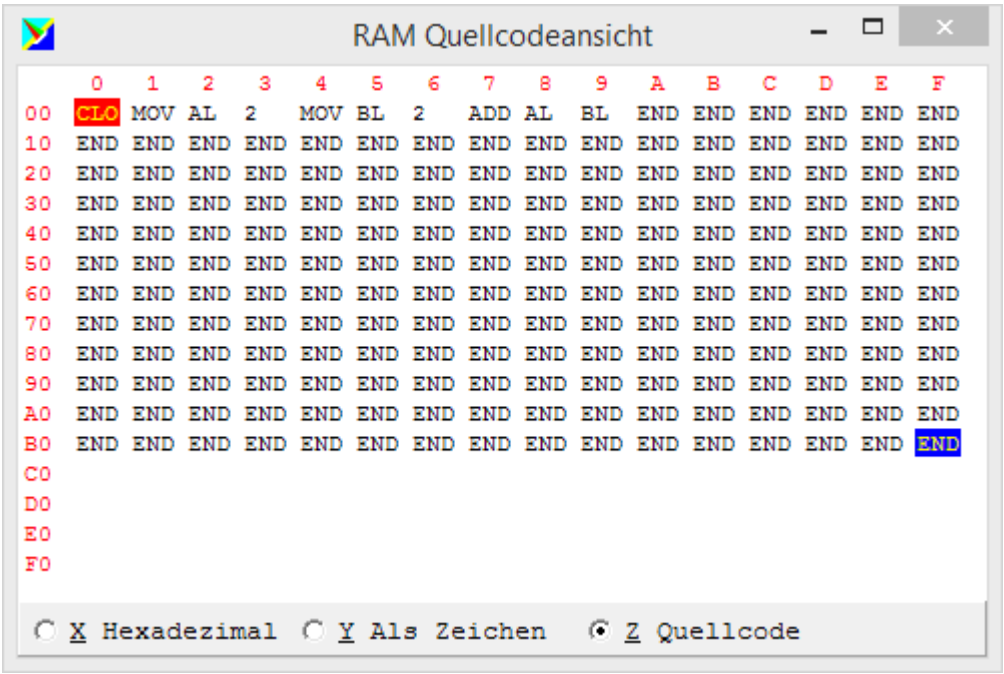
Im Hauptfenster des Simulators können Sie **Assembler**-Programme schreiben. Die Befehle dieser Programmiersprache beziehen sich jeweils speziell auf die verwendete CPU und ihre Möglichkeiten. Das erste Programm laden wir fertig herunter. Öffnen Sie über das Datei-Menü das abgebildete Programm 🖫 01FIRST.ASM.

Was Sie im Hauptfenster sehen, nennt man **Quellcode.** Befehle sind hier in **Mnemonics** notiert. Das sind verständliche Abkürzungen wie ADD (Addiere) oder MOV (Move). Manche Befehle stehen allein, andere benötigen ein oder zwei **Operanden** (Daten oder deren Speicheradresse) in der Reihenfolge erst Ziel, dann Was oder Woher.

```
CLO             ; Nicht gewünschte Fenster schließen.
MOV     AL,2    ; Schreibe eine 2 in das AL-Register.
MOV     BL,2    ; Schreibe eine 2 in das BL-Register.
ADD     AL,BL   ; Addiere AL und BL.
                ; Die Antwort findet sich in AL.

END             ; Programmende
```

Damit die CPU dieses Programm versteht, muss es aus der für Menschen verständlichen Assemblerspreche in die für die CPU verständlichen binären Befehlscodes umgewandelt werden. Dies übernimmt der so genannte **Assembler**.

- Klicken Sie auf die Taste **Assembler**. Die Befehle des Programms werden in Maschinencode übersetzt und in den Arbeitsspeicher geschrieben. Was hinter einem Semikolon steht, bleibt beim Assemblieren unbeachtet.
- Der Arbeitsspeicher **RAM** des Modellcomputers ist von 00 bis FF durchnummeriert. Der nächste Befehl (Adresse im Befehlszeiger IP) ist im RAM-Fenster rot markiert. die Adresse, auf die der Stapelzeiger SP zeigt, blau.



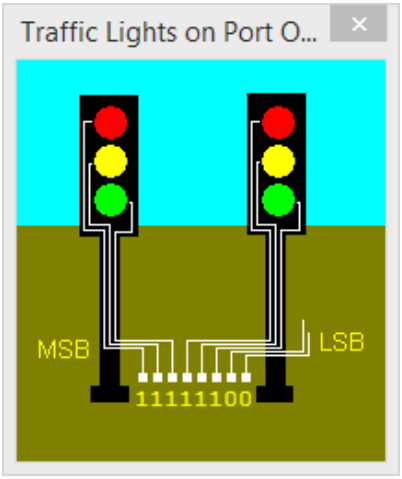
- Klicken Sie auf **Schritt.** Jedes Mal wird ein Befehl des Programms ausgeführt. Beobachten Sie, wie Zahlen in die Register des Prozessors geholt und dort verarbeitet werden. Mit **Start** beginnt ein ganzer Programmdurchlauf, mit **Langsamer** und **Schneller, Stopp** und **Weiter** können Sie das Tempo regeln.

Aufgaben

1. Vielleicht versuchen Sie jetzt, die am Ende des Textes stehende Aufgabe zu bearbeiten. Hier können Sie weitere Erklärungen finden (einstweilen in Englisch)
2. Testen Sie auch die anderen Programme. Das Programm simuliert nicht nur Prozessor und RAM, sondern auch die Kommunikation mit Peripheriegeräten. Das Programm 🖫 DEMO.ASM liefert einen schönen Überblick.

Geräte steuern

Im ersten Programm hat unsere CPU mit dem RAM kommuniziert, nun werden wir versuchen ein simuliertes Gerät zu steuern. Wenn die CPU die Adresse 01 auf den Adressbus legt und gleichzeitig die IO-Leitung auf 1 setzt, fühlt sich nicht der Arbeitsspeicherpalt 01, sondern der Port 01 angesprochen. Wenn die CPU außerdem RW auf 1 setzt, heißt das, dass sie den Port nicht auslesen, sondern beschreiben will. In der Praxis gibt es für jeden Port ein gesondertes Register, in dem festgelegt wird, ob er als Eingang oder Ausgang genutzt wird.



Nehmen wir an, an Port 1 sei eine Ampel angeschlossen. Das geht normalerweise nicht direkt. Mit den Strömen, die durch eine CPU fließen, kann man die Lampen einer Ampel nicht betreiben. Deshalb sitzt zwischen dem Port der CPU, den man sich durchaus als Stecker vorstellen kann, und dem Stecker, an den die Ampel angeschlossen ist, noch ein Interface mit Relais als Verstärker.

Laden Sie die Datei 🖫·02TLIGHT.ASM.

Mit dem Befehl OUT 01 kann die CPU den Inhalt des Akkumulators in den Port 1 übertragen und der Ampel mitteilen.

In einem anderen Programm könnte die Steuerung z.B. mit IN 02 den Port 2 auslesen und feststellen, ob ein externes Gerät den Port auf einen bestimmten Wert gesetzt hat.

```
; ------------------------------------------------------------
; Steuerung einer Ampel
; ------------------------------------------------------------

        CLO             ; Schließe nicht gewünschte Fenster.
Start:                  ; Dies ist ein Label (ein Sprungziel)
                        ; Alle Lampen der Ampeln abstellen:
        MOV     AL,0    ; Schreibe 00000000 in das AL-Register.
        OUT     01      ; Schicke AL an den Port 1 (die Ampeln).
                        ; Alle Lampen der Ampeln anstellen:
        MOV     AL,FC   ; Schreibe 11111100 in das AL-Register.
        OUT     01      ; Schicke AL an den Port 1 (die Ampeln).
        JMP     Start   ; Springe zurück zum Label Start.
        END             ; Programende.
```

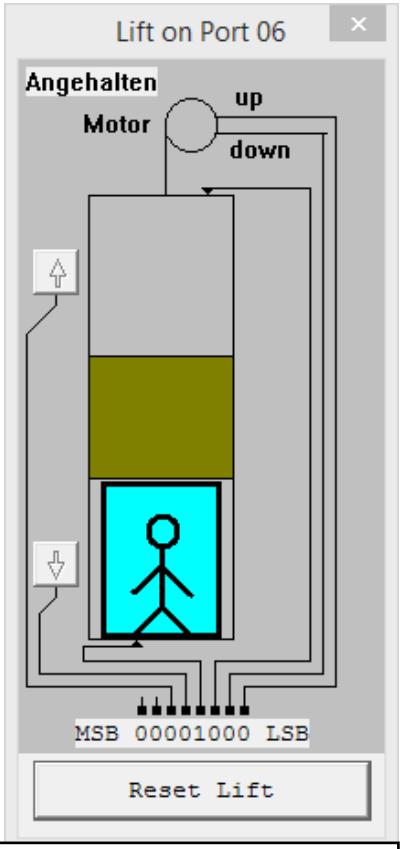
Außerdem finden Sie in diesem Programm einen Wiederholungsbefehl. Im Programm steht das **Label** „Start“. Ein Label ist kein Befehl an den Computer, sondern eine Anweisung an das Übersetzungsprogramm. Sobald der Assembler beim Übersetzen auf ein solches Label stößt, merkt er sich die Adresse des nächsten Befehls (hier MOV AL,0. Wenn dann im Programm mit JMP zu diesem Label gesprungen wird, dann setzt der Assembler die Speicheradresse dieses Befehls als Sprungziel ein. So ergibt sich ein endloser Kreislauf.

Aber sehen Sie selbst…

Aufgaben

1. Finden Sie heraus, welche Zahlen Sie in A0 ablegen müssen, um jeweils eine der 6 Lampen anzusprechen.
2. Erweitern Sie das Programm so, dass beide Ampeln im gleichen Takt laufen: Rot -> Rot+Gelb -> Grün -> Gelb -> Rot und dann alles wieder von vorn.
3. Schreiben Sie ein Programm, das die beiden Ampeln gegenläufig schaltet—als die beiden Richtungen einer Baustellenampel.
4. Bearbeiten Sie die drei Dateien: 🖫 03MOVE.ASM, 🖫 04INCJMP.ASM und 🖫 05KEYBIN.ASM der Reihen nach. Dann dürfen Sie beliebig weiter forschen.

Nur unter der Bedingung, dass ...



Drei Fähigkeiten muss eine Maschine haben, um als Computer zu gelten: Sie muss

- eine **Sequenz** (Folge) von Befehlen ohne Eingriff von außen abarbeiten können,
- eine gegebene Befehlsfolge in Form einer **Schleife** wiederholen können,
- Bedingungen prüfen können und eine **Verzweigung** der Bearbeitung zulassen.

Die ersten beiden Punkte waren im Ampelprogramm schon gegeben. Den dritten Punkt soll das folgende Programm veranschaulichen:

Ein Lift, der an Port 6 unseres Simulators angeschlossen ist, besitzt einen Motor, der bei gesetztem Einerbit (LSB = Least Significant Bit) aufwärts und bei gesetztem Zweierbit abwärts fährt. Darüber hinaus kann der Simulator prüfen, ob einer oder mehrere der vier Schalter am oberen Anschlag (4er-Bit), unteren Anschlag (8er-Bit), Hochfahrwunsch (16er-Bit) oder Herunterfahrwunsch (32er-Bit) gesetzt ist. Das 64er-Bit und das 128er-Bit (MSB=Most Significant Bit) sind nicht angeschlossen.

Unser noch ziemlich schlichtes Programm 04LIFT.ASM stoppt zunächst den Motor . Da die Taste am unteren Anschlag gedrückt ist, liefert die Portabfrage IN 06 den Wert 8. Der Befehl AND 20 betrachtet isoliert das 32er-Bit. Das Ergebnis ist 0.

Der Befehl JNZ (Jump on not Zero, also AL ≠ 20) verzweigt zum Label „Start:“. Wenn nun die Taste „Hochfahren“ gedrückt wird, dann wird der das Resultat 32 (also hexadezimal 20) und das Programm geht unten weiter.

Der Motor wird eingeschaltet, und da kein Schalter mehr gedrückt ist, liefert die Abfrage IN 06 jetzt 1 das isolierte 4er-Bit liefert 0 und CMP AL,4 ergibt ungleich. Das Programm verzweigt hoch zum Label „Weiter“.

Irgendwann aber stößt der Aufzug am oberen Schalter an (4) und IN 06 liefert nun 5 und das 4er-Bit ist gesetzt. Dann ist es Zeit, mit MOV AL,0 den Motor auszuschalten, sonst crasht der Aufzug an der Decke.

```
; ------------------------------------------------------
; Liftsteuerung
; ------------------------------------------------------

        MOV     AL,0    ; Motor stoppen
        OUT     06
Start:
        IN      06      ; Alle Bits abfragen
        AND     AL,20   ; 32er-Bit isolieren (20h ist 32d)
        CMP     AL,20   ; Vergleich mit AL
        JNZ     Start   ; Wiederholen bis 32er-Bit gesetzt
        MOV     AL,1    ; Motor ein
        OUT     06
Weiter:
        IN      06      ; Alle Bits abfragen
        AND     AL,4    ; 4er Bit isolieren
        CMP     AL,4    ; Vergleich mit AL
        JNZ     Weiter  ; Wiederholen bis 4er-Bit gesetzt
        MOV     AL,0    ; Motor ausschalten
        OUT     06
        END
```

Aufgaben

1. Studieren Sie das Verhalten des Programms ausgiebig im Schrittmodus.
2. Versuchen Sie das Programm so zu ergänzen, dass der Aufzug auf einen weiteren Knopfdruck wieder abwärts fährt bis zum unteren Anschlag.
3. Testen Sie das ausführlichere Programm 🖫 LIFT.ASM (ohne 04_) im RUN– und im Schrittmodus und versuchen Sie es zu verstehen.
4. Untersuchen Sie andere Beispielprogramme und lösen Sie die Aufgaben.

Die CPU des Arduino

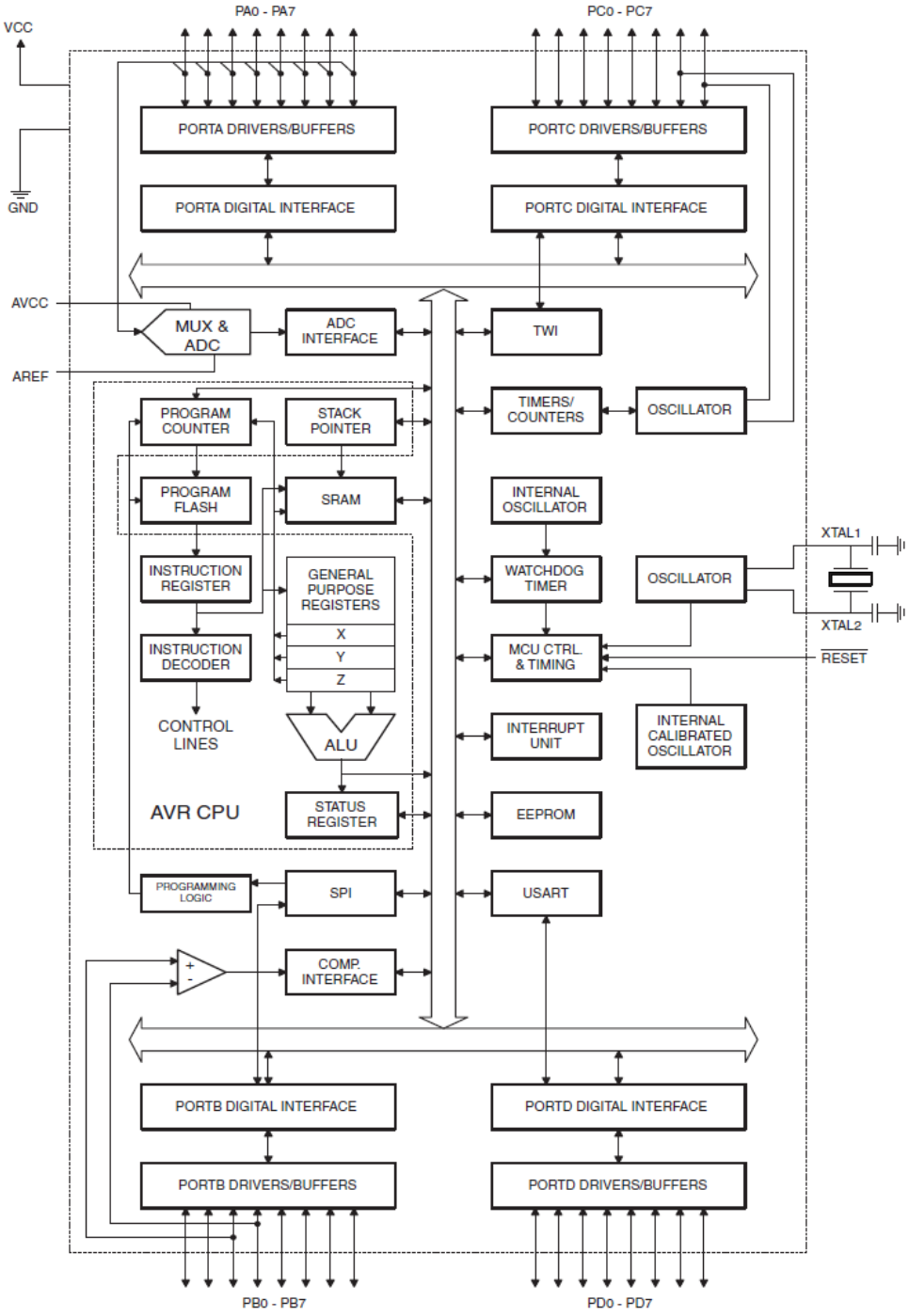
Die CPU, die Sie auf Ihrem Arduino finden, gehört zur Familie des **Atmel ATmega**. Auf dem abgebildeten Blockschaltbild (es stammt aus dem Datenblatt des ATmega32) werden Sie bekannte Elemente entdecken:

- das Rechenwerk ALU,
- eine Reihe von Prozessorregistern, eines davon ist der Akkumulator,
- Befehlszähler, Befehlsregister, Decoder und Statusregister,
- mehrere Ports zum Anschluss externer Geräte,
- den Datenbus, dargestellt durch die drei Doppelpfeile.

Da der Chip außer der CPU auch den Arbeitsspeicher enthält (flüchtigen SRAM als Datenspeicher und stromausfallresistenten Flash-Speicher als Programmspeicher), spricht man von einem „**System on a chip**“ oder einem „**Mikrocontroller**“.

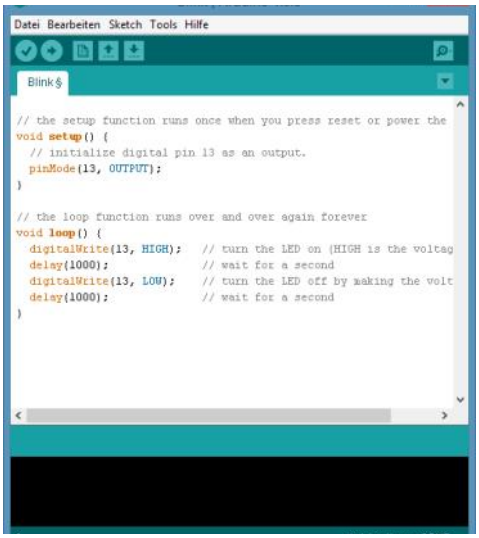
Anders als in einem PC-Prozessor sind hier weder Adressbus noch Datenbus nach außen geführt. Die Kommunikation mit der Außenwelt erfolgt über die Ports. Von außen kommen außerdem die Stromversorgung (VCC/GND), eine Resetleitung und der Arbeitstakt für den Prozessor. Der Taktgeber ist ein Oszillator, der als kleines silbernes Oval auf der Platine aufgelötet ist.

Der Chip selbst wird in zwei Bauformen verkauft: als kleines Quadrat zum Auflöten auf die Platine oder in der DIL-Bauform (Dual-In-Line) in der Fassung.



Aufgaben

1. Finden Sie heraus:
   - wie viel Bit breit der Datenbus des ATmega32 ist,
   - wie viele Prozessorregister (General Purpose Registers) der ATmega32 besitzt,
   - wie groß der Programmspeicher (Flash), der Arbeitsspeicher (SRAM) und der nichtflüchtige Datenspeicher (EEPROM) dimensioniert sind,
   - wie viele Befehle die CPU des ATmega-Prozessors versteht,
   - wie viele Befehlszyklen pro Sekunde ausgeführt werden,
   - was die Abkürzung ADC bedeutet.



Level 4:
Hochsprache

In diesem Kapitel erfahren Sie, ...

- wie man einen Mikrocontroller vom PC aus programmiert
- was eine Hochsprache von Maschinensprache unterscheidet
- wie der Arduino Aktoren (Lampen oder Motoren) in Gang setzt
- wie der Arduino mithilfe von Sensoren seine Umwelt erkundet

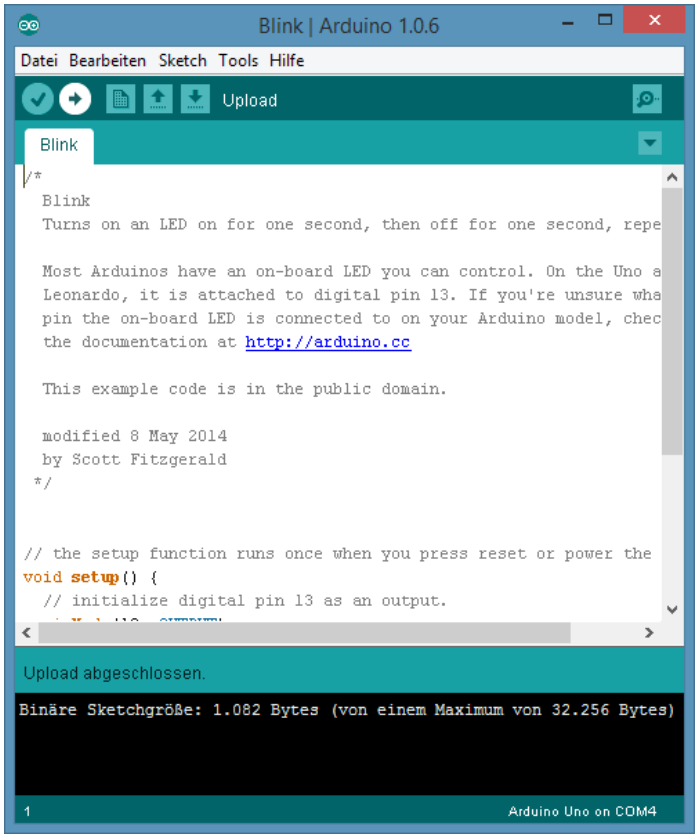
Programme hochladen

Den Simulator haben wir auf dem PC-Bildschirm mit Maschinenspracheprogrammen gefüttert. Unseren echten Arduino haben wir bisher nur als Stromquelle genutzt. Aber wie kommen wir an seine CPU und seinen Programmspeicher heran?

Unter http://arduino.cc finden Sie alles, was Sie brauchen: eine komplette Programmierumgebung, die auch das Hochladen von Programmen zum Arduino regelt. Laden Sie sich die neueste Version der Arduino-Software herunter, installieren Sie sic auf Ihrem PC und starten Sie „Arduino.exe“

Das Programmfenster enthält von oben nach unten:
die Menüleiste (weiß)
die Symbolleiste (dunkelgrün)
die Programmtabs (hellgrün mit kleinem Menü)
den Programmeditor (weiß)
die Meldungszeile (grün)
den Fehlerhinweis-Bereich (schwarz)
die Statusleiste (grün)

Wenn Sie mit der Maus über die Symbolleiste fahren, werden die Befehle kurz erklärt.



Verbindung aufnehmen

Schließen Sie den Arduino an das USB-Kabel an. Klicken Sie *Tools -> Board* und wählen Sie den richtigen Arduino-Typ aus. In diesem Kurs wird mit dem UNO gearbeitet, aber Sie besitzen ja vielleicht einen anderen Mega oder einen Leonardo.

Klicken Sie *Tools -> Serieller Port*. Hier sollte außer den internen Ports COM1 und COM2 Ihres PC noch ein weiterer Port erscheinen, an den Ihr Arduino angeschlossen ist. Wählen Sie diesen aus. Das Ergebnis lesen Sie in der Statuszeile.

Das erste Programm

Klicken Sie nun *Datei -> Beispiele - > 01 Basics -> Blink* und laden Sie das abgebildete Beispielprogramm. Speichern Sie es am besten gleich mit *Datei -> Speichern unter ...*. Nun können Sie das Programm nach Herzenslust manipulieren, ohne einem eventuellen Nachfolger ein Chaos zu hinterlassen. Arduino-Programme heißen **Sketch** und die Arduino-Software speichert sie standardmäßig in einem eigenen Ordner in Ihrem persönlichen „**Sketchbook**“.

Ein Klick auf *Neu* (den Zettel in der Symbolleiste) erzeugt einen neuen Sketch in einem zusätzlichen leeren Fenster. Wenn Sie das Symbol *Öffnen* (Pfeil nach oben) anklicken, dann sehen Sie oben die Programme Ihres Sketchbooks, darunter die Beispielprogramme und unten einige Libraries (Bibliotheken zu bestimmten Bauteilen). Ein Klick auf *Speichern* (Pfeil nach unten) führt in Ihr Sketchbook.

Ein Klick auf *Überprüfen* (Haken) überprüft das Programm im Editor auf grammatikalische Korrektheit, ein Klick auf *Upload* (Rechtspfeil) lädt es hoch zum Arduino und startet es. Und schon sollte eine der LEDs auf dem Arduino anfangen zu blinken. Beachten Sie die erzeugten Meldungen.

Ab jetzt sprechen wir C++

Die Sprache, in der wir bei der Arduino-Software unsere Programme formulieren müssen, ist ein an den Arduino angepasster Teil der höheren Programmiersprache C++. Der Begriff **höhere Programmiersprache** meint dabei, dass das Programm in einer dem Englischen ähnlichen Sprache formuliert wird. Das heißt, wir brauchen den verwendeten Prozessor und seine Maschinensprache nicht zu kennen. Unser Programm wird vor dem Hochladen von einem in der Arduino-Software enthaltenen **Compiler** (Übersetzer) in die Sprache des Prozessors übertragen.

Das eigentliche Programm besteht aus den in Schwarz, Orange oder Türkis gedruckten Wörtern. Die grau gedruckten Teile sind **Kommentare**. Sie dienen zur Erläuterung und als Gedächtnisstütze, sind aber für den Arduino nicht ausführbar. Kommentare, die mit // eingeleitet werden, gelten für den Rest der Zeile. Mehrzeilige Kommentare werden mit /* eingeleitet und mit */ beendet.

Das Programm besteht aus zwei mit Namen benannten **Funktionen** (Befehlsfolgen), die in jedem Arduino-Programm definiert werden müssen: Die Funktion *setup()* wird ein einziges Mal nach dem Einschalten ausgeführt, die Funktion *loop()* wird danach in einer Endlosschleife wiederholt. Vor dem Namen beider Funktionen steht der Begriff *void*. Er zeigt an, dass die Funktion nach der Ausführung keine Daten zurückgibt. Hinter dem Namen beider Funktionen steht jeweils eine leere Klammer. Sie zeigt an, dass die Funktion zu ihrer Ausführung keine **Argumente** (notwendige Zusatzinformationen) braucht.

Blink

```
/*
  Blink
  Turns on an LED on for one second, then off for one second, repeatedly.

  Most Arduinos have an on-board LED you can control. On the Uno and
  Leonardo, it is attached to digital pin 13. If you're unsure what
  pin the on-board LED is connected to on your Arduino model, check
  the documentation at http://arduino.cc

  This example code is in the public domain.

  modified 8 May 2014
  by Scott Fitzgerald
 */


// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin 13 as an output.
  pinMode(13, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(13, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);              // wait for a second
  digitalWrite(13, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);              // wait for a second
}
```

Der Programmcode beider Funktionen steht jeweils zwischen zwei geschweiften Klammern. Im vorliegenden Programm setzt *setup()* mithilfe der Funktion *pinMode()* den Digitalpin 13 auf OUTPUT, sie teilt also dem Arduino mit, dass über Pin 13 Daten ausgegeben werden sollen. Die Funktion *pinMode()* benötigt zwei Parameter: auf welchen Pin sie sich bezieht und ob dieser Pin als Eingang oder als Ausgang benutzt werden soll. Der Aufruf jeder Funktion schließt mit einem Semikolon.

loop() besteht aus vier **Funktionsaufrufen**: *digitalWrite(13,HIGH);* setzt den Ausgang 13 unter 5 Volt Spannung, *delay(1000);* veranlasst den Prozessor, 1000 Millisekunden lang Däumchen zu drehen. *digitalWrite(I13, LOW)* setzt den Ausgang auf 0 Volt. Danach wird wieder 1000 Millisekunden gewartet, bevor das Spiel neu beginnt. HIGH und LOW sind dabei vordefinierte **Konstanten** für die Zahl 1 (HIGH) und 0 (LOW). Solche Konstanten machen ein Programm besser lesbar.

In der Mathematik bezeichnet das Wort „**Funktion**“ eine Formel, mit der man aus einer oder mehreren Argumenten (Eingaben) einen Funktionswert berechnen kann. In der Sprache C++ steht es auch für solche Befehle oder Befehlsfolgen, die kein Ergebnis liefern. Solche Funktionen werden in C++ als void (leer) deklariert.

Die geschweiften Klammern können Sie auf der PC-Tastatur mit <AltGr>+7 (öffnende Klammer) und <AltGr>+0 (schließende Klammer) eingeben.

Aufgaben

1. Ändern Sie die Länge der Pausen.
2. Lassen Sie den Arduino SOS blinken:
   kurz kurz kurz lang lang lang kurz kurz kurz Pause

Der Rest geht auch ohne PC

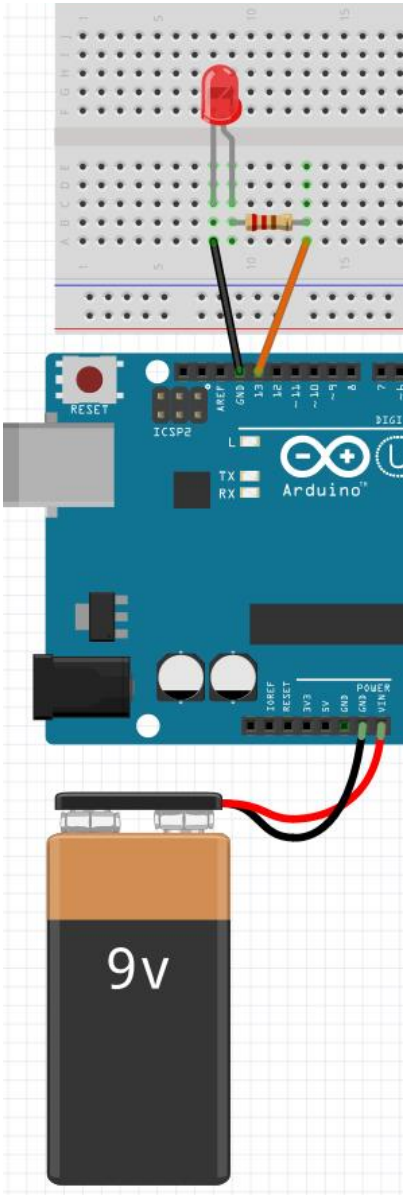
Unser Arduino ist zwar immer noch über das USB-Kabel mit dem PC verbunden, aber das Programm wird auf dem Arduino ausgeführt, nicht auf dem PC.

Aufgaben

1. Beenden Sie die Arduino-Software auf dem PC. Der Arduino sollte weiter blinken.
2. Trennen Sie den Arduino vom USB-Stecker. Die LED erlischt. Sobald Sie ihn wieder anschließen, setzt das Blinken wieder ein. Vorher aber flackert die LED kurz.
3. Drücken Sie die rote **Reset-Taste** neben dem USB-Anschluss. Der Effekt ist der gleiche. Wieder flackert die LED, um dann wieder regelmäßig zu blinken.
4. Trennen Sie den Arduino vom PC und schließen Sie, falls vorhanden, eine externe Stromversorgung an. Das Programm beginnt sofort wieder.

Das Flackern der LED kommt diesmal nicht von unserem Programm, sondern es ist ein Zeichen dafür, dass der Arduino zurückgesetzt wurde. Ein **Reset** löscht die Prozessorregister und die Datenregister im SRAM. Das Programm jedoch, das im nichtflüchtigen Speicher abgelegt wurde, bleibt erhalten. Der Befehlszähler der CPU zeigt dorthin, wo vom Compiler die *setup()*-Funktion abgelegt wurde.

Als **externe Stromversorgung** können Sie wahlweise ein Netzteil mit 5 bis 12 Volt Gleichstrom, eine 9V-Blockbatterie oder einen Batteriehalter mit vier bis acht AA oder AAA-Batterien anschließen. Verbinden Sie **das rote Kabel mit Vin** und **das schwarze Kabel mit GND** des Arduino. Besser ist es aber, Sie verwenden gleich einen **DC-Hohlstecker** an der großen schwarzen Buchse des Arduino. (Maße innen 2,1 mm an rot außen 5,5 mm an schwarz). Dann können Sie die Versorgungsspannung bei Bedarf über den Vin-Anschluss an andere Bauteile weitergeben.



Und was ist mit den anderen Pins?

Der Arduino besitzt zwar 14 digitale Anschlüsse (nummeriert mit 0 bis 13), aber nur Ausgang 13 ist mit einer LED auf der Arduino-Platine verbunden. Wir wollen nun eine beliebige LED oder mehrere LEDs auf dem Breadboard blinken lassen.

Aufgaben

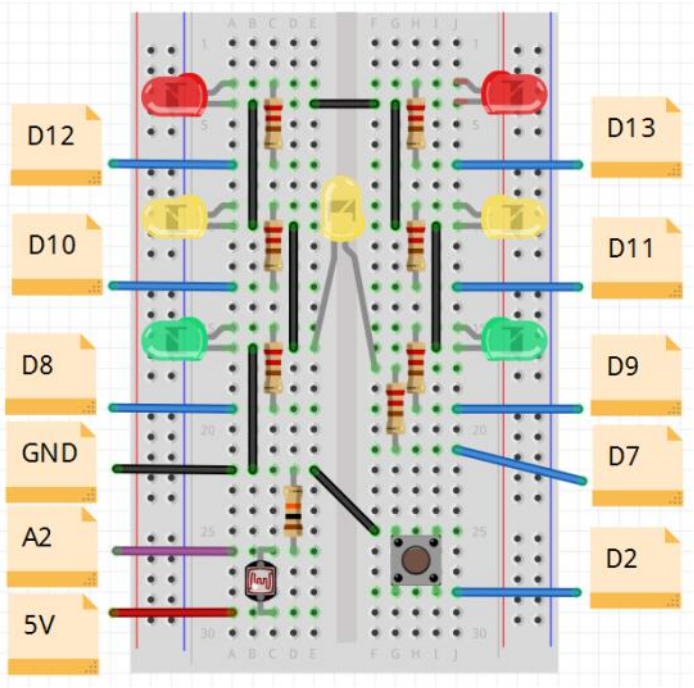
5. Stecken Sie eine LED und einen 220Ω-Widerstand so auf das Breadboard, dass die Anode der LED (das längere Bein) mit dem Widerstand verbunden ist. Verbinden Sie wie abgebildet das freie Ende des Widerstands mit Digitalanschluss 13 des Arduino und den freien Pin der LED mit GND. Die LED sollte jetzt zusammen mit der LED auf dem Arduino blinken.
6. Ändern Sie das Programm so ab, dass nicht mehr der Digitalausgang 13, sondern einer der anderen Digitalausgänge verwendet wird. Schließen Sie die LED an diesen Ausgang an und laden Sie das Programm erneut hoch.
7. Setzen Sie eine zweite LED samt Vorwiderstand auf das Breadboard und lassen Sie die beiden LEDs im Wechsel blinken.
8. Denken Sie sich selbst etwas aus.

Eine Ampel für den Arduino

Auf den nächsten Seiten werden wir verschiedene Aspekte der Programmierung des Arduino kennen lernen. Damit wir uns dabei voll auf die Software konzentrieren können, bauen wir uns zunächst ein etwas komplexeres Hardware-Modell auf. Legen Sie das folgende Material bereit:

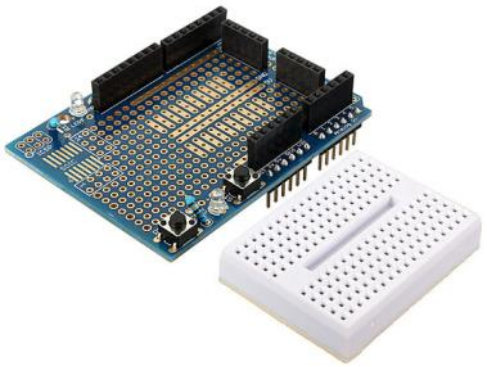
Stückliste

Arduino UNO, Breadboard, 11 Kabel m/m
2 LED rot, 3 LED gelb, 2 LED grün
7 Widerstände 220 Ω, 1 Widerstand 10kΩ
Draht für Steckbrücken, Zange
1 Mikroschalter
1 Fotozelle



Sie können so wie rechts oben abgebildet Ihr Breadboard verwenden. Die elf Verbindungen zum Arduino realisieren Sie hier mit Steckkabeln Das „D“ bedeutet „Digitalanschluss“. „A2“ ist der Analogeingang Nummer 2 und „5V“ die schon öfter verwendete 5V-Leitung des Arduino. Die mittlere gelbe LED, den Schalter und die Fotozelle links benötigen wir erst später, wir bauen die Teile aber jetzt schon ein. Der zur Fotozelle gehörige Widerstand beträgt 10kΩ.

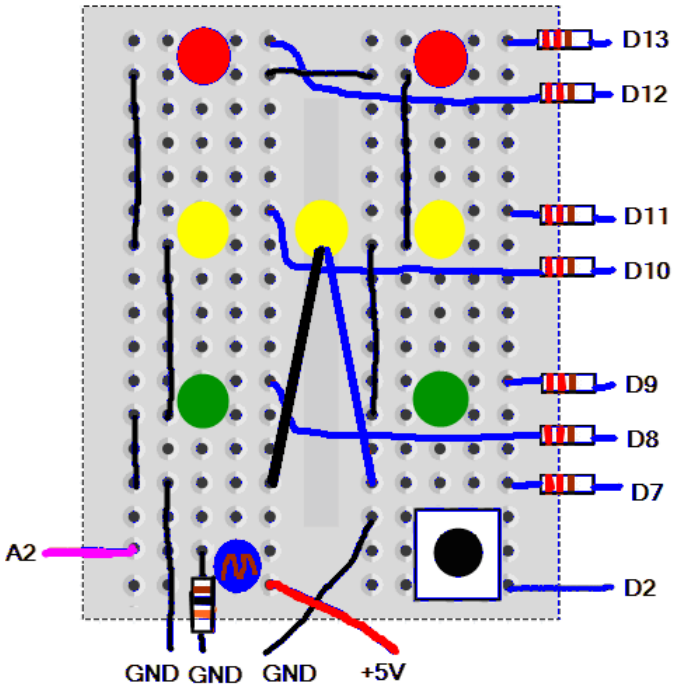
Ich schlage vor, dass Sie die verwendeten Bauteile (LEDs, Fotozelle und Widerstände so weit kürzen, dass sie fest im Breadboard sitzen. Schneiden Sie die LED so ab, dass die Anode etwas länger bleibt als die Kathode. Für die (schwarz eingezeichneten) Steckbrücken sollten Sie passende Drahtstücke zuschneiden und an den Enden abisolieren.



Eine kompakte Alternative

Etwas übersichtlicher und kompakter wird die Sache, wenn Sie sich für ein paar Euro ein so genanntes **Prototyping Shield** besorgen. Das wird auf den Arduino aufgesteckt und das mitgelieferte Mini-Breadboard wird oben zwischen den Anschlussleisten eingeklebt. So brauchen wir keine Kabelverbindungen mehr, sondern können ausschließlich mit Steckbrücken und individuell zugeschnittenen Widerständen arbeiten.

Wir legen die Widerstände von der Anode (Pluspol) jeder LED zum Arduino-Digitalpin. Die Kathoden der LEDs verbinden wir reihum miteinander über Steckbrücken und legen das Ende an GND. Bei der gelben LED in der Mitte werden die beiden langen Anschlussbeine zurechtgebogen. Da sich die Anschlussbeine mit dem Widerstand an D8 kreuzen, müssen entweder dieser Widerstand oder die beiden Anschlussbeine der LED mit einem Stück Kabelhülle isoliert werden.



Software mit System

Sie können sich vorstellen, dass sich auf einem Modell mit den zwei Ampeln und zwei Sensoren bereits relativ komplexe Projekte verwirklichen lassen. Deshalb wollen wir von Anfang an möglichst so programmieren, dass unser Code

- verständlich,
- pflegeleicht und
- wiederverwendbar ist.

Der Verständlichkeit dienen in erster Linie Kommentare, die den Zweck des ganzen Programms und seiner Teile beschreiben. In diesem Heft werden diese Kommentare aus Platzgründen auf das Wichtigste beschränkt bleiben, in Ihren Programmen sollten Sie eher großzügig kommentieren.

Leichter verständlich und pflegeleichter wird ein Programm auch dadurch, dass Sie Hardwareadressen nicht als Zahl, sondern als Text ins Programm schreiben. Im Beispiel rechts wird unter anderem statt der Pinadresse 12 die Konstante *lirot* verwendet. Der Vorteil für den Programmierer ist erstens, dass er bei der Lektüre des Programms ohne Kommentar gleich sieht, welche Lampe gemeint ist. Zweitens aber braucht er, falls er sein Modell später einmal umbaut, also etwa die Anschlüsse der linken und rechten roten LED vertauscht, das Programm nur an einer Stelle zu ändern.

Ampel

```
/*
Steuerung einer einzelnen Ampel in Endlosschleife
Autor: Wastl Kraxlhuber
Programm erstellt 15.02.2014
letzte Änderung 21.02.2014
*/
                    // Ampel links
const int lirot = 12;   // rot  : D12
const int ligelb = 10;  // gelb : D10
const int ligruen = 8;  // grün : D8

void setup() {    // Digitalpins auf Ausgang
  pinMode(lirot,OUTPUT);
  pinMode(ligelb,OUTPUT);
  pinMode(ligruen,OUTPUT);
}

void loop() {
  // Linke Ampel rot
  digitalWrite(lirot, HIGH);
  digitalWrite(ligelb, LOW);
  digitalWrite(ligruen, LOW);
  delay(6000);
  // Linke Ampel rot+gelb
  // ...
}
```

Wissenswert

- Der Editor hebt reservierte **Schlüsselwörter** der Programmiersprache orange hervor. Vordefinierte Konstanten werden blau markiert, selbst definierte Programmcodes schwarz, Kommentare grau. Erklärungen für die Schlüsselwörter finden Sie in über den Menüpunkt *Hilfe -> Referenz* als Webseite in Ihrem Browser.
- Die Anweisung **const** definiert einen Namen für eine **Konstante**. Der Compiler setzt überall dort, wo im Quellcode *lirot* steht, im Maschinencode die Zahl 12 ein.
- Der Datentyp **int** bezeichnet eine ganze Zahl zwischen –32768 und +32767
- C++ ist für die **Groß-/Kleinschreibung** sensibel. lirot, liRot und Lirot sind nicht das Gleiche. Schreiben Sie die vordefinierten und die selbst definierten Funktions– und Zahlennamen immer in derselben Weise.
- Mithilfe des Menübefehls *Tools -> Automatisch formatieren* können Sie verrutschte **Einrückungen korrigieren** lassen.
- Die Verwendung der **Umlaute** ä, ö, ü und des ß ist in Kommentaren erlaubt, ansonsten aber verboten. Auch Satzzeichen und ähnliche **Zeichen** sind in Konstanten– und Funktionsnamen unzulässig. Erlaubt ist dagegen der Unterstrich.

Aufgaben

1. Tippen Sie das abgebildete Programm ab und vervollständigen Sie es so, dass die linke Ampel endlos läuft: sechs Sekunden rot, eine Sekunde rot+gelb, drei Sekunden grün, zwei Sekunden gelb, ...).
   Speichern Sie Ihr Ergebnis unter 🖫 Ampel.ino

Die Arduino-Software speichert Ihr Programm „Ampel“ als „Ampel.ino“ im Ordner „Ampel“ Ihres Sketchbooks. Den Ort des Sketchbooks können Sie unter Datei -> Einstellungen verändern.

Vorsicht Baustelle!

Um den Verkehr an einer Baustelle zu regeln, benötigt man in der Regel mindestens zwei miteinander synchronisierte Ampeln. Und die laufen etwa nach dem rechts abgedruckten Schema ab:

- Während eine Ampel die bekannten Phasen durchläuft, zeigt die andere Ampel rot.
- Es gibt eine Phase, in der die beide Ampeln rot zeigen. Die Dauer dieser Phase hängt von der Zeit ab, die ein langsames Fahrzeug benötigt, um die Baustelle zu passieren. Nur für unser Modell verkürzen wir diese Zeit auf drei Sekunden.
- Die Phase, in der eine Ampel rot+gelb zeigt, ist praktisch immer eine Sekunde lang. Die Dauer der Gelbphase einer Ampel muss auf einer schnellen Straße länger ausfallen als innerorts. Für unsere Zwecke wählen wir für die Gelbphase 2 Sekunden.
- Die Dauer der Grünphase richtet sich nach der Verkehrsdichte und nach der Länge der Baustelle. An einer langen Baustelle wird die Grünphase häufig länger als 20 Sekunden sein müssen sein, damit das Verhältnis zwischen Rot- und Grünphasen nicht zu ungünstig ausfällt. Für unser Modell wählen wir 5 Sekunden.

Phase	Dauer	Ampel links	Ampel rechts
1	1 s	rot+gelb	rot
2	> 5 s	grün	rot
3	2 s	gelb	rot
4	> 3s	rot	rot
5	1 s	rot	rot+gelb
6	> 5 s	rot	grün
7	2 s	rot	gelb
8	> 3 s	rot	rot

Eigene Funktionen definieren

Nicht nur Zahlen lassen sich mit Namen versehen. Auch Programmteile kann man benennen und auslagern. Im Beispiel rechts wurden die Befehle, die für die erste Ampelphase notwendig sind, unter dem Namen *void rotgelb_rot()* in einer eigenen Funktion deklariert und von der *loop()* aus mit diesem Namen aufgerufen.

Zur Erinnerung: *void* bedeutet leer. Diese Funktion gibt also keine Zahl aus. Die runde Klammer ist das Kennzeichen von Funktionsnamen. Im nächsten Schritt werden wir in dieser Klammer der Funktion Zahlen übergeben, die sie für ihre Ausführung benötigt.

Baustellenampel

```
      // Ampel links West-Ost-Richtung
const int lirot = 12;   // rot  : D12
const int ligelb = 10;  // gelb : D10
const int ligruen = 8;  // grün : D8
      // Ampel rechts Ost-West-Richtung
const int rerot = 13;   // rot  : D13
const int regelb = 11;  // gelb : D11
const int regruen = 9;  // grün : D9

void setup() {  // Alle Pins auf Ausgang
  pinMode(lirot,OUTPUT);
  pinMode(ligelb,OUTPUT);
  pinMode(ligruen,OUTPUT);
  pinMode(rerot,OUTPUT);
  pinMode(regelb,OUTPUT);
  pinMode(regruen,OUTPUT);
}

void rotgelb_rot() {
  digitalWrite(lirot,HIGH);
  digitalWrite(ligelb,HIGH);
  digitalWrite(ligruen,LOW);
  digitalWrite(rerot,HIGH);
  digitalWrite(regelb,LOW);
  digitalWrite(regruen,LOW);
  delay(1000);
}
// Weitere Methodendefinitionen ...

void loop() {
  rotgelb_rot();
  // Weitere Methodenaufrufe ...
}
```

Aufgaben

1. Ergänzen Sie die Funktionsdefinitionen *gruen_rot(), gelb_rot(), rot_rot (), rot_rotgelb(), rot_gruen() und rot_gelb()* und rufen Sie sie in der *loop*() auf.

Selbstverständlich können Sie beim Entwerfen des Programms die Zwischenablage benutzen: Markieren, mit <Strg>+C kopieren, mit <Strg>+V einfügen. Bevor Sie aber ein und denselben Code an verschiedenen Stellen einfügen, sollten Sie lieber für diesen Programmteil eine Funktion definieren und mehrfach aufrufen.

Haben Sie bemerkt, dass die Funktion *rot_rot()* nur einmal deklariert werden muss, aber mehrfach aufgerufen wird? Sollte sich in die Funktion *rot_rot()* ein Fehler eingeschlichen haben, brauchen Sie ihn nur an einer Stelle zu korrigieren. Hätten Sie den Programmabschnitt kopiert, müssten Sie zwei Stellen ändern.

Funktionen mit Argumenten

Manchmal unterscheiden sich zwei Funktionsaufrufe im geforderten Zahlenwert. Nehmen wir an, es geht in der einen Richtung bergauf und in der anderen bergab. Dann könnte es erforderlich werden, die Rot-Rot-Phase in der einen Richtung länger zu takten als in der anderen. Wir könnten trotzdem die gleiche Funktion verwenden, wenn wir mit **Argumenten** arbeiten.

Dazu müssen wir der Funktion in der Klammer einen Wert übergeben. Wir definieren dazu eine Variable, d.h. wir lassen den Compiler den Speicherplatz für eine Zahl reservieren, ohne diese bereits zu kennen. Bei der Variable soll es sich um eine ganze Zahl handeln, und wir wollen sie „dauer“ nennen.

In der *loop()* wird ab sofort nun der Funktion *rot_rot()* in der Klammer ein Zahlenwert übergeben. Die Funktion *rot_rot()* merkt sich diesen Zahlenwert und setzt ihn in die *delay()*-Anweisung ein.

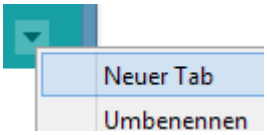
Auch die delay()-Anweisung ist eine Funktion mit einem Argument. Andere Funktionen, z.B. *pinMode()* benötigen mehrere Argumente.

```
void rot_rot(int dauer) {
  digitalWrite(lirot,HIGH);
  digitalWrite(ligelb,LOW);
  digitalWrite(ligruen,LOW);
  digitalWrite(rerot,HIGH);
  digitalWrite(regelb,LOW);
  digitalWrite(regruen,LOW);
  delay(dauer);
}

void loop() {
  //...
  rot_rot(3000); // bergab
  //...
  rot_rot(4000); // bergauf
  //...
}
```

Bibliotheken

Oben haben wir uns vorgenommen, den Code so zu schreiben, dass er möglichst übersichtlich und wiederverwendbar ist. Wenn wir unsere Ampelphasen in späteren Programmen recyclen wollen, dann sollten wir die dazugehörigen Funktionsdefinitionen in eine eigene Datei auslagern:

- Erzeugen Sie mit Klick auf das kleine Dreieck in der Symbolleiste einen neuen Tab und geben Sie ihm den Namen „Ampelphasen.h“. Denken Sie bei der Benennung daran, Umlauten und ß zu vermeiden.
- Verschieben Sie mittels Ausschneiden und Einfügen Ihre selbst definierten Funktionen aus dem Hauptprogramm in diese Datei.



Baustellenampel | Ampelphasen.h

```
#include <Arduino.h>

void rotgelb_rot(int dauer) {
  digitalWrite(lirot,HIGH);
```

Baustellenampel | Ampelphasen.h

```
#include "Ampelphasen.h"

const int lirot = 12;
const int ligelb = 10;
const int ligruen = 8;
const int rerot = 13;
const int regelb = 11;
const int regruen = 9;

void setup() {
  pinMode(lirot,OUTPUT);
  pinMode(ligelb,OUTPUT);
  pinMode(ligruen,OUTPUT);
  pinMode(rerot,OUTPUT);
  pinMode(regelb,OUTPUT);
  pinMode(regruen,OUTPUT);
}

void loop() {
  rotgelb_rot(1000);
  gruen_rot(5000);
  gelb_rot(2000);
  rot_rot(3000);
  rot_rotgelb(1000);
  rot_gruen(5000);
  rot_gelb(2000);
  rot_rot(3000);
}
```

Damit die Arduino-Software Ihre Bibliothek einbinden kann, müsse Sie in Ihr Programm die Compileranweisung **#include “Ampelphasen.h“** einfügen. In die eingebundene Datei Ampelphasen.h sollten Sie wiederum mit **#include <Arduino.h>** die Arduino Standardbibliothek einbinden.

Wenn Sie anschließend ein neues Projekt mit der Ampel beginnen wollen, dann speichern Sie es gleich ab und kopieren die Datei *Ampelphasen.h* über die Zwischenablage in den Projektordner Sobald Sie das Projekt wieder öffnen, wird *Ampelphasen.h* im zweiten Tab erscheinen und verfügbar sein.

Später werden Sie eine andere Art von Bibliotheken entdecken: An verschiedenen Stellen Ihres Arduino-Ordners finden Sie ein Unterverzeichnis namens „libraries“. Darin befinden sich Dateien mit den Namenserweiterungen .h (Header) und .cpp (C Preprocessor), die z. B. beim Kauf von Hardware mitgeliefert wurden. Solche Bibliotheken werden einmalig ins library-Verzeichnis importiert und mit der Anweisung #include <Dateiname.h> in eigene Programme eingebunden. Danach kann man die darin definierten Funktionen wie normale Arduino-Funktionen verwenden.

Aufgaben

1. Stellen Sie Ihr Programm 🖫 Baustellenampel.ino auf die Verwendung von Parametern um, lagern Sie die Prozeduren in die Bibliothek Ampelphasen.h aus und speichern Sie dann alles erneut ab.

Auf Knopfdruck wird's grün

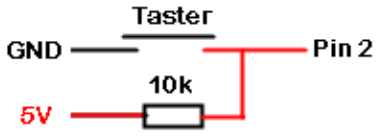
Neben Ampeln, die ihr Programm immer stur im gleichen Rhythmus gleich abspulen, gibt es auch solche, die auf Sensoren reagieren. Nehmen wir die Ampel an einem Fußgängerübergang. Normalerweise zeigt die Ampel für die Autos dauernd grün und für die Fußgänger rot. Wenn aber ein Fußgänger auf den Knopf drückt, geht die Auto-Ampel erst auf Gelb, dann auf Rot und dann die Fußgängerampel auf Grün. Nach einiger Zeit geht die Fußgängerampel wieder auf Rot und die Autos bekommen erst Rot+Gelb, dann Grün.

Für diese Anlage wollen wir jetzt ein Programm erstellen. Wir verwenden dazu den Taster, der an den Digitalpin 2 des Arduino angeschlossen ist. Dieser Digitalpin muss dazu als Eingang geschaltet werden. Machen wir einen Versuch: Das abgebildete Programm überträgt den Zustand, der an Pin 2 gelesen wird, an Pin 13. Wenn Sie also das Kabel vom Taster abziehen und Pin 2 an 5V anschließen, sollte die LED an Pin 13 leuchten. Wenn Sie Pin 2 an GND anschließen, sollte die LED an Pin 13 dunkel bleiben. Das funktioniert auch.

EinAus

```
void setup() {
  pinMode(2,INPUT);
  pinMode(13,OUTPUT);
}

void loop() {
  digitalWrite(13,digitalRead(2));
}
```

Schließen wir das Kabel wieder am Taster an, so wird die Sache komplizierter: Der Arduino kann, solange die Taste nicht gedrückt wird, gar nicht erkennen, ob hinter dem Taster ein HIGH- oder ein LOW-Signal lauert. Der Normalzustand eines unbeschalteten Eingangs im Arduino ist weder HIGH noch LOW, sondern unbestimmt. Dies könnten wir mit einem „Pullup"-Widerstand ändern, der parallel zum Schalter an 5V angeschlossen ist und ein schwaches HIGH-Signal liefert, das bei gedrücktem Taster auf GND heruntergezogen wird. Der Arduino nimmt uns aber diese Arbeit ab. Die Pullup-Schaltung ist im Arduino intern vorhanden, wir müssen sie nur aktivieren, indem wir im *setup()* den *pinMode(2,INPUT_PULLUP)* setzen.



Die zweite, ausführlichere Fassung unseres Testprogramms besagt:
WENN Pin 2 LOW liefert (der Taster also gedrückt ist), soll die LED leuchten.
SONST aber (wenn der Taster nicht gedrückt ist und Pin 2 deshalb HIGH liefert), soll die LED dunkel bleiben.

EinAus2

```
void setup() {
  pinMode(2,INPUT_PULLUP);
  pinMode(13,OUTPUT);
}

void loop() {
  if (digitalRead(2) == LOW) {
    digitalWrite(13,HIGH);
  }
  else {
    digitalWrite(13,LOW);
  }
}
```

Wissenswert:

- Auf einen **if-Befehl** folgt zunächst in **runden Klammern** eine Bedingung, die geprüft werden soll. Das Gleichheitszeichen in Bedingungen ist doppelt zu schreiben (das einfache Gleichheitszeichen hat eine andere Funktion).
- Auf die runde Klammer folgt eine **geschweifte Klammer**, in der die Befehle stehen, die ausgeführt werden sollen, wenn die Bedingung zutrifft.
- Anschließend kann nach **else** ein weiterer Anweisungsblock folgen, der ausgeführt werden soll, wenn die Bedingung nicht erfüllt ist.

Aufgaben

1. Testen Sie das Programm 🖫 EinAus2 an Ihrer Ampel.
2. Denken Sie sich andere Effekte aus, die auf Tastendruck ablaufen.
3. Realisieren Sie das im ersten Absatz beschriebene Programm 🖫 Fussgaengerampel. Greifen Sie dabei auf Ihre Funktionsbibliothek 🖫 Ampelphasen.h zurück. Die linke Ampel Ihres Modells ist für die Autos, die rechte für die Fußgänger. Von der Fußgängerampel verwenden wir nur die rote und die grüne LED.

Exkurs: Interrupts

Vielleicht haben Sie schon einmal das Stichwort Multitasking gehört: Ein PC kann sich in der Regel nicht auf eine Sache konzentrieren. Während er das durch den Benutzer gestartete Programm ausführt, muss er gleichzeitig die Tastatur und die Maus beobachten und nebenbei aktualisiert die Systemuhr. Vielleicht druckt er auch noch, während der Benutzer weiterarbeitet, eine Datei aus oder er fragt das Mailkonto ab. In Wirklichkeit macht er das nicht gleichzeitig, sondern er wirft in kurzen, regelmäßigen Abständen einen Seitenblick auf mögliche andere Anforderungen. Auch auf dem Arduino lässt sich das realisieren.

Ein Problem unseres Fußgängerampel-Programms ist, dass der Fußgänger praktisch immer Grün anfordern. kann Sobald die Autos Grün bekommen, ist seitens der Fußgänger eine neue Grünanforderung möglich. Da aber niemand weiß, wie viele Autos die Straße benutzen, werden diese dabei möglicherweise benachteiligt.

Eine Möglichkeit wäre nun, den Delay in gruen_rot() auf sagen wir 5 Sekunden zu verlängern. Allerdings wird dann auch nur alle fünf Sekunden einmal die Schleife abgefragt. Um sicherzustellen, dass der Taster wahrgenommen wird, muss man ihn also mindestens fünf Sekunden gedrückt halten.

Dem weichen wir aus: Die Digitaleingänge D2 lässt sich nämlich auch als **Interrupt 0** definieren. Gesetzte Interrupts werden vom Arduino im Hintergrund eines laufenden Programms beobachtet. Das Signal von einer Interruptleitung löst jederzeit während eines laufenden Programms eine Unterbrechung und einen Sprung zu einer im Interruptbefehl definierten Prozedur aus. Bei uns ist das die Prozedur "signalKommt". Diese setzt eine Variable, die der Hauptschleife mitteilt, ob seit dem letzten Durchlauf eine Taste gedrückt worden ist.

Die Reaktion auf die Anforderung muss dann nicht sofort erfolgen, sondern sie lässt sich im Programm an passender Stelle einbauen.

Interrupt | Ampelphasen.h

```
// Steuerung der Fußgängerampel über Interrupts

 int lirot = 12;   // rot : D12   Ampel für Autos
 int ligelb = 10;  // gelb : D10
 int ligruen = 8;  // grün : D8
 int rerot = 13;   // rot : D13  Fußgängerampel
 int regelb = 11;
 int regruen = 9;  // grün : D9
 int taster = 2;   // Fußgängertaster auf interruptfähigem Pin
 int warten = 7;   // Gelbe LED in der Mitte als Signal "Bitte warten"
 int dauer;
 int anforderung=0;

  #include "Ampelphasen.h" // hier eingefügt, da Konstanten deklariert sein müssen

 void setup() {              // Alle Pins außer Taster als Ausgang programmieren
   pinMode(lirot,OUTPUT);
   pinMode(ligelb,OUTPUT);
   pinMode(ligruen,OUTPUT);
   pinMode(rerot,OUTPUT);
   pinMode(regruen,OUTPUT);
   pinMode(warten,OUTPUT);

   pinMode(taster,INPUT_PULLUP);       // Taster an D2 steuert Interrupt 0
   attachInterrupt(0,signalKommt,LOW); // LOW startet die Funktion "signalKommt()"
 }

 void signalKommt() {            // wird aufgerufen, sobald Taster gedrückt wird
   anforderung=1;                // setzt Variable anforderung
   digitalWrite(warten,HIGH);    // schaltet gelbe LED ein "Bitte warten!"
}

 void loop() {
   gruen_rot(10000);             // minimale Grünphase für Hauptrichtung
   if (anforderung == 1) {       // ist 1, falls in dieser Zeit Taster gedrückt wurde
     anforderung=0;
     digitalWrite(warten,LOW);  // setzt "Grün kommt"-LED
     gelb_rot(2000);
     rot_rot(3000);
     rot_rotgelb(1000);
     rot_gruen(5000);
     if (anforderung == 1) {     // Wenn während Grünphase Taster gedrückt wird,
       anforderung=0;            // wird Grünphase um 2 Sekunden verlängert
       digitalWrite(warten,LOW);
       delay(2000);
     }
     rot_rot(3000);
     rotgelb_rot(1000);
   }
 }
```

Aufgaben

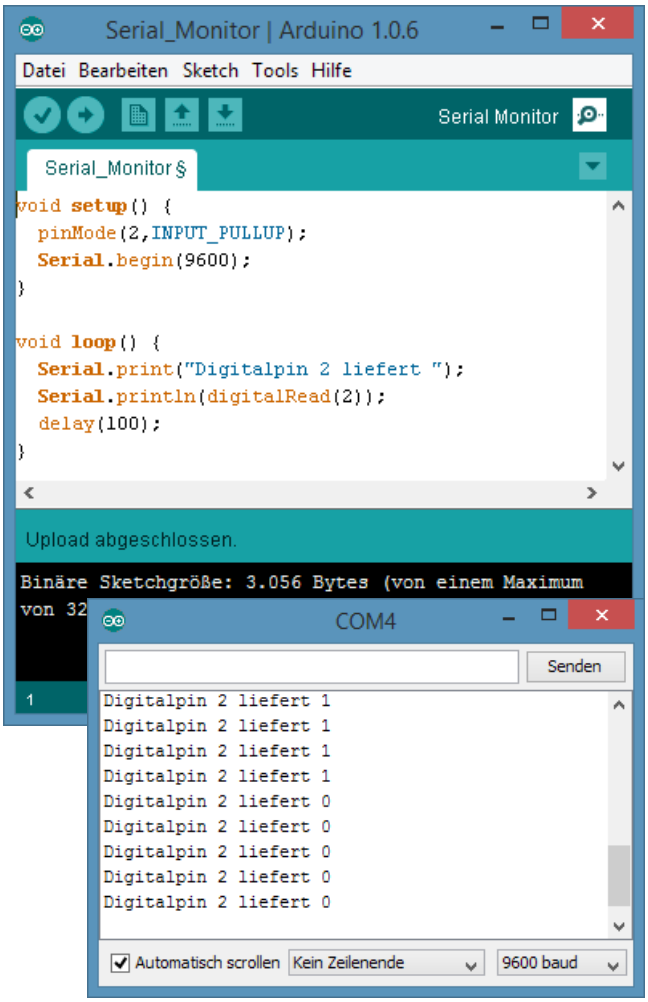
1. Laden Sie das Programm 🖫 Interrupt.ino und probieren Sie es aus. Später werden Sie vielleicht eigene Interruptroutinen schreiben wollen.

Arduino ruft PC

Normalerweise ist der Arduino ja dafür konstruiert, programmiert zu werden und anschließend seine Arbeit unabhängig vom PC zu verrichten Das heißt aber nicht, dass er nicht von sich aus auch Infos zum PC zurückliefern kann. Das einfachste Mittel hierfür ist der Serielle Monitor. Er öffnet sich bei einem Klick auf das Symbol ganz rechts in der Symbolleiste und dient dazu, während des Programmlaufs Daten zum PC zu senden oder auch von dort abzufragen. Da er dazu die Digitalpins 0 und 1 benutzt, können diese, während der Serielle Monitor aktiv ist, nicht anderweitig verwendet werden.

Um den Zustand unseres Ampeltasters auf dem Seriellen Monitor anzuzeigen, ist zunächst erforderlich, dass der Arduino und der PC die Daten in der gleichen Geschwindigkeit schreiben bzw. lesen. Dazu dient im *setup()* des Programms der Befehl *Serial.begin()* und im Fenster des Seriellen Monitors die Auswahlbox rechts unten.

Um Daten zu senden, verwenden wir die Funktion *Serial.print()* ohne oder *Serial.println()* mit anschließender Zeilenschaltung. An unserer Ampel wird das abgebildete Programm normalerweise 1 und bei gedrücktem Taster den Wert 0 zurückgeben. Sie müssen nach jedem Programmstart erneut manuell den Seriellen Monitor öffnen.



Aufgaben

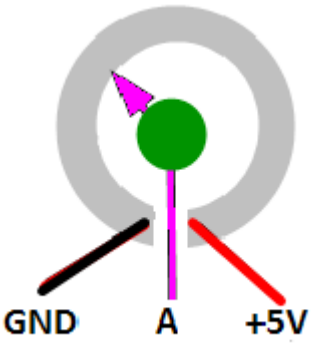
1. Auf dem Seriellen Monitor kann man auch Text ausgeben. Programmieren Sie eine „Ampel“, bei der der Serielle Monitor die Wörter "rot", "rot+gelb", "grün" und "gelb" in einer Endlosschleife anzeigt. 🖫 Textampel.ino
2. Probieren Sie das unter *Datei -> Beispiele -> Communication -> ASCII-Table* befindliche Programm.
3. Mit dem Seriellen Monitor können Sie bei Bedarf auch Daten an den Arduino senden. Ein Beispiel hierzu finden Sie unter *Hilfe -> Referenz* ganz rechts unten unter *Communication -> Serial* auf der Beispielseite zu *Serial.read()*. Rechts ist eine Übersetzung abgebildet. Probieren Sie das Programm aus.
4. Erstellen Sie ausgehend vom abgebildeten Programm eine manuell geschaltete Ampel:
   Wenn Sie 1 eintippen, soll die Ampel auf Rot schalten.,
   bei 2 zeigt sie Rot+Gelb, bei 3 Grün und bei 4 Gelb.
   🖫 Manuelle_Ampel.ino

Eingabe_vom_S-Monitor

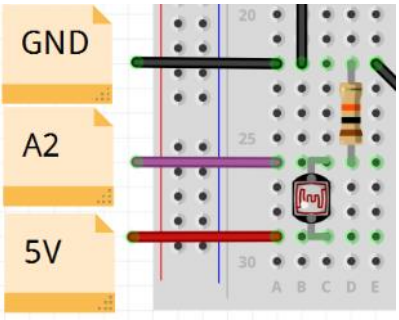
```
// Daten vom Seriellen Monitor lesen

int incomingByte;  // Variable deklarieren,
                   // um sich Eingaben zu merken

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

void loop() {
  // Nur wenn Eingabe gesendet wurde,
  // wird Auswertung durchgeführt:
  if (Serial.available() > 0) {
    // Code des eingegebenenen Zeichens
    // als Zahl zwischenspeichern:
    incomingByte = Serial.read();
    // ASCII-Zeichencode als Dezimalzahl ausgeben:
    Serial.print("Das Zeichen hat den Code: ");
    Serial.println(incomingByte, DEC);
  }
}
```

Analogwerte einlesen

Vielleicht besitzen Sie ja noch so ein altmodisches Radio, an dem man die Lautstärke über einen Drehknopf regelt. Hinter einem solchen Knopf sitzt meist ein Potentiometer. Das ist ein Ausschnitt aus einem Kreisring, hergestellt aus einem Material, das einen elektrischen Widerstand aufweist. Mit dem grünen Knopf wird ein Kontakt über diesen Widerstand geführt und die Spannung geteilt. Drehe ich den Knopf nach links, so erhalte ich auf der Ausgangsleitung A eine geringe Spannung, weil der größere Widerstand zur 5V-Leitung geht. Drehe ich aber den Knopf ganz nach rechts, so liefert die Ausgangsleitung 5V.

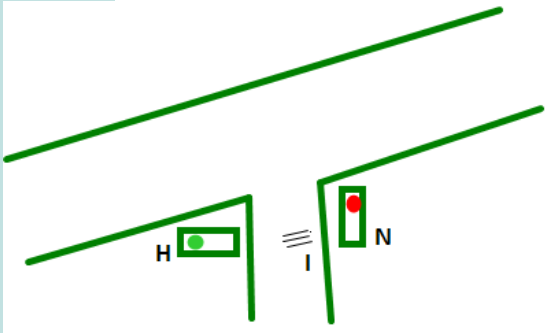


Auch auf unserer Ampel existiert ein solcher Spannungsteiler. Zwischen GND und A2 sind zwei Widerstände hintereinander geschaltet. Der eine ist ein fester Widerstand von 10 Kiloohm, der andere besteht aus einem lichtempfindlichen Widerstand und liefert bei Dunkelheit einen hohen, bei hoher Helligkeit dagegen einen geringen Widerstand. Über den Analogpin A2 lässt sich Wert des Widerstands messen und damit die Helligkeit über der Fotozelle bestimmen. Wenn Sie sich über *analogRead(0)* den Eingangswert anzeigen lassen, werden Sie je nach Helligkeit einen Wert zwischen 0 und 1023 erhalten,.



Aufgaben

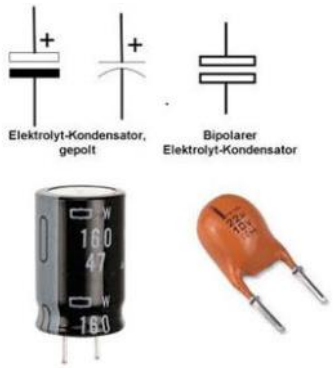
1. Ändern Sie das Beispielprogramm *Datei -> Beispiele -> 01.Basics -> AnalogReadSerial* so ab, dass es den Wert von Analogeingang 2 auf dem Seriellen Monitor ausgibt, und experimentieren Sie mit der Helligkeit. Analogpins sind immer Eingänge. Sie benötigen keinen *pinMode()*-Befehl im Setup.
2. Ihr Ampel-Board muss ja nicht zwangsläufig eine Ampel darstellen. Machen Sie daraus den Belichtungsmesser einer Kamera: Kann die Kamera das Bild richtig belichten, so leuchtet die grüne LED. Bei greller Sonne kommen die Farben nicht so gut, die Kamera schaltet wegen Überbelichtung die gelbe LED ein. Wenn es zu dunkel zum Fotografieren ist, zeigt die Kamera rot. 🖫·Belichtungsmesser.ino.
3. Eine kleine Straße N mündet in eine größere Straße H ein. Normalerweise hat die größere Straße grün. Wenn aber auf der Induktionsschleife I ein Auto steht, so soll die Nebenrichtung kurzzeitig auf Grün geschaltet werden.
   Die Induktionsschleife I wird in unserem Ampelmodell repräsentiert durch die Fotozelle am Analogeingang 2. Das Auto ist Ihr Finger, mit dem Sie die Fotozelle abschatten.
   🖫 Bedarfsampel.ino



4. Manche Ampeln werden nachts abgeschaltet. Dies geschieht in der Regel um eine festgelegte Uhrzeit. Bei uns soll es bei Einbruch der Dunkelheit automatisch passieren. Programmieren Sie Ihre Ampel so, dass sie nach Einbruch der Dunkelheit vom normalen Ablauf auf Blinken umschaltet. Wählen Sie einen passenden Schwellenwert. 🖫Nachtabschaltung.ino".
5. Schaffen Sie es, das Bedarfsampel-Programm von Aufgabe 3 so zu ändern, dass während der Grünphase der Hauptrichtung die gelbe LED in der Mitte blinkt, um Abbieger vor Fußgängern zu warnen? 🖫Bedarfsampel_m_F.ino
6. Expertenaufgabe: Steuern Sie die Ampel über einen Interrupt.
   🖫Bedarfsampel_2.ino (Erklärung siehe dort)

Exkurs: Ein einfacher A-D-Wandler

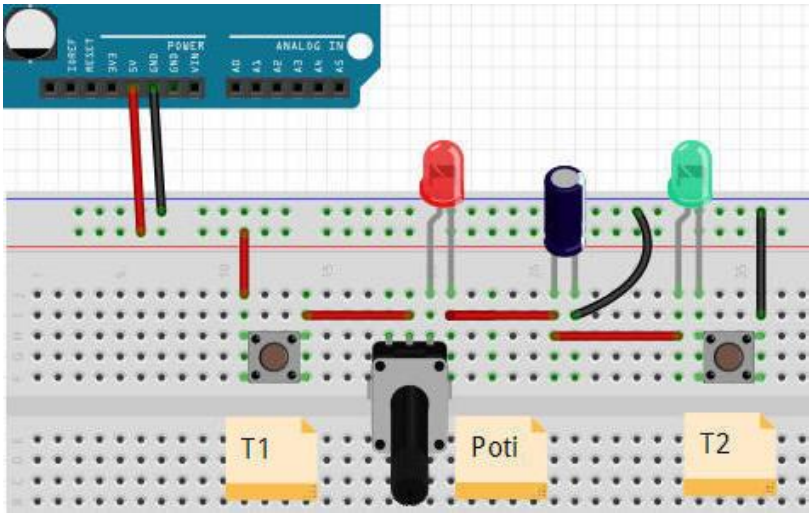
Wie funktioniert eigentlich ein Analogeingang? Genau wie an den Digitaleingängen ist an den Analogeingängen jeweils nur eine Leitung angeschlossen. Und trotzdem interpretiert der Prozessor die Stärke des Signals auf dieser Leitung nicht nur als 0 oder als 1, sondern übersetzt sie in fast stufenlos aufgelöste Werte. Wie macht er das? Im Arduino-Schaltbild findet sich, beschriftet mit ADC (Analog-Digital-Converter) ein Bauteil, das in vielen technischen Schaltungen enthalten ist: ein Analog-Digital-Umsetzer. Wie ein solcher ADC aussehen könnte, könnten wir, falls Sie die entsprechenden Teile übrig haben, in einem kleinen Experiment erforschen:



Kern des abgebildeten ADC ist ein Elektrolyt-Kondensator mit 22 Mikrofarad. Wie andere Bauteile auch, muss der Elko richtig herum eingesetzt werden; auf der Minusseite befindet sich dazu eine farbige Markierung. Außer solchen gepolten Elkos gibt es auch bipolare, die in beiden Richtungen funktionieren.

Im Unterschied zu einer Batterie kann ein Kondensator nur wenig Energie speichern, er kann diese dafür aber viel schneller abgeben. Das macht man sich zum Beispiel im Elektronenblitz von Kameras zunutze, wo ein Kondensator die Energie, die er für den Lichtblitz braucht, in vier bis fünf Sekunden von der Batterie lädt und in einer zehntausendstel Sekunde an die Blitzröhre abgibt.

In der oben abgebildeten Schaltung können wir einen solchen Elektronenblitz simulieren. Die beiden Taster T1 und T2 unterbrechen den Stromkreis. Wird der Schalter T1 gedrückt, so wird der Kondensator geladen. Gleichzeitig leuchtet die rote LED. Ist der Kondensator geladen, so erlischt die rote LED. Sobald der Kondensator geladen ist, sperrt er den Stromfluss.



Nun lassen wir den Taster T1 los. Die Verbindung zum Pluspol des Arduino ist jetzt unterbrochen, aber der Kondensator ist geladen.

Sobald wir den Taster T2 drücken, wird die Ladung an die grüne LED abgegeben. Diese leuchtet kurz auf.

Mit dem Poti können Sie den Stromfluss vom Arduino zum Kondensator regeln und die Ladezeit verlängern. Die rote LED leuchtet dann während des Ladevorgangs entsprechend schwächer, aber länger. Der Entladevorgang ist davon unabhängig. Die grüne LED leuchtet immer kurz und intensiv.

Was das alles mit einem Analog-Digital-Umsetzer zu tun hat? Ganz einfach: Wenn Sie den Computer die Zeit messen lassen, die ein Kondensator zum Aufladen braucht, dann haben Sie ein Maß für die Spannung auf der Eingangsleitung (bei uns die Einstellung des Potentiometers). Je länger der Kondensator für eine Ladephase braucht, desto weniger Spannung ist auf der Leitung.

Aufgaben

1. Bauen Sie - sofern Sie dafür Ihre Ampel nicht zerstören müssen - die Schaltung nach und beobachten Sie die Wirkung verschiedener Einstellpositionen des Potentiometers.

D-A-Wandler: Quasi analoge Ausgabe

Im vorigen Kapitel haben wir einen analogen Wert an einem Eingang in eine Zahl zwischen 0 und 1023 umgesetzt. Also müsste doch der umgekehrte Weg auch zu bewerkstelligen sein. Während sich aber die Digitalports des Arduino mithilfe des Befehls *pinMode ()* als INPUT oder als OUTPUT konfigurieren lassen, sind die Analoganschlüsse nur als Ausgang verwendbar. Trotzdem aber enthält die Befehlsreferenz des Arduino auch die Funktion *analogWrite(),* es müsste also möglich sein, eine LED nicht nur aus– und einzuschalten, sondern sie stufenlos zu dimmen. Wie geht das?

Die Lösung heißt PWM - Pulse Width Modulation oder auf Deutsch Pulsbreitenmodulation, und die Analogausgabe erfolgt über bestimmte Digitalausgänge, die auf der Arduino-Platine mit einer kleinen Tilde gekennzeichnet sind. Einer der Ausgänge, an denen dies möglich ist, ist D11, und daran hängt bei uns die gelbe LED der rechten Ampel.

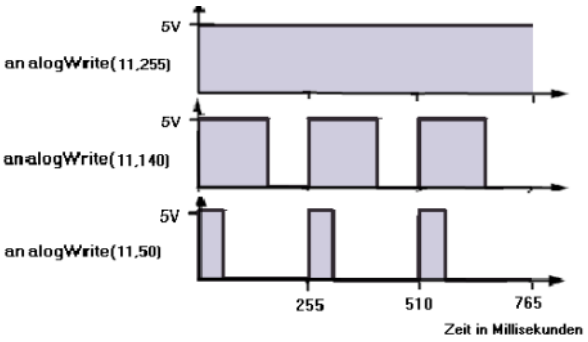
Schreiben Sie das folgende Programm:

```
analogWrite

// adalogWrite()-Demo

const int pin = 11; '// Möglich: 3, 5, 6, 9, 11, 12
const int wert = 50; // Möglich: 0 bis 255

void setup() {
  pinMode(pin,OUTPUT);
}

void loop () {
  analogWrite(pin, wert);
}
```

Setzen Sie *wert* auf verschieden Werte zwischen 0 und 255 und *pin* auf 11, 10 oder 9.

Laden Sie das Programm jedes Mal neu hoch und beobachten Sie die Wirkung. Je nach der Eingabe in *wert* leuchtet die betreffende LED heller oder weniger hell. Probieren Sie auch aus, was passiert, wenn Sie *pin* auf 13, 12 oder 8 setzen.

Es sieht so aus, als ob man die Spannung auf einigen digitalen Leitungen praktisch stufenlos regeln könnte. Ein digitaler Wert (gegeben durch eine Zahl zwischen 0 und 255) wird auf einer einzelnen Leitung in eine analoge Größe (Helligkeit) umgesetzt.



In Wirklichkeit bleibt die Ausgabe digital. Man nutzt die Geschwindigkeit des Computers und die Genauigkeit der Zeitmessung. Nehmen wir an, die Balken in der folgenden Abbildung wären jeweils eine Zehntelsekunde lang. Wenn wir die LED in der vollen Zehntelsekunde einschalten, wird sie maximal hell leuchten. Schalten wir sie aber in schneller Folge mehrmals aus und ein, so nimmt das menschliche Auge, das mehr als zwanzig Eindrücke pro Sekunde nicht auseinanderhalten kann, die Folge der Einzelbilder wie einen Film wahr. Während die LED in Wirklichkeit mehrmals kurz an und ausgeht, glaubt unser Auge, ihr Licht leuchte weniger hell.

Aufgaben

1. Schreiben Sie ein Programm, das eine beliebige LED per *delay()*-Anweisung ein- und nach kurzer Zeit wieder ausschaltet. Spielen Sie mit den Werten in den beiden Delays.
   a) Das „Tastverhältnis“ soll dabei immer 50% sein, d.h. die LED soll genauso lange ein- wie ausgeschaltet sein. Wie kurz muss die Verzögerung ausfallen, damit Sie das Flackern nicht mehr wahrnehmen?
   b) Ändern Sie das Verhältnis der beiden Delays und erzeugen Sie ohne Verwendung von *analogWrite()* eine matt glimmende LED.

Variablen

Laden Sie das Programm unter *Datei -> Beispiele -> 01.Basics -> Fade* und führen Sie es aus. In diesem Beispiel wird demonstriert, wie man eine LED ein- und ausblenden kann. Rechts steht eine Übersetzung.

Bereits in unserem allerersten Arduino-Programm haben wir einen Zahlenwert als **Konstante** definiert. Das heißt, wir haben der Zahl einen Namen gegeben und diesen Namen später anstelle der Zahl benutzt. Im Anfangsteil dieses Programms werden nun drei **Variablen** deklariert und mit Anfangswerten versehen. Im Gegensatz zum Wert einer Konstanten kann der Wert einer Variablen im Laufe des Programms verändert werden. Verändert werden hier allerding nur *fadeAmount* und *brightness*. Die Variable *led* wird eigentlich als Konstante benutzt.

Variablennamen können in Formeln wie Zahlen verwendet werden, Betrachten wir die einzelnen Befehle, in denen Variablen vorkommen:

- Mit *int brightness* wird das Wort *brightness* als Name einer Variablen deklariert, d.h. es wird Speicherplatz für diese Zahl zugewiesen ...
- ... und mit *brightness = 0;* wird der Wert der Variablen auf 0 gesetzt.
- Die Zeile *int brightness = 0;* kombiniert hier beide Anweisungen.
- Das Zeichen = wird in C++ als Wertzuweisung benutzt (deshalb muss für Gleichungen in if-Bedingungen das Zeichen == benutzt werden).
- In der Zeile *analogWrite(led, brightness)* werden die aktuellen Werte der beiden Variablen *led* und *brightness* ausgelesen und als Argumente für die Funktion *analogWrite*() verwendet.
- In der Zeile *brightness = brightness + fadeAmount;* werden rechts vom Gleichheitszeichen die Werte von *brightness* und von *fadeAmount* ausgelesen. Dann werden beide Werte addiert und das Ergebnis wird der Variable *brightness* als neuer Wert zugewiesen.
- In der Zeile *fadeAmount = - fadeAmount* wird der Wert von *fadeAmount* gelesen, das Vorzeichen wird umgekehrt und das Ergebnis wird der Variable *fadeAmount* als neuer Wert zugewiesen.
- Die Bedingung *if (brightness ==0 || brightness == 255)* bedeutet „Wenn die Helligkeit den Wert 0 oder die Helligkeit den Wert 250 hat, dann ...“. Man beachte das doppelte Gleichheitszeichen i

Bei jedem Durchlauf der Loop wird also *brightness* um 5 erhöht, bis 255 erreicht ist, dann wird die Variable um 5 vermindert, bis 0 erreicht ist, um dann erneut zu wachsen.

Fade

```
// Dieses Programm demonstriert analogWrite():

int led = 9;           // der mit der LED verbundene Pin
int brightness = 0;    // die Helligkeit der LED
int fadeAmount = 5;    // Veränderung der Helligkeit

// Die Setup-Routine läuft einmalig nach jedem Reset:
void setup()  {
  // Pin 9 als Ausgang deklarieren:
  pinMode(led, OUTPUT);
}

// Die Loop-Routine läuft endlos:
void loop()  {
  // Helligkeit der LED an Pin 9 setzen:
  analogWrite(led, brightness);

  // Helligkeit für den nächsten Durchlauf ändern:
  brightness = brightness + fadeAmount;

  // Richtung der Veränderung umkehren:
  if (brightness == 0 || brightness == 255) {
    fadeAmount = -fadeAmount ;
  }
  // Um Ablauf zu bremsen 30 Millisekunden warten:
  delay(30);
}
```

Wertzuweisungen und Rechenoperationen:

a = 17;	a bekommt den Wert 17
b = -23;	b bekommt den Wert -23
c = a + b;	c bekommt den Wert von a + b;
a = a + 5;	Der Wert von a wird um 5 erhöht
b = (a+c)/2;	b soll der Mittelwert von a und c sein
a = - a;	Das Vorzeichen von v wird umgekehrt
c = abs(c);	Das Vorzeichen von c wird entfernt
a++;	Der Wert von a wird um 1 erhöht
a- -;	Der Wert von a wird um 1 vermindert
c = b/2;	c soll die Hälfte von b sein (ganzer Anteil)
d = b%5	d soll der Rest sein, der bei der Division von b durch 5 auftritt

Namen von Variablen beginnen in C++ normalerweise mit Kleinbuchstaben. Zur besseren Lesbarkeit werden Wortgrenzen oft mit Großbuchstaben markiert, z.B. wertBeimStart

Datentypen ganzer Zahlen

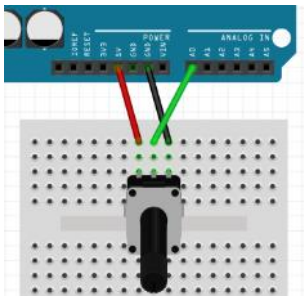
byte	Ganze Zahlen zwischen 0 und 255 (belegt 1 Byte Speicherplatz)
int	Ganze Zahl zwischen -32768 und +32767 (belegt 2 Byte Speicherplatz)
long	für größere Werte (belegt 4 Byte Speicherplatz)

Aufgaben

1. Fügen Sie in der setup()-Funktion die Zeile *Serial.begin(9600);* und in der loop()-Routine die Zeilen *Serial.println(brightness);* und *Serial.println(fadeAmount);* ein und beobachten Sie im Seriellen Monitor, wie sich *brightness* und *fadeAmount* verändern.
2. Probieren Sie die im Kasten aufgezählten Rechenoperationen aus: z. B. mit *int a = 17; int b = -23; Serial.println(a+b);*

Analoge Zahlenwerte

Laden Sie das Programm *Datei -> Beispiele -> 01 Basics -> ReadAnalogVoltage*. Verbinden Sie dann wie im Programm vorgegeben die beiden äußeren Pins eines Potentiometers mit 5V und GND und den inneren mit A0. Starten Sie das Programm und sehen Sie sich im Seriellen Monitor die Ausgabe an, während Sie am Potentiometer drehen.

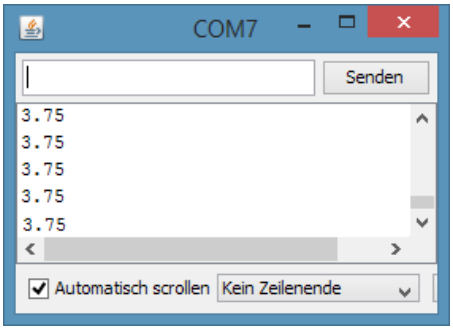


Das Programm liefert Ihnen nicht, wie Sie vielleicht erwartet haben, Zahlenwerte zwischen 0 und 1023 zurück, sondern Kommazahlen. Das liegt daran, dass das Programm den vom Poti gelieferten „Sensorwert“ in eine Voltzahl umrechnet: *float voltage = sensorValue * (5.0 / 1023.0);*

Klar: Aus der 5V-Leitung kommen 5 Volt. Wenn der Poti voll aufgedreht ist, kommt also beim Analogeingang 0 der Wert 1023 an. Ist der Poti aber nur halb offen, so liefert A0 noch etwa den Wert 512 und der wird von der Formel zu 2,5 V umgerechnet.

Bisher haben wir immer mit Integerzahlen gearbeitet. Das sind ganze Zahlen zwischen –37768 und +37767. Für der „analogen“ Zwischenwert 2,5 ist da kein Platz. Aber der Arduino kann auch mit Kommazahlen umgehen. Die Variable *voltage* in unserem Programm wird nicht als *int*, sondern als *float* deklariert. Das ist eine Abkürzung für „floating point value“, also eine „Fließpunktzahl“. Das weist darauf hin, dass wir solche Zahlen in Programmen nicht mit Komma, sondern mit Punkt eingeben müssen.

```
ReadAnalogVoltage

/*
  ReadAnalogVoltage
  Reads an analog input on pin 0,
  converts it to voltage, and prints the result
  to the serial monitor.
  Attach the center pin of a potentiometer to pin A0,
  and the outside pins to +5V and ground.

 This example code is in the public domain.
 */

// the setup routine runs once when you press reset:
void setup() {
  // initialize serial communication at 9600 bits/second:
  Serial.begin(9600);
}

// the loop routine runs over and over again forever:
void loop() {
  // read the input on analog pin 0:
  int sensorValue = analogRead(A0);
  // Convert the analog reading (which from 0 - 1023)
  // to a voltage (0 - 5V):
  float voltage = sensorValue * (5.0 / 1023.0);
  // print out the value you read:
  Serial.println(voltage);
}
```



Schauen Sie nach unter *Hilfe -> Referenz*. Da finden Sie unter *Data Types* neben *int* und *long* noch eine ganze Reihe weiterer ganzzahliger Variablentypen und neben *float* auch noch den Kommazahlentyp *double*. Im Arduino decken beide Datentypen denselben Zahlenbereich ab (10 hoch –38 bis 10 hoch +38) und besitzen dieselbe Genauigkeit (ca. 7 Kommastellen).

Wenn Sie Probleme mit Rundungsfehlern vermeiden wollen, sollten Sie Fließkommazahlen nur da einsetzen, wo es nötig ist. Rechnen Sie so lange wie möglich mit Integer- oder Long-Werten und wandeln Sie diese erst zum Schluss für die Anzeige in Kommazahlen um.

Wenn Sie wollen, dass ein ganzzahliger Wert im Programm wie eine Kommazahlen behandelt wird, dann hängen Sie .0 an.

Aufgaben

1. Ändern Sie die Deklaration in der angegebenen Zeile von *float* auf *int*. Was wird ausgegeben?
2. Ändern Sie Zeile auf *float voltage = sensorValue * (5 / 1023);* Was ändert sich ? Warum?
3. Hängen Sie eine 1,5V-Batterie zwischen GND und A0 und geben Sie auf dem Seriellen Monitor die Spannung aus. Wegwerfbatterien sollten mindestens 1,3 Volt, Akkus mindestens 1,2 Volt liefern.

Hier regiert der Zufall

Zum Abschluss unserer Versuchsreihe mit der Ampel werden wir jetzt Ihre Ampel zum Würfel umfunktionieren. Ihre Ampel kann nämlich auch Würfelbilder anzeigen, und Ihr Arduino kann auch würfeln

Dazu verwenden wir die C++-Funktion *random(int grenze).* Sie liefert eine ganze Zahl zwischen 0 (einschließlich) und *grenze* (ausschließlich). Durch die *Wertzuweisung x = random(6);* wird x also mit einer ganzen Zahl zwischen 0 und 5 belegt. Und wenn wir mit der Anweisung *x = random(6)+1;* zum Ergebnis der random()-Funktion jeweils noch +1 addieren, verschiebt sich der Zahlenbereich auf 1 bis 6.

In Wirklichkeit kann kein Computer würfeln, weil alle programmierbaren Vorgänge im Computer streng determiniert ablaufen. Man behilft sich damit, dass die Funktion random() aus einer vorhandenen Zahl durch Anwendung von Rechenoperationen eine folgende „Pseudo"-Zufallszahl erzeugt, die der Mensch nicht vorhersagen kann. Die random()-Funktion ist aber so programmiert, dass die erzeugten Pseudo-Zufallszahlen gleich verteilt sind, also gleich häufig vorkommen.

Wenn wir ein Programm, das die random()-Funktion enthält, auf dem Arduino starten, würde er bei wiederholtem Aufrufen von random() normalerweise immer dieselbe Zahlenfolge erzeugen. Deshalb holen wir uns den allerersten Wert aus einem Umstand, der nicht vorhersagbar ist, zum Beispiel aus der Spannung auf einem nicht angeschlossenen Analogeingang und setzen die random()-Funktion mithilfe von *randomSeed()* einmalig auf diesen Anfangswert.

random

```
// Demoprogramm für random()

void setup() {
  Serial.begin(9600);
  randomSeed(analogRead(0);
}

void loop() {
  int zufallsZahl = random(6)+1)
  Serial.println(zufallsZahl);
  delay(500);
    int k=0;
  }
}
```

Aufgaben

1. Generieren Sie Würfelzahlen, Lottozahlen oder Zufallszahlen in größeren Zahlbereichen und geben Sie sie auf dem Seriellen Monitor aus. Testen Sie mit einer Strichliste, ob die Zahlen gleich verteilt sind.
2. Erarbeiten Sie eine Reihe von Prozeduren: *void eins(), void zwei(), void drei(), void vier(), void fuenf(), void sechs().* Diese sollen jeweils das Würfelbild der Zahl erzeugen. Gliedern Sie die Prozeduren in eine Bibliothek wuerfel.ino aus.
3. Schreiben Sie ein Programm, das bei Druck auf den Taster eine Würfelzahl generiert und auf die Ampel-LEDs ausgibt.
4. Verwenden Sie ein Reihe von if-Anweisungen, um anhand einer Würfelzahl das entsprechende Würfelbild zu erzeugen: *if (zufallsZahl == 1) {eins();}* usw.
   Sie können aber auch bei dieser Gelegenheit die *Switch-Case*-Anweisung ausprobieren, die in der Arduino-Referenz beschrieben ist.
5. Animieren Sie das Programm, so dass beim Druck auf den Taster nicht gleich die gewürfelte Zahl erscheint, sondern erst einmal auf den LEDs eine Art Würfelbewegungs-Feuerwerk abläuft.
6. Ihren Würfel kann man auch als Belichtungsmesser verwenden: Schreiben Sie ein Programm, dass die Zahlen 0, 1, 2, 3, 4, 5, 6 abhängig von der Helligkeit anzeigt. Setzen Sie die Grenzwerte mit der Hand über dem Fotowiderstand. Wenn es dunkel ist (der Fotowiderstand abgeschattet), zeigt das Instrument 0, wenn es hell ist, die 6.

Schleifen

Zu den elementaren Fähigkeiten jedes Computers gehört es, Schleifen und Wiederholungsanweisungen abzuarbeiten. Jedes Arduino-Programm geht nach der *setup()*-Befehlsfolge in die Endlosschleife der *loop()*-Funktion. Häufig möchte man aber auch innerhalb eines Programmablaufs einige Schritte mehrfach wiederholen, ohne sie mehrfach aufzuschreiben. Dafür stellt uns C++ die *while()*-Funktion zur Verfügung. Sie wiederholt eine Reihe von Anweisungen, *während* eine bestimmte Bedingung erfüllt ist. (Vorsicht: In anderen Programmiersprachen, z.B. BASIC ist die entsprechende Anweisung oft andersherum definiert: Wiederhole, *bis* die Bedingung erfüllt ist.)

Geben Sie jeweils das abgebildete Programm ein, starten Sie es und öffnen Sie den Seriellen Monitor.

while (Bedingung) {Befehle}

„Solange es dir zu kalt ist, musst du heizen“. Der Aufbau dieser Anweisung ist der gleiche wie bei *if():* Ist die Bedingung nicht erfüllt, so wird der Befehlsblock übersprungen. Trifft sie aber zu, so wird der Befehlsblock nicht nur einmal ausgeführt wie bei *if()*, sondern so lange wiederholt, bis sich die Verhältnisse geändert haben und die Bedingungsprüfung scheitert.

Im abgebildeten Programm ist es die Variable n, die sich innerhalb des Befehlsblocks verändert und schließlich nicht mehr kleiner als 10 ist. In anderen Programmen wird der Befehlsblock ausgeführt, während ein Sensor einen bestimmten Wert liefert reagiert oder eine Zeitspanne noch nicht verstrichen ist. Weil hier vor Ablauf der Befehlsfolge erst die Bedingung geprüft wird, spricht man von einer **kopfgesteuerten** Schleife.

do {Befehle} while (Bedingung)

„Heize, solange es zu kalt ist.“ bewirkt fast das Gleiche, die Schleife ist aber hier **fußgesteuert**. Der Befehlsblock wird mindestens einmal ausgeführt, da die Bedingung erst nach der ersten Ausführung überprüft wird. Vergleichen Sie den Unterschied der von *while()* und von *do {} while()* gelieferten Zahlenreihen.

Aufgaben

1. Spielen Sie in einer Schleifen mit Variablen und Bedingungen. Produzieren Sie im Seriellen Monitor Zahlenreihen wie 99, 98, 97, ... 3, 2, 1, oder 1 , 2, 4, 8, ... oder 1, 1, 2, 3, 5, 8, 13, ... o.ä.
2. Analysieren Sie die rechts abgedruckt Programme 🖫 Alarm.ino und 🖫 Blinkblink.ino Probieren Sie sie an Ihrer Ampel aus.
3. Schreiben Sie eine optische Sirene: Beim Druck auf die Ampeltaste soll die mittlere LED immer heller werden, bis die maximale Helligkeit erreicht ist. Dann soll sie dunkler werden, bis sie ganz ausgeht. Das Ganze soll dreimal ablaufen und dann enden. (Mit einem Druck auf die Reset-Taste am Arduino können Sie das Programm erneut starten),
4. Bei kurzem Druck auf die Ampeltaste soll die mittlere LED allmählich heller werden, bis das Maximum erreicht ist. Bei einem zweiten kurzen Druck auf den Taster soll sie dunkler werden, bis sie erloschen ist.

while

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

void loop() {
  int n = 0; // lokale Variable n
  while (n<10) {
    Serial.println(n);
    n++;      // kurz für n=n+1;
    delay(500);
  } while (n<10);
  Serial.println();
}
```

do_while

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

void loop() {
  int n = 0; // lokale Variable n
  do {
    Serial.println(n);
    n++;      // kurz für n=n+1;
    delay(500);
  } while (n<10);
  Serial.println();
}
```

Alarm

```
void setup() {
  pinMode(2,INPUT_PULLUP);
  pinMode(7,OUTPUT);
}

void loop() {
  while(digitalRead(2)==LOW) {
    digitalWrite(7,HIGH);
    delay(100);
    digitalWrite(7,LOW);
    delay(100);
  }
}
```

Blinkblink

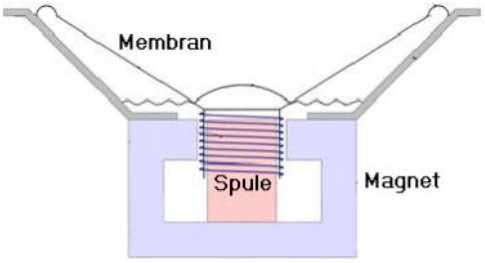
```
void setup() {
  pinMode(7,OUTPUT);
}

void loop() {
  long startZeit = millis();
  while(millis() < startZeit+2000) {
    digitalWrite(7,HIGH);
    delay(100);
    digitalWrite(7,LOW);
    delay(100);
  }
  delay(2000);
}
```

Hast du Töne, Arduino?

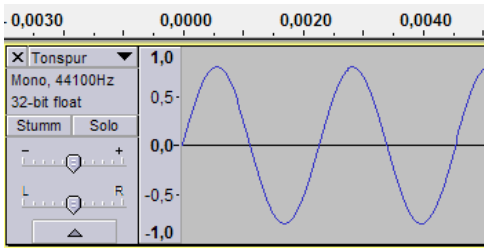
Stückliste

Arduino UNO, Breadboard, Kabel m/m,
Piezo-Beeper oder kleiner Computerlautsprecher, Potentiometer

Wir beginnen mit einem Versuch: Schließen Sie einen kleinen Lautsprecher mit dem einen Kabel an GND und mit dem anderen an Digitalpin 1 an und laden Sie ein beliebiges Programm hoch. Und schon kommt das, was Sie sonst nur als Blinken einer LED wahrnehmen, als Ton aus dem Lautsprecher. Ihr PC hat das Programm kompiliert und schickt es beim Upload über USB zum Arduino. Damit der es versteht, werden ihm die Nullen und Einsen schön langsam über den Digitalpin 1 eingeflößt, etwa 1000 Bit pro Sekunde. Der Lautsprecher macht diesen Vorgang hörbar.

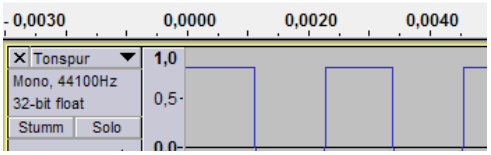


Aber wie funktioniert eigentlich so ein Lautsprecher? In einem Dauermagneten sitzt eine Spule, die durch das Magnetfeld berührungslos in der Mittelposition gehalten wird. An diese Spule sind die Kabelangeschlossen. Füttert man die Spule mit einer Spannung von +5V, so baut sie ein Magnetfeld auf und schiebt sich im Dauermagneten nach vorn, füttert man sie mit -5V, so schiebt sie sich nach hinten. Bei der Bewegung nimmt sie eine trichterförmige Membran aus Pappe mit, und diese wiederum schiebt die Luft vor und zurück. Bei schnellem Wechsel, sagen wir 440 Wechseln von +5 V nach -5V pro Sekunde entsteht eine Schwingung, die unser Ohr als Ton wahrnimmt.



Mit dem Programm Audacity kann man Töne sichtbar machen:
Erzeugen ->Tongenerator 1 ->Wellenform Sinus / Frequenz 440 Hz.

Mit unserem Arduino können wir das Gleiche, etwas weniger elegant, ebenfalls erreichen. Den Lautsprecher hängen wir jetzt an Pin 2 und GND, und damit entfällt das Gequietsche beim Hochladen.



Die Anweisung *digitalWrite(outPin,HIGH)* schiebt die Spule nach vorn und hält sie dort fest, bis der delay abgelaufen ist und *digitalWrite(outPin,LOW)* die Spule wieder in Mittelposition fallen lässt. Nach einem erneuten delay beginnt die *loop()* von Neuem. Um eine Frequenz von 440 Schwingungen pro Sekunde zu erzeugen, dauert jeder der beiden delays eine halbe Sekunde durch 440, und das sind 1135 Mikrosekunden.

Dauerton

```
const int outPin = 2;
int freqenz = 440;

void setup() {
  pinMode(outPin,OUTPUT);
}

void loop() {
  digitalWrite(outPin,HIGH);
  delayMicroseconds(500000/freqenz);
  digitalWrite(outPin,LOW);
  delayMicroseconds(500000/freqenz);
}
```

Das Ergebnis sieht etwa so aus wie in beim Audacity-Befehl *Erzeugen -> Tongenerator 1 -> Wellenform Rechteck. / Frequenz 440 Hz*. Ein Rechtecksignal klingt nicht so weich wie ein Sinuston, aber sonst recht ähnlich. Beim Arduinio wechselt die Spannung allerdings nicht zwischen +5V und -5V wie bei der Soundkarte, sondern nur zwischen +5V und 0 V.

Aufgaben

1. Ändern Sie die Frequenz ab und erzeugen Sie andere Töne. Bis zu welcher tiefen / hohen Frequenz können Sie mit dem Arduino gehen, ohne dass Seiteneffekte auftreten, die den Ton verfälschen?
2. Die Lautstärke können Sie beim Arduino programmtechnisch nicht beeinflussen. Sie können aber eins der Lautsprecherkabel an einen Spannungsteiler anschließen. Schließen Sie ein Potentiometer links an +5V und rechts an GND und mit dem mittleren Pin an den Lautsprecher an.

Vom Ton zur Melodie

Bisher erzeugt Ihr Programm bei jedem Start nur einen einzigen Dauerton. Ihr Arduino besitzt aber eine eingebaute „Uhr“. Mit ihrer Hilfe können wir in einer Loop mehrere Töne unterbringen und deren Dauer variieren.

Die Funktion *millis()* gibt die seit dem Einschalten des Arduino vergangenen Millisekunden an. Mit ihr können wir innerhalb der Dauerschleife loop() kurze Schleifen mit festgelegter Dauer definieren. Das abgebildete Programm sollte sich wie das Besetztzeichen des Telefons anhören.

Mit *startZeit = millis() merkt* sich das Programm die aktuelle Uhrzeit. Da die Dimensionen einer int-Variablen (16 Bit, Zahlbereich von -32768 bis 32767) für die hier vorkommenden Werte nicht ausreichen, muss die Variable *startZeit* als **long** deklariert werden. Eine long-Variable ist 32 Bit lang und kann ganzzahlige Werte bis etwa ±2 Milliarden speichern.

Die folgende *while()*-Schleife produziert einen Ton. Die Schleife läuft so lange, bis die Uhr des Arduino 1000 Millisekunden weitergerückt ist. Danach folgt eine Sekunde Pause und die Loop beginnt von Neuem.

Besetzt

```
const int outPin = 2;
int freqenz = 440;
long startZeit;

void setup() {
  pinMode(outPin,OUTPUT);
}

void loop() {
  startZeit = millis();
  while (millis() < startZeit + 1000)
    digitalWrite(outPin,HIGH);
    delayMicroseconds(500000/freqenz);
    digitalWrite(outPin,LOW);
    delayMicroseconds(500000/freqenz);
  }
  delay(1000);
}
```

Aufgaben

1. Verändern Sie die Tondauer so, dass das Ergebnis sich wie ein Freizeichen anhört.
2. Definieren Sie mit zwei while-Schleifen zwei verschiedene Töne innerhalb der loop() und bauen Sie damit ein Martinshorn (Tatü ... Tatü...), einen Kuckucksruf oder einen Schulgong. 🖫 Tatue.ino

Das rechts abgedruckte Programm 🖫 Halbtonreihe.ino enthält zwei verschachtelte Schleifen. Um die while-Schleife herum, die den Ton erzeugt, ist eine for-Schleife gelegt, die die Frequenz verändert. (Die Frequenzvariable ist hier aus Platzgründen nur mit f benannt).

Bei jedem Durchlauf der for-Schleife wird die Frequenz mit dem Faktor 1,06 multipliziert (ein grober Näherungswert für die 12. Wurzel aus 2). So ergibt sich die Frequenzreihe 466, 493, 522, 553, ... bis die for-Schleife abbricht, weil die 1000 überschritten ist. Danach beginnt die Loop erneut.

Halbtonreihe

```
const int outPin = 2;
long startZeit;

void setup() {
  pinMode(outPin,OUTPUT);
}

void loop() {
  long freq = 440;
  while (freq<1000) {
    startZeit = millis();
    while (millis() < startZeit + 100) {
      digitalWrite(outPin,HIGH);
      delayMicroseconds(500000/freq);
      digitalWrite(outPin,LOW);
      delayMicroseconds(500000/freq);
    }
    freq=freq*106/100;
    delay(100);
  }
}
```

Aufgaben

3. Verändern Sie das Programm so, dass die Dauer der Folge unverändert bleibt, aber die Töne nicht mehr so abgehackt nebeneinander stehen.
4. Lassen Sie die Töne abwärts schreiten. 🖫 Tontreppe.ino
5. Erzeugen Sie ein noch sanfteres Gleiten, indem Sie den Tonabstand vermindern.
6. Erzeugen Sie einen Feueralarm: einen Sirenenton von ganz tief drei Mal hoch und wieder herunter, danach ist Ende.
7. Auf simple Art einen Piepser erzeugen, das geht auch so wie rechts gezeigt. Erklären Sie, wieso.

Piep

```
void setup() {
  pinMode(11,OUTPUT);
  analogWrite(11,100);
}

void loop() {
}
```

Ein Lied, zwei, drei ...

Für eine Melodie braucht man klar definierte Tonhöhen. Der Kammerton A der Stimmgabel ist definiert als Ton mit der Frequenz 440 Hertz. Das A der nächsthöheren Oktave hat die doppelte Frequenz, 880 Hertz, das A der darunter liegenden Oktave hat die halbe Frequenz, 220 Hertz. Aber wie finden wir die Töne dazwischen?

Wenn eine ganze Oktave eine Verdoppelung der Frequenz bedeutet und die Oktave aus zwölf Halbtönen besteht, so muss ich, um von einem Ton zum nächsthöheren Halbton zu kommen (also vom A zum Ais bzw. B) die Frequenz mit der 12. Wurzel aus 2 multiplizieren: 440 * 2^(1/12) = 466 Hz. Um zum nächstniedrigeren Halbton zu kommen, muss ich die Frequenz durch die 12. Wurzel aus 2 dividieren: 440 / 2^(1/12) = 415 Hz. Die Frequenzen runden wir auf ganze Zahlen.

Zum Abspielen der Töne benutzen wir jetzt statt einer while-Schleife die eingebaute tone()-Funktion des Arduino. Sie verlangt zwei Argumente: den Digitalpin, an dem der Lautsprecher hängt, und die Frequenz. Die Tondauer wird duch ein Delay festgelegt, danach wird der Ton durch die Funktion noTone() beendet.

Tonleiter

```
const int outPin = 2;

void setup() {
  pinMode(outPin,OUTPUT);
}

void loop() {
  tone(outPin,262); // C
  delay(250);
  noTone(outPin);

  tone(outPin,294); // D
  delay(250);
  noTone(outPin);
  //...
}
```

Aufgaben

1. Legen Sie eine Excel-Tabelle an und berechnen Sie vom A ausgehend die Frequenzen der darüber und darunter liegenden Halbtöne. Die Halbtonleiter ist C, C#/Db, D, D#/Eb, E, F, F#/Gb, G, G#/Ab, A, A#/Bb, H, C .
   Ergänzen Sie das abgebildete Fragment zur C-Dur-Tonleiter.
2. Bringen Sie dem Arduino eine einfache Melodie bei: (z.B. Kuckuck, Kuckuck ruft's aus dem Wald) und spielen Sie sie mit dem Arduino ab. 🖫 Melodie.ino

Wir trennen Programm und Daten

Auch bei Verwendung der tone-Funktion werden Programm schnell unübersichtlich. Deswegen versuchen wir, die Daten aus dem Programmcode herauszunehmen. Mit so genannten Feldvariablen kann eine ganze Liste gleichartiger Daten unter einem einzigen Namen aufbewahrt werden. Über einen Index kann man auf jedes beliebige Element der Liste zugreifen, über eine for-Schleife kann man den Index hochzählen und alle Elemente nacheinander bearbeiten. Ein Beispiel:

Durch die eckige Klammer in der Deklaration wird die Variable als Feldvariable ausgewiesen. Die Anzahl der Elemente kann wie im Beispiel offen gelassen werden. Bei uns werden die Werte gleich bei der Deklaration aufgezählt, es ist aber auch möglich, Sie erst durch das Programm mit Werten zu füllen. Listenindizes beginnen immer mit 0. Der sechste Ton hat also die Frequenz 392 Hz (freq[5]=392) und die Dauer 500 ms (dauer[5] = 500).

Haenschenklein §

```
const int outPin = 2;

int freq[] = {523,440,440,466,
              392,392,349,392,
              440,466,523,523,
              523};

int dauer[] = {250,250,500,250,
               250,500,250,250,
               250,250,250,250,
               500};

void setup() {
  pinMode(outPin,OUTPUT);
}

void loop() {
  int n=0;
  while(n<14) {
    tone(outPin, freq[n]);
    delay(dauer[n]);
    noTone(outPin);
    n++;
  }
  delay(2000);
}
```

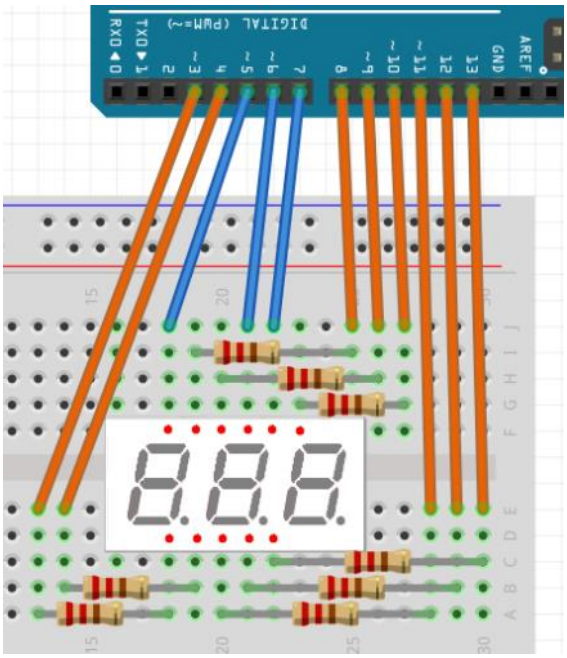
Aufgaben

1. Programmieren Sie eine eigene Melodie. Die Deklaration einer Feldvariablen darf auch über mehrere Zeilen gehen.
2. Laden und starten Sie das Meisterwerk Melody.ino. Analysieren Sie das Programm und codieren Sie anschließend mit dem in den Kommentaren beschriebenen Code eine eigene Melodie.

Zu wenig Pins! Was nun?

Stückliste

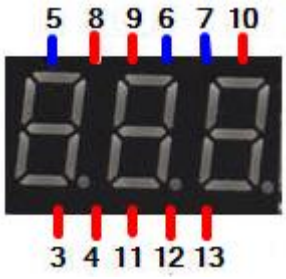
Arduino UNO, Breadboard, 11 Kabel m/m oder Prototype-Shield
8 Widerstände 220 Ω
dreistellige Siebensegmentanzeige (gemeinsame Kathode)



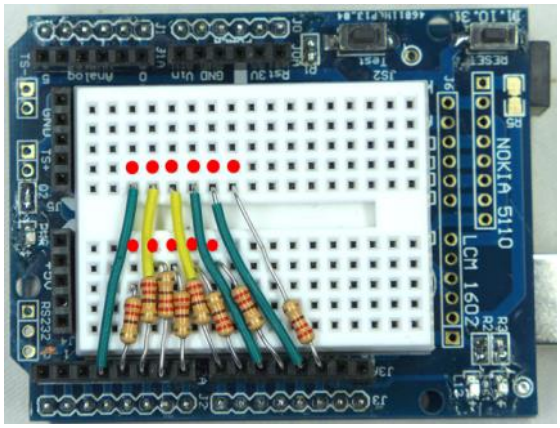
Wie man LEDs mit dem Arduino ansteuert, haben Sie in den Ampelkapiteln gründlich untersucht. In diesem Kapitel wollen wir damit eine Siebensegmentanzeige betreiben. Die folgende Beschreibung bezieht sich auf ein preiswertes handelsübliches Anzeigeelement. Bei anderen Anzeigeelemente können die Pins anders verbunden sein. Hier hilft nur, das Datenblatt zu studieren.

Aber wie schaltet man insgesamt 24 LEDs, wenn man nur 14 Digitalausgänge zur Verfügung hat? Die Lösung:: Alle Segmente einer Ziffer benutzen eine gemeinsame Kathode und alle Segmente an der entsprechenden Position dieselbe Anode. So bleiben drei Kathodenanschlüsse und acht Anodenanschlüsse zu verbinden.

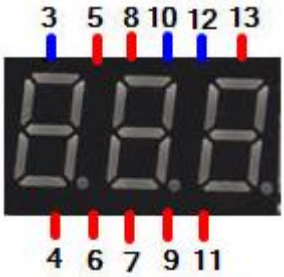
Rechts finden Sie zwei Entwürfe, oben für ein normales Breadboard, unten für das Mini-Breadboard auf einem Prototype-Shield. Ihre Siebensegmentanzeige besitzt unten zwei Reihen mit Anschlüssen: einmal sechs und einmal fünf. Diese werden über die Rille hinweg auf dem Breadboard eingesteckt. Die Position, an der das Anzeigeelement eingesteckt wird, ist jeweils mit roten Punkten markiert.



Zunächst jedoch sollten Sie die Widerstände und Drahtbrücken (hier grün) auf die erforderliche Länge kürzen und einstecken. Die Anschlussdrähte der Widerstände dürfen sich nicht berühren und sie dürfen auch keinen Kontakt zu den Pins des Anzeigeelements bekommen. Biegen Sie sie mit einem Schraubenzieher hin oder ziehen Sie notfalls ein Stück Isolation (hier gelb) über die Drähte. Stecken Sie zum Schluss das Anzeigeelement in die rot markierten Löcher.



Um die Verbindungen zu optimieren, sind die einzelnen Pins des Anzeigeelements mit den Digitalpins des Arduino bei beiden Lösungen auf unterschiedliche Weise verbunden (siehe Schema).



Aufgaben

1. Starten Sie den Arduino. Setzen Sie jeweils den Anschluss einer Kathode (im Anschlussschema blau markiert) auf LOW und dann nacheinander die Anschlüsse aller Anoden auf HIGH. Notieren Sie: Welche Kathode gehört zur linken, zur mittleren Ziffer? Welche Anode gehört zum Segment oben, oben links, oben rechts, mitte, unten links, unten rechts, unten, Punkt?
2. Zeigen Sie auf der mittleren Ziffer eine 3 am.
3. Zeigen Sie die Zahl 999 an. 🖫 Schnapszahl.ino

Mehrstellige Zahlen

Das rechts abgebildete Programm ist für die zweite Variante (Prototype-Shield) eingerichtet. Wenn Sie es mit dem Breadboard betreiben möchten, dann setzen Sie die Kommentarzeichen so, dass die Pinbelegung für das Breadboard aktiviert und die für das Prototype-Shield deaktiviert wird.

Aufgaben

1. Analysieren Sie das vorhandene Programm 🖫 Ziffertest.ino. Welche Zahl zeigt es an? Was müssen Sie tun, um die Stelle zu ändern?
2. Ergänzen Sie Funktionen zur Anzeige der Ziffern 0 bis 9. Komplettieren Sie anschließend die Funktion *mitte*() und ergänzen Sie die Funktionen *links*() und *rechts*().
3. Zeigen Sie die Zahl 123 an. Wenn Sie diese Aufgabe nicht in angemessener Zeit lösen können, dann beschreiben Sie, bevor Sie weiterlesen, möglichst genau das Problem.

Multiplexing

Unser Problem liegt darin, dass die Ziffern sich durch die gemeinsamen Leitungen gegenseitig beeinflussen. Um das Problem zu lösen, müssen Sie Ihre Augen überlisten.

Sie übertragen jede drei Ziffern zeitversetzt über dieselbe Leitung und geben jeder Ziffer in kleinen zeitversetzten Häppchen jeweils ein Drittel der Anzeigezeit. Methoden, mit denen verschiedene Informationen quasi gleichzeitig über denselben Kanal übertragen werden, nennt man **Multiplexverfahren** (siehe dazu auch die Wikipedia)

Aufgaben

4. Lassen Sie in schneller Folge die 1 auf der Hunderterstelle aufleuchten, dann die 2 auf der Zehnerstelle und schließlich die 3 auf der Einerstelle. Und dann beginnen Sie von vorn. Finden Sie durch Experimente die optimale Standzeit, bei der die Anzeige nicht mehr flackert.
5. Beschreiben Sie die rechts abgebildete Ausgaberoutine *zeige()*, der Sie nur noch eine maximal dreistellige Zahl übergeben müssen, die dann angezeigt wird. Erklären Sie anhand der Referenz, was der Operator % bewirkt.
6. Bauen Sie die Routine in Ihr Programm ein. Verschieben Sie dann wie bei der Ampel alle eigenen Funktionen und auch die Konstantendefinition in eine Bibliothek namens LED.h. Vergessen Sie nicht, am Beginn Ihres Programms die Anweisung *#include „LED.h“* und am Beginn der Bibliothek die Anweisung *#include Arduino.h*> hinzuzufügen. Speichern Sie Ihr Werk unter dem Titel 🖫 Siebensegment.ino.

Ziffertest

```
/*Pinbelegung Breadboard-Variante
const int li=5, mi=6, re=7,
      o=8, lo=9, ro=10, m=13, lu=3, ru=12, u=4, p=11;
*/
/* Pinbelegung Prototype-Shield-Variante */
const int li=3, mi=10, re=12,
      o=5, lo=8, ro=13, m=11, lu=4, ru=9, u=6, p=7;

void setup() {
  pinMode( 3,OUTPUT);
  pinMode( 4,OUTPUT);
  pinMode( 5,OUTPUT);
  pinMode( 6,OUTPUT);
  pinMode( 7,OUTPUT);
  pinMode( 8,OUTPUT);
  pinMode( 9,OUTPUT);
  pinMode(10,OUTPUT);
  pinMode(11,OUTPUT);
  pinMode(12,OUTPUT);
  pinMode(13,OUTPUT);
}

void P1() {
  digitalWrite(o,LOW);
  digitalWrite(lo,LOW); digitalWrite(ro,HIGH);
  digitalWrite(m,LOW);
  digitalWrite(ru,HIGH); digitalWrite(lu,LOW);
  digitalWrite(u,LOW); digitalWrite(p,LOW);
}

void mitte(int ziffer) {
  digitalWrite(li,HIGH);
  digitalWrite(mi,LOW);
  digitalWrite(re,HIGH);
  // if (ziffer==0) P0();
  if (ziffer==1) P1();
  // if (ziffer==2) P2();
  // ...
}

void loop() {
  mitte(1);
}
```

```
void zeige(int zahl) {
  int hunderter, zehner, einer, rest;
  hunderter = zahl/100;
  rest = zahl%100;
  links(hunderter); delay(5);
  zehner = rest/10;
  einer = rest%10;
  mitte(zehner); delay(5);
  rechts(einer); delay(5);
}

void loop() {
  zeige(123);
}
```

Schleifen für Fortgeschrittene

Im Zusammenhang mit der Ampel haben wir bereits die *while()*-Anweisung kennengelernt, die es erlaubt, Befehlssequenzen zu wiederholen, solange eine bestimmte Bedingung erfüllt ist. Wenn es bei der Bedingung um Zahlen geht, ist diese Konstruktion oft etwas umständlich: Man muss zunächst außerhalb der Schleife eine Variable initialisieren, diese innerhalb der Schleife verändern und nach Ablauf der Schleife abfragen. Um einen Befehlsblock mehrfach (sagen wir zehn Mal) auszugeführen, ist das zu umständlich. Hier bietet C++ eine bessere Anweisung an:

for (Initialisierung; Bedingung; Veränderung) {Befehle}

In der *for() {}*-Anweisung werden alle Rahmenbedingungen der Schleife in einem einzigen Block zusammengefasst. Hinter dem for folgt eine runde Klammer mit drei durch Semikolon getrennten Teilen:

- der Deklaration (int n) und Initialisierung (n=0) einer **Laufvariablen**,
- der Bedingung, unter der der Befehlsblock ausgeführt werden soll,
- die Veränderung der Laufvariablen bei jedem Schleifendurchlauf.

Anschließend folgt in geschweiften Klammern der Befehlsblock. Unser Demoprogramm zählt von 0 bis 9 und gibt die Zahlen im Seriellen Monitor aus.

for

```
// Demoprogramm für for(..; ..; ..)

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

void loop() {
  for(int n=0; n<10; n++) {
    Serial.println(n);
    delay(500);
  }
  Serial.println();
}
```

Aufgaben

1. Ändern Sie die Initialisierung so, dass das Programm bei 1 beginnt.
2. Ändern Sie die Bedingung so, dass das Programm bei 10 endet.
3. Ändern Sie alle drei Abschnitte der runden Klammer so, dass das Programm in einem Countdown von 10 bis 0 rückwärts zahlt.
4. Ersetzen Sie in den Programmen 🖫 Halbtonreihe.ino (S.56) und 🖫 Haenschenklein.ino (S.57) die while()-Schleife durch eine for()-Schleife.
5. Ändern Sie den setup()-Abschnitt des Programms 🖫 Zifferntest.ino auf der vorigen Seite so, dass die Pins von 3 bis 13 in einer einzigen Programmzeile auf den Pinmode OUTPUT gesetzt werden.

Lokale und globale Variablen

Jede Variablen hat einen **Gültigkeitsbereich**. Während die **globale Variable a** im ganzen Programm sichtbar ist, gelten die **lokalen Variablen b, c, d** nur für den jeweiligen Befehlsblock und werden danach entfernt. Deshalb erhalten beim Kompilieren des rechts abgebildeten Programms 🖫 Lokal.ino vier Fehlermeldungen: Die Variable b gilt nur im Setup, d gilt nur in der for-Schleife. Entfernen Sie die vier fehlerhaften Zeilen, starten Sie dann das Programm und den Seriellen Monitor. Über die Durchläufe der Loop hinweg hochgezählt wird nur die Variable a. Die Variable c wird bei jedem Durchlauf der Loop neu initialisiert und immer als 1 angezeigt.

Lokal

```
int a=0;

void setup() {
  Serial.begin(9600);
  int b=0;
}

void loop() {
  int c=0;
  for (int d=0;d<10;d++) {}
  a++;
  b++;
  c++;
  d++;
  Serial.print("a= "); Serial.println(a);
  Serial.print("b= "); Serial.println(b);
  Serial.print("c= "); Serial.println(c);
  Serial.print("d= "); Serial.println(d);
  delay(1000);
}
```

Aufgaben

6. Schreiben Sie ein Programm, das auf der Siebensegmentanzeige bis 10 zählt.
7. Erzeugen Sie auf der Siebensegmentanzeige Lottozahlen.
8. Erzeugen Sie mit einer for-Schleife die Zahlenfolge 1, 1 , 2, 3, 5, 8, 13, 21, ...
9. Erzeugen Sie mit for-Schleifen die Zahlenfolge 1, 2, 1, 2, 3, 2, 1, 2, 3, 4, 3, ...

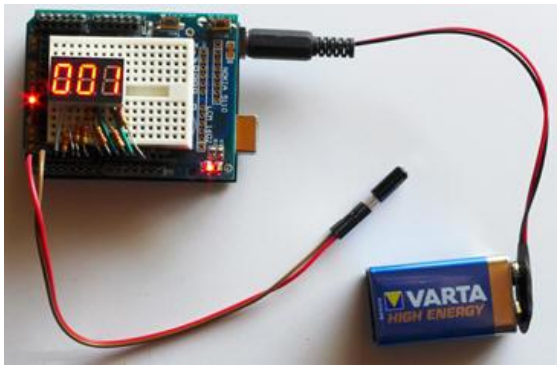
Messen mit Arduino

Hier finden Sie nun einige Bauvorschläge für Messgeräte auf Basis Ihrer Siebensegmentanzeige. Programmieren Sie die Geräte zunächst ganz normal am USB-Kabel. Später können Sie eine Batterie anschließen und sich vom PC befreien.

Stückliste

Arduino UNO, Board mit Siebensegmentanzeige
2 Verlängerungskabel m/w, Tilt-Switch / Heißleiter 10 kΩ
Ultraschall-Entfernungsmesser HC-SR04 / 3 Mikroschalter

Rüttelzähler



Vielleicht finden Sie in Ihrem Inventar einen kleinen schwarzen Metallzylinder mit zwei Anschlüssen. Das ist ein einfacher Neigungssensor, auch "Tilt-Switch" genannt, der erfunden wurde, um Manipulationen an Flipperautomaten zu verhindern. Er enthält ein kleines Metallkügelchen in einer Röhre. In der Röhre sind zwei Kontakte, die bei aufrechter Position durch das Kügelchen verbunden werden. Wenn Sie das Teil aber neigen oder schütteln, öffnet sich der Schalter.

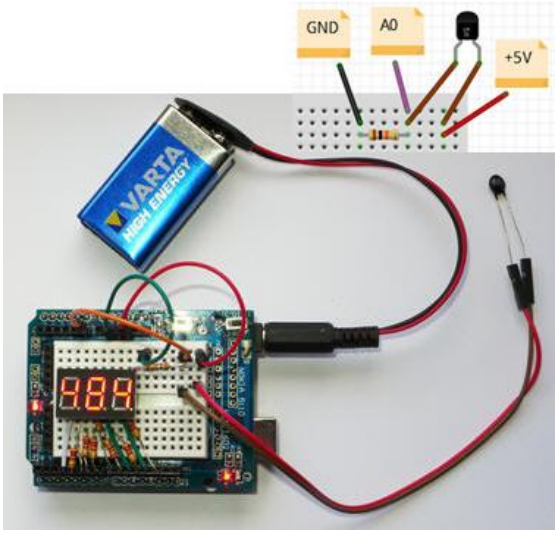
Schließen Sie den **Tilt-Switch** über ein Verlängerungskabel an die Digitalanschlüsse 1 und 2 an. Hier ist ein Programm, das bei liegendem Switch 1 und bei stehendem Switch 0 anzeigt.

Schreiben Sie ein Programm, das die Anzahl der Unterbrechungen zählt und anzeigt Veranstalten Sie einen Wettbewerb, wer am schnellsten schütteln kann.

Tilt | LED.h

```
int outPin = 1;
int inPin = 2;

#include "LED.h"

void setup() {
  pinMode(outPin,OUTPUT);
  pinMode(inPin,INPUT_PULLUP);
  for (int n=3; n<=13 ;n++) pinMode(n,OUTPUT);
}

void loop() {
  digitalWrite(1,LOW);
  if (digitalRead(2)==HIGH) zeige(1); else zeige(0);
}
```

Digitalthermometer



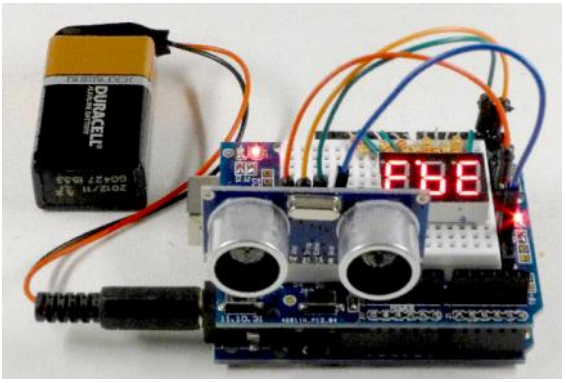
Wenn Sie ein schwarzes Teil mit der Aufschrift "103" besitzen, ist das vermutlich ein so genannter Heißleiter, der je nach Temperatur einen Widerstand von 0 bis 10000 Ohm erzeugt.

Bauen Sie damit einen Spannungsteiler: Verbinden Sie das eine Bein des **Heißleiters** mit +5V und das andere mit einem 10K-Widerstand mit GND, verbinden Sie die andere Seite des Widerstands und das eine Bein des Heißleiters zusammen mit A0 und das andere Bein des Heißleiters mit +5V. Lassen Sie sich den Messwert anzeigen. Rubbeln Sie den Heißleiter dann zwischen Daumen und Zeigefinger und beobachten Sie die Temperaturveränderung. Für weitere Messungen sollten Sie den Heißleiter über ein Verlängerungskabel an das Breadboard anschließen). Folgende Vergleichswerte wurden experimentell ermittelt:
Der Heißleiter liefert bei -12 °C den Wert 110 und bei 38°C den Wert 670. Die Messwerte steigen linear an. Definieren Sie mithilfe eines Mathematikbuchs (-> Punktsteigungsform) eine Umrechnungsfunktion, die die Heißleiterwerte in Grad Celsius umrechnet und programmieren Sie dann den Arduino so, dass er nicht den Messwert, sondern die Temperatur in Grad Celsius ausgibt.

Heissleiter | LED.h §

```
int messWert = 0;

#include "LED.h"

void setup() {
  for (int n=3;n<14;n++) pinMode(n,OUTPUT);
}

void loop() {
  zeige(analogRead(messWert));
}
```

Entfernungsmesser



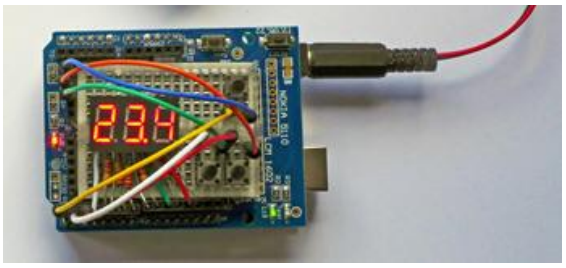
Für den Arduino gibt es ein **Ultraschallmodul**. HC-SR04 Es besitzt außer den zwei Pins für Vcc (+5V) und GND die Anschlüsse TRIG und ECHO. Das abgebildete Programm 🖫 Ultraschall.ino schickt bei jedem Durchlauf der Loop über den *TrigPin* einen kurzen Tonimpuls. von 10 Mikrosekunden Dauer. Danach misst es mit der Anweisung *pulseIn()*, wie viele Mikrosekunden es dauert, bis das Echo des Tons zurückkommt und den *EchoPin* auf HIGH setzt . Damit die zurückgegebene Zahl in die Anzeige passt, wird sie hier durch 100 geteilt.

- Experimentieren Sie mit dem Gerät und verändern Sie den Divisor so, dass der Abstand vom Hindernis in Dezimeter angezeigt wird.
- Hängen Sie an den Digitalpin 1 einen Lautsprecher und warnen Sie, wenn der Abstand vom Hindernis einen Meter unterschreitet.
- Laden Sie das Programm 🖫 Entfernungsmesser.ino. Dessen Bibliothek LED2.h enthält eine verbesserte LED-Ansteuerungsroutine "output()", die die Anzeige besser ausnutzt und auch Kommazahlen von 0,01 bis 999 ausgeben kann.

Ultraschall | LED.h

```
const int TrigPin = 1;
const int EchoPin = 2;
#include "LED.h"

void setup() {
  for (int n=3; n<=14; n++) pinMode(n,OUTPUT);
  pinMode(TrigPin,OUTPUT);
  pinMode(EchoPin,INPUT);
  digitalWrite(TrigPin,LOW);
}

void loop() {
  digitalWrite(TrigPin,HIGH);
  delayMicroseconds(10);
  digitalWrite(TrigPin, LOW);
  long ms = pulseIn(EchoPin,HIGH);
  zeige(ms/100);
}
```

Stoppuhr



Setzen Sie drei **Mikroschalter** auf das Breadboard und programmieren Sie eine einfache Stoppuhr mit drei Tastern. 🖫 Stoppuhr.ino

- Nach dem Programmstart zeigt das Programm 000 an.
- Nach Druck auf den ersten Taster (an GND und Digitalpin 0) beginnt es die Sekunden zu zählen, bei Druck auf den zweiten Taster (an GND und Digitalpin 1) hält es an. Bei Druck auf den dritten Taster (an GND und Digitalpin 2) wird die Anzeige auf 000 zurückgestellt.
- Wenn Sie LED.h so verändern, dass in der mittleren Ziffer der Dezimalpunkt eingeschaltet ist, können Sie die Anzeige auch auf Zehntelsekunden umstellen.

Stoppuhr § | LED.h

```
const int startPin = 0;
const int stoppPin = 1;
const int resetPin = 2;

long startZeit, dauer;

#include "LED.h"

void setup() {
  for (int n=3;n<14;n++) pinMode(n,OUTPUT);
  pinMode(startPin,INPUT_PULLUP);
  pinMode(stoppPin,INPUT_PULLUP);
  pinMode(resetPin,INPUT_PULLUP);
}

void loop() {
  //...
  zeige(dauer);
}
```

Der Eyecatcher

Ein HD-Bildschirm besteht aus 1080 Zeilen zu je 1920 Pixeln. Jedes Pixel wird durch je eine LED in den Farben Rot, Blau, Grün dargestellt. Das macht 6 220 800 LEDs auf dem Bildschirm und ebenso viele Bytes Speicherplatz im Grafikspeicher. 18 Millionen kleine Pünktchen auf ein paar Quadratdezimeter Display, das geht offenbar zu machen, ebenso wie 18 Millionen Bytes im Arbeitsspeicher des Computers. Was nicht geht, sind 18 Millionen Leitungen von hier nach da.

Die Lösung für dieses Problem ist aber nicht mehr ganz neu. Schon 1884 erhielt der Student Paul Nipkow ein Patent auf ein Verfahren, das ein Bild punktweise abtastete und wieder auf dem Schirm einer Elektronenröhre erzeugte. Voraussetzung war, dass jeder Punkt der Elektronenröhre nach dem Beschuss durch den Elektronenstrahl noch eine Weile nachleuchtete. Dass andererseits das Auge des Betrachters so träge ist, dass es bei schneller Bildfolge keine Einzelbilder, sondern einen kontinuierlichen Ablauf „sieht“, das hatte man schon vorher entdeckt.

Multiplexverfahren wurden entwickelt, um eine optimale Ausnutzung der Leitungen und Frequenzen zu erreichen, die in der Elektronik und Kommunikationstechnik als Übertragungswege zur Verfügung stehen.

Hierdurch werden die Kosten verringert und die Zuverlässigkeit erhöht, da zum Beispiel weniger Anschluss- und Verbindungsleitungen erforderlich sind. Manche technische Lösungen sind überhaupt nur mit multiplexer Signalübertragung realisierbar (z. B. die Ansprache bzw. das Auslesen einzelner Pixel digitaler Kameras und Flachbildschirme)."

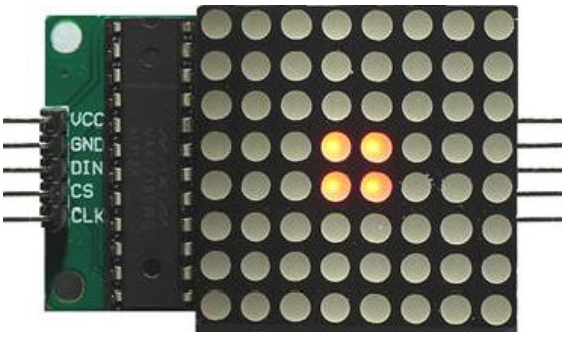
(Wikipedia-Artikel "Multiplexverfahren",

Punktmatrix für den Hausgebrauch

Stückliste

Arduino UNO, Board , 4 Verlängerungskabel m/w, 8x8-Matrix mit MAX7219

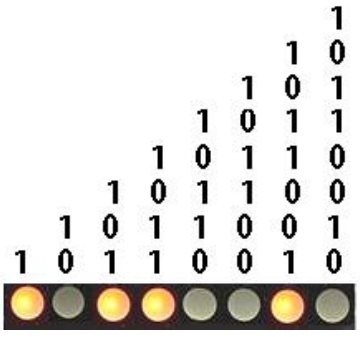
Fangen wir bei unseren Videoübertragungen mal klein an— mit dem abgebildeten Punktmatrix-Modul. Bei 8 Zeilen zu je 8 Spalten müsste das Teil selbst bei Multiplexbetrieb eigentlich 16 Anschlussleitungen haben. Die sehen Sie auch, wenn Sie das Anzeigemodul aus der Fassung herausnehmen.



Zwischen Matrix und Arduino ist ein Chip (MAX7219) geschaltet, der die Daten annimmt, die übertragenen Daten aufbewahrt und den Betrieb der Anzeige aufrechthält, während der Arduino nicht sendet. Im Gegensatz zur Siebensegmentanzeige ist der Arduino also bei diesem Anzeigemodul nicht permanent beschäftigt, sondern kann sich zwischen den Anzeigeänderungen anderen Dingen zuwenden.

Die Platine als Ganzes wird aber, abgesehen von der Stromversorgung über Vcc (5V) und GND, nur über eine **serielle Datenverbindung** aus drei Leitungen gesteuert:

- die eigentliche Datenleitung, beschriftet mit DIN,
- eine zweite Leitung, beschriftet mit CS oder LOAD, mit der der Arduino dem MAX-Chip mitteilt, dass eine Datenübertragung ansteht,
- und den Taktgeber, beschriftet CLK, mit dem der Arduino dem Max-Chip anzeigt, dass das nächste Bit auf der Datenleitung liegt.



Die Datenübertragung funktioniert seriell. Der Arduino sendet eine für die erste Zeile eine Binärzahl beginnend mit der höchsten Stelle Bit für Bit. Der MAX7219-Chip schiebt die Nullen und Einsen von rechts nach links in ein **Schieberegister** ein. Dann folgen die anderen Zeilen. Wenn die achte Zeile übertragen ist, beginnt der MAX7219-Chip selbstständig mit der Multiplexansteuerung der acht Displayzeilen.

Schließen Sie Vcc beim MAX7219 an +5V beim Arduino, GND an GND, DIN an D02, CS an D03 und CLK an D04. Auf der nächsten Seite finden Sie ein Programm.

Matrix reloaded

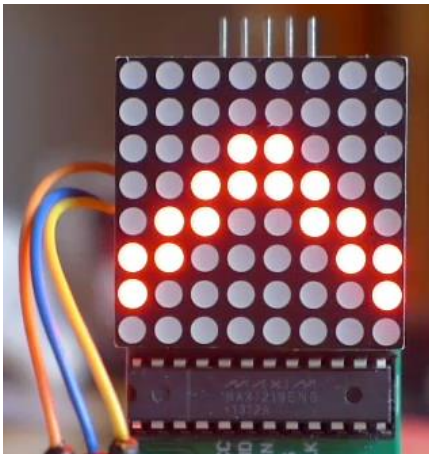
Hier ist ein Programm, mit dem Sie die LED-Matrix ansprechen können:

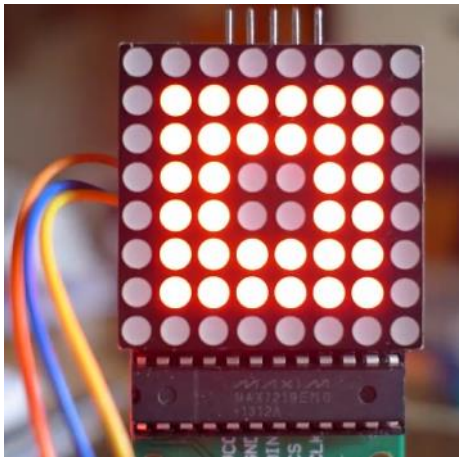
Aufgaben

1. Studieren Sie das abgebildete Programm Dot8x8Demo.ino. Wenn Sie meinen, die Abläufe einigermaßen verstanden zu haben, laden Sie Dot8x8.ino und starten es.
2. Schreiben Sie statt B00011000 eine Dezimalzahl in den Set-Befehl. Welche müssen Sie nehmen?
3. Erzeugen Sie ein eigenes Bild.
4. Verschieben Sie das Setzen der Datenregister in die Loop. Lassen Sie zwei Bilder im Wechsel anzeigen oder ein Logo blinken.
5. Mit drei Bildern oder mehr kann man Bewegungseffekte erzielen: Pfeil Herzschlag Erzeugen Sie aus drei oder mehr Bildern eine eigene Animation.
6. Probieren Sie das Folgende:
   void loop() {
   set(random(8)+1, random(256))
   }
   Erklären Sie (schon bevor Sie es ausprobieren), was da passiert.
7. Weitere Ideen unter http://www.planetarduino.org/?cat=435

Dot8x8Demo

```
// Ansteuerung einer 8x8-Matrix mit Maxim 7219-Chip
// 3.5.2014
// Reinhard Atzbach

int din = 2;  // Data in:     Datenleitung (kann nur 0 oder 1 sein)
int cs  = 3;  // Chip Select: LOW=Übertragung im Gange
int clk = 4;  // Clock:       LOW->HIGH schiebt jeweils ein Bit zum Chip

void putByte(byte data) {          // Serielle Übertragung eines Bytes
  for (byte i=128 ; i>0 ; i=i/2) { // von höchster zu niedrigster Stelle
    digitalWrite(clk,LOW);
    if (data & i){                 // Wenn im Datenbyte eine 1 steht,
      digitalWrite(din,HIGH);      // dann setze Datenleitung auf 1
    } else {
      digitalWrite(din,LOW);       // sonst setze Datenleitung auf 0
    }
    digitalWrite(clk,HIGH);        // Wenn clk von LOW auf HIGH geht
  }                                // schiebe alle Bits eine Stelle nach links
}                                  // und setze die Einerstelle.

void set(byte regNum, byte dataByte) {  //setzt Register des Chips
  digitalWrite(cs,LOW);                 // Übertragung beginnt
  putByte(regNum);                      // sendet Adresse
  putByte(dataByte);                    // sendet Daten
  digitalWrite(cs,HIGH);                // Übertragung beendet, Daten gültig
}

void setup () {
  pinMode(din,OUTPUT); // Alle drei Leitungen auf Ausgabemodus
  pinMode(clk,OUTPUT);
  pinMode(cs,OUTPUT);
                       // Initialisiere die Steuerregister des MAX7219-Chips
  set( 9,0);   // decodeMode:  0=LED-Matrix
  set(10,5);   // inensity:    LED-Helligkeit
  set(11,7);   // ScanLimit:   Spalte 0-7, 8x8-Matrix angeschlossen
  set(12,1);   // Shutdown:    aktiviert alle LED
                       // 1 bis 8 sind die Datenregister für die Zeilen 1 bis 8
  set(1,B00000000);
  set(2,B00000000);
  set(3,B00000000);
  set(4,B00011000);
  set(5,B00011000);
  set(6,B00000000);
  set(7,B00000000);
  set(8,B00000000);
}

void loop () {
}
```





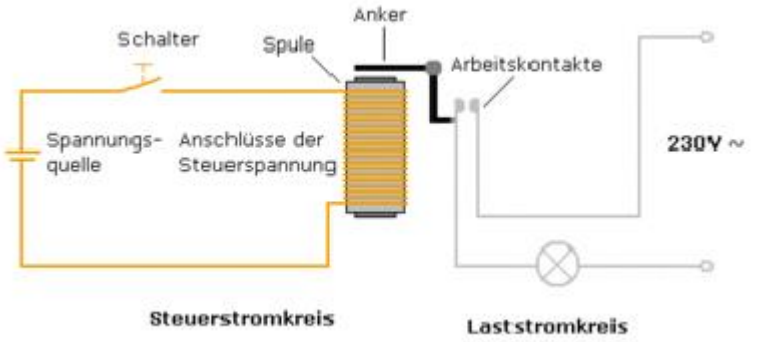
Jetzt geht's rund

Bisher haben Sie mit Ihrem Arduino Lämpchen zum Leuchten gebracht und Lautsprecher in Schwingungen versetzt. Da muss noch mehr Action drin sein! Das Problem ist nur: Bewegung kostet Kraft, und allzu viel Power dürfen wir der USB-Schnittstelle des PCs oder einer keinen Batterie nicht abverlangen, sonst macht sie die Grätsche. Wir merken uns: Der Arduino kann - wie jeder Computer - zwar größere Lasten schalten, aber er kann sie nicht selbst mit Strom versorgen. An den Stromkreis auf der Platine kann man eine Leuchtdiode anschließen, aber keine 230 -Volt Glühbirne und erst recht keinen starken Motor.

Stückliste

Arduino UNO, Board 2 Verlängerungskabel m/w,
Baustein mit zwei Relais, Gleichstrommotor, Batteriepack mit Kabelenden

Die Lösung des Problems besteht in einem Bauteil, bei dem ein schwacher Stromkreis einen Kontakt in einem starken Stromkreis betätigt: das Relais. Im Artikel „Relais „der Wikipedia sehen Sie eins in Aktion. Auf der linken Seite der Abbildung (Quelle: Wikimedia) ist in gelb der Steuerstromkreis eingezeichnet. Dieser Stromkreis wird über eine Batterie mit einem schwachen Gleichstrom versorgt. Ein Druck auf den Schalter schließt den Stromkreis und setzt die Spule unter Spannung. Die Spule wiederum verwandelt den Spulenkern in einen Elektromagneten und der wiederum schließt über den beweglichen Anker den Kontakt im Laststromkreis mit 230 V Wechselstrom. Und schon leuchtet die Glühbirne. Wir merken: Der Laststromkreis in einem Relais ist vom Steuerstromkreis völlig unabhängig.



Auch für den Betrieb mit dem Arduino gibt es preiswerte Relais. Lesen Sie die Aufschrift: Die untere Zeile besagt, dass der Steuerstromkreis mit 5 Volt Gleichstrom betrieben wird. Die beiden Zeilen darüber nennen Beispiele für den Laststromkreis. Mit einem der üblichen Relaisbausteine könnte der Arduino durchaus 230-Volt Wechselstrom mit 10 Ampère schalten, was sogar für einen Staubsaugermotor reichen würde. Mit zwei weiteren Relaisbausteinen könnten wir dann ein Fahrwerk schalten und fertig wäre der Arduino-gesteuerte Staubsaugerroboter.

Das machen wir aber nicht. Bei uns führt der Steuerstromkreis durch den Arduino und im Laststromkreis sitzt als Stromquelle eine Batterie und als Verbraucher ein kleiner Gleichstrommotor. Auf dessen Achse kann man, damit die Wirkung besser sichtbar wird, einen zum Propeller geformten Aufkleber befestigen.

Zunächst der Steuerstromkreis:

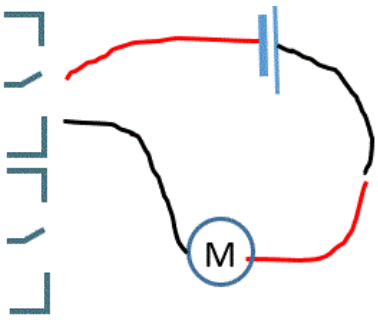
Verbinden Sie den Arduino mit dem 4er-Stecker des Relaisbausteins, der nicht durch den kleinen Jumper (Steckbrücke) belegt ist, wie folgt:

- 5V am Arduino mit Vcc am Relaisbaustein
- GND am Arduino mit GND am Relaisbaustein
- Digitalausgang 7 am Arduino mit IN2 am Relaisbaustein

Starten Sie dann das abgebildete Programm. Das Relais sollte jede Sekunde einmal hörbar schalten und auf dem Baustein sollte eine LED leuchten.

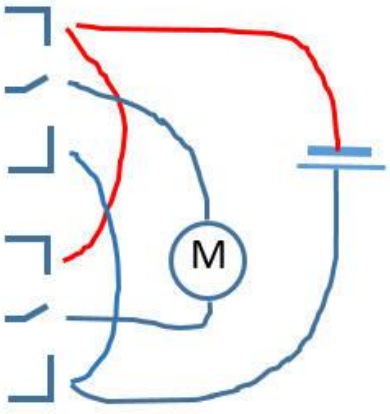
```
void setup() {
   pinMode(7,OUTPUT);
}

void loop() {
  digitalWrite(7,LOW);
  delay(1000);
  digitalWrite(7,HIGH);
  delay(1000);
}
```

Der Laststromkreis:



An der anderen Seite des Relais sehen Sie drei Anschlüsse, die mit Haken markiert sind. Wenn das Relais knackt, wechselt der mittlere Kontakt vom oberen zum unteren oder zurück. Schließen Sie das eine Kabel der Batterie an den oberen Kontakt und das eine Kabel des Motors an den unteren Kontakt an. Verbinden Sie die beiden anderen Kabel miteinander. Nun sollte der Motor beim ersten Knacken des Relais loslaufen, beim zweiten wieder anhalten.

Für einen Ein-Aus-Schalter benötigen wir nur einen der beiden äußeren Anschlüsse des Relais. Unser Relais ist aber in Wirklichkeit ein Wechselschalter. Der mittlere Kontakt schaltet zwischen den beiden äußeren hin und her.



Für eine echte Rechts-Linkslauf-Schaltung benötigt man zwei Wechselschalter-Relais. Damit baut man eine so genannte H-Schaltung auf. Die beiden Anschlüsse des Motors werden mit dem jeweils mittleren Kontakt der beiden Relais verbunden, der eine Pol der Stromquelle mit den beiden oberen, der andere mit den beiden unteren Kontakten. Schließen Sie das zweite Relais über den Digitalausgang D6 des Arduino und den IN2-Eingang des Relaisbausteins an. Dann haben Sie folgende Möglichkeiten:

- Wenn Sie sowohl D6 als auch D7 auf LOW schalten, steht der Motor still,
- wenn Sie D6 auf HIGH und D7 auf LOW schalten, dreht der Motor rechtsherum,
- wenn Sie D6 auf LOW und D7 auf HIGH schalten, dreht der Motor linksherum,
- wenn Sie sowohl D6 als auch D7 auf HIGH schalten, steht der Motor still.

Aufgaben

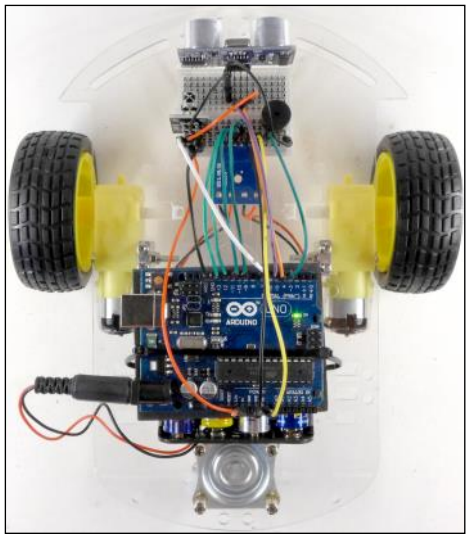
1. Schreiben Sie ein Programm, das den Motor eine Sekunde nach rechts dreht, ausschaltet, nach links dreht, ausschaltet...
2. Ändern Sie das Programm so ab, dass Sie mit einem kleinen Schalter, den Sie in D3, D4 und D5 eingesteckt haben, den Motor auf Vorlauf, Rücklauf und Stopp schalten können.

Eine Nummer kleiner

Im Roboter verbaut ist ein anderer Motortreiber für zwei Motoren. Die Pins links werden an Digitalausgänge des Arduino und die Stromversorgung, an den Klemmen rechts zwei Motoren. In den beiden angeschlossenen Chips sitzt jeweils eine H-Schaltung mit zwei **Leistungstransistoren** als Verstärker.

Wechselstrommotoren und 230-Volt-Glühbirnen kann man mit diesen Bausteinen natürlich nicht betreiben, aber die 5 Volt und 40 Milliampere der Digitalausgänge des Arduino schalten hier einen Strom von bis zu 12 V und 800 Milliampère der Motoranschlüsse, was für kleine Gleichstrommotoren ausreicht.

Da diese Treiber nicht mechanisch, sondern elektronisch funktionieren, arbeiten sie wesentlich schneller als Relais. So ist es möglich. die Geschwindigkeit der angeschlossenen Motoren über Pulsweitenmodulation stufenlos zu regulieren, was mit Relais nicht funktioniert. Weil der Schaltstromkreis in diesen Treibern aber nicht völlig vom Leistungsstromkreis getrennt ist, kommt es hier allerdings auch leichter zu chipzerstörender Überlast.



Level 5:
Klassen und Objekte

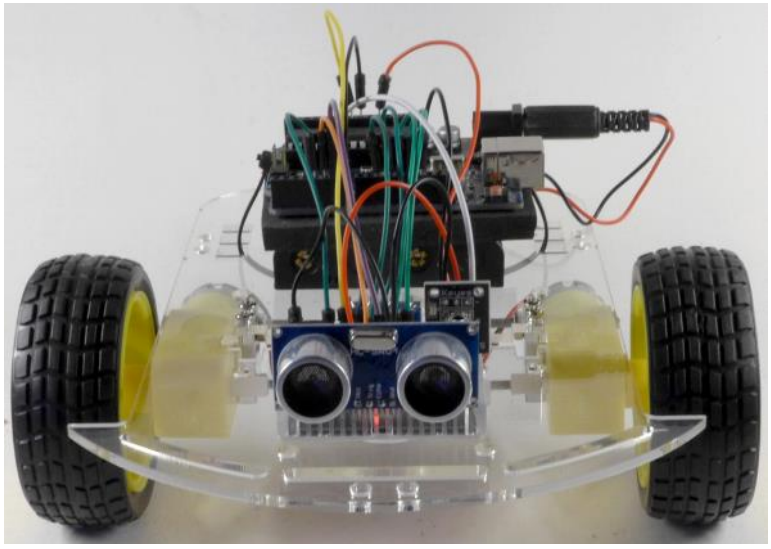
In diesem Kapitel erfahren Sie, ...

- dass in der Welt der Informatik alle Dinge in Klassen eingeteilt werden, die gemeinsame Attribute aufweisen und auf die gleichen Methoden reagieren,
- dass man anhand einer Klassendefinition neue Objekte dieser Klasse erzeugen kann.

Gestatten, mein Name ist Adi!

Stückliste

Arduino UNO, Mini-Breadboard,
9 Kabel m/m 10 cm, 5 Kabel m/m 15 cm.
Bausatz für ein „Smart Car“ mit zwei DC-Getriebemotoren, zwei Rädern, einer Rolle und Batteriebox
DC-Hohlstecker 5,5 mm 2,1 mm
oder Adapter zum Betrieb mit 9V-Block
L9110S-Dual-DC-Motor-Treiber-Board
Piezo-Beeper, am besten die aktive Variante,
Ultraschall-Entfernungsmesser HC-SR04
Infrarot-Fernbedienungs-Kit NEC-kompatibel



Der Zauberlehrling hat einen Besenstiel, Daniel Düsentrieb hat sein Helferlein und was haben Sie? Nichts! Das muss sich ändern! Als krönenden Abschluss unserer Arduino-Serie präsentieren wir hier Adi, den mobilen Roboter.

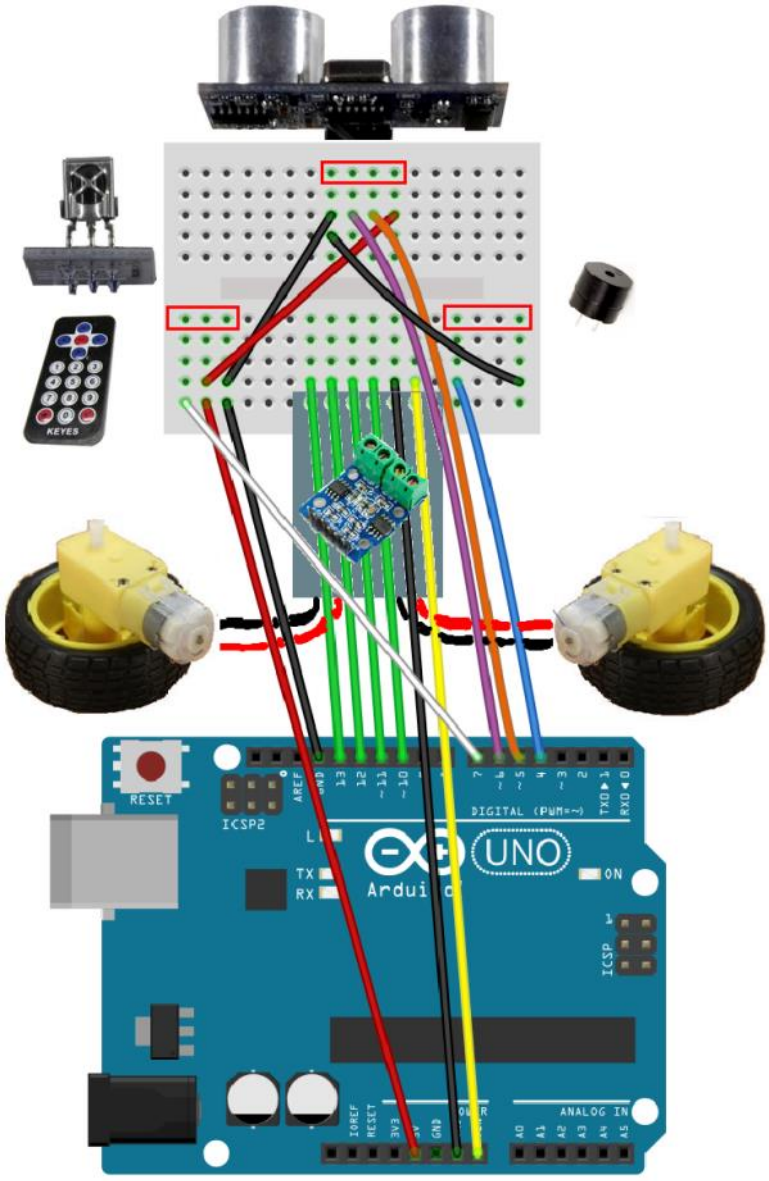
Anders als ein Auto besitzt er nur zwei Räder und ein Stützrad oder eine Gleitrolle. Gelenkt wird er vorläufig dadurch, dass er sich auf der Stelle dreht (ein Rad vorwärts, das andere rückwärts).

Als Energiequelle besitzt Adi einen 4,8V-Batteriepack. Auf diesem sitzt huckepack auf einer isolierenden Unterlage Adis Hirn: der Arduino. Dieser steuert über die Digitalpins 10 und 11 den rechten und über die Digitalpins 12 und 13 den linken Motor (Kabel orange markiert). Dazu brauchen wir den kleinen Motortreiber-Baustein. Er steckt kopfüber im Breadboard und erhält vom Vin-Pin des Arduino (Kabel gelb markiert) die volle Batteriespannung für die beiden Motoren.

Gesteuert wird Adi durch eine kleine Infrarot-Fernbedienung. Der Empfänger ist links in das Breadboard eingesteckt. Ihr Signal geht an Digitalpin 9 (weiße Leitung). Die Spannung bekommt der Empfänger vom 5V-Anschluss des Arduino (Kabel rot markiert). Vorsicht: Nicht alle IR-Empfänger haben dieselbe Pinbelegung!

Als Hindernis-Warngerät besitzt Adi einen Ultraschall-Entfernungsmesser, dessen Triggerpin an Digitalpin 5und dessen Echopin an Digitalpin 6 angeschlossen ist. 5V-Spannung und Masse holen wir vom Infrarotempfänger herüber.

Schließlich ist als Hupe ein Piezo-Beeper oder ein kleiner Computerlautsprecher angeschlossen. Sein Signal bekommt er vom Digitalausgang 4, die Masse vom Ultraschallempfänger.



Klasse Motor

Eine **Klasse** im Sinne der Informationstechnik ist eine Gruppe von realen oder gedachten Gegenständen oder Lebewesen, die gleichartige Eigenschaften aufweisen und die gleiche Funktion haben. So betrachtet sind Elektromotoren eine Klasse.

Wenn wir die Motoren unseres Arduino in Gang setzen wollen, so interessiert uns dabei nicht die Farbe ihres Gehäuses. Auch der Durchmesser der Antriebswelle ist für die Funktion eines Motors nicht weiter relevant. Entscheidend aber sind die Fragen: Wo muss ich das rote und wo das schwarze Kabel anschließen? Wie bringe ich den Motor dazu, dass er sich dreht und wieder anhält?

Die relevanten Informationen hält man in einer **Klassendeklaration**, fest und speichert sie in der Datei 🖫 Motor.h. Ein Motor in unserem Sinne ist danach ein **Objekt**, das seinem Benutzer vier verschiedene **Methoden** (=Befehle) anbietet:
Motor(): Mache dich betriebsbereit. (Die Methode heißt **Konstruktor**.)
rechts(): Drehe die Welle rechtsherum.
links(): Drehe die Welle linksherum.
aus(): Schalte die Drehung aus.

Um seinerseits Befehle an den Arduino weitergeben zu können, muss jedes Objekt sich zwei **Attribute** (=Eigenschaften) merken, nämlich an welchen Pin des Arduino seine rote und an welchen seine schwarze Leitung angeschlossen ist. Für diese beiden Attribute wird hier je eine Integervariable deklariert.

Nun wird die Klasse **implementiert**. Im ersten Abschnitt (Konstruktor) steht hier, was konkret zu tun ist, wenn ein neues Objekt der Klasse Motor erzeugt wird: Die Nummern der beiden Anschlüsse müssen abgespeichert, der pinMode der Anschlüsse muss auf OUTPUT und die Spannung auf LOW gesetzt werden. Danach wird festgehalten, was der Motor bei den weiteren Methodenaufrufen jeweils erledigen soll.

Die eigentliche Programmdatei 🖫 Adi_1.ino bindet zunächst die Datei *Motor.h* ein, um die Klasse verwenden zu können. Dann werden zwei verschiedene Objekte dieser Klasse erzeugt und ihr Konstruktor aufgerufen. Die konkreten Objekte erhalten die Namen MR (rechter Motor) und ML (linker Motor).

In der Funktion *vorwaerts()*, die Ali eine für eine Sekunde geradeaus vorwärts bewegt, können Sie sehen, wie die beiden Objekte verwendet werden: Man gibt den Namen des Objekts an und dann nach einem Punkt den Namen der Methode, die dieses Objekt ausführen soll. Laden Sie 🖫 Adi_1.ino und probieren Sie es aus.

Aufgaben

1. Ergänzen Sie im Programm 🖫 Adi_1 den Code der Funktion *rueckwaerts()*.
2. Ergänzen Sie die Funktion *drehe()* , die eine Drehung Adis um die eigene Achse auslöst. Das Maß der Drehung wird über einen Delay gesteuert. Dieser muss so kalibriert sein, dass etwa die eingegebene Gradzahl herauskommt.
3. Programmieren Sie einen Bewegungsablauf: Eine Sekunde vor, 30 Grad drehen, zwei Sekunden rückwärts, ,...
4. Verbessern Sie anschließend die Funktion drehe() so, dass eine positive Gradzahl eine Rechtsdrehung und eine negative Gradzahl eine Linksdrehung auslöst.
5. Ihre Turtle bekommt eine Hupe. Modellieren Sie dafür das Objekt *Hupe*, indem Sie einen Tab *Hupe.h* ergänzen und in Ihr Programm einbinden.

Adi_1 | Motor.h

```
#include <Arduino.h>

// Deklaration der Klasse Motor
class Motor {
public:
  Motor(int, int);
  void rechts();
  void links();
  void aus();
private:
  int rot, schwarz;
}; //Semikolon nicht vergessen!

// Implementierung der Klasse Motor
Motor::Motor(int r, int s) {
  rot = r;
  schwarz = s;
  pinMode(rot,OUTPUT);
  digitalWrite(rot,LOW);
  pinMode(schwarz,OUTPUT);
  digitalWrite(schwarz,LOW);
}

void Motor::rechts() {
  digitalWrite(rot,HIGH);
  digitalWrite(schwarz,LOW);
}

void Motor::links() {
  digitalWrite(rot,LOW);
  digitalWrite(schwarz,HIGH);
}

void Motor::aus() {
  digitalWrite(rot,LOW);
  digitalWrite(schwarz,LOW);
}
```

Adi_1 | Motor.h

```
#include "Motor.h"
Motor MR = Motor(10,11);
Motor ML = Motor(12,13);

void vorwaerts(int ms) {
  MR.links();
  ML.rechts();
  delay(ms);
  MR.aus();
  ML.aus();
}

void rueckwaerts(int ms) {
//...
}

void drehe(int grad) {
//...
}

void setup() {
vorwaerts(1000);
}

void loop() {
}
```

Adi bekommt Ohren zum Sehen …

Ein halbwegs intelligenter Roboter braucht Sensoren, mit denen er seine Umwelt erfassen kann. Deshalb spendieren wir Adi einen Ultraschall-Entfernungsmesser. Wir verbinden seinen Trig-Pin mit Digitalpin 5und seinen Echo-Pin mit Pin6 des Arduino und schreiben zum Betrieb des Ganzen die Klasse *EMesser*:

Zunächst die Klassendeklaration: Um ein Objekt dieser Klasse zu erzeugen, müssen die beiden Pins angegeben werden, an denen das konkrete Teil angeschlossen ist. Der Konstruktor *EMesser()* muss also zwei Argumente übergeben, die sich das jeweilige Objekt in den privaten Variablen *trigPin* und *echoPin* merkt.

Mitteilen soll uns unser Entfernungsmesser, wie groß der Abstand Adis zu einem vor ihm gelegenen Hindernis ist. Dafür definieren wir die Methode *abstand*(). Bisher waren unsere Methoden meist als *void* deklariert, da sie keinen Ergebniswert liefern, sondern nur weitere Funktionen ausführen sollten. Die Methode *abstand()* soll aber eine Zahl ausgeben. Also deklarieren wir sie mit dem Ausgabewert *long*.

Wie oben bereits beschrieben sendet der E-Messer einen kurzen Schallimpuls, sobald trigPin für 10 Millisekunden auf HIGH gesetzt wird. Dann misst er mit *pulseIn(9)* die Zeit, bis das Echo dieses Impulses vom Hindernis zurückkommt. Wenn wir den Rückgabewert der Funktion *pulseIn*() durch 58 dividieren, erhalten wir den Abstand vom Hindernis in cm. Diesen gibt die Methode *abstand*() mithilfe der return-Anweisung an das aufrufende Programm zurück. Dort kann der Wert an den Seriellen Monitor weitergereicht, in einer Variable abgespeichert oder in eine if-Bedingung eingebaut werden.

Im Hauptprogramm, aus dem wir die Bewegungsbefehle hier der Übersicht halber in die Bibliothek Adi.h ausgelagert haben, muss natürlich

- die E-Messer-Bibliothek eingebunden werden,
- aus der Klasse EMesser das konkrete Objekt EM erzeugt werden,
- die Methode *abstand()* dieses Objekts aufgerufen werden.

Unser realer Entfernungsmesser ist also für Programmierer zu einem Objekt geworden, das er abfragen kann, ohne sich um dessen internen Aufbau zu kümmern, und dessen Code gegenüber dem Rest des Programms in einer eigenen „Kapsel“ eingeschlossen ist.

Adi_2 | Adi.h | EMesser.h | Hupe.h | Motor.h

```
#include <Arduino.h>

//Klassendeklaration
class EMesser {
  public:
    EMesser(int, int);
    long abstand();
  private:
    int trigPin;
    int echoPin;
};

//Implementierung
EMesser::EMesser(int t, int e) {
  trigPin = t;
  echoPin = e;
  pinMode(trigPin,OUTPUT);
  pinMode(echoPin,INPUT);
  digitalWrite(trigPin,LOW);
}

long EMesser::abstand() {
  digitalWrite(trigPin,HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);
  long ms = pulseIn(echoPin,HIGH);
  return ms/58;
}
```

Adi_2 | Adi.h | EMesser.h | Hupe.h | Motor.h

```
#include "Motor.h"
#include "Hupe.h"
#include "EMesser.h"

Motor MR = Motor(10,11);
Motor ML = Motor(12,13);
Hupe  Hl = Hupe(4);
EMesser EM = EMesser(5,6);

#include "Adi.h"
// vorwaerts(), rueckwaerts, drehe()

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

void loop() {
  Serial.println(EM.abstand());
}
```

Aufgaben

1. Fügen Sie Ihrem Programm 🖫 Adi_1.ino die neue Klasse hinzu und speichern Sie das Programm unter 🖫 Adi_2.ino ab. Probieren Sie den Entfernungsmesser aus. Richten Sie Adi auf Hindernisse und kontrollieren Sie, ob der Serielle Monitor den Abstand korrekt angibt.
2. Programmieren Sie Adi neu: Adi soll eine kurze Strecke fahren, dann soll er überprüfen, ob ein Hindernis vor ihm aufgetaucht ist. Wenn ja, dann soll er ein Stück zurückweichen und die Richtung ändern. Wenn nein, dann soll er seinen Weg fortsetzen.

… und Augen zum Hören

Hindernisse auf dem Weg würde ein Mensch sehen. Adis Ultraschallempfänger hört sie. Kommandos dagegen würde ein Mensch meist hören. Adi dagegen sieht sie—mit einem Infrarotsensor.

Das Objekt zur Abfrage der Fernbedienung dürfen Sie sich herunterladen: Steuerung.h. Kopieren Sie die Datei in Ihren Programmordner Adi_2, öffnen Sie das Programm Adi_2.ino, binden Sie Steuerung.h ein und deklarieren Sie das Objekt S1 wie rechts abgebildet, Schreiben Sie in die Loop ein kurzes Hauptprogramm und speichern Sie das Ganze

Aufgaben

1. Halten Sie die Fernbedienung vor eine Digitalkamera. Drücken Sie eine Taste und sehen Sie auf dem Monitor der Kamera das Signal.
2. Starten Sie Adi_3.ino Drücken Sie Tasten der Fernbedienung und beobachten Sie auf dem Seriellen Monitor die Codes. Ergänzen Sie Steuerung.h um eine Liste aller Tastencodes der Fernbedienung.
3. Schreiben Sie ein Programm, das die Hupe betätigt, wenn Sie die Rautetaste (#) drücken.

Die Funktionsweise der command()-Methode werden Sie kaum enträtseln können. Deshalb hier eine Erklärung: Wenn Sie auf irgendeine Taste der Fernbedienung drücken, dann gibt diese zunächst ein Lichtsignal von 9 Millisekunden Dauer ab. Anschließend schaltet sie die Infrarot-LED für 4,5 ms aus. Sobald die *command()*-Methode dieses Signal erkennt, weiß sie: Jetzt kommt der eigentliche Code.

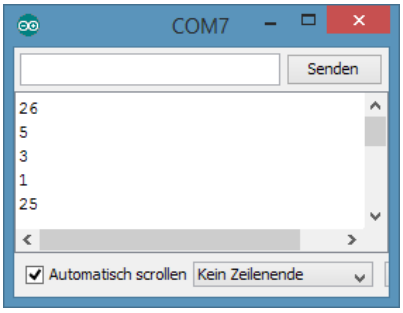
Jeder Code besteht aus der Adresse des angesprochenen Geräts und dem Code der gedrückten Taste. Diese werden vom niedrigsten Bit (LSB, die Einer) bis zum höchsten (MSB, die 128er). Eine 1 besteht dabei aus einem Lichtsignal von 560 Mikrosekunden und einer Pause von 2,5 Millisekunden. Eine Null besteht aus einem Lichtsignal von 560 µs und einer Pause von 2,5 ms. Jedes Byte wird zweimal übertragen: einmal normal und einmal zur Kontrolle invertiert.

Aufgaben

4. An welche Gerätenummer (Address) ist der abgedruckte Code adressiert, welche Kommandonummer (Command) wird übertragen?
5. Steuern Sie Adi über das Cursorkreuz: Ein Schritt vorwärts, Ein Schritt rückwärts, 30° Drehung links, 30°-Drehung rechts.
6. Steuern Sie Adi ohne anzuhalten: Anfahren, Lenken, schneller, langsamer, stop rückwärts,....
7. Steigern Sie Adis Fähigkeiten. Programmieren Sie z. B. Umschaltung auf Autopilot oder automatisches Einparken.

Adi_3 | Adi.h | EMesser.h | Hupe.h | Motor.h | Steuerung.h

```
#include "Motor.h"
#include "Hupe.h"
#include "EMesser.h"
#include "Steuerung.h"


Motor MR = Motor(10,11);
Motor ML = Motor(12,13);
Hupe  Hl = Hupe(4);
EMesser EM1 = EMesser(5,6);
Steuerung S1 = Steuerung(7);


#include "Adi.h"

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

void loop() {
 byte befehl = S1.command();
 if (befehl>0) {Serial.println(befehl);}
}
```



Adi_3L | Adi.h | EMesser.h | Hupe.h | Motor.h | Steuerung.h

```
#include <Arduino.h>

//Klassendeklaration
class Steuerung {
  public:
    Steuerung(int);
    byte command();
  private:
    int pin;
};

//Implementierung
Steuerung::Steuerung(int p) {
  pin = p;
  pinMode(pin,INPUT);
}

byte Steuerung::command() {
  // Diese Funktion
  // - wartet auf ein Signal der Fenbedienung
  // - empfängt es
  // - dekodiert es
  // - liefert den Code der gedrückten Taste
```

Nähreres zu NEC-kompatiblen Infrarotsteuerungen unter http://www.sbprojects.com/knowledge/ir/nec.php

Index