Make:
makezine.com

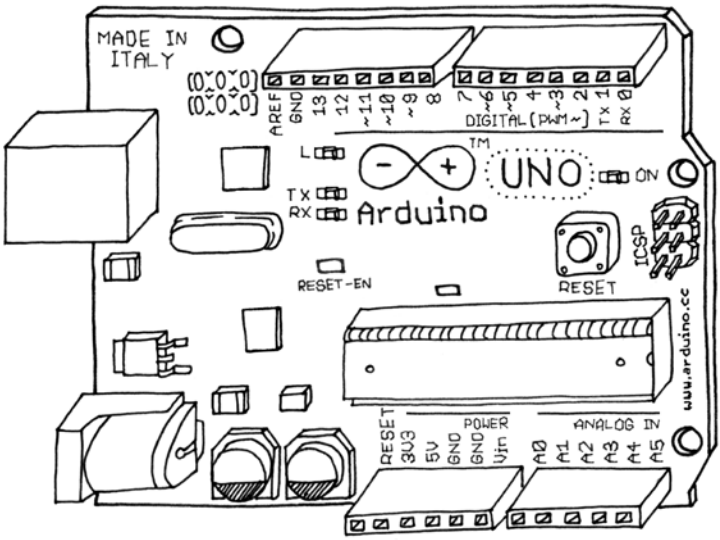
DIE OPEN-SOURCE-ELEKTRONIK-PROTOTYPING-PLATTFORM

Für Arduino 1.0

Arduino für Einsteiger

Massimo Banzi Mitbegründer von Arduino

Übersetzung von Tanja Feder



O'REILLY®

Arduino für Einsteiger

Massimo Banzi

Deutsche Übersetzung von Tanja Feder

O'REILLY®
BEIJING · CAMBRIDGE · FARNHAM · KÖLN · SEBASTOPOL · TOKYO

Kommentare und Fragen können Sie gerne an uns richten:
O'Reilly Verlag GmbH & Co. KG
Balthasarstr. 81
50670 Köln
E-Mail: kommentar@oreilly.de
2012 O'Reilly Verlag GmbH & Co. KG

1. Auflage 2012

Die Originalausgabe erschien 2011 unter dem Titel
Getting Started with Arduino, 2nd Edition bei O'Reilly Media, Inc.

Bibliografische Information Der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über *http://dnb.d-nb.de* abrufbar.

Übersetzung: Tanja Feder, Köln
Lektorat: Volker Bombien, Köln
Korrektorat: Eike Nitz, Köln
Produktion: Karin Driesen, Köln
Umschlaggestaltung: Micheal Oreal, Köln
Satz: Reemers Publishing Services GmbH, Krefeld, www.reemers.de
Druck: Mediaprint, Paderborn

ISBN: 978-3-86899-232-8

Dieses Buch ist auf 100% chlorfrei gebleichtem Papier gedruckt.

Inhalt

Vorwort

Vor einigen Jahren stand ich vor einer sehr interessanten Herausforderung: Ich sollte Designern die einfachsten Grundlagen der Elektronik vermitteln, sodass sie anschließend in der Lage wären, interaktive Prototypen der Objekte, an denen sie gerade arbeiteten, herzustellen.

Ich folgte unterbewusst meinem Instinkt, Elektronik auf die Weise zu lehren, wie ich sie von der Schule her kannte. Später realisierte ich dann, dass das nicht so gut funktionierte, wie ich das gerne gehabt hätte und ich erinnerte mich an die Stunden im Klassenzimmer, in denen jede Menge Theorie ohne praktischen Bezug auf mich eingeprasselt war und ich mich zu Tode gelangweilt hatte.

Eigentlich kannte ich die Elektronik zur Schulzeit bereits, und diese Kenntnisse hatte ich auf sehr empirische Weise erworben: mit herzlich wenig Theorie, aber mit viel praktischer Erfahrung.

Ich begann also, darüber nachzudenken, mittels welcher Prozesse ich Elektronik wirklich verstanden habe:

- Ich nahm alle elektronischen Geräte auseinander, die ich in die Finger bekommen konnte.
- So lernte ich langsam all die einzelnen Komponenten kennen.
- Ich begann, mit ihnen herumzubasteln, einige ihrer inneren Verbindungen zu verändern und dann zu beobachten, wie das Gerät reagierte, üblicherweise mit einer Art von Explosion oder mit einer Rauchwolke.
- Ich baute Bausätze, die Beilagen von Elektrozeitschriften waren, zusammen.
- Ich kombinierte Geräte, die ich gehackt hatte, und zweckentfremdete Bausätze und andere Schaltungen, die ich in Zeitschriften gefunden hatte, um aus ihnen etwas Neues herzustellen.

Als kleines Kind war ich fasziniert davon herauszufinden, wie Dinge funktionieren, daher habe ich sie immer auseinandergebaut. Dieses Interesse wuchs immer weiter, je mehr nicht benutzte elektronische Objekte, derer ich im Haus irgendwie habhaft werden konnte, ich in ihre Einzelteile zer-

legte. Schließlich brachten die Leute alle möglichen Geräte zu mir, damit ich sie auseinandernehmen konnte. Meine größten Objekte zu dieser Zeit waren eine Geschirrspülmaschine und ein früher Computer aus einem Versicherungsbüro, der über einen Drucker, Elektronikkarten, Magnetkartenleser und viele andere Teile verfügte, deren komplette Zerlegung sich als äußerst interessant und knifflig erwies.

Nach umfangreichen Untersuchungen wusste ich, was elektronische Komponenten sind und auch ungefähr, was sie tun. Obendrein war unser Haus voll von elektronischen Zeitschriften, die mein Vater irgendwann Anfang der 1970er gekauft haben musste. Ich habe Stunden damit verbracht, die Artikel zu lesen und mir die Schaltskizzen anzuschauen, ohne allerdings sonderlich viel zu begreifen.

Dieser Prozess des immer wieder erneuten Lesens der Artikel auf der äußerst hilfreichen Grundlage des Wissens, das ich durch das Zerlegen von Schaltungen erworben hatte, erwies sich als langsamer, wirkungsvoller Zyklus.

Ein großer Durchbruch kam an einem Weihnachtstag, als mein Vater mir einen Bausatz schenkte, mit dem Teenagern Wissen über die Elektronik vermittelt werden sollte. Jede Komponente war in einem Plastikwürfel untergebracht, der magnetisch an den anderen Würfeln haften konnte, sodass eine Verbindung entstand. Oben auf diesen Würfeln war das jeweilige elektronische Symbol angeführt. Ich wusste noch wenig davon, dass dieses Spielzeug auch ein Meilenstein von Design Made in Germany war, denn es war bereits in den 1960ern von Dieter Rams entwickelt worden.

Mit diesem neuen Tool konnte ich auf schnelle Weise Schaltkreise zusammenbauen, diese dann ausprobieren und mir das Resultat ansehen. Der Prototyping-Zyklus wurde dabei immer kürzer.

Danach baute ich Radios, Verstärker, Schaltkreise, die fürchterlichen Lärm oder auch schöne Töne produzierten, Regensensoren und kleine Roboter.

Ich habe lange nach einem englischen Begriff gesucht, der diese Arbeitsweise ohne speziellen Plan, bei der einfach von einer bestimmten Idee ausgegangen wird und bei der man bei einem völlig unerwarteten Resultat landet, wiedergibt. Schließlich stieß ich auf das Wort „Tinkering“, das sich etwa mit dem Begriff „Tüfteln“ ins Deutsche übertragen ließe. Ich verstand, wie dieser Begriff in vielen anderen Bereichen verwendet wurde, um eine Arbeitsweise zu beschreiben und Menschen zu porträtieren, die ausgetretene Pfade verlassen und Neuland erkundet hatten. Auch die französischen Regisseure, die die Nouvelle Vague begründeten, wurden im englischen

Sprachraum als Tinkerer bezeichnet. Die beste Definition, die ich kenne, habe ich im Rahmen einer Ausstellung im Exploratorium in San Francisco gefunden:

Tinkering is what happens when you try something you don't quite know how to do, guided by whim, imagination, and curiosity. When you tinker, there are no instructions – but there are also no failures, no right or wrong ways of doing things. It's about figuring out how things work and reworking them.

Contraptions, machines, wildly mismatched objects working in harmony – this is the stuff of tinkering.

Tinkering is, at its most basic, a process that marries play and inquiry..

http://www.exploratorium.edu/tinkering

Von meinen früheren Experimenten wusste ich bereits, wie viel Erfahrung nötig ist, um einen Schaltkreis von den Basiskomponenten aus aufzubauen, der dann auch noch das tut, was Sie möchten.

Ein weiterer Durchbruch erfolgte im Sommer 1982, in dem ich mit meinen Eltern in London viele Stunden lang das Science Museum besichtigte. Hier war gerade ein neuer Bereich eröffnet worden, der Computern gewidmet war und in dem angeleitete Experimente vorgestellt wurden. Indem ich diesen folgte, lernte ich die Grundlagen der Binärmathematik und der Programmierung.

Dabei stellte ich fest, dass bei vielen Anwendungen Ingenieure keine Schaltkreise mehr aus Basiskomponenten bauten, sondern viele intelligente Möglichkeiten mittels Mikroprozessoren in ihre Produkte implementierten. Software ersparte dabei viele Arbeitsstunden beim elektronischen Design und ermöglichte kürzere Zyklen beim Tüfteln.

Nach der Rückkehr begann ich damit, Geld zu sparen, weil ich mir einen Computer kaufen und das Programmieren lernen wollte.

Mein erstes und wichtigstes Objekt war ein brandneuer ZX81-Computer, mit dem ich eine Schweißmaschine steuerte. Das klingt sicher nicht nach einem besonders spannenden Projekt, aber es bestand ein gewisser Bedarf und für mich war es eine große Herausforderung, weil ich gerade erst das Programmieren erlernt hatte. Zu diesem Zeitpunkt wurde mir klar, dass

das Schreiben von Codezeilen weniger zeitaufwendig ist als das Aufbauen komplexer Schaltungen.

Mehr als 20 Jahre später denke ich, dass diese Erfahrung mir ermöglicht, Menschen zu unterrichten, die sich nicht einmal daran erinnern, irgendeine Mathematikstunde besucht zu haben und ihnen die gleiche Begeisterung für das Tüfteln und die entsprechenden Fähigkeiten zu vermitteln, die ich in meiner Jugend erworben und seitdem immer bewahrt habe.

Massimo

Danksagung

Dieses Buch ist Luisa und Alexandra gewidmet.

Zuallererst möchte ich meinen Partnern im Arduino-Team danken: David Cuartielles, David Mellis, Gianluca Martino und Tom Igoe. Jungs, es ist immer wieder eine erstaunliche Erfahrung, mit Euch zu arbeiten!

Als Nächstes möchte ich mich bei Barbara Ghella bedanken. Sie weiß es vielleicht nicht, aber ohne ihre präzise Beratung wären Arduino und dieses Buch vielleicht niemals zustande gekommen.

Ein weiterer Dank gebührt Bill Verplank, der mich weit mehr gelehrt hat als Physical Computing.

Bei Gillian Crampton-Smith möchte ich mich dafür bedanken, dass sie mir eine Chance gegeben hat und für all das, was ich von ihr gelernt habe.

Hernando Barragan möchte ich für die Arbeit beim Verdrahten danken.

Ein Dank gebührt auch Brian Jepson für seine großartige Arbeit als Lektor und für seine unermüdliche enthusiastische Unterstützung.

Mein nächstes Dankeschön gilt Nancy Kotary, Brian Scott, Terry Bronson und Patti Schiendelman dafür, dass sie noch eingepflegt haben, was ich in einem bereits fertigen Buch geändert hatte.

Es wären hier sicherlich noch zahlreiche weitere Menschen zu nennen, aber Brian gibt mir an dieser Stelle zu verstehen, dass ich damit den Rahmen des Buches sprengen würde, daher hier nur eine kleine Liste von Personen, denen ich aus vielerlei Gründen danken möchte:

Adam Somlai-Fisher, Ailadi Cortelletti, Alberto Pezzotti, Alessandro Germinasi, Alessandro Masserdotti, Andrea Piccolo, Anna Capellini, Casey Reas, Chris Anderson, Claudio Moderini, Clementina Coppini, Concetta Capecchi, Csaba Waldhauser, Dario Buzzini, Dario Molinari, Dario Parravicini, Donata Piccolo, Edoardo Brambilla, Elisa Canducci, Fabio Violante, Fabio Zanola, Fabrizio Pignoloni, Flavio Mauri, Francesca Mocellin, Francesco Monico, Giorgio Olivero, Giovanna Gardi, Giovanni Battistini, Heather Martin, Jennifer Bove, Laura Dellamotta, Lorenzo Parravicini, Luca Rocco, Marco Baioni, Marco Eynard, Maria Teresa Longoni, Massimiliano Bolondi,

Matteo Rivolta, Matthias Richter, Maurizio Pirola, Michael Thorpe, Natalia Jordan, Ombretta Banzi, Oreste Banzi, Oscar Zoggia, Pietro Dore, Prof Salvioni, Raffaella Ferrara, Renzo Giusti, Sandi Athanas, Sara Carpentieri, Sigrid Wiederhecker, Stefano Mirti, Ubi De Feo und Veronika Bucko.

1/Einführung

Arduino ist eine Open-Source-Plattform für Physical Computing, die auf einem einfachen Input/Output-(I/O-)Board und einer Enwicklungsumgebung basiert, die die Sprache Processing (**www.processing.org**) implementiert. Mit Arduino lassen sich autonome interaktive Objekte entwickeln, man kann aber auch eine Verbindung mit der Computersoftware (z.B. Flash, Processing, VVVV oder Max/MSP) herstellen. Man kann das Board manuell zusammenbauen oder es auch vormontiert kaufen. Die Open-Source-IDE (Integrated Development Environment, Integrierte Entwicklungsumgebung) steht unter **www.arduino.cc** zum kostenlosen Download bereit.

Arduino unterscheidet sich durch folgende Features von andern Plattformen auf dem Markt:

- Arduino ist betriebssystemunabhängig und kann unter Windows, Macintosh und Linux betrieben werden.

- Arduino basiert auf der Progammier-IDE Processing, einer leicht handhabbaren Entwicklungsumgebung, die von Künstlern und Designern verwendet wird.

- Arduino wird über ein USB-Kabel und nicht über einen seriellen Anschluss programmiert. Dieses Feature ist äußerst nützlich, da viele moderne Computer keine seriellen Anschlüsse haben.

- Es handelt sich um eine Open-Source-Software. Sie können Schaltdiagramme herunterladen, alle betreffenden Komponenten kaufen oder eigene herstellen, ohne dass Sie dafür etwas an die Entwickler von Arduino zahlen müssten.

- Die Hardware ist preisgünstig. Der Preis für das USB-Board liegt bei etwa 25 Euro. Das Ersetzen von durchgebrannten Chips auf dem Board ist einfach und kostet nicht mehr als 5 Euro oder 4 US-Dollar. Man kann es sich also leisten, auch mal einen Fehler zu machen.

- Es gibt eine entsprechende Community, sodass man bei vielen anderen Nutzern Hilfe finden kann.

» Das Arduino-Projekt wurde in einer Lernumgebung entwickelt und eignet sich daher bestens für Einsteiger, die Dinge schnell ans Laufen bringen möchten.

Dieses Buch ist dazu gedacht, Einsteigern ein Verständnis von den Vorteilen zu vermitteln, die sie durch das Erlernen der Handhabung der Arduino-Plattform und das Übernehmen der entsprechenden Philosophie gewinnen.

An wen sich das Buch richtet

Dieses Buch ist für die ursprünglichen Arduino-Benutzer geschrieben: Designer und Künstler. Daher werden Dinge in einer Weise erklärt, die einigen Ingenieuren möglicherweise die Haare zu Berge stehen lassen wird. Einer nannte die einleitenden Kapitel meines ersten Entwurfs sogar Staubflocken. Das ist genau der Punkt, denn seien wir mal ehrlich: Die meisten Ingenieure können anderen Ingenieuren und erst recht fachfremden Personen, das, was sie tun, nicht erklären. Wir wollen nun tief in diese „Staubflocken" eintauchen.

Hinweis: Arduino baut auf der Diplomarbeit von Hernando Barragan auf, bei der er an der Wiring-Plattform arbeitete, während er unter Casey Reas und mir am IDII Ivrea studierte.

Nachdem Arduino allmählich beliebt wurde, sah ich, wie Experimentatoren, Hobbybastler und alle Arten von Hackern schöne und verrückte Objekte herstellten. Ich erkannte, dass Ihr selbst alle Künstler und Designer seid. Daher ist dieses Buch Euch gewidmet.

Arduino wurde aus der Idee geboren, Interaction Design, eine Designdisziplin, bei der das Prototyping im Zentrum der Methodik steht, zu vermitteln. Es gibt viele Definitionen für den Begriff Interaction Design, ich bevorzuge die Folgende:

Interaction Design ist das Design einer jeglichen interaktiven Erfahrung.

In unserer heutigen Welt befasst sich Interaction Design mit dem Erzeugen bedeutsamer Erfahrungen zwischen uns (Menschen) und Objekten. Dies ist eine gute Methode, um das Entstehen von schönen – und vielleicht auch kontroversen – Erfahrungen im Umgang mit der Technik zu erschließen. Beim Interaction Design erfolgt Design in einem Iterationsprozess, basierend auf Prototypen und mit ständig wachsender Präzision. Dieser Ansatz – der teilweise auch beim konventionellen Design zu finden ist – kann so

erweitert werden, dass Prototyping in die Technologie eingebunden wird, speziell im Bereich Elektronik.

Der spezielle Bereich von Interaction Design, der bei Arduino zum Tragen kommt, ist das Physical Computing (oder auch Physical Interaction Design).

Was ist Physical Computing?

Beim Physical Computing wird Elektronik verwendet, um Prototypen von neuen Arbeitsmaterialien für Designer und Künstler herzustellen.

Dies umfasst auch das Design von interaktiven Objekten, die über Sensoren und Aktoren, die mittels einer vorgegebenen Verhaltensweise gesteuert werden, mit den Menschen kommunizieren können. Diese Verhaltensweise ist als Software implementiert, die in einem Mikrocontroller (ein kleiner Computer auf einem einzelnen Chip) ausgeführt wird.

In der Vergangenheit bedeutete der Einsatz von Elektronik gleichzeitig das ständige Arbeiten mit Ingenieuren und gleichzeitig den langwierigen Aufbau von Schaltkreisen, immer eine kleine Komponente nach der anderen. Dadurch wurden kreative Menschen daran gehindert, mit dem Medium direkt zu experimentieren. Die meistens Tools waren für Ingenieure gedacht und setzten ein erhebliches Wissen voraus. In den letzten Jahren wurden Mikrocontroller billiger und einfacher in der Handhabung, wodurch die Herstellung besserer Tools ermöglicht wurde.

Der Fortschritt, der mit Arduino erfolgte, bestand darin, dass diese neuen Tools Neulingen nähergebracht wurden, sodass es ihnen möglich wurde, nach nur einem zwei- oder dreitägigen Workshop bereits irgendwelche Dinge zu bauen.

Mit Arduino können sich Designer und Künstler die Grundlagen von Elektronik und von Sensoren sehr schnell aneignen und ohne große Investitionen mit dem Bau von Prototypen beginnen.

2/Die Philosophie von Arduino

Die Philosophie von Arduino besteht darin, Designs zu erstellen, anstatt über sie zu sprechen. Sie besteht in einem andauernden Suchen nach schnelleren und leistsungsstärkeren Möglichkeiten, um bessere Prototypen zu bauen. Wir haben viele Prototyping-Techniken erkundet und so quasi mit unseren Händen neue Denkansätze geschaffen.

Das klassische Engineering beruht auf einem strikten Prozess, der von A nach B führt; bei Arduino besteht der Spaß in der Möglichkeit, auf diesem Weg verlorenzugehen und stattdessen bei C zu landen.

Dies ist der Prozess des Tüftelns, den wir so lieb gewonnen haben – genzenlos mit einem Medium herumexperimentieren und dabei das Unerwartete entdecken. Bei dieser Suche nach Wegen, bessere Prototypen herzustellen, haben wir auch eine Reihe von Softwarepaketen ausgewählt, die einen Prozess der ständigen Veränderung des Software- oder des Hardwaremediums ermöglichen.

In den nächsten Abschnitten werden einige philosophische Aspekte, Ereignisse und Pioniere vorgestellt, durch die die Philosophie von Arduino inspiriert wurde.

Prototyping

Prototyping ist das Herzstück der Arduino-Philosophie: Wir stellen Dinge her und bauen Objekte, die mit anderen Objekten, Menschen oder Netzwerken interagieren. Wir sind bestrebt, einen einfacheren und schnelleren Weg für das Prototyping zu finden, der außerdem möglichst kostengünstig sein soll.

Viele Neulinge gehen zunächst mit der Vorstellung an Elektronik heran, dass sie lernen müssen, alles von Grund auf selbst zu bauen. Das ist reine Energieverschwendung: Was man wirklich möchte, ist die Bestätigung, dass etwas sehr schnell funktioniert, sodass man selbst motiviert ist, den nächsten Schritt zu unternehmen, oder dass man sogar jemand anderen motiviert, entsprechend goßzügig in einen selbst zu investieren.

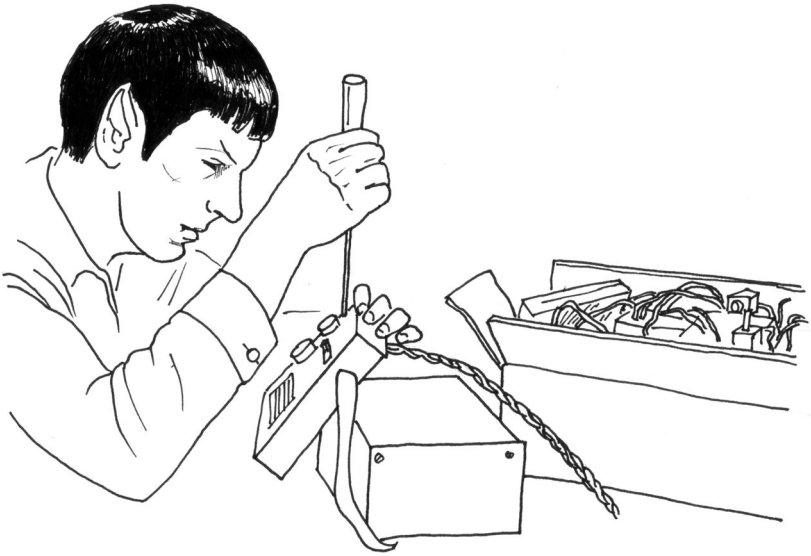
Daher haben wir das »Opportunistische Prototyping« entwickelt: Warum sollten wir Zeit und Energie darauf verschwenden, Dinge von Grund auf zu bauen, ein Prozess, der viel Zeit und tiefgehendes technisches Wissen erfordert, wenn wir fertige Geräte hacken und so die harte Arbeit nutzen können, die von großen Unternehmen und fähigen Ingenieuren bereits getan wurde?

Unser Held ist James Dyson, der 5127 Prototypen seines Vakuumstaubsaugers baute, bevor er mit dem Resultat zufrieden war (**http://www.international.dyson.com/jd/1947.asp**).

Tüfteln

Wir glauben, dass es essentiell ist, mit Technologie herumzuexperimentieren und verschiedene Möglicheiten direkt mit der Hard- oder Software auszuprobieren – manchmal ohne dabei ein wirklich definiertes Ziel zu haben.

Das Verwerten von bereits vorhandener Technologie ist eine der besten Möglichkeiten beim Tüfteln. Durch das Sammeln und Hacken von billigem Spielzeug und alten, ausgemusterten Geräten lassen sich tolle Resultate erzielen.

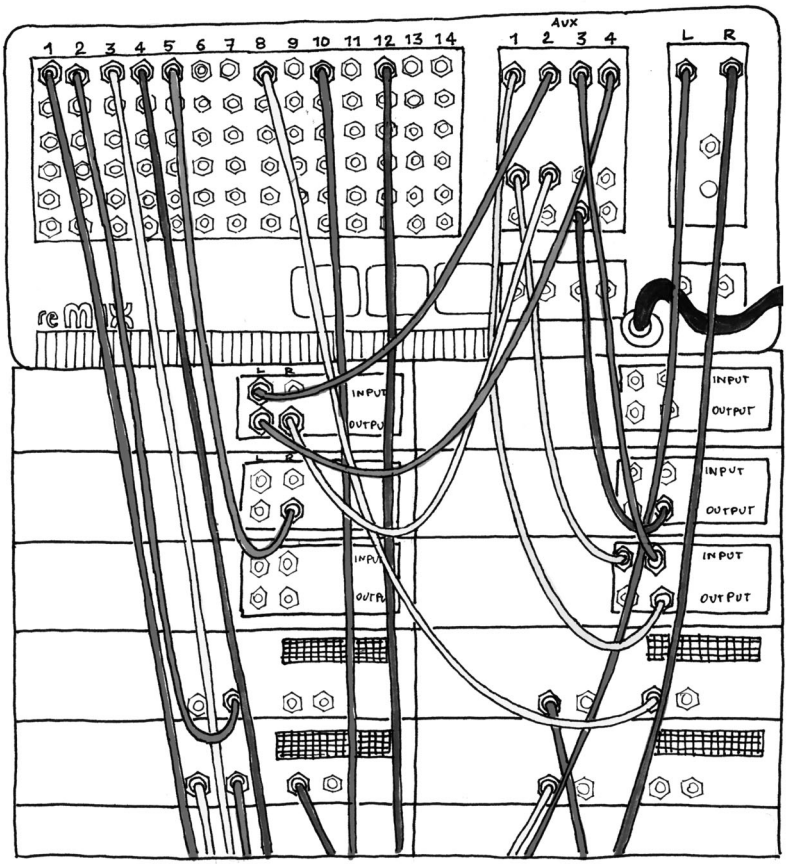


Patching

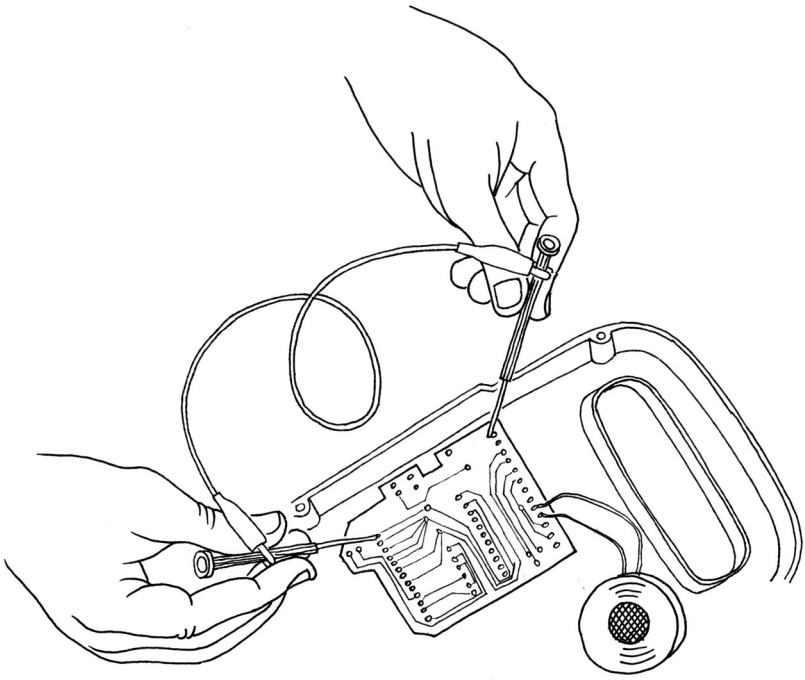
Ich war immer fasziniert vom Baukastenprinzip und von der Möglichkeit, durch das Verbinden einfacher Geräte komplexe Systeme aufzubauen. Dieses System wird sehr gut durch Robert Moog und seine analogen Synthesizer repräsentiert. Musiker erzeugten Sounds, indem sie verschiedene Module mit Kabeln zusammenschusterten und so unzählige Kombinationen herstellten. Durch diesen Ansatz hatten Synthesizer oft das Aussehen von alten Telefon-Switchboards, die aber mit zahlreichen Finessen ausgestattet waren und so eine perfekte Plattform für das Herumexperimentieren mit Sound und für musikalische Innovationen darstellten. Moog beschrieb dies als einen Prozess zwischen Beobachten und Entdecken. Ich bin sicher, dass die Musiker zu Beginn nicht wussten, wozu all die Hunderte von Knöpfen dienten, aber sie experimentierten unaufhörlich und entwickelten ihren Stil ständig weiter, ohne Unterbrechung dieses Prozesses.

Das Reduzieren der Anzahl an Unterbrechungen im Prozess ist sehr wichtig für die Kreativität – je nahtloser der Prozess ist, desto besser funktioniert das Tüfteln.

Diese Technik wurde in Form von Entwicklungsumgebungen für das visuelle Programmieren wie Max, Pure Data oder VVVV in die Welt der Software übertragen. Diese Tools lassen sich als Behälter für verschiedene Funktionen, die sie bereitstellen, visualisieren, mit denen der Nutzer dann Patches erstellt, indem er diese Behälter verbindet. Diese Umgebung bietet dem Nutzer die Möglichkeit, mit der Programmierung zu experimentieren, ohne dabei ständig den Zyklus aus Programmeingabe, Kompilierung, „Verdammt – da ist ein Fehler“, Fehlerbehebung, Kompilierung und schließlich Programmausführung zu unterbrechen. Wenn Sie also eher visuell orientiert sind, empfehle ich Ihnen, solche Entwicklungsumgebungen auszuprobieren.

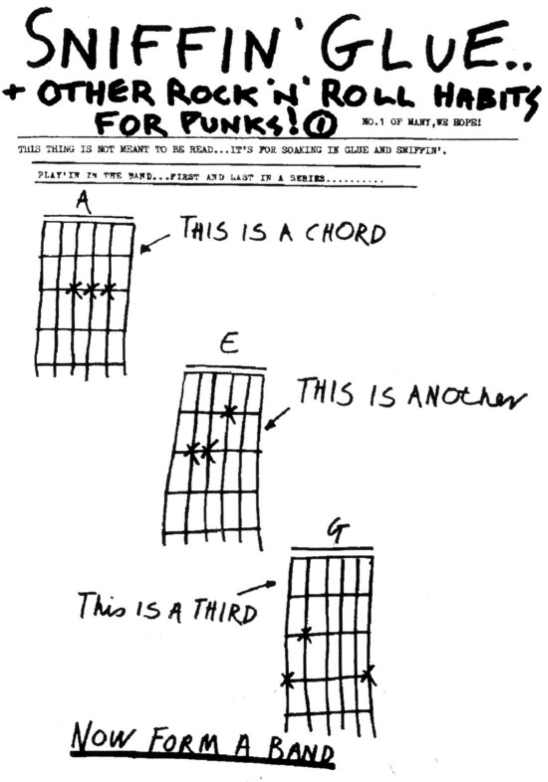

1 2 3 4 5 6 7 8 9 10 11 12 13 14
AUX
1 2 3 4
L R
remix
INPUT
OUTPUT
INPUT
OUTPUT
INPUT
OUTPUT
INPUT
OUTPUT

Modifizieren von Schaltkreisen



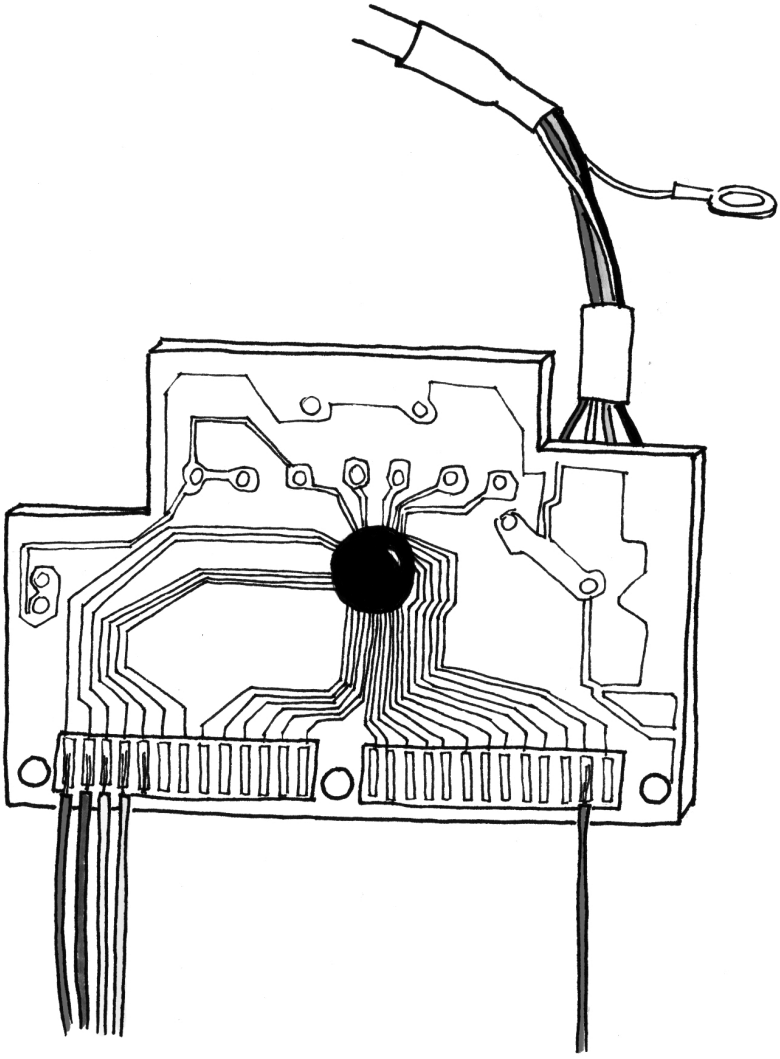
Circuit Bending ist eine der interessantesten Formen des Tüftelns. Es handelt sich um das kreative Kurzschließen von akustischen Geräten, die mit einer Niederspannungsbatterie betrieben werden, z. B. Pedale für Gitarreneffekt, Kinderspielzeug und kleine Synthesizer, um neue Musikinstrumente und Schallgeber zu kreieren. Das Herzstück dieses Prozesses ist die „Kunst des Zufalls“. Es begann 1966, als Reed Ghazala zufällig einen Spielzeugverstärker an einem Metallobjekt an seiner Schreibtischschublade kurzschloss, was in einer Flut von ungewöhnlichen Tönen resultierte. Was ich am Circuit Bending mag, ist die Möglichkeit, durch das Zweckentfremden oder Modifizieren von Technologie die wildesten Geräte zu

erschaffen, ohne dass dazu notwendigerweise ein Verständnis erforderlich wäre, wie sie rein theoretisch funktionieren.



Es ist ein wenig wie beim Fanzine *Sniffin Glue*, aus dem hier ein Auszug zu sehen ist: Während der Punk-Ära war die Kenntnis von drei Gitarrenakkorden ausreichend, um eine Band zu gründen. Lassen Sie nicht zu, dass Experten in einem bestimmten Bereich Ihnen vermitteln, dass Sie niemals zu ihnen gehören werden. Ignorieren Sie sie und überraschen sie dann.

Keyboard-Hacks



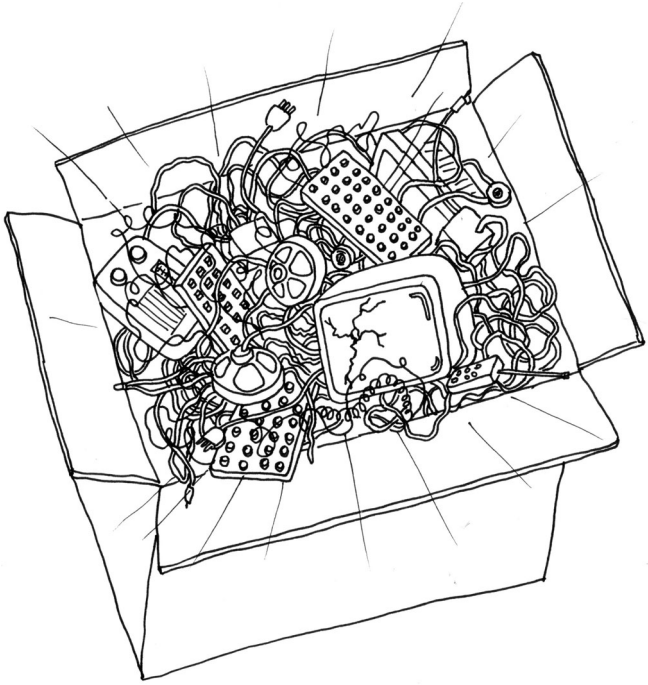
Nach mehr als 60 Jahren sind Computertastaturen immer noch der am weitesten verbreitete Weg, mit dem Computer zu interagieren. Alex Pentland, der akademische Leiter des MIT Media Laboratory, bemerkte einst Folgendes: „Entschuldigen Sie die Ausdrucksweise, aber Männerurinale sind intelligenter als Computer. Computer sind von ihrer Umgebung isoliert.“ [1]

Als Tüftler können wir neue Wege für die Interaktion mit dem Computer einführen, indem wir die Tasten durch Bauteile ersetzen, die ihre Umgebung sensorisch erfassen. Beim Auseinandernehmen der Computertastatur offenbart sich ein sehr einfaches (und preiswertes) Gerät. Das Kernstück besteht in einem kleinen Brett. Dabei handelt es sich üblicherweise um einen Schaltkreis in einem hässlichen Grün oder in Braun, mit zwei Gruppen von Kontakten, die zu zwei Plastikschichten führen, die die Verbindungen zwischen den einzelnen Tasten beherbergen. Wenn Sie den Schaltkreis entfernen und zwei Kontake mit einem Draht verbinden, erscheint ein ganzer Roman auf dem Computerbildschirm. Wenn Sie nun einen Bewegungssensor kaufen und mit der Tastatur verbinden, sehen Sie, dass jedes Mal, wenn jemand vor dem Computer herläuft, eine Taste gedrückt wird. In Verbindung mit der bevorzugten Software wird Ihr Computer so intelligent wie ein Urinal. Keyboard-Hacking ist eine Schlüsseldisziplin beim Prototyping und beim Pysical Computing.

1 Sara Reese Hedberg, MIT Media Lab's quest for perceptive computers, *Intelligent Systems and Their Applications*, IEEE, Jul/Aug 1998.

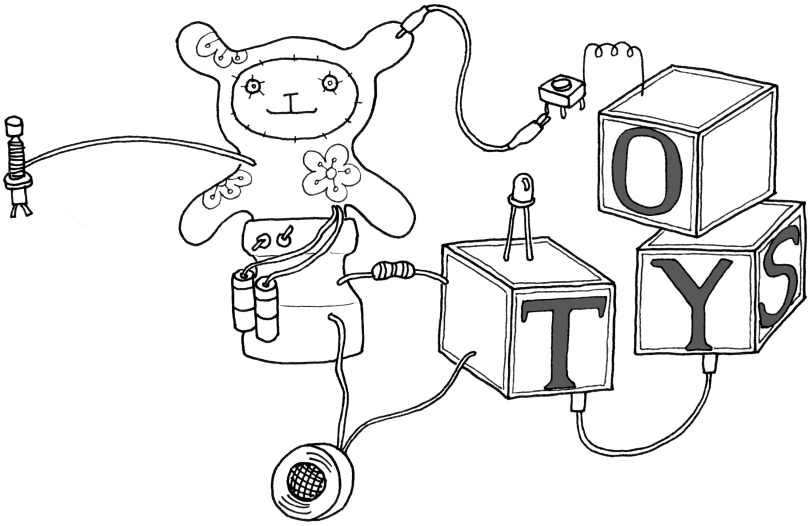
Wir lieben Elektroschrott

Die Leute werfen heutzutage viel Technologie weg: alte Drucker, Computer, seltsame Büromaschinen, technisches Equipment und sogar vieles aus dem militärischen Bereich. Es gab schon immer einen Markt für diese ausgemusterte Technologie, besonders bei jungen und/oder wenig vermögenden Hackern oder Geeks, die erst noch Hacker werden wollen. Dieser Markt wurde augenfällig in Ivrea, wo wir Arduino entwickelt haben. Die Stadt war Hauptsitz des Unternehmens Olivetti, das seit den 1960ern Computer hergestellt hatte. Mitte der 90er entsorgte die Firma alles auf Schrottplätzen in der Gegend, unter anderem ganze Computerteile, elektronische Komponenten und seltsame Geräte aller Art. Wir verbrachten unzählige Stunden auf diesen Schrottplätzen, kauften für kleines Geld verschiedene Apparate und hackten uns in unsere Prototypen. Wenn man Tausende von Lautsprechern für wenig Geld kaufen kann, drängt sich einem schließlich folgende Idee auf: Elektroschrott zu sammeln und ihn durchzuschauen, bevor man etwas von Grund auf neu baut.



Hacken von Spielzeug

Spielzeug ist eine fantastische Quelle für billige Technologie zum Hacken und Weiterverwenden, wie es auch schon vorher beim Circuit Bending angeführt wurde. Mit der aktuellen Flut Tausender billiger Spielzeuge aus China kann man auf schnelle Weise kleine Ideen mit beispielsweise wenigen Spielzeugkatzen, die Geräusche von sich geben, und einigen Laserschwertern umsetzen. Ich habe das einige Jahre lang getan, um meinen Studenten zu zeigen, dass Technologie nichts Furchteinflößendes hat und auch nicht schwer zu begreifen ist. Eine meiner liebsten Quellen ist das Buch *Low Tech Sensors and Actuators* von Usman Haque und Adam Somlai-Fischer (**http://lowtech.propositions.org.uk**). Ich denke, dass besagte Technik in diesem Handbuch perfekt beschrieben wurde, und ich verwende sie schon seit jeher.



Kooperation

Die Kooperation von Nutzern ist eines der Schlüsselprinzipien der Arduino-Welt – über das entsprechende Forum unter **http://www.arduino.cc** helfen sich Menschen aus aller Welt gegenseitig beim Erkunden der Plattform. Das Arduino-Team ermutigt Menschen dazu, auf lokaler Ebene zusammenzuarbeiten, hilft ihnen aber auch dabei, Benutzergruppen in jeder Stadt, die sie besuchen, zu gründen. Wir haben auch einen , wie er bei Wiki genannt wird, Playground eingerichtet (**http://www.arduino.cc/playground**), auf dem Benutzer ihre Erkenntnisse dokumentieren können. Es ist wirklich erstaunlich, wie viele Informationen diese Leute im Web bereitstellen, sodass sie sich jeder zunutze machen kann. Diese Kultur des Teilens und gegenseitigen Helfens ist einer der Aspekte, der mich im Hinblick auf Arduino am meisten mit Stolz erfüllt.

3/Die Arduino-Plattform

Arduino besteht aus zwei Hauptteilen: dem Arduino-Board, d. h. der Hardware, mit der man arbeitet, wenn man seine Objekte herstellt, und der Arduino-IDE, also der Software, die man auf seinem Computer ausführt. Mit der IDE kann man einen Sketch (ein kleines Computerprogramm) erstellen, der dann auf das Arduino-Board übertragen wird. Der Sketch übermittelt dem Board, was zu tun ist.

Es ist noch gar nicht lange her, da bedeutete die Arbeit mit Hardware das Aufbauen von Schaltkreisen mit Hunderten von verschiedenen Komponenten mit seltsamen Namen wie Widerstand, Kondensator, Induktor, Transistor usw. – und das von Grund auf.

Jeder Schaltkreis war für eine bestimmte Verwendung „verdrahtet“ und Änderungen waren mit dem Zuschneiden von Verbindungsdrähten, dem Herstellen von Lötverbindungen und weiteren Arbeitsschritten verbunden.

Mit dem Aufkommen von digitalen Technologien und Mikroprozessoren wurden Funktionen, die vorher über eine entsprechende Verdrahtung implementiert wurden, nun mittels Softwareprogrammen umgesetzt.

Software lässt sich leichter modifizieren als Hardware. Mit wenigen Tastatureingaben kann die Logik eines Bauteils oder Geräts radikal geändert werden und es können zwei oder drei Versionen mit demselben Zeitaufwand ausprobiert werden, der für das Löten von ein paar Widerständen erforderlich wäre.

Die Arduino-Hardware

Beim Arduino-Board handelt es sich um ein kleines Mikrocontroller-Board, d.h. einen kleinen Schaltkreis (das Board), der einen kompletten Computer auf einem kleinen Chip (der Mikrocontroller) enthält. Dieser Computer verfügt über eine tausendfach geringere Leistungsfähigkeit als das MacBook, mit dem ich dieses Buch schreibe, ist aber wesentlich billiger und sehr nützlich, wenn es darum geht, interessante Geräte herzustellen. Schauen Sie sich Ihr Arduino-Board einmal an: Sie können einen Chip mit

28 Beinchen erkennen – dieser Chip ist vom Typ ATmega328 und das Herzstück Ihres Boards.

Wir (das Arduino-Team) haben auf diesem Board alle Komponenten platziert, die ein Mikrocontroller für eine einwandfreie Funktionsweise und für die Kommunikation mit dem Computer benötigt. Es gibt viele Versionen dieses Boards; wir werden im gesamten Buch Arduino Uno verwenden, das im Hinblick auf seine Handhabung am einfachsten ist und auf dem man am besten lernen kann. Die Anweisungen gelten für frühere Versionen des Boards, einschließlich des Arduino Duemilanove von 2009. Abbildung 3-1 zeigt das Arduino Uno; in Abbildung 3-2 ist das Arduino Duemilanove dargestellt.

In beiden Abbildungen sehen Sie das Arduino-Board. Auf den ersten Blick sind all diese Anschlüsse möglicherweise ein wenig verwirrend. Hier eine Erläuterung der Funktion eines jeden Elements auf dem Board:

14 digitale I/O-Pins (Pins 0-13)
Hierbei kann es sich um Eingangs- oder Ausgangspins handeln, abhängig davon, wie sie im betreffenden Sketch definiert wurden.

6 analoge In-Pins (Pins 0-5)
Diese dedizierten analogen Eingangspins übernehmen analoge Werte (z.B. elektrische Spannung, die von einem Sensor gemessen und dann ausgelesen wurde) und wandeln sie in eine Zahl zwischen 0 und 1023 um.

6 analoge Ausgangspins (Pin 3,5,6,9,10 und 11)
Hierbei handelt es sich eigentlich um sechs der digitalen Pins, die mittels eines Sketches, den Sie in der IDE erstellen, als analoge Ausgangspins programmiert werden können.

Das Board kann mit dem USB-Anschluss Ihres Computers (meistens sind das USB-Stecker) oder einem AC-Adapter (9 Volt, 2,1-mm-Klinkenstecker, Zentrum positiv) mit Strom versorgt werden. Wenn keine Verbindung zur Steckdose besteht, erfolgt die Stromzufuhr über das USB-Board. Sobald aber eine Verbindung zur Steckdose hergestellt wird, wird sie auch automatisch vom Board genutzt.

Hinweis: Wenn Sie das ältere Arduino-NG oder Arduino Diecimila nutzen, müssen Sie den Power Selection Jumper (auf dem Board mit PWR_Sel gekennzeichnet) für eine entsprechende Stromversorgung entweder auf EXT (extern) oder USB einstellen. Den Jumper finden Sie zwischen dem Anschluss für den AC-Adapter und dem USB-Anschluss.

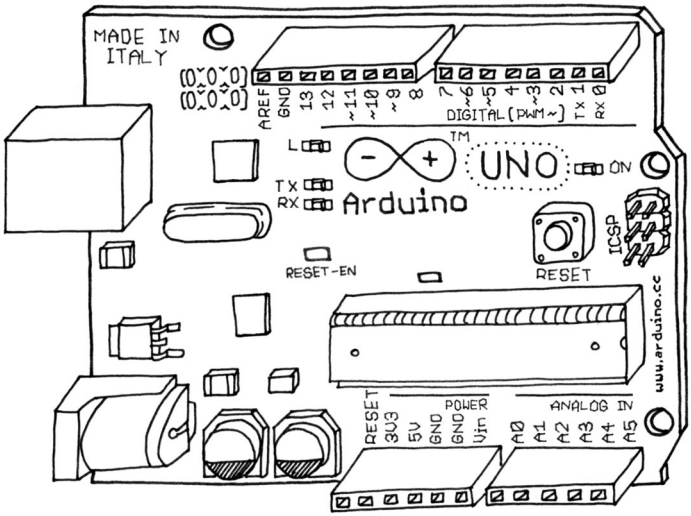


Abbildung 3-1.
Das Arduino Uno-Board

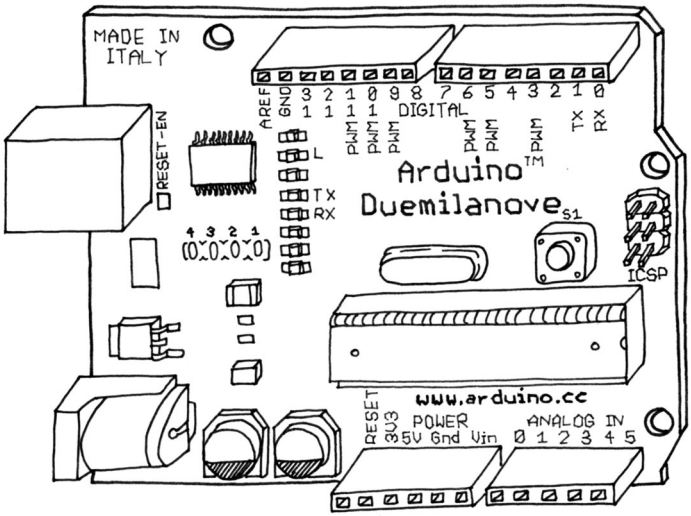


Abbildung 3-2.
Das Arduino Duemilanove-Board

Die Software (IDE)

Die IDE (Integrierte Entwicklungsumgebung) ist ein spezielles Programm, das auf Ihrem Computer ausgeführt wird und Ihnen ermöglicht, Sketches für Ihr Arduino-Board in einer einfachen Sprache zu schreiben, die auf der Sprache Processing (**http://www.processing.org**) basiert. Die Magie offenbahrt sich, wenn Sie die entsprechende Schaltfläche drücken, um den Sketch auf das Board zu laden: Der Code, den Sie geschrieben haben, wird in die Sprache C übersetzt (die im Allgemeinen für einen Anfänger recht schwierig zu verstehen ist) und dann zum avr-gcc-Compiler übertragen, bei dem es sich um einen sehr wichtigen Bestandteil von Open-Source-Software handelt und der die endgültige Übersetzung in die Sprache, die vom Mikrocontroller verstanden wird, vornimmt. Dieser letzte Schritt ist sehr wichtig, da genau an dieser Stelle Arduino die Sache erheblich erleichtert, indem die Komplexität, die mit der Programmierung von Mikrocontrollern verbunden ist, in größtmöglichem Maße ausgeblendet wird.

Die Programmierung von Arduino umfasst folgende Schritte:

- Das Board an einen USB-Anschluss Ihres Computers anschließen.
- Einen Sketch schreiben, der dem Board Leben einhaucht.
- Den Sketch über die USB-Verbindung auf Ihr Board laden und einige Sekunden auf den Restart des Boards warten.
- Das Board führt den von Ihnen geschriebenen Sketch aus.

Hinweis: Die Installation unter Linux ist zum Zeitpunkt der Entstehung dieses Buches sehr kompliziert. Entsprechende Anweisungen finden Sie unter http://www.arduino.cc/playground/Learning/Linux.

Die Installation von Arduino auf dem Computer

Um das Arduino-Board zu programmieren, müssen Sie zunächst die Entwicklungsumgebung (die IDE) herunterladen. Diese finden Sie unter **http://www.arduino.cc/en/Main/Software**. Wählen Sie die für Ihr Betriebssystem passende Version.

Laden Sie die Datei herunter und öffnen Sie sie mit einem Doppelklick. Daraufhin wird ein Ordner mit dem Namen *arduino-[Version]* erstellt, z.B. *arduino-1.0*. Ziehen Sie den Ordner an die gewünschte Stelle, z.B. auf die Benutzeroberfläche, in den Ordner *Program Files* (unter Windows) usw.

Auf einem Mac wird durch den Doppelklick ein Diskettensymbol mit einer Arduino-Anwendung aufgerufen (ziehen Sie es in den Ordner *Applications*). Wenn Sie nun die Arduino-IDE ausführen möchten, öffnen Sie hierzu den Ordner *arduino* (Windows oder Linux) oder den Ordner *Applications* (Mac) und fahren dann mit einem Doppelklick auf das Arduino-Symbol fort. Führen Sie dies aber an dieser Stelle noch nicht aus, weil noch ein weiterer Schritt erforderlich ist.

Hinweis: Falls Probleme bei der Ausführung der Arduino-IDE auftreten, schauen Sie sich das Kapitel 7 an.

Nun müssen Sie die Treiber installieren, die Ihrem Computer ermöglichen, via USB-Anschluss mit Ihrem Board zu kommunizieren.

Installation der Treiber unter Macintosh

Wenn das Arduino Uno mit einem Mac betrieben wird, werden die vom Betriebssystem bereitgestellten Treiber verwendet. Die Prozedur ist also recht simpel. Sie müssen einfach das Board an den Computer anschließen.

Das PWR-Lämpchen sollte nun leuchten und die gelbe LED mit der Bezeichnung L sollte zu blinken beginnen.

Möglicherweise wird Ihnen ein Popup-Fenster angezeigt, das Sie darüber informiert, dass eine neue Netzwerkschnittstelle gefunden wurde.

Klicken Sie in diesem Fall auf *Network Preferences* und im dann aufgerufenen Dialogfenster auf *Apply*. Das Uno-Board wird als *Not Configured* angezeigt, es funktioniert aber einwandfrei. Schließen Sie dann das Fenster *System Preferences*.

Falls Sie ein älteres Arduino-Board besitzen, finden Sie hier entsprechende Anweisungen: **http://www.arduino.cc/en/Guide/MacOSX**.

Falls das Arduino-Board nicht funktioniert, schauen Sie sich das Kapitel 7 an.

Installation der Treiber unter Windows

Schließen Sie das Arduino-Board an Ihrem Computer an. Wenn der Found New Hardware-Wizard angezeigt wird, versucht Windows, die Treiber zunächst auf der Windows Update-Seite zu finden.

Bei Windows XP werden Sie gefragt, ob Windows Update durchsucht werden soll. Wenn Sie Windows Update nicht nutzen möchten, wählen Sie die Option *No, not at this time* aus und klicken Sie auf *Next*.

Wählen Sie im nächsten Bildschirm *Install from a list or specific location* aus und klicken Sie auf *Next*.

Navigieren Sie zur Treiberdatei für das Uno-Board, die *ArduinoUNO.inf* heißt und sich im Ordner *Drivers* (nicht zu verwechseln mit dem Unterverzeichnis *FTDI USB Drivers*) der heruntergeladenen Arduino-Software befindet, und wählen Sie sie hier entsprechend aus. Von da an wird Windows die Installation zu Ende bringen.

Falls Sie ein älteres Board haben, finden Sie hier entsprechende Anweisungen: **http://www.arduino.cc/en/Guide/Windows**.

Sobald die Treiber installiert sind, kann die Arduino-IDE gestartet und Arduino genutzt werden.

Als Nächstes müssen Sie herausfinden, welcher serielle Anschluss Ihrem Arduino-Board zugewiesen ist – diese Information ist für dessen spätere Programmierung erforderlich. Wie Sie an diese Informationen gelangen, wird in den folgenden Abschnitten beschrieben.

Port-Identifikation unter Macintosh

Wählen Sie im Menü *Tools* der Arduino-IDE die Option Serial Port und hier den Anschluss, der mit */dev/cu.usbmodem* beginnt, aus. Mit diesem Namen referenziert Ihr Computer das Arduino-Board. In Abbildung 3-3 wird eine Liste von Anschlüssen gezeigt.

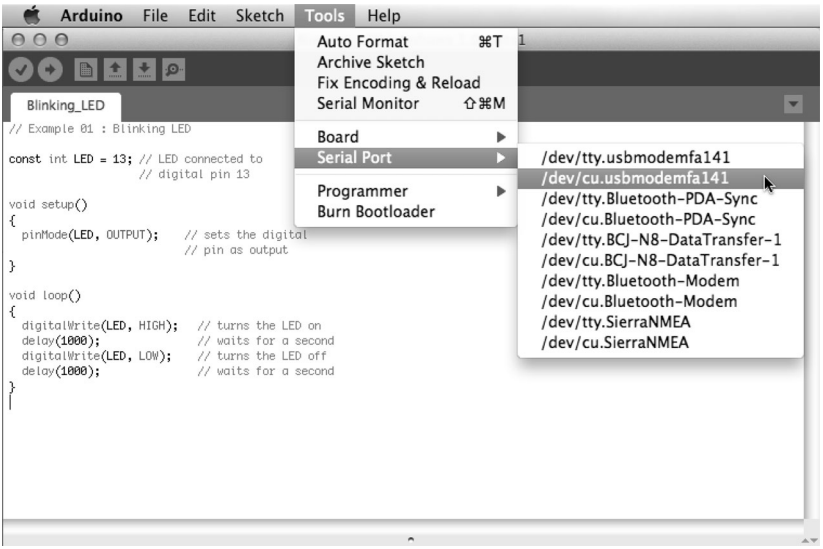


Abbildung 3-3.

Die Liste der seriellen Anschlüsse in der Arduino-IDE

Port-Identifikation unter Windows

Unter Windows ist der Prozess ein wenig komplizierter – zumindest am Anfang. Rufen Sie den Device Manager auf, indem Sie im Menü *Start* mit der rechten Maustaste auf *Computer* (Vista) oder *My Computer* (XP) klicken und dann *Properties* auswählen. Klicken Sie unter Vista dann auf Device Manager (auf der linken Seite des Fensters wird dann eine Liste mit allen Features angezeigt).

Das Arduino-Board ist unter *Ports (COM & LPT)* aufgelistet. Es wird hier als Arduino UNO angezeigt und wird einen Namen wie beispielsweise COM7 aufweisen, wie es in Abbildung 3-4 zu sehen ist.

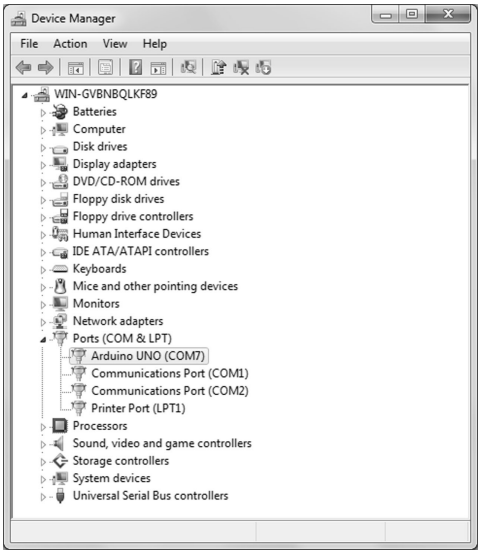


Abbildung 3-4.

Im Windows Device Manager werden alle verfügbaren seriellen Anschlüsse angezeigt

---

Hinweis: Auf einigen Windows-Computern weist der COM-Anschluss eine Nummer auf, die größer als 9 ist. Diese Nummerierung führt zu einigen Problemen, wenn Arduino versucht, mit ihnen zu kommunizieren. Eine entsprechende Hilfestellung findeen Sie im Kapitel 7.

Wenn Sie die Zuweisung für den COM-Anschluss ermittelt haben, können Sie diesen Anschluss über *Tools → Serial Port* in der Arduino-IDE auswählen.

Nun können Sie über die Arduino-Entwicklungsumgebung mit dem Arduino-Board kommunizieren und es programmieren.

4/Die ersten Schritte mit Arduino

Als Nächstes werden Sie erfahren, wie Sie ein interaktives Gerät bauen und programmieren können.

Der Aufbau eines interaktiven Geräts

Alle Objekte, die wir bauen werden, basieren auf einem simplen Muster, das wir Interactive Device nennen. Hierbei handelt es sich um elektronische Schaltungen, die mithilfe von Sensoren (elektronische Komponenten, die Messwerte aus der realen Welt in elektrische Signale umwandeln) die Umgebung erfassen. Das Gerät verarbeitet die Daten, die von den Sensoren geliefert werden, mittels eines Verhaltens, das als Software implementiert wird. Das Gerät ist dann in der Lage, mithilfe von Aktoren, das sind elektronische Komponenten, die ein elektrisches Signal in physikalisches Verhalten umwandeln können, mit der Welt zu interagieren.

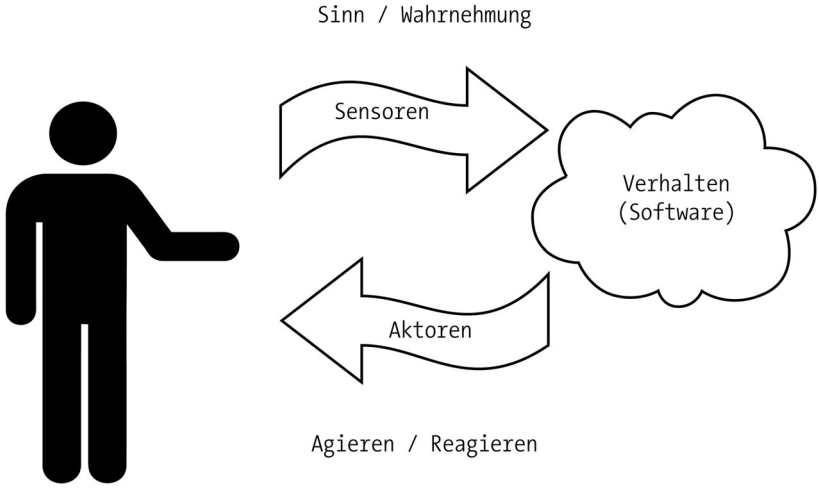


Abbildung 4-1.
Das interaktive Gerät

Sensoren und Aktoren

Sensoren und Aktoren sind elektronische Komponenten, mit deren Hilfe eine elektronische Komponente mit der Umwelt interagieren kann.

Da es sich bei einem Mikrocontroller um einen sehr einfachen Computer handelt, kann er nur elektrische Signale verarbeiten (ähnlich wie bei elektrischen Impulsen, die zwischen den Neuronen unseres Gehirns übertragen werden). Um Licht, Temperatur oder andere physikalische Größen erfassen zu können, müssen diese in Elektrizität umgewandelt werden. In unserem Körper wandelt das Auge Licht in Signale um, die mittels der Nerven an das Gehirn weitergeleitet werden. In der Elektronik können wird dazu ein spezielles Bauteil verwenden, nämlich einen lichtabhängigen Widerstand (einen LDR oder einen Fotowiderstand), der die auftreffende Lichtmenge messen und als Signal, das der Mikrocontroller versteht, wiedergeben kann.

Wenn die Sensoren ausgelesen wurden, verfügt das Gerät über die Informationen, die erforderlich sind, um zu entscheiden, wie es reagieren soll. Der Prozess der Entscheidungsfindung wird vom Mikrocontroller abgewickelt, und die Reaktion erfolgt über die Aktoren. In unserem Körper beispielsweise erhalten die Muskeln elektrische Signale vom Gehirn, die sie dann in Bewegung umsetzen. Im Bereich der Elektronik könnten diese Funktionen z.B. durch Licht oder einen elektrischen Motor ausgeführt werden.

In den folgenden Abschnitten werden Sie erfahren, wie unterschiedliche Typen von Sensoren ausgelesen und unterschiedliche Arten von Aktoren gesteuert werden.

Eine LED zum Blinken bringen

Der Sketch, mit dem eine LED zum Blinken gebracht wird, ist der erste Sketch, den Sie ausführen sollten, um zu testen, ob Ihr Board einwandfrei arbeitet und richtig konfiguriert ist. Dies ist üblicherweise auch die erste Übung für das Programmieren eines Mikrocontrollers. Eine Leuchtdiode (LED) ist eine kleine elektronische Komponente, die einer kleinen Glühbirne ähnelt, jedoch effektiver ist und eine geringere Betriebsspannung benötigt.

Ihr Arduino-Board wird mit einer vorinstallierten LED geliefert. Diese ist mit einem L gekennzeichnet. Sie können auch Ihre eigene LED hinzufügen. Schließen Sie sie so an, wie es in Abbildung 4-2 dargestellt ist.

Hinweis: Wenn die LED über eine längere Zeitdauer leuchten soll, sollten Sie einen Widerstand verwenden, wie auf Seite 54 beschrieben wird.

K kennzeichnet die Kathode (negativ) oder den kürzeren Anschluss, A die Anode (positiv) oder den längeren Anschluss.

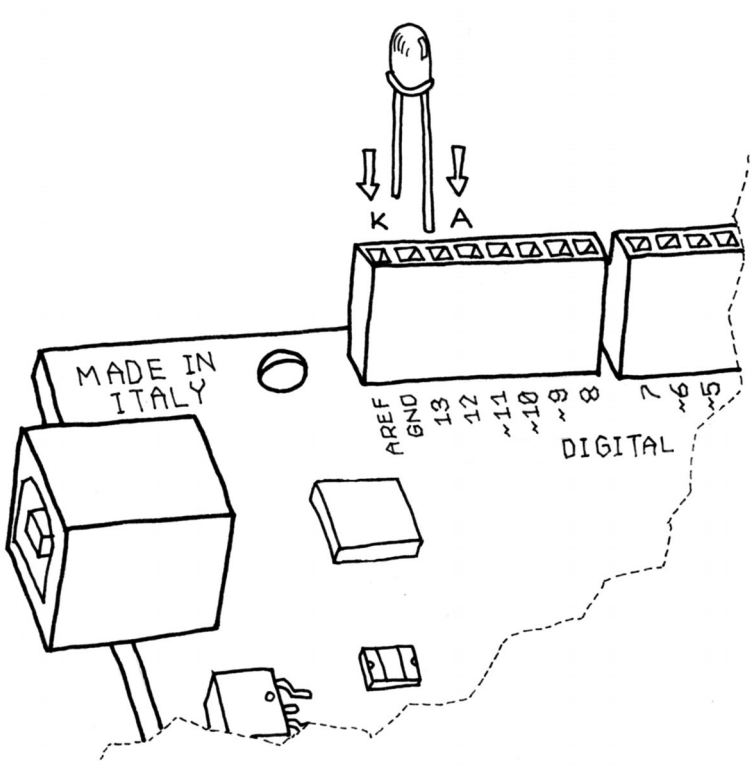


Abbildung 4-2.
Anschließen einer LED an das Arduino-Board

Wenn die LED angeschlossen ist, muss Arduino mitgeteilt werden, was zu tun ist. Dies erfolgt mithilfe von Code, einer Liste von Anweisungen, die wir wiederum dem Mikrocontroller übermitteln und mit der wir ihn dazu bringen, das zu tun, was wir möchten.

Öffnen Sie auf Ihrem Computer den Ordner, in den Sie die Arduino-IDE kopiert haben. Starten Sie die IDE mit einem Doppelklick auf das entsprechende Symbol. Wählen Sie *File → New* aus. Sie werden nun aufgefordert,

einen Ordnernamen für den Sketch anzugeben. Hier wird der Sketch dann gespeichert. Nennen Sie ihn *Blinking_LED* und klicken Sie auf *OK*. Geben Sie dann die folgenden Zeilen (Beispiel 4-1) in den Sketch-Editor von Arduino (das Hauptfenster der Arduino-IDE) ein. Sie können den Code auch unter **http://www.makezine.com/getstartedarduino** herunterladen. Er sollte wie in Abbildung 4-3 aussehen.

```
// Beispiel 4-1: blinking_led

const int LED = 13; // LED connected to
                    // digital pin 13

void setup()
{
  pinMode(LED, OUTPUT);    // sets the digital
                           // pin as output
}

void loop()
{
  digitalWrite(LED, HIGH);   // turns the LED on
  delay(1000);               // waits for a second
  digitalWrite(LED, LOW);    // turns the LED off
  delay(1000);               // waits for a second
}
```

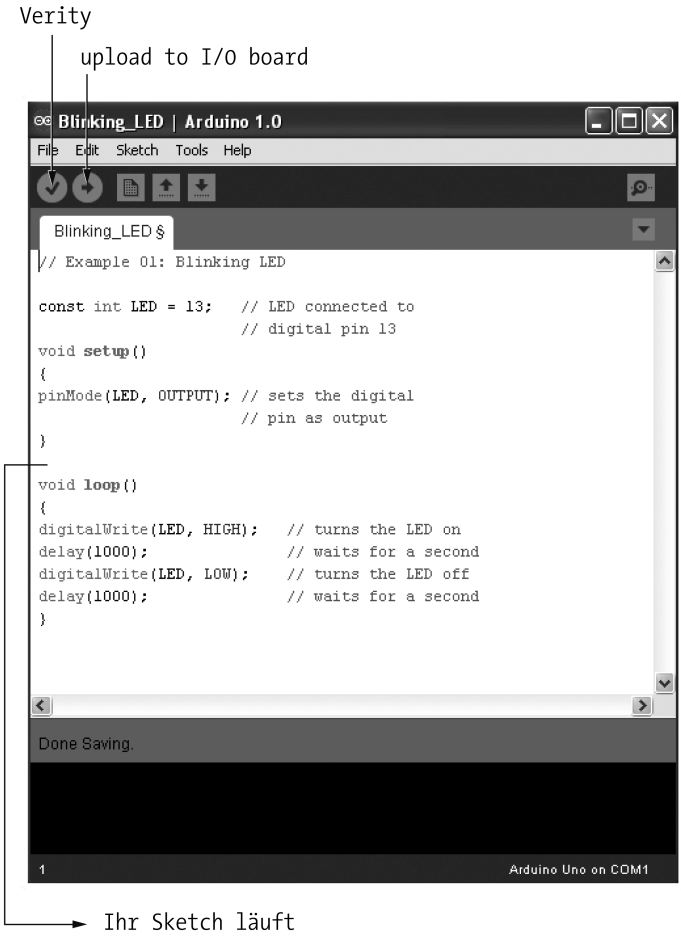


Abbildung 4-3.

Die Arduino-IDE mit dem ersten geladenen Sketch

Nun, da sich der Code in Ihrer IDE befindet, müssen Sie ihn auf Fehler überprüfen. Klicken Sie auf die Schaltfläche *Verify* (Abbildung 4-3 zeigt, wo sie sich befindet); wenn alles korrekt ist, wird die Nachricht *Done compiling* am unteren Rand der Arduino-IDE angezeigt. Diese Nachricht bedeutet, dass die Arduino-IDE Ihren Sketch in ein ausführbares Programm übersetzt hat, das auf Ihrem Board ausgeführt werden kann, ähnlich wie das bei einer .exe-Datei unter Windows oder bei einer .app-Datei unter Mac der Fall ist.

Nun können Sie Ihren Sketch in Ihr Board laden: Klicken Sie auf die Schaltfläche *Upload to I/O Board* (siehe Abbildung 4-3). Dadurch wird das Board zurückgesetzt und alle laufenden Prozesse werden beendet. Das Board wartet nun auf Instruktionen, die vom USB-Port kommen. Die Arduino-IDE sendet den aktuellen Sketch zum Board, das ihn wiederum speichert und schließlich ausführt.

Sie werden einige Nachrichten im schwarzen Bereich am unteren Rand des Fensters sehen, und genau über diesem Bereich die Meldung *Done uploading*, mit der Sie darüber informiert werden, dass der Prozess erfolgreich abgeschlossen wurde. Es sind zwei LEDs mit den Bezeichnungen RX und TX auf dem Board vorhanden. Diese flackern jedes Mal auf, wenn ein Byte vom Board geschickt oder empfangen wird. Während des Upload-Prozesses flackern sie kontinuierlich.

Falls Sie kein Flackern der LEDs erkennen können oder anstelle der Nachricht *Done Uploading* eine Fehlermeldung erhalten, besteht ein Kommunikationsproblem zwischen Ihrem Computer und Arduino. Vergewissern Sie sich, dass Sie den richtigen seriellen Anschluss (siehe Kapitel 3) unter *Tools > Serial Port* ausgewählt haben. Überprüfen Sie außerdem, ob unter *Tools > Board* das richtige Arduino-Modell ausgewählt wurde.

Wenn weiterhin Probleme bestehen, schauen Sie sich das Kapitel 7 an.

Sobald der Code auf Ihr Arduino-Board übertragen wurde, verbleibt er dort, bis Sie ihn mit einem anderen Sketch überschreiben. Der Code bleibt gespeichert, wenn Sie beim Board einen Reset duchführen oder es ausschalten, ähnlich wie bei den Daten auf Ihrer Computerfestplatte.

Wenn der Sketch korrekt geladen wurde, wird die LED L für eine Sekunde aufleuchten und dann für eine Sekunde dunkel bleiben. Wenn Sie eine sepatate LED installiert haben, wie vorher in Abbildung 4-6 zu sehen ist, wird auch diese LED blinken. Das, was Sie geschrieben haben, ist ein Computerprogramm oder auch Sketch, wie ein Arduino-Programm genannt wird. Wie schon erwähnt handelt es sich bei Arduino um einen kleinen Computer, der sich nach Bedarf programmieren lässt. Dazu wird eine Programmiersprache verwendet, um eine Serie von Anweisungen in die Arduino-IDE einzugeben, die diese dann so umwandelt, dass sie vom Arduino-Board ausgeführt werden können.

Als Nächstes möchte ich Ihnen ein Veständnis des Codes vermitteln. Zunächst ist zu erwähnen, dass Arduino den Code von oben nach unten ausführt. Die erste Zeile zuoberst ist also die, die zuerst gelesen wird. Dann wird der Prozess nach unten fortgesetzt. Dies erinnert ein wenig an die

Statusanzeige bei einem Video-Player, z.B. Quick Time Player oder Windows Media Player, bei dem die Statusanzeige allerdings nicht von oben nach unten, sondern von links nach rechts verläuft um anzuzeigen, wo im Film Sie sich gerade befinden.

Reich mir den Parmesan

Achten Sie auf die geschweiften Klammern, die dazu dienen, Codezeilen zusammenzufassen. Diese sind besonders dann sehr nützlich, wenn Sie eine Gruppe von Anweisungen mit einem Namen versehen möchten. Mit der Aufforderung „Bitte reich mit den Parmesan!“ beim Abendessen beispielsweise werden eine Reihe von Aktionen angestoßen, die in diesem kleinen Satz zusammengefasst sind. Weil wir Menschen sind, erfassen wir das ganz selbstverständlich, bei Arduino hingegen müssen alle einzelnen kleinen Aktionen ausformuliert werden, weil die Plattform nicht so leistungsfähig wie unser Gehirn ist. Um also eine Anzahl von Anweisungen in einer Gruppe zusammenzufassen, platzieren Sie ein *{* vor dem Code und ein *}* hinter dem Code.

Sie sehen, dass in unserem Beispiel zwei Blöcke auf diese Weise definiert wurden. Vor jedem dieser Blöcke ist ein merkwürdiger Befehl angeführt:

```
void
 setup()
```

Mit dieser Zeile wird dem Codeblock ein Name zugewiesen. Wenn Sie eine Liste von Anweisungen schreiben würden, die Arduino beibringen, Ihnen den Parmesankäse zu reichen, würden Sie *void passTheParmesan()* am Anfang des Blocks schreiben, und dieser Block würde zu einer Anweisung, die Sie von jeder beliebigen Stelle im Arduino-Code aus aufrufen könnten. Solche Blöcke werden als **Funktionen** bezeichnet. Wenn Sie also anschließend *passTheParmesan()* irgendwo im Code schreiben, wird Arduino die betreffenden Anweisungen ausführen und dann an der Stelle fortfahren, an der der Code vor den Anweisungen verlassen wurde.

Arduino ist nichts für Zögerliche

Bei Arduino wird das Vorhandensein von zwei Anweisungen erwartet – die eine heißt *setup()* und die andere *loop()*.

setup() ist die Funktion, in der all der Code untergebracht wird, der zu Beginn des Programms ausgeführt werden soll, und *loop()* enthält das Kernstück des Programms, das kontinuierlich immer wieder ausgeführt wird. Dies liegt darin begründet, dass Arduino sich nicht wie ein normaler Computer verhält – es können nicht mehrere Programme gleichzeitig

ausgeführt werden, und es ist auch nicht möglich, ein Programm abzubrechen. Wenn das Board an eine Stromversorgung angeschlossen ist, wird der Code ausgeführt. Wenn Sie die Ausführung beenden möchten, Sie dazu einfach das Board ausschalten.

Wirkliche Tüftler schreiben Kommentare

Jeglicher Text, der mit *//* beginnt, wird von Arduino ignoriert. Diese Zeilen sind Kommentare, die Sie für sich selbst im Programm hinterlassen, um sich daran zu erinnern, was Sie mit dem Code bezweckt haben, oder die Sie für andere schreiben, damit sie den Code verstehen.

Es ist sehr üblich (und ich weiß das, weil ich es ständig tue), einen Codeabschnitt zu schreiben, ihn auf das Board zu laden und sich dann zu sagen: „Okay, diesen Kram werde ich nie wieder anfassen!!“, nur um sechs Monate später festzustellen, dass der Code aktualisiert werden muss oder noch ein Fehler zu beheben ist. Sie werden sich den Code anzeigen lassen, und wenn Sie dann keine entsprechenden Kommentare in Ihrem ursprünglichen Programm eingefügt haben, werden Sie sehr schnell denken: „Oh Mann, was für ein Chaos! Wo fange ich denn da bloß an?“ Wenn wir in diesem Buch weiter voranschreiten, werden Sie noch einige Tricks kennenlernen, wie Sie Ihr Programm besser lesbar und einfacher im Hinblick auf die Wartung gestalten.

Der Code – Schritt für Schritt

Womöglich wird Ihnen diese Art von Erläuterung ein wenig überflüssig vorkommen, ähnlich wie in meiner Schulzeit, als ich Dantes *Göttliche Komödie* lesen musste (jeder italienische Schüler muss sie durcharbeiten, genauso wie ein anderes Buch mit dem Titel *Die Brautleute* oder *The Betrothed* – oh, was für ein Albtraum). Für jede Textzeile gab es hundert Zeilen an Kommentar. Wenn Sie allerdings damit beginnen, eigene Programme zu schreiben, sind solche Erläuterungen wesentlich nützlicher.

```
// Example 01 : Blinking LED
```

Ein Kommentar ist eine hilfreiche Möglichkeit, kleine Hinweise anzuführen. Der vorangestellte Titelkommentar erinnert uns daran, dass dieses Programm, Beispiel 4-1, eine LED zum Blinken bringt.

```
const int LED = 13; // LED connected to
                    // digital pin 13
```

const int bedeutet, dass es sich bei LED um eine Ganzzahl handelt, die nicht geändert werden kann (d.h. sie ist eine Konstante) und für die der Wert 13 festgelegt wurde. Das ist vergleichbar mit einem automatischen Suchen-

und-Ersetzen-Vorgang im Code. In unserem Fall wird Arduino angewiesen, jedes Mal, wenn das Wort LED erscheint, die Zahl 13 zu schreiben. Der Befehl wird hier verwendet um festzulegen, dass die LED, die wir zum Blinken bringen, an Pin 13 des Arduino-Boards angeschlossen ist.

```
void setup()
```

Mit dieser Zeile wird Arduino mitgeteilt, dass der nächste Codeblock *setup()* heißt.

```
{
```

Mit der öffnenden geschweiften Klammer wird ein Codeblock eingeleitet.

```
pinMode(LED, OUTPUT); // sets the digital
                      // pin as output
```

Endlich, eine wirklich interessante Anweisung. *pinMode* teilt Arduino mit, wie ein bestimmter Pin konfiguriert werden soll. Digitale Pins können entweder als INPUT oder OUTPUT verwendet werden. In unserem Beispiel benötigen wir einen Ausgangspin, um die LED zu steuern, daher fügen wir die Pinnummer und den „Verwendungszweck" in Klammern an.

pinMode ist eine Funktion, und die in ihr angegebenen Wörter (oder Zahlen) sind **Argumente** . INPUT und OUTPUT werden in der Arduino-Sprache als Konstanten bezeichnet. (Wie Variablen werden auch Konstanten Werte zugewiesen, wobei aber die Werte von Konstanten vordefiniert sind und sich niemals ändern.)

```
}
```

Die schließende geschweifte Klammer zeigt das Ende der *setup()*-Funktion an.

```
void loop()
{
```

In *loop()* wird das hauptsächliche Verhalten des interaktiven Geräts festgelegt. Die Funktion wird immer weiter wiederholt, und zwar so lange, bis Sie das Board ausschalten.

```
digitalWrite(LED, HIGH); // turns the LED on
```

Wie der Kommentar schon besagt ist *digitalWrite()* in der Lage, jeden Pin, der als OUTPUT konfiguriert wurde, ein- oder auszuschalten. Das erste Argument (in unserem Beispiel *LED*) gibt an, welcher Pin ein- oder ausgeschaltet werden soll (erinnern Sie sich daran, dass es sich bei *LED* um einen konstanten Wert handelt, der auf Pin 13 verweist, dies ist also der Pin,

der entsprechend geschaltet wird). Mit dem zweiten Argument wird der Pin eingeschaltet (HIGH) oder ausgeschaltet (LOW).

Stellen Sie sich vor, dass der Ausgangspin eine kleine Steckdose ist, wie die in den Wänden Ihrer Wohnung. Europäische Steckdosen liefern 230 Volt, amerikanische 110 Volt und Arduino arbeitet mit gemäßigten 5 Volt. Die Magie offenbart sich hier, wenn die Software zur Hardware wird. Wenn Sie *digitalWrite(LED, HIGH*) schreiben, wird der Ausgangspin auf 5 V gesetzt. Schließen Sie dann die LED an, leuchtet sie. An dieser Stelle im Code bewirkt eine Anweisung in der Software eine Reaktion in der physikalischen Welt, indem der Stromfluss zum Pin gesteuert wird. Das Ein- und Ausschalten des Pins lässt sich in etwas für den Menschen besser Sichtbares übertragen; die LED ist unser Aktator.

`delay(1000); // waits for a second`

Arduino hat eine sehr elementare Struktur. Wenn Sie daher möchten, dass irgendetwas mit einer bestimmten Regelmäßigkeit erfolgen soll, weisen Sie Arduino an, sich ruhig zu verhalten und nichts zu tun, bis es an der Zeit ist, mit dem nächsten Schritt fortzufahren. Mit *delay()* weisen Sie im Grunde genommen den Prozessor an zu pausieren und nichts zu tun, und zwar für die Zeitdauer in Millisekunden, die Sie als Argument übergeben. Eine Millisekunde ist ein Tausendstel einer Sekunde, also sind 1000 Millisekunden eine Sekunde. In unserem Beispiel wird die LED also eine Sekunde lang leuchten.

`digitalWrite(LED, LOW); // turns the LED off`

Mit dieser Anweisung wird die LED, die wir vorher eingeschaltet haben, ausgeschaltet. Warum verwenden wir eigentlich HIGH oder LOW? Nun, es handelt sich um eine alte Konvention in der digitalen Elektronik. HIGH bedeutet, dass der Pin eingeschaltet ist, was im Falle von Arduino bedeutet, dass er auf 5 V gesetzt ist. Bei LOW ist er auf 0 V gesetzt. Sie können diese beiden Argumente mental einfach durch EIN und AUS ersetzen.

`delay(1000); // waits for a second`

An dieser Stelle bauen wir eine weitere Verzögerung von einer Sekunde ein. Die LED bleibt eine Sekunde lang ausgeschaltet.

`}`

Die schließende geschweifte Klammer zeigt das Ende der loop-Funktion an.

Zusammengefasst tut das Programm Folgendes:

- Pin 13 wird als Ausgangspin definiert (nur ein Mal zu Beginn).
- Es erfolgt der Eintritt in eine Schleife.
- Die LED, die mit Pin 13 verbunden ist, wird eingeschaltet.
- Es folgt eine Wartezeit von einer Sekunde.
- Die LED, die mit Pin 13 verbunden ist, wird ausgeschaltet.
- Es folgt eine Wartezeit von einer Sekunde.
- Es wird ein Sprung zurück an den Anfang der Schleife durchgeführt.

Ich hoffe, dass Ihnen dieser Code noch keine allzu großen Kopfschmerzen bereitet hat. Sie werden in den späteren Beispielen noch mehr zum Thema Programmierung erfahren.

Bevor wir zum nächsten Abschnitt kommen, wollen wir noch ein wenig mit dem Code spielen. Wir könnten zum Beispiel die Anzahl der Verzögerungen reduzieren und dabei verschiedene Zahlen für die Ein- und Ausphasen verwenden, sodass wir unterschiedliche Blinkmuster beobachten können. Insbesondere sollten Sie darauf achten, was geschieht, wenn die Verzögerungen sehr klein sind und sich bei den Ein- und Ausphasen unterscheiden ... Sie können dabei nämlich für einen Moment etwas beobachten, das später in diesem Buch, wenn wir zum Stichwort Pulsweitenmodulation kommen, noch sehr nützlich sein wird.

Was wir bauen werden

Ich war immer fasziniert von Licht und der Möglichkeit, verschiedene Lichtquellen mittels Technologie zu steuern. Ich hatte das Gück, an einigen interessanten Projekten zu arbeiten, die damit befasst waren, Licht zu steuern und es mit lebenden Personen interagieren zu lassen. Arduino bietet diesbezüglich wirklich gute Möglichkeiten. Im gesamten Buch werden wir uns damit befassen, wie sich „interaktive Lampen“ herstellen lassen. Anhand von Arduino, das wir hier verwenden, werden wir die Grundlagen kennenlernen, die erforderlich sind, um interaktive Geräte zu bauen.

Im nächsten Abschnitt werde ich versuchen, die Grundlagen der Elektrizität auf eine Art und Weise zu erläutern, die zwar einen Ingenieur sicherlich

langweilen würde, aber dafür auch einen neuen Arduino-Programmierer nicht sofort abschreckt.

Was ist Elektrizität?

Wenn Sie zu Hause schon mal Klempnerarbeiten durchgeführt haben, werden Sie in puncto Elektronik keine Verständnisschwierigkeiten haben. Der beste Weg zu vermitteln, wie Elektrizität und elektrische Schaltungen funktionieren, ist die Wasseranalogie. Nehmen wir ein einfaches Gerät wie den batteriebetriebenen, tragbaren Ventilator, der in Abbildung 4-4 gezeigt wird.

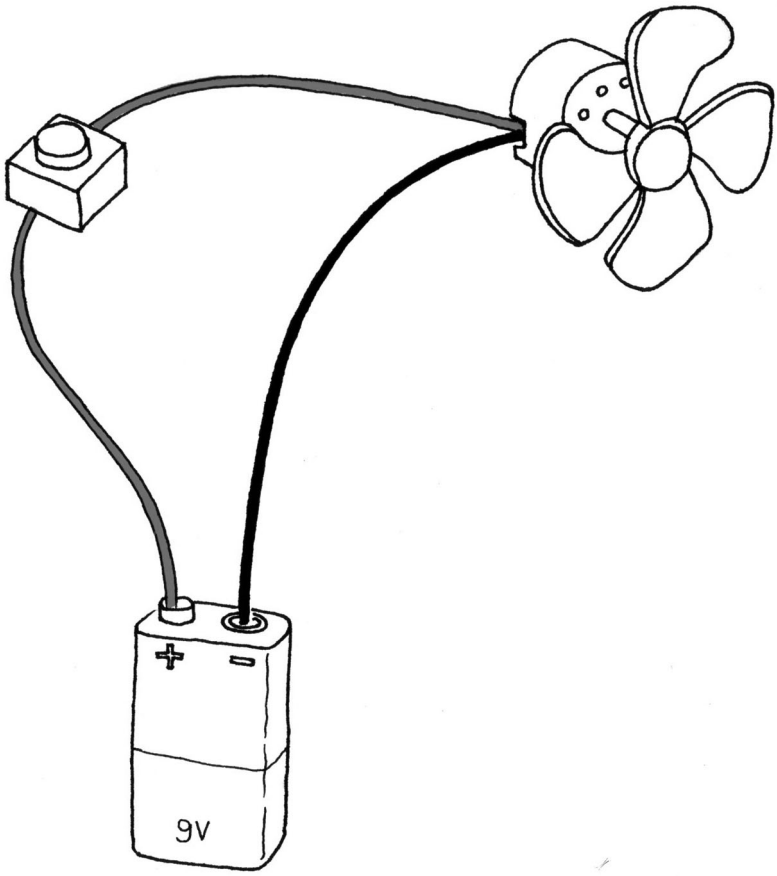


Abbildung 4-4.
Ein portabler Ventilator

Wenn Sie den Ventilator auseinanderbauen, werden Sie sehen, dass er eine kleine Batterie, einige Drähte und einen elektrischen Motor enthält, und dass einer der Drähte, die zum Motor führen, durch einen Schalter unterbrochen ist. Wenn die Batterie voll ist und Sie den Schalter betätigen und den Motor einschalten, beginnt dieser, sich zu drehen, und sorgt so für die nötige Abkühlung. Wie funktioniert das? Stellen Sie sich einfach vor, die Batterie sei zugleich ein Wasserreservoir und eine Pumpe, der Schalter ein Ventil und der Motor eines von diesen Wasserrädern, die Sie sicher schon bei Windmühlen gesehen haben. Wenn Sie das Ventil öffnen, fließt das Wasser von der Pumpe zum Waserrad und treibt es an.

Bei diesem einfachen Beispiel, das in Abbildung 4-5 dargestellt ist, sind zwei Faktoren wichtig: der Wasserdruck (der von der Leistung der Pumpe bestimmt wird) und die Wassermenge, die durch die Leitung fließt (die vom Durchmesser der Leitung und vom Widerstand, den das Wasserrad dem auftreffenden Wasserstrom entgegesetzt, abhängt).

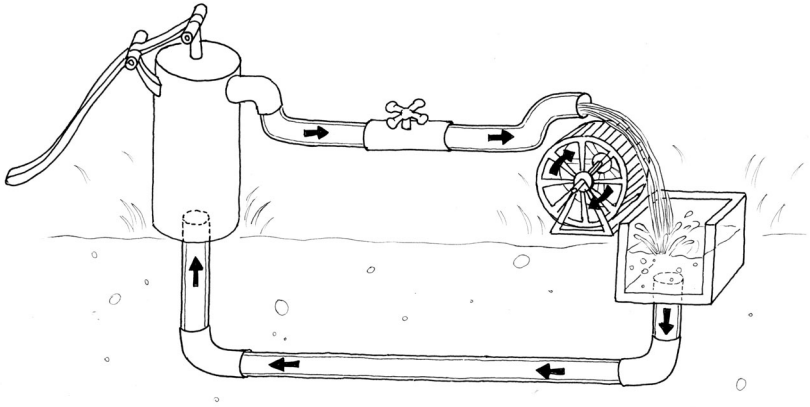


Abbildung 4-5.
Ein Hydrauliksystem

Sie werden schnell bemerken, dass zur Erhöhung der Drehgeschwindigkeit des Rades erforderlich ist, den Durchmesser der Leitungen zu vergrößern (was nur bis zu einem bestimmten Punkt funktioniert) und den Druck zu erhöhen, der durch die Pumpe erzielt wird. Durch das Vergrößern des Durchmessers der Leitungen kann mehr Wasser durch sie hindurchfließen. Durch diesen größeren Durchmesser wird der Widerstand in Bezug auf den Wasserdurchfluss verringert. Dieser Ansatz funktioniert bis zu einem bestimmten Punkt, ab dem sich das Rad nicht mehr schneller dreht, weil der Wasserduck nicht ausreicht. Wenn dieser Punkt erreicht wurde, muss die Pumpleistung erhöht werden. Diese Möglichkeit der Beschleunigung des

Wasserrades funkioniert so lange, bis das Rad wegen des zu starken Wasserdrucks auseinanderbricht und zerstört wird. Ein anderer Aspekt, der sich beobachten lässt, ist die Wärmeentwicklung an der Achse, die entsteht, wenn sich das Rad dreht. Egal, wie gut das Wasserrad montiert ist, durch die Reibung zwischen der Achse und der Vorrichtung, in der sie montiert ist, wird Wärme erzeugt. Es ist wichtig zu verstehen, dass bei einem System wie diesem nicht alle zugeführte Energie in Bewegung umgewandelt wird, sondern ein Teil der Energie verloren geht. Diese zeigt sich dann als Wärme, die von einzelnen Komponenten im System abgegeben wird.

Was sind also die wichtigen Aspekte bei diesem System? Einer ist der durch die Pumpe erzeugte Druck, die anderen sind der Widerstand, der dem Wasserstrom durch die Leitung und das Wasserrad entgegengesetzt wird, und der eigentliche Wasserdurchfluss (der dargestellt wird als die Anzahl an Litern, die pro Sekunde fließt). Elektrizität funktioniert ein wenig wie Wasser. Man hat eine Art Pumpe (jede Art von Energiequelle, z.B. eine Batterie oder eine Steckdose in der Wand), die elektrische Ladungen (die Sie sich am besten als kleine elektrische Tropfen vorstellen) durch Leitungen drückt, die in Form von Drähten realisert sind – diese werden von einigen Geräten verwendet, um Wärme zu produzieren (Großmutters Heizdecke), Licht zu erzeugen (Ihre Nachttischlampe), Sound zu produzieren (Ihre Stereoanlage), Bewegung anzustoßen (unser Ventilator) und für viele weitere Dinge.

Wenn Sie also auf einer Batterie die Angabe 9 V lesen, dann stellen Sie sich diese elektrische Spannung einfach als Wasserdruck vor, der mittels einer kleinen Pumpe erzeugt wird. Elektrische Spannung wird in Volt gemessen. Diese Einheit wurde nach Alessandro Volta benannt, dem Erfinder der ersten Batterie.

Wie der Wasserdruck hat auch die Durchflussmenge des Wassers ein Äquivalent in der Elektrizität. Sie wird als Strom bezeichnet, der in Ampere gemessen wird (nach André-Marie Ampère, einem Pionier des Elektromagnetismus). Das Verhältnis von elektrischer Spannung und Strom kann wieder anhand des Beipiels mit dem Wasserrad veranschaulicht werden: Ein höherer Wasserdruck (elektrische Spannung) bewirkt eine schnellere Drehung des Rades, mit einer höheren Durchflussrate (Strom) lässt sich ein größeres Rad antreiben.

Der Widerstand schließlich, der dem Stromfluss auf jedem Weg, den er zurücklegt, entgegengesetzt wird, heißt, wie Sie bestimmt schon erraten haben, auch in der Elektronik Widerstand und wird in Ohm gemessen (nach einem deutschen Physiker). Herr Ohm formulierte auch das wichtigste

Gesetz in der Elektrizität – und die betreffende Formel ist auch die einzige, die Sie sich wirklich merken müssen. Er konnte nachweisen, dass in jedem Schaltkreis eine Beziehung zwischen Strom und Widerstand besteht, genauer gesagt, dass bei einer gegebenen Spannung die Strommenge, die durch einen Schaltkreis fließt, vom vorhandenen Widerstand abhängt.

Bei genauerem Nachdenken ist das recht intuitiv zu verstehen. Schließen Sie eine 9-V-Batterie an einen einfachen Schaltkreis an. Wenn Sie nun den Strom messen, werden Sie feststellen, dass er umso geringer wird, je mehr Widerstände Sie einbauen. Wenn wir nochmal auf das Beispiel mit dem Wasserdurchfluss in den Leitungen zurückkommen, heißt das Folgendes: Wenn Sie hier ein Ventil einbauen (das sich mit einem **variablen Widerstand** in der Elektrizität vergleichen lässt) und dieses Ventil immer weiter schließen, erhöhen Sie den Widerstand in Bezug auf den Wasserdurchfluss und es fließt immer weniger Wasser durch die Leitungen. Ohm hat dieses Gesetz in folgender Formfel zusammengefasst:

```
R (Widerstand) = V (Spannung) / I (Strom)
V = R * I
I = V / R
```

Dies ist die einzige Regel, die Sie sich merken und anwenden können müssen, weil sie die einzige ist, die Sie für Ihre Arbeit mit Arduino wirklich brauchen.

Steuerung einer LED mit einem Drucktaster

Eine LED zum Blinken zu bringen, war recht einfach, aber ich glaube nicht, dass Sie glücklich werden, wenn Ihre Nachttischlampe ständig blinkt, während Sie versuchen, ein Buch zu lesen. Daher müssen Sie sie irgendwie steuern können. In unserem vorherigen Beispiel war die LED ein Aktator, der von Arduino gesteuert wurde. Was uns also fehlt, ist ein Sensor.

Für unser Beispiel verwenden wir die einfachste verfügbare Ausführung eines Sensors: einen Drucktaster.

Wenn Sie einen Drucktaster in seine Einzelteile zerlegen würden, wäre Ihnen sehr schnell klar, dass es sich um ein sehr einfaches Bauteil handelt. Er besteht aus zwei Metallplättchen, die durch eine Feder voneinander separiert werden, und einer Plastikkappe, die, wenn sie gedrückt wird, die zwei Metallplättchen miteinander verbindet. Wenn keine Verbindung zwischen den Metallplättchen besteht, erfolgt keine Stromzirkulation im Drucktaster (ähnlich wie bei einem geschlossenen Ventil). Wenn Sie den Taster aber drücken, stellen Sie eine Verbindung her.

Um den Status eines Schalters zu überwachen, möchte ich hier eine neue Arduino-Anweisung einführen: die Funktion *digitalRead()*.

digitalRead() überprüft, ob irgendeine Spannung an dem Pin anliegt, den Sie in den runden Klammern angegeben haben, und gibt einfach den Wert HIGH oder LOW zurück, je nachdem, was die Überprüfung ergeben hat. Die anderen Anweisungen, die wir bisher verwendet haben, geben keinerlei Informationen zurück – sie führen nur das aus, was wir von ihnen möchten. Diese Art von Anweisungen ist aber ein wenig begrenzt, da wir bei einer sehr leicht vorhersagbaren Abfolge von Anweisungen stehenbleiben, ohne Input aus der sie umgebenden Welt. Mittels *digitalRead()* können wir an Arduino eine Frage richten und wir erhalten eine Antwort, die wiederum irgendwo gespeichert und sofort im Anschluss oder auch später zur Entscheidungsfindung herangezogen wird.

Bauen Sie die Schaltung, die in Abbildung 4-6 dargestellt ist. Hierzu benötigen Sie einige Bauteile (diese werden auch bei anderen Projekten nützlich sein):

- Eine lötfreie Steckplatine: Maker Shed (**http://www.makershed.com**), Teilnummer MKKN3, im Arduino Store (**bit.ly/ArduinoStoreBreadBoard**). Anhang A bietet eine Einführung zum Thema lötfreie Steckplatinen.

- Einen Satz vorgefertigter Steckbrücken: Maker Shed MKKN4, Arduino Store (sind im Lieferumfang der lötfreien Steckplatine enthalten).

- Einen 10-K-Ohm-Widerstand: Maker Shed JM691 104 (100er-Pack), Arduino Store (**bit.ly/ArduinoStore10k**, 10er-Pack).

- Einen Drucktastschalter: Maker Shed JM119011, Arduino Store (**bit.ly/ArduinoStorePushButtons**)

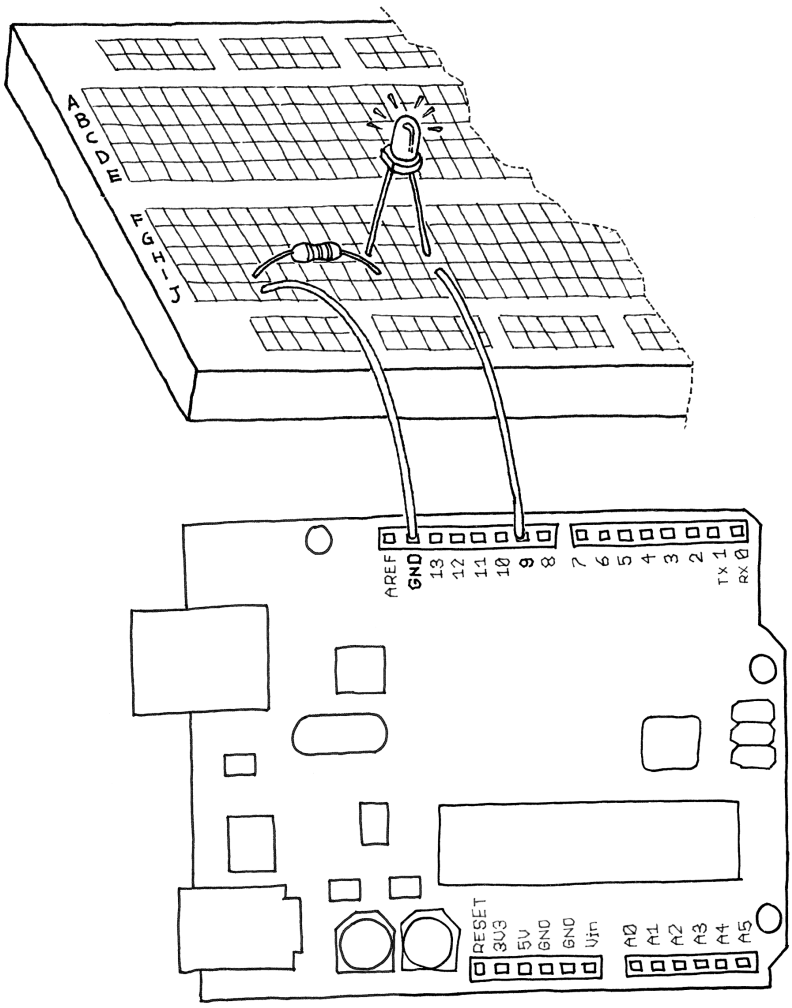


Abbildung 4-6.
Anschließen eines Drucktasters

Hinweis: Alternativ zum Kauf vorgefertigter Steckbrücken können Sie auch festen Schaltdraht vom Typ 22 AWG, der auf kleinen Spulen aufgewickelt ist, verwenden und ihn mit Drahtschneider und Abisolierzange abisolieren.

Werfen wir nun einen Blick auf den Code, den wir verwenden, um die LED mit dem Drucktaster zu steuern:

```
// Beispiel 4-2: turn_on_led_while_the_button_is_pressed

const int LED = 13;   // the pin for the LED
const int BUTTON = 7; // the input pin where the
                      // pushbutton is connected
int val = 0;          // val will be used to store the state
                      // of the input pin

void setup() {
  pinMode(LED, OUTPUT);   // tell Arduino LED is an output
  pinMode(BUTTON, INPUT); // and BUTTON is an input
}

void loop(){
  val = digitalRead(BUTTON); // read input value and store it

  // check whether the input is HIGH (button pressed)
  if (val == HIGH) {
    digitalWrite(LED, HIGH); // turn LED ON
  } else {
    digitalWrite(LED, LOW);
  }
}
```

Wählen Sie in Arduino *File > New* aus (falls Sie einen anderen Sketch geöffnet haben, schließen Sie ihn gegebenenfalls). Wenn Sie von Arduino aufgefordert werden, einen Namen für den neuen Sketch-Ordner anzugeben, geben Sie *PushButtonControl* ein. Geben Sie dann den Code für Beispiel 4-2 ein (oder laden Sie ihn von der Webadresse **http://www.makezine.com/getstartedarduino** herunter und kopieren ihn in die Arduino-IDE). Wenn alles korrekt abgelaufen ist, wird die LED leuchten, wenn Sie den Taster drücken.

Erläuterung der Funktionsweise

Mit diesem Beispielprogramm habe ich zwei neue Konzepte eingeführt: Funktionen, die das Ergebnis ihrer Arbeit zurückliefern, und die *if*-Anweisung.

Die *if*-Anweisung ist wahrscheinlich die wichtigste Anweisung in einer Programmiersprache, weil sie dem Computer (und denken Sie immer daran, dass Arduino ein kleiner Computer ist) ermöglicht, Entscheidungen zu treffen. Nach dem Schlüsselwort *if* muss eine Frage, die in Klammern eingeschlossen ist, angefügt werden. Wenn die Antwort bzw. das Ergebnis wahr ist, wird der erste Codeblock ausgeführt, anderenfalls der nachfolgende Codeblock. Beachten Sie hier, dass ich das Symbol == anstelle von = verwendet habe. Das Erstere wird verwendet, wenn zwei Einheiten miteinander verglichen werden. Es wird dann entsprechend das Ergebnis TRUE oder FALSE zurückgeliefert; mit dem Letzeren wird einer Variablen ein Wert zugewiesen. Achten Sie darauf, das korrekte Zeichen zu verwenden, denn diese Fehlerquelle ist recht groß, und im Falle eines Fehlers an dieser Stelle wird das Programm niemals funktionieren. Ich weiß, wovon ich spreche, denn nach 25 Jahren Programmiererfahrung unterläuft mir dieser Fehler immer noch.

Es ist natürlich sehr unpraktisch, mit dem Finger die ganze Zeit den Taster gedrückt halten zu müssen, wenn Sie Licht benötigen. Auch wenn man bedenkt, wie viel Energie verschwendet wird, wenn Sie sich von der Lampe fortbewegen und sie nicht nutzen, sie aber eingeschaltet lassen, wollen wir doch herausfinden, wie wir bewirken können, dass der Taster im Modus „gedrückt“ fixiert wird.

Ein Schaltkreis – 1000 Verhaltensweisen

Der große Vorteil von digitaler, programmierbarer Elektronik gegenüber klassischer Elektronik wird hier offensichtlich: Ich werde Ihnen nun zeigen, auf welche Weise viele verschiedene Verhaltensweisen unter Verwendung desselben Schaltkreises wie im vorherigen Abschnitt implementiert werden können, einfach, indem die Software entsprechend geändert wird.

Wie ich bereits erwähnt habe, ist es nicht sehr praktisch, die ganze Zeit den Taster mit dem Finger gedrückt halten zu müssen, um Licht zu haben. Wir müssen also eine Art von Gedächtnis implementieren, und zwar in Form eines Softwaremechanismus, der speichert, wann wir den Taster gedrückt haben, und der die Lampe weiter leuchten lässt, auch wenn wir den Finger vom Taster genommen haben.

Hierzu verwenden wir eine sogenannte **Variable**. (Wir haben sie bereits einmal verwendet, aber ich habe sie noch nicht erläutert.) Eine Variable ist ein Ort im Arduino-Speicher, an dem Daten gespeichert werden können. Sie können sie sich als Post-it vorstellen, das Sie verwenden, um sich an etwas zu erinnern, z.B. eine Telefonnummer: Sie schreiben beispielsweise Luisa 02555 1212 darauf und kleben ihn an Ihren Computerbildschirm oder an den Kühlschrank. In der Arduino-Sprache ist das ähnlich simpel: Sie entscheiden einfach, welcher Datentyp gespeichert werden soll (z.B. eine Zahl oder Text) und vergeben einen Namen. Sie können diese Daten dann speichern oder abrufen. Hier ein Beispiel:

```
int val = 0;
```

int bedeutet, dass die Variable eine Ganzzahl speichert. Dabei ist *val* der Name der Variablen, und mit = *0* wird ein Anfangswert von 0 zugewiesen.

Eine Variable kann, wie der Name schon sagt, überall im Code geändert werden, sodass Sie später im Programm Folgendes schreiben können, wodurch Ihrer Variable ein neuer Wert, 112, zugewiesen wird:

```
val = 112;
```

Hinweis: Haben Sie bemerkt, dass in Arduino jede Anweisung mit einem Semikolon endet? Auf diese Weise wird dem Compiler (dem Teil von Arduino, der Ihren Sketch in ein vom Mikrocontroller ausführbares Programm umwandelt) angezeigt, dass eine Anweisung beendet ist und eine neue beginnt. Vergessen Sie also nicht, immer das Semikolon anzufügen.

Im folgenden Programm wird *val* verwendet, um das Ergebnis von *digitalRead()* zu speichern; alle Informationen, die Arduino vom Input-Pin erhält, landen in der Variablen und bleiben dort gespeichert, bis sie von einer anderen Codezeile geändert werden. Beachten Sie, dass Variablen einen Speichertyp verwenden, der als RAM bezeichnet wird. Diese Art von Speicher ist recht schnell, doch wenn Sie Ihr Board ausschalten, gehen alle im RAM gespeicherten Daten verloren (d.h. jede Variable wird auf ihren Anfangswert zurückgesetzt, wenn das Board wieder mit Energie versorgt wird). Ihre Programme selbst werden in einem Flash-Speicher – dieselbe Art von Speicher, wie sie auch bei Mobiltelefonen zum Speichern von Telefonnummern verwendet wird – gespeichert. Hierin bleiben die Daten erhalten, auch wenn das Board ausgeschaltet wird.

Nun wollen wir eine andere Variable verwenden, mit der gespeichert wird, ob die LED ein- oder ausgeschaltet bleiben soll, nachdem wir den Finger

vom Taster genommen haben. Beispiel 4-3 ist ein erster Versuch, dies zu erreichen:

```
// Beispiel 4-3: turn_on_led_when_the_button_is_pressed_a

const int LED = 13;   // the pin for the LED
const int BUTTON = 7; // the input pin where the
                      // pushbutton is connected
int val = 0;    // val will be used to store the state
                // of the input pin
int state = 0;  // 0 = LED off while 1 = LED on

void setup() {
  pinMode(LED, OUTPUT);   // tell Arduino LED is an output
  pinMode(BUTTON, INPUT); // and BUTTON is an input
}

void loop() {
  val = digitalRead(BUTTON); // read input value and store it

  // check if the input is HIGH (button pressed)
  // and change the state
  if (val == HIGH) {
    state = 1 - state;
  }

  if (state == 1) {
    digitalWrite(LED, HIGH); // turn LED ON
  } else {
    digitalWrite(LED, LOW);
  }
}
```

Testen Sie nun diesen Code. Sie werden dann sehen, dass er funktioniert … irgendwie. Das Licht wechselt allerdings so schnell, dass Sie es gar nicht zuverlässig mit einem Tastendruck ein- oder ausschalten können.

Schauen wir uns einmal die interessanten Zeilen des Codes an: *state* ist eine Variabe, die entweder 0 oder 1 speichert, um sich so zu merken, ob die

LED ein- oder ausgeschaltet ist. Wenn der Taster freigegeben wurde, wird sie auf den Anfangswert 0 (LED aus) gesetzt.

Später lesen wir den aktuellen Zustand des Tasters aus, und wenn er gedrückt ist (*val == HIGH*), ändern wir diesen von 0 in 1 oder umgekehrt. Im Hinblick darauf, dass state immer nur 1 oder 0 sein kann, verwende ich hier einen kleinen Trick. Dieser beinhaltet einen kleinen mathematischen Ausdruck, der auf der Idee basiert, dass 1 - 0 = 1 ist, und 1 - 1 = 0:

```
state = 1 - state;
```

Die Zeile ergibt mathematisch gesehen vielleicht keinen Sinn, bei der Programmierung hingegen schon. Das Symbol = bedeutet „Weise das Ergebnis von dem, was nach mir folgt, der Variablen zu, die vor mir angeführt ist." – in unserem Beispiel wird *state* als neuer Wert das Ergebnis von 1 minus dem alten Wert von *state* zugewiesen.

Später in diesem Programm können Sie sehen, dass wir *state* verwenden, um zu ermitteln, ob die LED ein- oder ausgeschaltet sein muss. Wie bereits erwähnt wurde, führt das zu eher ungenauen Ergebnissen.

Dies liegt an der Art und Weise, wie der Taster ausgelesen wird. Arduino ist wirklich schnell. Die eigenen internen Anweisungen werden mit einer Rate von 16 Millionen pro Millisekunde ausgeführt – das sind gut einige Millionen Codezeilen pro Sekunde. Während Sie also mit Ihrem Finger den Taster drücken, liest Arduino die Position des Schalters möglicherweise einige tausend Male und ändert dabei *state* entsprechend. Das Ergebnis wird also letztendlich unvorhersehbar; es könnte „Aus" lauten, wenn es eigentlich „An" lauten sollte oder umgekehrt. So wie selbst eine kaputte Uhr zwei Mal am Tag die Zeit korrekt wiedergibt, kann auch das Programm gelegentlich ein korrektes Verhalten aufweisen, meistens aber wird es falsch sein.

Wie können wir dieses Problem beheben? Nun, wir müssen den exakten Zeitpunkt ermitteln, an dem der Taster gedrückt wird – das ist dann der einzige Moment, in dem *state* geändert werden muss. Dazu möchte ich den Wert von *val* speichern, bevor ich einen neuen Wert auslese. Dadurch wird es möglich, die aktuelle Position des Tasters mit der vorherigen zu vergleichen und *state* nur dann zu ändern, wenn für den Taster ein HIGH nach einem vorherigen LOW ermittelt wird.

Beispiel 4-4 enthält den entsprechenden Code:

```
// Beispiel 4-4: turn_on_led_when_the_button_is_pre-id001

const int LED = 13;   // the pin for the LED
const int BUTTON = 7; // the input pin where the
                  // pushbutton is connected
int val = 0;    // val will be used to store the state
              // of the input pin
int old_val = 0; // this variable stores the previous
              // value of "val"
int state = 0;   // 0 = LED off and 1 = LED on

void setup() {
  pinMode(LED, OUTPUT);   // tell Arduino LED is an output
  pinMode(BUTTON, INPUT); // and BUTTON is an input
}
void loop(){
  val = digitalRead(BUTTON); // read input value and store it
                         // yum, fresh

 // check if there was a transition
 if ((val == HIGH) && (old_val == LOW)){
   state = 1 - state;
 }

 old_val = val;  // val is now old, let's store it

 if (state == 1) {
   digitalWrite(LED, HIGH); // turn LED ON
 } else {
   digitalWrite(LED, LOW);
 }
}
```

Probieren Sie das Programm aus, wir haben es fast geschafft!

Möglicherweise haben Sie bemerkt, dass dieser Ansatz nicht ganz perfekt ist, was an einem anderen Problem mit mechanischen Schaltern liegt. Bei Drucktastern handelt es sich um recht einfache Bauteile: zwei Metallplätt-

chen, die durch eine Feder auseinandergehalten werden. Wenn Sie den Schalter drücken, wird eine Verbindung zwischen diesen beiden Kontakten hergestellt und Strom fließt. Dies klingt nach einem einfachen und wirksamen Mechanismus, aber in der Realität ist die Verbindung nicht so perfekt, besonders dann nicht, wenn der Taster nicht richtig gedrückt wurde, und es werden einige Störsignale erzeugt. Dieser Effekt wird als **Prellen** bezeichnet.

Wenn der Taster geprellt wird, erhält Arduino eine Reihe von rasch aufeinanderfolgenden Ein- und Aussignalen. Es wurden viele Möglichkeiten zum Entprellen entwickelt, aber in diesem einfachen Codeabschnitt ist es meiner Erfahrung nach völlig ausreichend, eine Verzögerung von 10 bis 50 Millisekunden einzubauen, wenn vom Code ein Wechsel ermittelt wurde.

Beispiel 4-5 enthält den finalen Code:

```
// Beispiel 4-5: turn_on_led_when_the_button_is_pre-id002

const int LED = 13;    // the pin for the LED
const int BUTTON = 7;  // the input pin where the
                    // pushbutton is connected
int val = 0;     // val will be used to store the state
               // of the input pin
int old_val = 0; // this variable stores the previous
               // value of "val"
int state = 0;   // 0 = LED off and 1 = LED on

void setup() {
  pinMode(LED, OUTPUT);   // tell Arduino LED is an output
  pinMode(BUTTON, INPUT); // and BUTTON is an input
}

void loop(){
 val = digitalRead(BUTTON); // read input value and store it
                         // yum, fresh

 // check if there was a transition
 if ((val == HIGH) && (old_val == LOW)){
   state = 1 - state;
   delay(10);
 }
```

```
  old_val = val; // val is now old, let's store it

  if (state == 1) {
    digitalWrite(LED, HIGH); // turn LED ON
  } else {
    digitalWrite(LED, LOW);
  }
}
```

5/Erweiterter Input und Output

In Kapitel 4 haben wir die grundlegenden Operationen kennengelernt, die wir mit Arduino durchführen können: Steuern des digitalen Outputs und Auslesen des digitalen Inputs. Wenn es sich bei Arduino um eine menschliche Sprache handeln würde, wären dies zwei Buchstaben ihres Alphabets. Angesichts der Tatsache, dass dieses Aphabet aus nur fünf Buchstaben besteht, können Sie schon abschätzen, wie viel Arbeit uns bevorsteht, bevor wir Arduino-Poesie schreiben können.

Der Einsatz anderer Ein/Aus-Sensoren

Nachdem Sie nun erfahren haben, wie der Drucktastenschalter verwendet wird, sollten Sie wissen, dass es viele andere sehr einfache Sensoren gibt, die nach demselben Prinzip funktionieren:

Schalter
Sie funktionieren ähnlich wie ein Drucktaster, allerdings ändern sie nicht automatisch den Zustand, wenn sie freigegeben werden.

Thermostate
Hierbei handelt es sich um Schalter, die bei Erreichen eines festgelegten Wertes geöffnet werden.

Magnetische Schalter (auch als Reed-Relays bekannt))
Sie verfügen über zwei Kontake, die verbunden werden, wenn sie sich in der Nähe eines Magnets befinden. Diese Schalter kommen bei Alarmanlagen zum Einsatz um festzustellen, ob ein Fenster geöffnet ist.

Sensormatten
Dies sind dünne Matten, die unter einem Teppich oder unter der Türmatte platziert werden können, um die Anwesenheit von Personen (oder einer schwergewichtigen Katze) festzustellen.

Neigungsschalter
Hierbei handelt es sich um eine einfache elektronische Komponente, die aus zwei Kontakten und einem kleinen Metallball besteht (oder einem Quecksilbertropfen, aber ich empfehle, solche Schalter nicht zu verwen-

den). Ein Beispiel für einen Neigungsschalter ist ein Neigungssensor; Abbildung 5-1 zeigt ein typisches Modell. Wenn sich der Sensor in einer aufrechten Position befindet, werden die beiden Kontakte durch den Ball verbunden. Es ist dasselbe Prinzip wie beim Drücken eines Drucktasters. Wenn Sie den Sensor kippen, bewegt sich der Ball und der Kontakt wird geöffnet. Das hat denselben Effekt wie die Freigabe eines Drucktasters. Diese einfache Komponente kann zum Beispiel bei gestischen Schnittstellen verwendet werden, die reagieren, wenn ein Objekt bewegt oder geschüttelt wird (**bit.ly/ArduinoStoreTiltSensor**).

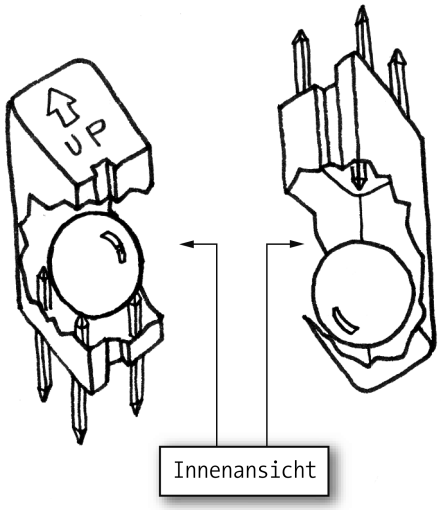


Abbildung 5-1.
Das Innere eines Neigungssensors

Ein weiterer Sensor, den Sie vielleicht verwenden möchten, ist der Infrarotsensor, wie Sie ihn bei Alarmanlagen finden (sie werden auch als passive Infrarotsensoren oder PIR-Sensoren bezeichnet, siehe Abbildung 5-2). Dieses kleine Bauteil löst einen Alarm aus, wenn eine Person (oder ein anderes Lebewesen) sich in der näheren Umgebung bewegt. Es bietet eine einfache Möglichkeit, Bewegung festzustellen.

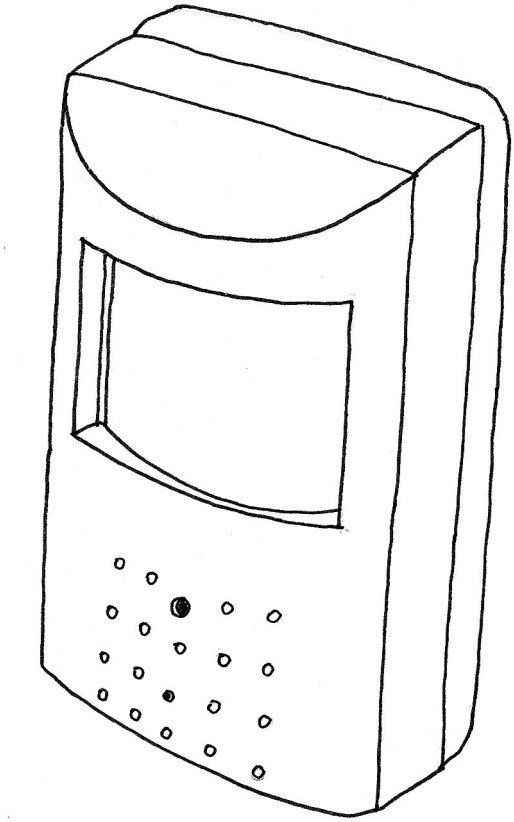


Abbildung 5-2.
Ein typischer PIR-Sensor

Sie sollten nun ein wenig mit all den möglichen Bauteilen, die über zwei solcher nah beieinanderliegenden Kontakte verfügen, herumexperimentieren, z.B. mit dem Thermostat, der die Raumtemperatur festlegt (verwenden Sie einen alten, der nicht mehr angeschlossen ist). Sie können auch zwei Kontakte nebeneinander platzieren und sie mit Wasser bespritzen.

Wenn Sie beispielsweise das letzte Projekt aus dem Kapitel 4 mit einem PIR-Sensor kombinieren, können Sie eine Lampe bauen, die auf die Anwesenheit von Personen reagiert. Sie könnten auch einen Neigungssensor

verwenden und so einen Leuchtkörper konzipieren, der sich ausschaltet, wenn Sie ihn zu einer Seite neigen.

Steuerung von Licht mittels PWM

Mit den bisher erworbenen Kenntnissen könnten Sie eine interaktive Lampe bauen – und zwar eine, die nicht nur über einen langweiligen Ein/Aus-Schalter verfügt, sondern vielleicht mit eleganteren Features aufwartet. Eine der Beschränkungen unserer Beispiele mit einer blinkenden LED war die, dass das Licht nur ein- oder ausgeschaltet werden konnte. Eine schicke Lampe muss aber auch dimmbar sein. Hierzu können wir uns eines Phänomens bedienen, das viele solcher Dinge wie Fernsehen oder Kino erst ermöglicht: die Trägheit des Auges.

Nach dem ersten Beispiel in Kapitel 4 wurde schon angedeutet, dass, wenn Sie die Angaben für die Verzögerungen im Code so ändern, dass Sie kein Blinken der LED mehr wahrnehmen können, das von den LEDs abgegebene Licht gegenüber ihrer normalen Helligkeit um 50 % reduziert erscheint. Ändern Sie nun die Angaben so, dass die LED ein Viertel der Zeit ausgeschaltet ist. Wenn Sie den Sketch ausführen, werden Sie sehen, dass die Helligkeit noch etwa 25% beträgt. Diese Technik heißt **Pulsweitenmodulation (PWM)**, eine hübsche Bezeichnung dafür, dass wenn Sie die LED nur schnell genug blinken lassen, Sie das Blinken nicht mehr wahrnehmen, aber die Helligkeit ändern können, indem Sie das Verhältnis von Ein- und Ausphasen entsprechend anpassen. In Abbildung 5-3 ist dargestellt, wie dies funktioniert.

Diese Technik funktioniert auch bei anderen Bauteilen als LEDs. Die Schnelligkeit eines Motors lässt sich auf dieselbe Weise ändern.

Wenn Sie herumexperimentieren, werden Sie bemerken, dass das Erzeugen von Blinkverhalten bei einer LED durch Einfügen von Verzögerungen im Code recht ungünstig sein kann, weil, sobald Sie einen Sensor auslesen oder Daten an den seriellen Anschluss schicken möchten, die LED zu flackern beginnt, während sie darauf wartet, dass das Auslesen des Sensors beendet wird. Glücklicherweise verfügt der Pozessor, der bei Arduino verwendet wird, über eine spezielle Hardware, die recht effizient drei LEDs blinken lassen kann, während der Sketch etwas anderes tut. Diese Hardware ist in den Pins 9, 10 und 11 implementiert, die wiederum über die Anweisung *analogWrite()* gesteuert werden können.

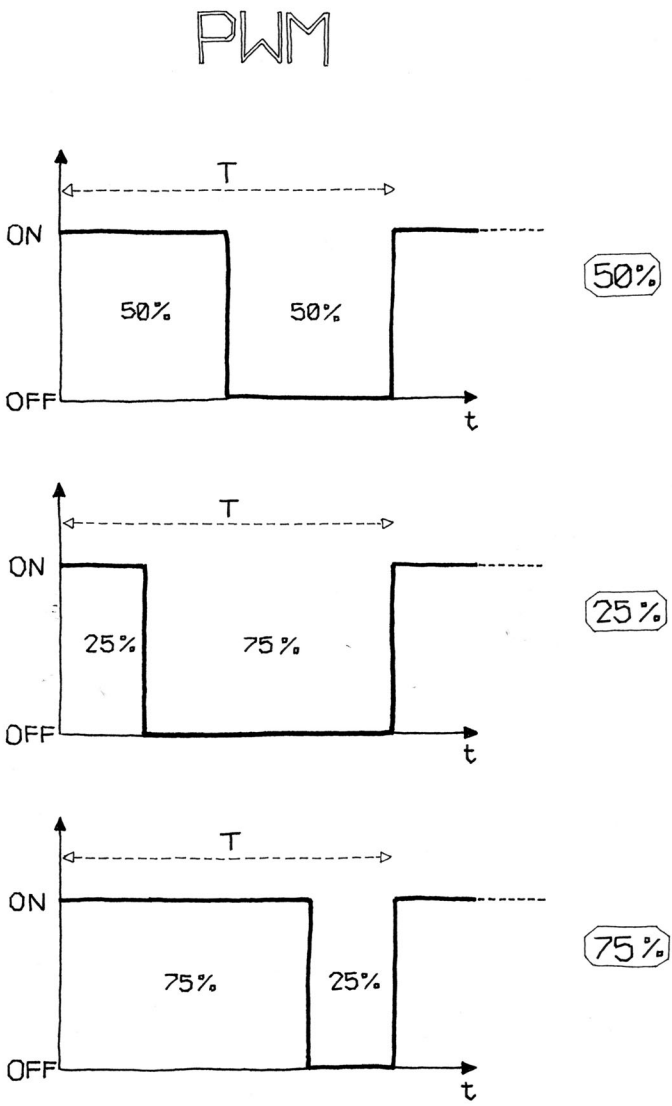


Abbildung 5-3.
PWM in Aktion

Wenn Sie zum Beispiel *analogWrite(9,128)* schreiben, wird die Helligkeit der LED, die an Pin 9 angeschlossen ist, auf 50% gedimmt. Warum 128? *analogWrite()* erwartet als Argument eine Zahl zwischen 0 und 255, wobei 255 volle Helligkeit bedeutet und bei 0 die LED ausgeschaltet ist.

Hinweis: Es ist eine feine Sache, dass Ihnen drei Kanäle zur Verfügung stehen, weil Sie so rote, grüne und blaue LEDs kaufen und das Licht dann nach Belieben mischen können!

Das wollen wir nun einmal ausprobieren. Bauen Sie den Schaltkreis, der in Abbildung 5-4 abgebildet ist. Beachten Sie, dass die LEDs gepolt sind: Der lange Pin (positiv) sollte nach rechts zeigen, der kurze Pin (negativ) nach links. Außerdem ist bei den meisten Widerständen die negative Seite abgeflacht, wie in der Abbildung zu sehen ist. Verwenden Sie einen 270-Ohm-Widerstand (rot-violett-braun).

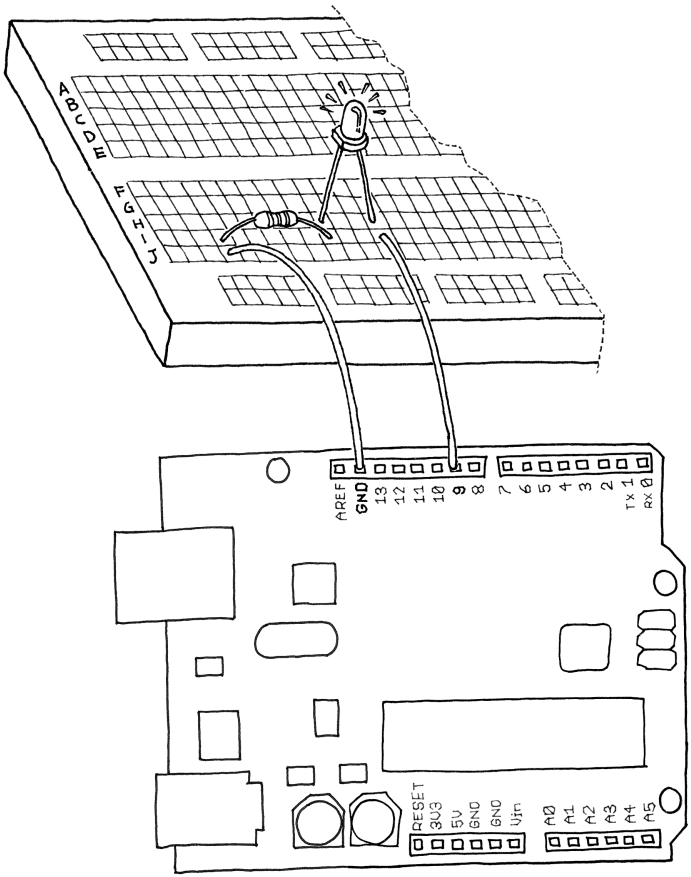


Abbildung 5-4.
LED am PWM-Pin

Erzeugen Sie dann in Arduino einen neuen Sketch und verwenden Sie Beispiel 5-1 (die Codebeispiele können auch unter **http://www.makezine.com/getstartedarduino** heruntergeladen werden):

```
// Beispiel 5-1: Fade an LED in and out

const int LED = 9; // the pin for the LED
int i = 0;        // We'll use this to count up and down

void setup() {
  pinMode(LED, OUTPUT); // tell Arduino LED is an output
}

void loop(){

  for (i = 0; i < 255; i++) { // loop from 0 to 254 (fade in)
    analogWrite(LED, i);     // set the LED brightness
    delay(10); // Wait 10ms because analogWrite
               // is instantaneous and we would
               // not see any change
  }

  for (i = 255; i > 0; i--) { // loop from 255 to 1 (fade out)

    analogWrite(LED, i); // set the LED brightness
    delay(10);          // Wait 10ms
  }

}
```

Sie haben nun ein nettes Laptop-Feature nachgebildet (vielleicht ist es auch ein wenig verschwenderisch, Arduino für so eine simple Angelegenheit zu nutzen). Wir wollen nun dieses Wissen einsetzen, um unsere Lampe zu verbessern.

Fügen Sie auf der Steckplatine die Schaltung hinzu, die wir verwendet haben, um den Drucktaster auszulesen (siehe weiter vorne in Kapitel 4). Versuchen Sie dabei, nicht auf die nächste Seite zu schauen, weil ich möchte, dass Sie damit anfangen, jeden hier gezeigten Basisschaltkreis als Baustein zu sehen, mit denen sich immer größere Projekte realisieren

lassen. Wenn Sie doch nachschauen müssen, macht das nichts; wichtig ist, dass Sie sich vorher ein paar Gedanken darüber gemacht haben, wie das Ganze aussehen könnte.

Um den gewünschten Schaltkreis zu bauen, müssen Sie die Schaltung, die Sie gerade gebaut haben (die aus Abbildung 5-4), mit der Drucktasterschaltung, die in Abbildung 4-6 dargestellt ist, kombinieren. Wenn Sie möchten, können Sie die beiden Schaltkreise auf verschiedene Bereiche der Platine bauen; es gibt hier viel Platz. Einer der Vorteile der Steckplatine (siehe Anhang A) besteht in den beiden Leiterbahnen, die horizontal über dem oberen und dem unteren Bereich verlaufen, eine ist rot (positiv) und die andere blau oder schwarz (Masse) gekennzeichnet.

Diese Leiterbahnen dienen dazu, Energie und Masse so zu verteilen, wie sie benötigt werden. Bei dem hier zu bauenden Schaltkreis müssen zwei Komponenten (beide sind Widerstände) mit dem Masse-(GND-)Pin des Arduino-Boards verbunden werden. Da Arduino über drei GND-Pins verfügt, können Sie die beiden Schaltkreise einfach auf die in jeder der beiden Abbildungen dargestellte Weise verbinden. Schließen Sie einfach beide an das Arduino-Board an. Sie können auch einen der Drähte der Masseleiterbahn der Steckplatine an einen der GND-Pins auf dem Arduio-Board anschließen und die Drähte, die in den Abbildungen an die GND-Pins angeschlossen sind, stattdessen mit der Masseleiterbahn auf der Steckplatine verbinden.

Wenn Sie noch nicht so weit sind, dies auszuprobieren, ist das nicht weiter schlimm: Verdrahten Sie einfach beide Schaltkreise mit dem Arduino-Board, wie es in den Abbildungen 4-6 und 5-4 dargestellt ist. Ein Beispiel, bei dem Masse und positive Leiterbahnen der Steckplatine genutzt werden, finden Sie im Kapitel 6.

Fahren wir nun mit unserem nächsten Beispiel fort. Wenn wir nur einen Drucktaster haben, wie können wir dann die Helligkeit der Lampe steuern? An dieser Stelle werden wir nun eine weitere Technik aus dem Interactive Design kennenlernen: das Ermitteln, wie lange ein Taster gedrückt wurde. Hierzu müssen wir ein Upgrade zu Beispiel 4-5 aus dem Kapitel 4 durchführen, um einen Dimm-Effekt einzubauen. Die Idee besteht darin, eine Schnittstelle herzustellen, über die mittels Drücken- bzw. Freigabe-Aktionen das Licht ein- oder ausgeschaltet und per Drücken- und Halten-Aktionen die Helligkeit geändert wird.

Schauen wir uns den entsprechenden Sketch an:

```
// Beispiel 5-2: Turn on LED when the button is pressed

const int LED = 9;    // the pin for the LED
const int BUTTON = 7;  // input pin of the pushbutton

int val = 0;    // stores the state of the input pin

int old_val = 0; // stores the previous value of "val"
int state = 0;   // 0 = LED off while 1 = LED on

int brightness = 128;       // Stores the brightness value
unsigned long startTime = 0; // when did we begin pressing?

void setup() {
  pinMode(LED, OUTPUT);   // tell Arduino LED is an output
  pinMode(BUTTON, INPUT); // and BUTTON is an input
}

void loop() {

  val = digitalRead(BUTTON); // read input value and store it
                         // yum, fresh

  // check if there was a transition
  if ((val == HIGH) && (old_val == LOW)) {

    state = 1 - state; // change the state from off to on
                   // or vice-versa

    startTime = millis(); // millis() is the Arduino clock
                      // it returns how many milliseconds
                      // have passed since the board has
                      // been reset.

    // (this line remembers when the button
    // was last pressed)
    delay(10);
  }
```

```
  // check whether the button is being held down
    if ((val == HIGH) && (old_val == HIGH)) {

      // If the button is held for more than 500ms.
      if (state == 1 && (millis() - startTime) > 500) {

        brightness++; // increment brightness by 1
        delay(10);    // delay to avoid brightness going
                      // up too fast

        if (brightness > 255) { // 255 is the max brightness

          brightness = 0; // if we go over 255
                          // let's go back to 0
        }
      }
    }

    old_val = val; // val is now old, let's store it

    if (state == 1) {
      analogWrite(LED, brightness); // turn LED ON at the
                                    // current brightness level

    } else {
      analogWrite(LED, 0); // turn LED OFF
    }
  }
```

Das wollen wir nun einmal ausprobieren. Wie Sie sehen, nimmt unser Interaktionsmodell Formen an. Wenn Sie den Taster drücken und sofort wieder loslassen, schalten Sie die Lampe ein bzw. aus. Wenn Sie den Taster gedrückt halten, ändert sich die Helligkeit. Nehmen Sie einfach den Finger vom Taster, wenn die gewünschte Helligkeit erreicht ist.

Als Nächstes wollen wir uns ein wenig näher mit der Verwendung einiger interessanter Sensoren beschäftigen.

Einsatz eines Lichtsensors anstelle eines Drucktasters

Wir werden nun ein kleines Experiment durchführen. Dazu benötigen Sie einen Lichtsensor, wie er in Abbildung 5-5 zu sehen ist. Solche Sensoren können Sie bei Maker Shed (Teilnummer JM169578) oder unter **bit.ly/ArduinoStoreLDR** kaufen.

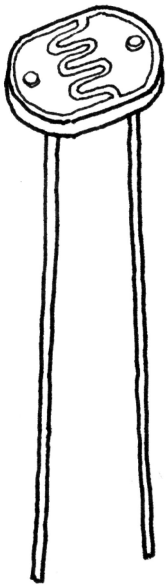


Abbildung 5-5.
Lichtabhängiger Widerstand (LDR)

Bei Dunkelheit weist der lichtabhängige Widerstand (Light Dependent Resistor, kurz LDR) einen hohen Widerstand auf. Wenn Licht auf ihn fällt, sinkt der Widerstand recht schnell und das Bauteil wird zu einem ziemlich guten Leiter für Elektrizität. Es handelt sich also um eine Art durch Licht aktivierten Schalter.

Bauen Sie den Schaltkreis aus Beispiel 4-2 (siehe Kapitel 4) und laden Sie dann den entsprechenden Code aus Beispiel 4-2 auf das Arduino-Board.

Stecken Sie nun anstelle des Drucktasters den LDR auf die Steckplatine. Ihnen wird sicherlich auffallen, dass die LED nicht mehr leuchtet, wenn Sie den LDR mit der Hand zudecken. Ist der LDR hingegen nicht abgedeckt, dann leuchtet die LED. Sie haben gerade Ihre erste sensorgesteuerte LED

konstruiert. Dies ist ein wichtiger Schritt, weil wir zum ersten Mal in diesem Buch eine elektronische Komponente verwenden, bei der es sich nicht einfach um ein mechanisches Bauteil handelt: Es ist ein wirklich komplexer Sensor.

Analoger Eingang

Wie Sie aus dem vorherigen Abschnitt bereits wissen, kann Arduino feststellen, ob eine Spannung an einem der Pins anliegt und das Ergebnis mittels der *digitalRead()*-Funktion zurückmelden. Diese Art von Entweder-oder-Antwort ist in vielen Anwendungen sinnvoll, aber der Lichtsensor, den wir hier verwenden, kann nicht nur mitteilen, ob Licht vorhanden ist, sondern auch wie viel. Das ist der Unterschied zwischen einem Ein/Aus-Sensor (der uns mitteilt, ob etwas vorhanden ist) und einem analogen Sensor, dessen Wert sich ständig ändert. Um einen solchen Sensor auszulesen, benötigen wir eine andere Art von Pin.

Im rechten unteren Bereich des Arduino-Boards befinden sich sechs Pins mit der Bezeichnung „Analog In“ Dabei handelt es sich um spezielle Pins, die uns nicht nur zurückliefern können, ob eine Spannung an ihnen anliegt, sondern gegebenenfalls auch den betreffenden Wert. Mithilfe der *analogRead()*-Funktion können wir die an einem der Pins anliegende Spannung auslesen. Sie liefert einen Wert zwischen 0 und 1023, der eine Spannung zwischen 0 und 5 Volt darstellt. Wenn beispielsweise eine Spannung von 2,5 Volt an Pin 0 anliegt, gibt *analogRead(0)* den Wert 512 zurück.

Wenn Sie nun den Schaltkreis bauen, der in Abbildung 5-6 dargestellt ist, dabei einen 10k-Widerstand verwenden und den Code ausführen, der in Beispiel 5-3 angeführt ist, werden Sie sehen, dass die Board-eigene LED (Sie können auch eine eigene LED verwenden und an Pin 13 und den GND-Pin anschließen, wie in Kapitel 4 dargestellt ist) in einer Geschwindigkeit blinkt, die von der Lichtmenge abhängt, die auf den Sensor trifft.

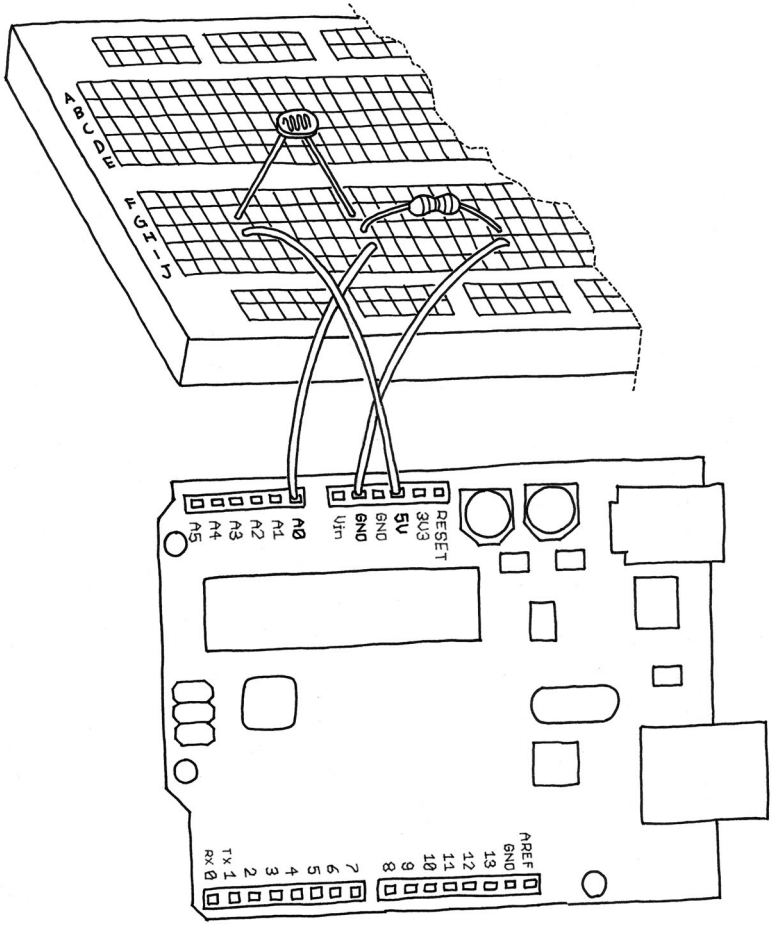


Abbildung 5-6.
Eine analoge Sensor-Schaltung

```
// Beispiel 5-3: Blink LED at a rate specified by the
                 value of the analogue input

const int LED = 13; // the pin for the LED

int val = 0;   // variable used to store the value
            // coming from the sensor
void setup() {
  pinMode(LED, OUTPUT); // LED is as an OUTPUT
```

```
  // Note: Analogue pins are
  // automatically set as inputs
}

void loop() {

  val = analogRead(0); // read the value from
                   // the sensor

  digitalWrite(LED, HIGH); // turn the LED on

  delay(val); // stop the program for
            // some time

  digitalWrite(LED, LOW); // turn the LED off

  delay(val); // stop the program for
            // some time
}
```

Probieren Sie nun Beispiel 5-4 aus, aber zuvor müssen Sie Ihren Schaltkreis modifizieren. Schauen Sie sich noch einmal Abbildung 5-4 an und verbinden Sie Ihre LED wie gezeigt mit Pin 9. Weil Sie bereits einige Komponenten auf Ihrer Steckplatine angebracht haben, müssen Sie erst eine Stelle finden, an der die LED, die Drähte und der Widerstand sich nicht mit dem LDR-Schaltkreis überschneiden.

```
// Beispiel 5-4: Set the brightness of LED

const int LED = 9;  // the pin for the LED

int val = 0;   // variable used to store the value
             // coming from the sensor

void setup() {

  pinMode(LED, OUTPUT); // LED is as an OUTPUT

  // Note: Analogue pins are
```

```
  // automatically set as inputs
}

void loop() {

  val = analogRead(0); // read the value from
                    // the sensor
  analogWrite(LED, val/4); // turn the LED on at
                        // the brightness set
                        // by the sensor

  delay(10); // stop the program for
           // some time
}
```

Hinweis: Wir legen die Helligkeit fest, indem wir *val* durch 4 teilen, weil *analogRead()* eine Zahl größer als 1023 zurückliefert und *analogWrite()* nur einen Maximalwert von 255 annehmen kann.

Der Einsatz anderer analoger Sensoren

In dem Schaltkreis aus dem vorherigen Abschnitt lassen sich auch zahlreiche andere resistive Sensoren einbauen, die alle mehr oder weniger nach demselben Prinzip funktionieren. Sie könnten beispielsweise einen Temperaturfühler anschließen, ein einfaches Bauteil, dessen Widerstand sich mit der Temperatur ändert. Ich habe bereits erläutert, wie Änderungen bei den Widerständen zu Änderungen bei der Spannung führen, die dann von Arduino gemessen werden können.

Wenn Sie einen Temperaturfühler verwenden, beachten Sie bitte, dass kein direkter Zusammenhang zwischen dem Wert, den Sie auslesen und der tatsächlich gemessenen Temperatur besteht. Wenn Sie eine exakte Angabe benötigen, sollten Sie die Werte auslesen, die vom analogen Pin kommen und dabei für die Messung ein reales Thermometer verwenden. Die entsprechenden Werte sollten Sie dann nebeneinander in einer Tabelle eintragen und einen Weg finden, wie sich die analogen Werte mit den realen Temperaturen abgleichen lassen.

Bis jetzt haben wir eine LED als Ausgabekomponente verwendet. Doch wie können wir die tatsächlichen Werte erhalten, die Arduino vom Sensor ausliest? Wir können das Board nicht veranlassen, die Werte im Morsealphabet zu blinken (d.h., das wäre schon möglich, es gibt aber einen

einfacheren Weg, die Werte zu lesen). Dazu müssen wir Arduino über einen seriellen Anschluss mit einem Computer kommunizieren lassen. Dies wird im nächsten Abschnitt beschrieben.

Serielle Kommunikation

Zu Beginn des Buches haben Sie erfahren, dass Arduino über einen USB-Anschluss verfügt, über den die IDE Code in den Prozessor lädt. Die gute Nachricht ist die, dass diese Verbindung auch von den Sketches, die in Arduino geschrieben werden, genutzt werden kann, um Daten zurück an den Computer zu liefern oder um Befehle von diesem zu empfangen. Zu diesem Zweck verwenden wir ein serielles Objekt (ein **Objekt** ist eine Sammlung von Fähigkeiten, die gebündelt wurden, um das Schreiben von Sketches zu erleichtern).

Dieses Objekt enthält all den Code, der für das Senden und Empfangen von Daten erforderlich ist. Wir werden nun den letzten Schaltkreis mit dem Fotowiderstand, den wir gebaut haben, verwenden und die ausgelesenen Daten zurück an den Computer senden. Erstellen Sie mit dem folgenden Code einen neuen Sketch (Sie können ihn auch unter **http://www.make-zine.com/getstartedarduino** herunterladen):

```
// Beispiel 5-5: Send to the computer the values read from
                 analogue input 0

// Make sure you click on "Serial Monitor"
// after you upload
const int SENSOR = 0;  // select the input pin for the
                    // sensor resistor

int val = 0; // variable to store the value coming
           // from the sensor

void setup() {

 Serial.begin(9600); // open the serial port to send
                  // data back to the computer at
                  // 9600 bits per second
}

void loop() {
```

```
  val = analogRead(SENSOR); // read the value from
                          // the sensor

  Serial.println(val); // print the value to
                    // the serial port

  delay(100); // wait 100ms between
            // each send
}
```

Wenn Sie den Code auf das Arduino-Board übertragen haben, klicken Sie auf die Schaltfläche *Serial Monitor* in der Arduino-IDE (die letzte Schaltfläche in der Tool-Leiste). Sie sehen dann die Zahlen an den unteren Rand des Fensters wandern. Nun kann jede Software, die vom seriellen Anschluss auslesen kann, auch mit Arduino kommunizieren. Es gibt viele Programmiersprachen, mit denen Sie Programme auf Ihrem Computer schreiben können, die mit dem seriellen Anschluss kommunizieren können. Processing (**http://www.processing.org**) ist eine großartige Ergänzung zu Arduino, weil die Sprachen und die IDEs sehr ähnlich sind.

Der Umgang mit größeren Lasten

Jeder der Pins des Arduino-Boards kann verwendet werden, um Bauteile zu betreiben, die bis zu 20 Milliampere verbrauchen: Das ist eine sehr geringe Strommenge, die gerade ausreicht, um eine LED zu betreiben. Wenn Sie z.B. einen Motor betreiben möchten, wird der Pin sofort seine Arbeit einstellen und möglicherweise den gesamten Prozessor durchbrennen. Um also größere Lasten wie Motoren oder Glühlampen zu betreiben, müssen wir auf eine externe Komponente zurückgreifen, die solche Bauteile ein- oder ausschalten kann und von einem Arduino-Pin betrieben wird. Ein solches Gerät ist der sogenannte MOSFET-Transistor – ignorieren Sie den merkwürdigen Namen einfach. Es handelt sich hierbei um einen elektronischen Schalter, der betrieben werden kann, indem einfach eine Spannung an einen seiner drei Pins angelegt wird. Diese Pins werden zusammen als Gate bezeichnet. Dieser Transistor funktioniert ähnlich wie unser Lichtschalter zu Hause. Die Fingerbewegung, mit der wir das Licht ein- und ausschalten, ist allerdings durch einen Pin auf dem Arduino-Board ersetzt, der elektrische Spannung an das Gate des MOSFET sendet.

Hinweis: MOSFET steht für Metal Oxide Semiconductor Field Effect Transistor. Es handelt sich um einen speziellen Transistortyp, der basierend auf dem Feldeffektprinzip funktioniert. Das heißt, dass Elektrizität durch einen Bereich aus Halbleitermaterial (zwischen den Drain- und den Source-Pins) fließt, wenn Spannung am Gate-Pin anliegt. Da der Gate-Pin durch eine Metalloxidschicht von den anderen isoliert ist, fließt kein Strom vom Arduino-Board in den MOSFET, wodurch dieser sich sehr einfach als Schnittstelle nutzen lässt. Diese Art von Transistor ist ideal, um größere Lasten in hoher Frequenz ein- oder auszuschalten.

In Abbildung 5-7 ist dargestellt, wie Sie einen MOSFET wie den IRF520 nutzen können, um einen kleinen Motor eines Ventilators an- oder auszuschalten. Sie sehen außerdem, dass der Motor seine Energiezufuhr von dem 9-V-Anschluss des Arduino-Boards bezieht. Dies ist ein weiterer Vorteil des MOSFET: Er ermöglicht das Betreiben von Geräten, deren Energiezufuhr sich von der von Arduino genutzten unterscheidet. Weil der MOSFET an Pin 9 angeschlossen ist, können wir außerdem *analogWrite()* verwenden, um die Motorgeschwindigkeit mittels PWM zu steuern. Um den Schaltkreis nachzubauen, benötigen Sie einen IRF520 MOSFET (**bit.ly/ArduinoStoreIRF520**) und eine 1N4007-Diode (**bit.ly/ArduinoStore1N4007**). Wenn der Motor während des Uploads ungewollt anspringt, platzieren Sie einen 10-K-Widerstand zwischen Pin 9 und den GND-Pin.

Komplexe Sensoren

Wir definieren **komplexe Sensoren** als Sensoren, die solche Informationen produzieren, deren Nutzung ein wenig mehr erfordert als den Einsatz einer *digitalRead()*- oder *analogRead()*-Funktion. Es handelt sich üblicherweise um kleine Schaltkreise, die einen kleinen Mikrocontroller enthalten, der die Informationen vorverarbeitet. Einige dieser komplexen Sensoren enthalten Ultraschall-Ranger, Infrarot-Ranger und Beschleunigungsmesser. Beispiele für ihre Verwendung finden Sie auf unserer Webseite im Abschnitt Tutorials (**http://www.arduino.cc/en/Tutorial/HomePage**).

Im Buch *Making Things Talk* – Die Welt hören, sehen, fühlen von Tom Igoe (erschienen bei O'Reilly, ISBN 978-3-86899-162-8) werden diese und andere komplexe Sensoren ausführlich erläutert.

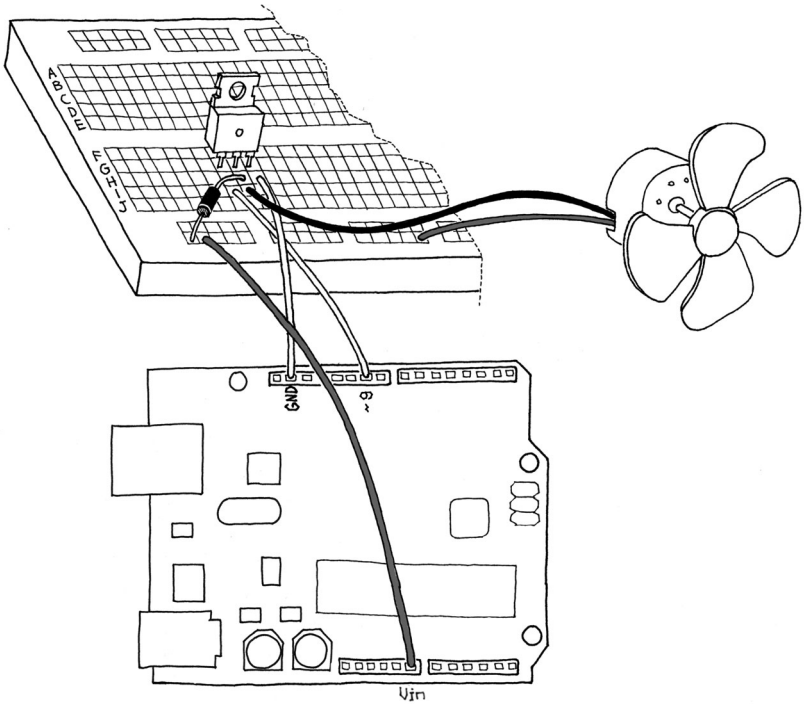


Abbildung 5-7.
Ein Motor-Schaltkreis für Arduino

6/Kommunikation mit der Cloud

In den vorangegangenen Kapiteln haben Sie die Grundlagen von Arduino und die grundlegenden zur Verfügung stehenden Bausteine kennengelernt. Ich möchte hier noch einmal die Bestandteile das Arduino-Alphabets auflisten:

Digitaler Output
Wir verwenden ihn zur Steuerung einer LED, aber mit einem entsprechenden Schaltkreis lassen sich hierüber auch Motoren steuern, Sounds erzeugen und viele weitere Dinge umsetzen.

Analoger Output
Er bietet uns die Möglichkeit, die Helligkeit einer LED zu steuern, anstatt sie nur ein- oder auszuschalten. Wir können hiermit sogar die Geschwindigkeit eines Motors steuern.

Digitaler Input
Dieser ermöglicht es uns, den Zustand einfacher Sensoren wie Drucktaster oder Neigungsschalter auszulesen.

Analoger Input
Wir können Signale von Sensoren auslesen, die kontinuierlich Daten senden, die nicht einfach „Ein“ oder „Aus“ bedeuten, z.B. solche von einem Potentiometer oder einem Lichtsensor.

Serielle Kommunikation
Dies ermöglicht es uns, mit einem Computer zu kommunizieren, Daten mit diesem auszutauschen oder einfach zu beobachten, was mit dem Sketch geschieht, der auf dem Arduino-Board ausgeführt wird.

In diesem Kapitel werden Sie erfahren, wie wir eine funktionierende Anwendung zusammenbauen, wobei wir die in den vorherigen Kapiteln gewonnenen Kenntnisse einfließen lassen. In diesem Kapitel sollte deutlich werden, wie jedes einzelne Beispiel als Baustein für ein komplexes Projekt genutzt werden kann.

An dieser Stelle wird der Möchtegerndesigner in mir geweckt. Wir werden eine Version des 21. Jahrhunderts einer Lampe meines italienischen Lieblingsdesigners Joe Colombo bauen. Das Objekt, das wir bauen werden, ist inspiriert von der Lampe „Aton“ aus dem Jahr 1964.

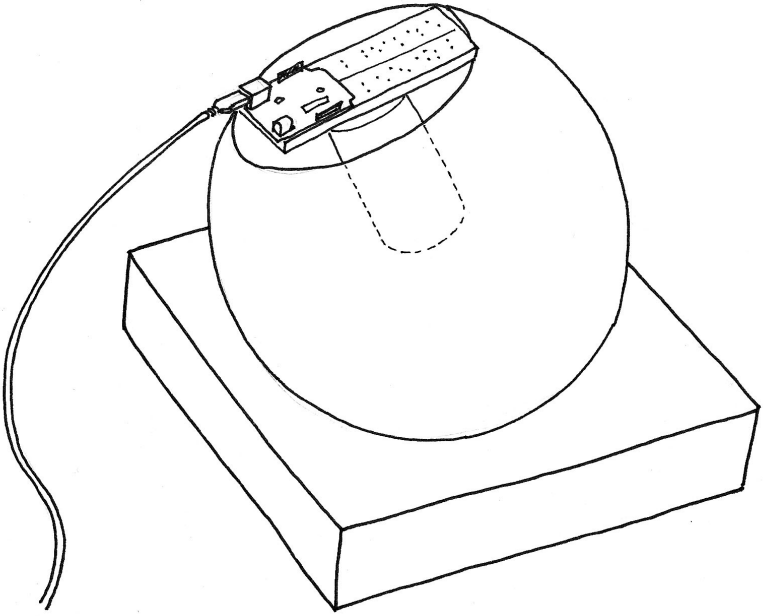


Abbildung 6-1.
Die fertige Lampe

Die Lampe ist, wie Sie in Abbildung 6-1 sehen können, eine einfache Kugel, die auf einem Sockel sitzt, der ein Loch hat, um zu verhindern, dass die Kugel von Ihrem Schreibtisch rollt. Durch dieses Design kann die Lampe in verschiedene Richtungen ausgerichtet werden.

Hinsichtlich der Funktionalität möchten wir ein Gerät bauen, das mit dem Internet verbunden werden kann, die aktuelle Liste der Artikel im Make-Blog (**blog.makezine.com**) abruft und zählt, wie oft die Wörter „peace“, „love“ und „Arduino“ vorkommen. Mit diesen Werten möchten wir dann eine Farbe erzeugen, die von der Lampe wiedergegeben wird. Die Lampe

selbst verfügt über einen Drucktaster zum Ein- und Ausschalten und einen Lichtsensor für eine automatische Aktivierung.

Planung

Schauen wir uns nun die Umsetzung sowie Bauteile und Komponenten an, die dazu erforderlich sind. Zuallererst benötigen wir Arduino, um uns mit dem Internet zu verbinden. Da Arduino nur über einen USB-Anschlusss verfügt, können wir keinen direkten Anschluss zum Internet herstellen, daher müssen wir uns eine entsprechende Überbrückung einfallen lassen. Normalerweise wird in einem solchen Fall eine Anwendung auf dem Computer ausgeführt, die sich mit dem Internet verbindet, die Daten verarbeitet und Arduino einige einfache, herausgefilterte Informationen sendet.

Das Arduino-Board ist ein einfacher Computer mit einem kleinen Speicher; das Verarbeiten von großen Dateien ist also schwierig, und wenn eine Verbindung zu einem RSS-Feed hergestellt wird, erhalten wir eine sehr umfangreiche XML-Datei, die sehr viel RAM erfordert. Wir werden daher mittels Processing ein Proxy-Programm implementieren, um die XML-Datei zu vereinfachen.

Processing

Processing war der Ursprung von Arduino. Wir lieben diese Sprache und nutzen sie, um Einsteigern das Programmieren beizubringen und um schönen Code zu schreiben. Processing und Arduino bilden eine perfekte Kombination. Ein weiterer Vorteil besteht darin, dass Processing als Open-Source zur Verfügung steht und auf allen größeren Plattformen verwendet werden kann (Mac, Linux und Windows). Es lassen sich hiermit auch eigenständige Anwendungen erzeugen, die auf diesen Plattformen ausgeführt werden können. Darüber hinaus gibt es eine lebhafte und hilfreiche Processing-Community und Sie finden Tausende von fertigen Beispielprogrammen.

Das Proxy-Programm erledigt folgende Aufgaben: Es lädt den RSS-Feed unter **makezine.com** herunter und extrahiert alle Wörter aus der resultierenden XML-Datei. Dann durchläuft es alle Wörter und zählt die Vorkommen von „peace“, „love“ und „Arduino“ im Text. Mit diesen drei Zahlen berechnen wir den Farbwert und senden ihn an Arduino. Das Board liefert die vom Sensor gemessene Lichtmenge zurück, die dann auf dem Computerbildschirm angezeigt wird.

Auf der Hardwareseite kombinieren wir die Beispiele Drucktaster, Lichtsensor und LED-Steuerung mittels PWM (mal 3!) und serieller Kommunikation.

Weil es sich bei Arduino um ein einfaches Gerät handelt, müssen wir die Farben auf einfache Weise kodieren. Wir nutzen dabei den Standard, nach dem Farben in HTML dargestellt werden: # gefolgt von sechs hexadezimalen Zahlen.

Hexadezimale Zahlen sind sehr praktisch, weil jede 8-Bit-Zahl in genau zwei Zeichen gespeichert wird; bei Dezimalzahlen reicht die Bandbreite von einem bis zu drei Zeichen. Vorhersagbarkeit macht den Code ebenfalls einfacher: Wir warten, bis wir ein # sehen, dann lesen wir die sechs nachfolgenden Zeichen in einen *Puffer* (eine Variable, die als temporärer Aufbewahrungsort von Daten dient) ein. Anschließend wandeln wir jede der Gruppen, die aus zwei Zeichen bestehen, in ein Byte um, das die Helligkeit einer der drei LEDs repräsentiert.

Der Code

Es werden zwei Sketches ausgeführt: ein Processing -Sketch und ein Arduino-Sketch. Im Folgenden sehen Sie den Code für den Processing-Sketch; Sie können ihn unter **http://www.makezine.com/getstarted arduino** herunterladen.

```
// Beispiel 6-1: Arduino networked lamp
// parts of the code are inspired
// by a blog by Tod E. Kurt (todbot.com)

import processing.serial.*;

String feed = "http://blog.makezine.com/index.xml";

int interval = 10;  // retrieve feed every 60 seconds;
int lastTime;      // the last time we fetched the content

int love    = 0;
int peace   = 0;
int arduino = 0;

int light = 0;  // light level measured by the lamp
```

```
Serial port;
color c;
String cs;

String buffer = ""; // Accumulates characters coming from Arduino

PFont font;

void setup() {
  size(640,480);
  frameRate(10);    // we don't need fast updates

  font = loadFont("HelveticaNeue-Bold-32.vlw");
  fill(255);
  textFont(font, 32);

  // IMPORTANT NOTE:
  // The first serial port retrieved by Serial.list()
  // should be your Arduino. If not, uncomment the next
  // line by deleting the // before it, and re-run the
  // sketch to see a list of serial ports. Then, change
  // the 0 in between [ and ] to the number of the port
  // that your Arduino is connected to.
  //println(Serial.list());
  String arduinoPort = Serial.list()[0];
  port = new Serial(this, arduinoPort, 9600); // connect to Arduino

  lastTime = 0;
  fetchData();
}

void draw() {
  background( c );
  int n = (interval - ((millis()-lastTime)/1000));

  // Build a colour based on the 3 values
  c = color(peace, love, arduino);
  cs = "#" + hex(c,6); // Prepare a string to be sent to Arduino

  text("Arduino Networked Lamp", 10,40);
  text("Reading feed:", 10, 100);
  text(feed, 10, 140);
```

```
text("Next update in "+ n + " seconds",10,450);
text("peace" ,10,200);
text(" " + peace, 130, 200);
rect(200,172, peace, 28);

text("love ",10,240);
text(" " + love, 130, 240);
rect(200,212, love, 28);

text("arduino ",10,280);
text(" " + arduino, 130, 280);
rect(200,252, arduino, 28);

// write the colour string to the screen
text("sending", 10, 340);
text(cs, 200,340);

text("light level", 10, 380);
rect(200, 352,light/10.23,28); // this turns 1023 into 100

if (n <= 0) {
  fetchData();
  lastTime = millis();
}

port.write(cs); // send data to Arduino

if (port.available() > 0) { // check if there is data waiting
  int inByte = port.read(); // read one byte
  if (inByte != 10) { // if byte is not newline
    buffer = buffer + char(inByte); // just add it to the buffer
  }
  else {

    // newline reached, let's process the data
    if (buffer.length() > 1) { // make sure there is enough data

      // chop off the last character, it's a carriage return
      // (a carriage return is the character at the end of a
      // line of text)
      buffer = buffer.substring(0,buffer.length() -1);
```

```
        // turn the buffer from string into an integer number
        light = int(buffer);

        // clean the buffer for the next read cycle
        buffer = "";

        // We're likely falling behind in taking readings
        // from Arduino. So let's clear the backlog of
        // incoming sensor readings so the next reading is
        // up-to-date.
        port.clear();
      }
    }
  }

void fetchData() {
  // we use these strings to parse the feed
  String data;
  String chunk;

  // zero the counters
  love    = 0;
  peace   = 0;
  arduino = 0;
  try {
    URL url = new URL(feed);  // An object to represent the URL
    // prepare a connection
    URLConnection conn = url.openConnection();
    conn.connect(); // now connect to the Website

    // this is a bit of virtual plumbing as we connect
    // the data coming from the connection to a buffered
    // reader that reads the data one line at a time.
    BufferedReader in = new
      BufferedReader(new InputStreamReader(conn.getInputStream()));

    // read each line from the feed
    while ((data = in.readLine()) != null) {

      StringTokenizer st =
        new StringTokenizer(data,"\"<>,.()[] ");// break it down
```

```
      while (st.hasMoreTokens()) {
        // each chunk of data is made lowercase
        chunk= st.nextToken().toLowerCase() ;

        if (chunk.indexOf("love") >= 0 ) // found "love"?
          love++;    // increment love by 1
        if (chunk.indexOf("peace") >= 0)   // found "peace"?
          peace++;   // increment peace by 1
        if (chunk.indexOf("arduino") >= 0) // found "arduino"?
          arduino++; // increment arduino by 1
      }
    }

    // Set 64 to be the maximum number of references we care about.
    if (peace > 64)   peace = 64;
    if (love > 64)    love = 64;
    if (arduino > 64) arduino = 64;
    peace = peace * 4;     // multiply by 4 so that the max is 255,
    love = love * 4;       // which comes in handy when building a
    arduino = arduino * 4; // colour that is made of 4 bytes (ARGB)
  }
  catch (Exception ex) { // If there was an error, stop the sketch
    ex.printStackTrace();
    System.out.println("ERROR: "+ex.getMessage());
  }

}
```

Zwei Dinge gilt es noch zu tun, bevor der Processing-Sketch einwandfrei läuft. Zunächst muss Processing angewiesen werden, die Schriftart zu erzeugen, die wir für den Sketch verwenden. Dazu müssen Sie den Sketch zunächst erstellen und speichern. Klicken Sie dann bei geöffnetem Sketch auf das *Tools*-Menü von Processing und dann auf *Create Font*. Wählen Sie die Schrift mit dem Namen HelveticaNeue-Bold aus, und geben Sie als Schriftgröße den Wert 32 an. Klicken Sie anschließend auf *OK*.

Als zweiten Schritt müssen Sie bestätigen, dass Arduino den korrekten seriellen Anschluss für die Kommunikation mit Arduino verwendet. Dazu müssen Sie zunächst den Arduino-Schaltkreis zusammenbauen und den Arduino-Sketch hochladen. Auf den meisten Systemen wird der Processing-Sketch einwandfrei laufen. Wenn allerdings auf dem Arduino-Board

nichts geschieht und auch keine Informationen vom Lichtsensor auf dem Bildschirm angezeigt werden, lesen Sie sich den Kommentar unter der Überschrift „IMPORTANT NOTE“ im Processing-Sketch durch und folgen Sie den betreffenden Anweisungen.

Hier nun der Arduino-Sketch (er ist auch unter **http://www.makezine.com/getstartedarduino** verfügbar):

```
// Beispiel 6-2: Arduino Networked Lamp

const int SENSOR = 0;
const int R_LED = 9;
const int G_LED = 10;
const int B_LED = 11;
const int BUTTON = 12;

int val = 0; // variable to store the value coming from the sensor

int btn = LOW;
int old_btn = LOW;
int state = 0;
char buffer[7] ;
int pointer = 0;
byte inByte = 0;

byte r = 0;
byte g = 0;
byte b = 0;

void setup() {
  Serial.begin(9600);  // open the serial port
  pinMode(BUTTON, INPUT);
}

void loop() {
  val = analogRead(SENSOR); // read the value from the sensor
  Serial.println(val);     // print the value to
                          // the serial port

  if (Serial.available() >0) {
```

```
  // read the incoming byte:
  inByte = Serial.read();

  // If the marker's found, next 6 characters are the colour
  if (inByte == '#') {

    while (pointer < 6) { // accumulate 6 chars
      buffer[pointer] = Serial.read(); // store in the buffer
      pointer++; // move the pointer forward by 1
    }

    // now we have the 3 numbers stored as hex numbers
    // we need to decode them into 3 bytes r, g and b
    r = hex2dec(buffer[1]) + hex2dec(buffer[0]) * 16;
    g = hex2dec(buffer[3]) + hex2dec(buffer[2]) * 16;
    b = hex2dec(buffer[5]) + hex2dec(buffer[4]) * 16;

    pointer = 0; // reset the pointer so we can reuse the buffer

  }
}

btn = digitalRead(BUTTON); // read input value and store it

// Check if there was a transition
if ((btn == HIGH) && (old_btn == LOW)){
  state = 1 - state;
}

old_btn = btn; // val is now old, let's store it

if (state == 1) { // if the lamp is on

  analogWrite(R_LED, r);  // turn the leds on
  analogWrite(G_LED, g);  // at the colour
  analogWrite(B_LED, b);  // sent by the computer
} else {

  analogWrite(R_LED, 0);  // otherwise turn off
  analogWrite(G_LED, 0);
  analogWrite(B_LED, 0);
 }
```

```
  delay(100);              // wait 100ms between each send
}

int hex2dec(byte c) { // converts one HEX character into a number
    if (c >= '0' && c <= '9') {
      return c - '0';
    } else if (c >= 'A' && c <= 'F') {
      return c - 'A' + 10;
    }
}
```

Das Zusammenbauen des Schaltkreises

In Abbildung 6-2 ist dargestellt, wie der Schaltkreis zusammenzubauen ist. Bei allen im Diagramm gezeigten und verwendeten Widerständen handelt es sich um 10K-Widerstände, obwohl Sie bei den Widerständen für die LEDs auch mit geringeren Werten arbeiten können.

Von dem PWM-Beispiel in Kapitel 5 wissen Sie, dass LEDs gepolt sind. In unserem Schaltkreis hier sollte der lange Pin (positiv) nach rechts und der kurze Pin (negativ) nach links zeigen. (Bei den meisten LEDs ist die negative Seite abgeflacht, wie auch in der Abbildung zu sehen ist.)

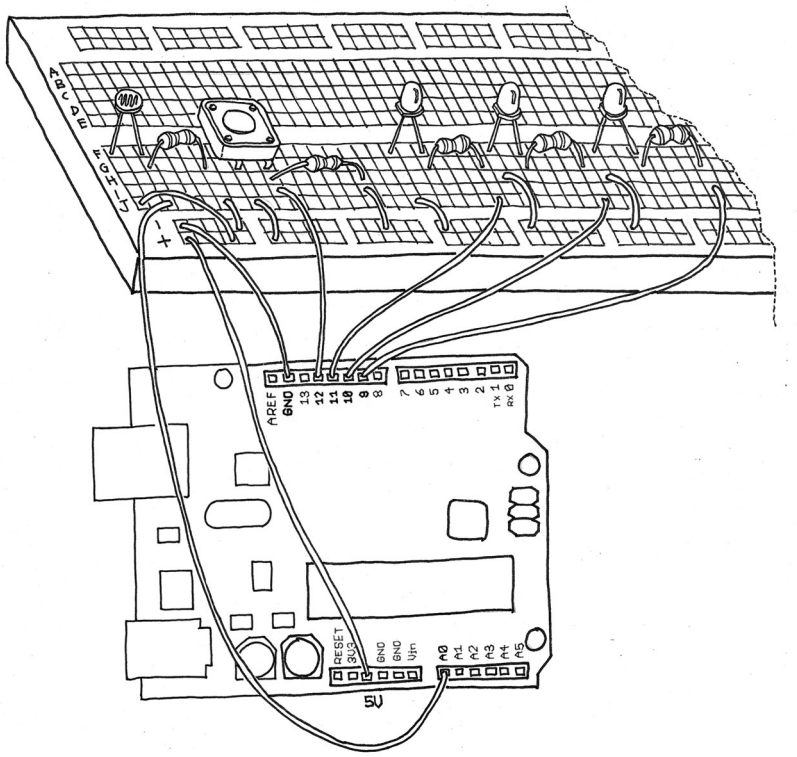


Abbildung 6-2.
Die "Arduino Networked Lamp"-Schaltung

Bauen Sie die Schaltung wie dargestellt nach und verwenden Sie dabei eine rote, eine grüne und eine blaue LED. Laden Sie die Sketches in Arduino und Processing und führen Sie sie dann aus. Falls Probleme auftreten, schlagen Sie im Kapitel 7 nach.

Nun wollen wir die Konstruktion vervollständigen, indem wir die Steckplatine in einer Glaskugel platzieren. Die preiswerteste Möglichkeit ist dabei die, bei IKEA einfach eine Tischlampe vom Typ Fado zu kaufen. Sie kostet zurzeit etwa 19,99 US-Dollar bzw. 14,99/11,99 Euro (ahh, the luxury of being European).

Anstelle von drei separaten LEDs können Sie auch eine einzelne RGB-LED verwenden, die über vier Anschlüsse verfügt. Sie lässt sich auf dieselbe Weise anschließen wie die LEDs aus Abbildung 6-2, mit einem Unterschied: Anstelle von drei separaten Verbindungen zum Masse-Pin auf dem Ardui-

no-Board gibt es nur einen einzigen Anschluss zur Masse (der als gemeinsame Kathode bezeichnet wird).

Im Arduino-Store wird eine RGB-LED mit vier Anschlüssen recht preisgünstig vertrieben (**bit.ly/ArduinoStoreRGBLed**). Anders als bei den separaten einfarbigen LEDs ist der längste Anschluss der RGB-LED der, der zur Masse führt. Die anderen drei werden mit Pin 9, 10 und 11 des Arduino-Boards verbunden (mit jeweils einem Widerstand zwischen den Anschlüssen und den Pins, genau wie bei den separaten roten, grünen und blauen LEDs).

So funktioniert das Zusammenbauen

Packen Sie die Lampe aus und entfernen Sie das Kabel, das in den Sockel der Lampe führt. Sie werden sie nicht mehr an die Steckdose anschließen.

Befestigen Sie das Arduino-Board auf einer Steckplatine und kleben Sie die Platine mit Heißkleber auf der Rückseite der Lampe fest.

Löten Sie längere Drähte an die RGB-LED und kleben Sie sie an die Stelle, an der sich normalerweise die Glühbirne befindet. Verbinden Sie die Drähte der LED mit dem Breadboard (an der Stelle, an der sie sich befanden, bevor Sie sie entfernt haben). Denken Sie daran, dass Sie nur eine Verbindung zur Masse benötigen, wenn Sie eine RGB-LED mit vier Anschlüssen verwenden.

Suchen Sie entweder ein hübsches Holzstück mit einem Loch in der Mitte, das als Sockel für die Kugel dienen kann oder schneiden Sie einfach den oberen Teil des Pappkartons, in dem die Lampe geliefert wurde, so zurecht, dass er eine Höhe von etwa 5 cm hat. Schneiden Sie dann ein Loch mit einem solchen Durchmesser aus, dass die Lampe gehalten wird. Verstärken Sie dann den Karton, indem Sie an den Innenkanten Heißleim auftragen, der den Sockel stabiler macht.

Platzieren Sie die Kugel auf dem Sockel, führen Sie das USB-Kabel an der Oberseite heraus und schließen es es an Ihren Computer an.

Starten Sie Ihren Processing-Code, drücken Sie den Ein/Aus-Schalter und sehen Sie zu, wie die Lampe zum Leben erwacht.

Als kleine Übung können Sie einmal Code hinzufügen, mit dem die Lampe angeschaltet wird, wenn der Raum dunkel wird. Hier einige weitere Möglichkeiten:

- Bauen Sie Neigungssensoren ein, um die Lampe durch Rotation in unterschiedliche Richtung ein- oder auszuschalten.

- Fügen Sie einen kleinen PIR-Sensor hinzu um festzustellen, ob jemand in der Nähe ist und um die Lampe auszuschalten, wenn niemand wahrgenommen wird.

- Erzeugen Sie verschiedene Modi, sodass Sie die Farbe manuell steuern oder mehrere Farben überblenden können.

Denken Sie sich verschiedene Projekte aus, experimentieren Sie und haben Sie Spaß!

7/Troubleshooting

Es wird bei Ihren Experimenten immer der Moment kommen, an dem nichts funktioniert und Sie herausfinden müssen, wie sich der Fehler beheben lässt. Troubleshooting und Debugging sind historische Disziplinen, bei denen ein paar einfache Regeln gelten, die meisten Resultate werden aber einfach durch viel Arbeit erzielt.

Je mehr Sie mit Elektronik und Arduino arbeiten, desto mehr werden Sie lernen und desto größer wird Ihr Erfahrungsschatz, wodurch der Prozess der Fehlerbehebung immer weniger zermürbend wird. Lassen Sie sich durch die Probleme, die auftreten, nicht entmutigen – alles ist einfacher, als es am Anfang aussieht.

Da sich jedes Arduino-basierte Projekt aus Hardware und Software zusammensetzt, gibt es immer mehrere Stellen, an denen Sie nachschauen müssen, ob etwas nicht richtig funktioniert. Wenn Sie einen Fehler suchen, sollten Sie sich an drei Richtlinien orientieren:

Verständnis
Versuchen Sie, so weit wie möglich zu verstehen, wie die Bauteile, die Sie verwenden, funktionieren und welchen Beitrag sie für das fertige Projekt leisten. Dieser Ansatz ermöglicht es Ihnen, Techniken für das separate Testen jeder einzelnen Komponente zu entwickeln.

Vereinfachung und Segmentierung
Von den alten Römern stammt der Ausspruch *Divide et impera*, „Teile und herrsche!“ Spalten Sie Ihr Projekt (mental) in seine Komponenten auf, indem Sie Ihr Wissen anwenden und finden Sie heraus, wo die Zuständigkeit jeder Komponente beginnt und wo sie endet.

Ausschließen und Sicherstellen
Testen Sie bei der Fehlersuche jede Komponente einzeln, sodass Sie absolut sicher sein können, dass jede für sich genommen einwandfrei funktioniert. Dadurch werden Sie schrittweise eine Sicherheit erlangen, welche Komponenten ihren Job erledigen und bei welchen Zweifel bestehen.

Debugging wird der Prozess genannt, der sich auf die Software bezieht. Der Legende nach wurde er erstmals in den 1940ern von Grace Hopper verwendet, als Computer noch größtenteils elektromechanisch waren und ein

Computer seine Arbeit einstellte, weil wirkliche Insekten (engl. *bugs*) in die Mechanik eingedrungen waren.

Viele der heutigen „Bugs“ sind nicht mehr physikalischer Natur. Sie sind virtuell und unsichtbar, jedenfalls teilweise. Daher ist manchmal ein langwieriger und langweiliger Prozess erforderlich, um sie ausfindig zu machen.

Testen des Boards

Nehmen wir einmal an, dass unser erstes Beispiel der blinkenden LED nicht funktioniert. Wäre das nicht ein wenig deprimierend? Wir wollen nun herausfinden, was in einem solchen Fall zu tun wäre.

Bevor Sie mit dem Projekt hadern, sollten Sie sich wie ein Pilot, der vor dem Start eine Checkliste durchgeht, um die Flugtüchtigkeit des Flugzeugs sicherzustellen, zunächst vergewissern, dass einige Dinge in Ordnung sind:

Schließen Sie Ihr Arduino-Board an einen USB-Anschluss Ihres Computers an.

- Stellen Sie sicher, dass der Computer eingeschaltet ist (ich weiß, das klingt dumm, aber hier lag schon des Öfteren ein vermeintlicher Fehler begründet). Wenn sich das grüne Licht einschaltet, das mit PWR gekennzeichnet ist, heißt das, dass der Computer das Board mit Strom versorgt. Ist das Licht sehr schwach, dann stimmt etwas nicht mit der Stromversorgung: Versuchen Sie es dann mit einem anderen USB-Kabel und inspizieren Sie den USB-Anschluss des Computers und die USB-Buchse des Arduino-Boards um sicherzustellen, dass sie nicht beschädigt sind. Wenn alles nichts hilft, verwenden Sie einen anderen USB-Anschluss Ihres Rechners oder gleich einen anderen Computer.

- Bei einem brandneuen Computer wird die gelbe LED mit der Bezeichnung L auf eine etwas nervöse Weise blinken. Dies ist das von Hause aus enthaltene Testprogramm, mit dem das Board getestet wird.

- Wenn Sie eine externe Stromversorgung nutzen und ein altes Arduino-Board (Extreme, NG oder Diecimila) verwenden, vergewissern Sie sich, dass der Stecker eingesteckt und die Steckbrücke mit der Bezeichnung SV1 die zwei Pins verbindet, die am nächsten am Anschluss für die externe Stromversorgung liegen.

Hinweis: Wenn es Probleme mit anderen Sketches gibt und Sie sicherstellen müssen, dass das Board funktioniert, öffnen Sie das erste Beispiel 4-1 in der Arduino-IDE und laden Sie es auf das Board. Die Board-eigene LED sollte nun in einem regelmäßigen Muster blinken.

Wenn Sie alle diese Schritte erfolgreich absolviert haben, können Sie darauf vertrauen, dass Arduino einwandfrei läuft.

Testen des Schaltkreises auf der Steckplatine

Verbinden Sie nun Ihr Board mit der Steckplatine, indem Sie mit einer Steckbrücke eine Verbindung vom 5-V-Anschluss und vom GND-Anschluss zu der positiven und der negativen Leiterbahn auf der Steckplatine herstellen. Wenn die grüne PWR-LED nicht mehr leuchtet, entfernen Sie sofort die Verdrahtung. In diesem Falle gibt es einen größeren Fehler im Schaltkreis und Sie haben irgendwo einen Kurzschluss verursacht. Ihr Board bezieht dann zu viel Strom und die Energieversorgung wird abgeschnitten, um das Board zu schützen.

Hinweis: Wenn Sie nun Angst haben, Ihren Computer zu beschädigen, sollten Sie bedenken, dass bei vielen Computern der Stromschutz sehr gut ist und auch rasch reagiert. Außerdem ist das Arduino-Board mit einer selbstrückstellenden Sicherung ausgestattet, einem speziellen Bauteil für den Stromschutz, das sich selbst zurücksetzt, wenn der Fehler behoben ist.

Wenn Sie schon fast paranoide Züge aufweisen, können Sie das Arduino-Board auch immer über einen Self-Powered-USB-Hub anschließen. In diesem Fall wäre das Schlimmste, was passieren kann, dass der USB-Hub zerstört wird. Der Computer bliebe unversehrt.

Wenn Sie einen Kurzschluss verursachen, müssen Sie mit dem Prozess „Vereinfachung und Segmentierung“ fortfahren. Überprüfen Sie jeden Sensor im Projekt und schließen Sie dabei immer nur einen an.

Beginnen Sie immer mit der Stromversorgung (die Verbindungen vom 5-V-Pin und vom GND-Pin). Schauen Sie sich alles an und vergewissere Sie sich, dass jedes Bauteil im Schaltkreis korrekt mit Strom versorgt wird.

Das Arbeiten in Einzelschritten mit niemals mehr als einer Änderung zur selben Zeit ist die goldene Regel bei der Behebung von Fehlern. Diese Regel wurde mir von meinem Dozenten und allerersten Arbeitgeber, Maurizio

Pirola, eingehämmert. Wenn es beim Debuggen nicht so gut läuft (und das geschieht oft), erscheint mir sein Gesicht und sagt: „Immer nur eine Änderung zur selben Zeit ... immer nur eine Änderung zur selben Zeit.“ Und das ist dann der Zeitpunkt, an dem ich das Problem tatsächlich behebe. (Nur allzu schnell geht der Überblick darüber verloren, durch welche Änderung das Problem tatsächlich behoben wurde. Deshalb ist es so wichtig, immer nur eine Änderung zur selben Zeit durchzuführen.) Mit jeder Debugging-Erfahrung entsteht in Ihrem Kopf Schritt für Schritt eine Wissensbasis im Hinblick auf Fehler und mögliche Lösungen, und bevor Sie es wissen, sind Sie schon ein Experte. Das verleiht Ihnen eine gewisse Souveränität, denn wenn sich dann ein Neuling darüber beklagt, dass etwas nicht funktioniert, schauen Sie sich die Sache kurz an und haben im Bruchteil einer Sekunde eine Lösung parat.

Das Isolieren von Problemen

Eine weitere wichtige Regel ist das Finden eines zuverlässigen Weges zur Problemreduzierung. Wenn Ihr Schaltkreis sich an scheinbar zufälligen Zeitpunkten merkwürdig verhält, müssen Sie sehr hart daran arbeiten, den exakten Zeitpunkt herauszufinden, an dem das Problem auftritt, um den Grund hierfür zu ermitteln. Dieser Prozess ermöglicht es Ihnen, über eine mögliche Ursache nachzudenken. Außerdem ist er hilfreich, wenn Sie jemand anderem erklären möchten, was eigentlich geschieht.

Eine möglichst präzise Beschreibung des Problems ist auch eine gute Methode, eine Lösung zu finden. Suchen Sie jemanden, dem Sie das Problem erläutern können – in vielen Fällen wird Ihnen eine Lösung einfallen, sobald Sie das Problem artikulieren. Brian W. Kernighan und Rob Pike, erzählen in ihrem Buch *The Practice of Programming* (Addison-Wesley, 1999) die Geschichte einer Universität, an der ein Teddybär beim Help-Desk aufbewahrt wurde. Studenten mit rätselhaften Bugs mussten zuerst diesem Stofftier das Problem erläutern, bevor sie sich an einen menschlichen Berater wenden durften.

Probleme mit der IDE

In einigen Fällen treten vielleicht Probleme bei der Verwendung der IDE auf, insbesondere unter Windows.

Wenn Ihnen nach einem Doppelklick auf das Arduino-Symbol eine Fehlermeldung angezeigt wird oder einfach nichts geschieht, versuchen Sie als alternative Methode, Arduino mit einem Doppelklick auf die Datei *run.bat* zu starten.

Windows-Nutzer werden außerdem mit einem Problem konfrontiert, wenn das Betriebssystem Arduino eine COM -Anschlussnummer COM10 oder größer zuweist. In diesem Fall können Sie normalerweise Windows dazu veranlassen, Arduino eine kleinere Anschlussnummer zuzuweisen. Öffnen Sie dazu den Device-Manager, indem Sie auf *Start* und dann mit der rechten Maustaste auf *Computer* (Vista) oder *My Computer* (XP) klicken und dann *Properties* auswählen. Klicken Sie unter Windows XP dann auf *Hardware* und wählen Sie *Device Manager* aus. Unter Vista müssen Sie auf *Device Manager* klicken (er wird in der Task-Liste in der linken Fensterhälfte aufgelistet).

Suchen Sie in der Liste unter *Ports (Com & LPT)* die seriellen Geräte. Wählen Sie ein serielles Gerät aus, das Sie nicht nutzen und dem COM9 oder niedriger zugewiesen wurde. Wählen Sie nach einem rechten Mausklick auf dieses Gerät *Properties* aus dem entsprechenden Menü aus. Klicken Sie auf die Registerkarte *Port Settings* und dann auf *Advanced*. Legen Sie für die COM- Anschlussnummer COM10 oder höher fest, klicken Sie auf *OK* und dann nochmal auf *OK*, um den *Dialog Properties* wieder zu schließen.

Führen Sie nun dieselbe Prozedur für das USB Serial Port-Gerät durch, das das Arduino-Board repräsentiert, mit einer Ausnahme: Weisen Sie ihm die COM-Anschlussnummer (COM9 oder niedriger) zu, die Sie gerade freigegeben haben.

Wenn diese Vorschäge alle nicht helfen oder ein Problem auftritt, das hier nicht beschrieben wurde, rufen Sie die Troubleshooting-Seite von Arduino unter **http://www.arduino.cc/en/Guide/Troubleshooting** auf.

So finden Sie Onlinehilfe

Wenn Sie bei einem Problem festhängen, sollten Sie nicht tagelang alleine herumbasteln – bitten Sie einfach um Hilfe. Einer der größten Vorteile bei Arduino ist seine Community. Sie können jederzeit Hilfe finden, wenn Sie das aufgetretene Problem gut beschreiben.

Gewöhnen Sie sich an, per Cut-and-Paste Stichworte in einer Suchmaschine einzugeben und so zu erfahren, ob jemand einen entsprechenden Beitrag veröffentlicht hat. Wenn beispielseise die Arduino-IDE eine merkwürdige Fehlermeldung anzeigt, können Sie sie einfach in die Google-Suchmaschine kopieren und sich die Suchergebnisse anschauen. Das funktioniert auch bei Ausschnitten aus Code, an dem Sie gerade arbeiten und bei speziellen Funktionsnamen. Schauen Sie sich einfach um: Alles wurde bereits erfunden und ist auf irgendeiner Webseite gespeichert.

Für weitere Nachforschungen können Sie auch die Hauptwebseite von Arduino **http://www.arduino.cc** besuchen und sich die FAQs unter **http://www.arduino.cc/en/Main/FAQ** anschauen und dann zum Playground (**http://www.arduino.cc/playground**) wechseln. Diese frei editierbare Wiki-Plattform kann von jedem Nutzer modifiziert werden, um Dokumentation beizusteuern. Dies ist einer der größten Vorteile der gesamten Open-Source-Philosophie. Personen steuern Dokumentation und Beispiele von allem bei, was sich mit Arduino umsetzen lässt. Bevor Sie mit einem Projekt beginnen, durchsuchen Sie zunächst den Playground, und Sie werden sicherlich Codeabschnitte und Schaltdiagramme finden, auf denen Sie aufbauen können.

Wenn Sie über all diese Wege immer noch keine Antwort erhalten haben, besuchen Sie das Forum (**http://www.arduino.cc/cgi-bin/yabb2/YaBB.pl**). Wenn Sie dort keine direkte Hilfe finden, können Sie auch eine entsprechende Frage posten. Wählen Sie das richtige Board für Ihre Frage aus: Es gibt verschiedene Bereiche für Software- und Hardwareprobleme und außerdem Foren in fünf verschiedenen Sprachen. Posten Sie bitte möglichst viele Informationen:

- Welches Arduino-Board verwenden Sie?
- Unter welchem Betriebssystem führen Sie die Arduino-IDE aus?
- Fügen Sie eine allgemeine Beschreibung dessen an, was Sie zu tun versuchen. Wenn Sie ungewöhnliche Bauteile verwenden, posten Sie Links zu den entsprechenden Datenblättern.

Die Anzahl der Antworten, die Sie erhalten, hängt davon ab, wie gut Sie Ihre Frage formulieren.

Ihre Chancen steigen, wenn Sie **folgende Dinge unter allen Umständen vermeiden** (diese Richtlinien gelten für alle Foren, nicht nur für das von Arduino):

- Das Schreiben der gesamten Nachricht in GROSSBUCHSTABEN. Das nervt die Leute gewaltig und Sie outen sich sofort als Neuling (bei Online-Communities wird das Schreiben in Großbuchstaben als Schreien aufgefasst).
- Das Posten ein- und desselben Beitrags in verschiedenen Bereichen des Forums.

- Das ständige Posten nachfassender Kommentare wie „Hey, warum antwortet niemand?“ oder, noch schlimmer, das nochmalige Senden derselben Frage. Wenn Sie keine Antwort erhalten, schauen Sie sich Ihr Posting noch einmal an. Wurde das Thema klar formuliert? War die Beschreibung Ihres Problems verständlich? Waren Sie freundlich? Seien Sie immer nett.

- Das Schreiben von Nachrichten wie „Ich möchte mit Arduino eine Raumfähre bauen. Wie mache ich das?“ Sie signalisieren damit, dass andere die Arbeit für Sie erledigen sollen, und dieser Ansatz ist für einen wirklichen Tüftler einfach nicht lustig. Es ist besser, eine spezifische Frage zu einem Teil des Projekts zu stellen und die bereitgestellten Informationen dann zu nutzen.

- Eine Variation des vorherigen Punktes ist es, wenn durch die Frage offensichtlich wird, dass die postende Person dafür bezahlt wird. Wenn Sie eine spezifische Frage stellen, helfen Ihnen die Leute gerne. Wenn sie aber Ihre Arbeit erledigen sollen (und Sie den betreffenden Lohn nicht teilen), wird die Antwort wahrscheinlich nicht sehr nett ausfallen.

- Das Posten von Nachrichten, die verdächtig nach Hausaufgaben aussehen und mit denen das Forum gebeten wird, Ihre Arbeit für Sie zu erledigen. Professoren wie ich durchkämmen das Netz und strafen solche Studenten gandenlos ab.

Anhang A/ Die Steckplatine

Wenn Sie einen Schaltkreis ans Laufen bringen möchten, ist das mit vielen Änderungen verbunden, bis alles wirklich funktioniert. Es handelt sich dabei um einen sehr schnellen, sich wiederholenden Prozess, der irgendwie ein elektronisches Pendant zum Schreiben von Sketches ist. Das Design entwickelt sich unter Ihren Händen, während Sie verschiedene Kombinationen ausprobieren. Um das bestmögliche Resultat zu erzielen, müssen Sie ein System entwickeln, das es ermöglicht, Verbindungen zwischen Komponenten auf die schnellstmögliche, praktischste und am wenigsten destruktive Weise zu ändern. Durch diese Anforderungen ist das Löten klar ausgeschlossen, weil es sehr zeitaufwendig ist und die Komponenten bei jedem Erhitzen und Abkühlen sehr beansprucht werden.

Die Antwort auf dieses Problem ist ein sehr praktisches Bauteil, das lötfreie Steckplatine genannt wird. Wie Sie in Abbildung A-1 sehen, handelt es sich dabei um ein kleines Plastikbrett, das komplett gelocht ist, wobei jedes Loch über einen federgelagerten Anschluss verfügt. Wenn Sie den Anschlusspin einer Komponente in eines der Löcher stecken, wird eine elektrische Verbindung mit allen anderen Löchern hergestellt, die sich in derselben vertikalen Spalte befinden. Jedes Loch weist einen Abstand von 2,54 mm zu den anderen Löchern auf.

Da bei den meisten Komponenten die Anschlussbeinchen (die von Geeks als Pins bezeichnet werden) einen Standardabstand aufweisen, passen Chips mit mehreren Beinchen gut. Nicht alle Kontakte auf dem Breadboard sind gleich gestaltet – es gibt einige Unterschiede. Die oberen und die unteren Reihen (die rot oder blau markiert und mit + oder – gekennzeichnet sind) sind horizontal verbunden und dienen dazu, den Strom über die Platine zu leiten. Wenn Sie also Strom oder Masse benätigen, können diese sehr schnell mittels einer *Steckbrücke* (ein kurzes Drahtstück, mit dem sich zwei Punkte auf der Platine verbinden lassen) bereitgestellt werden. Als letzten Fakt müssen Sie über Steckplatinen wissen, dass sich in der Mitte eine große Lücke befindet, die so groß ist wie ein kleiner Chip. Alle vertikalen Lochreihen sind hier unterbrochen. Wenn Sie also einen Chip anbringen, der über Pins auf beiden Seiten verfügt, wird bei diesen kein Kurzschluss erzeugt. Das ist doch mal eine clevere Idee, oder?

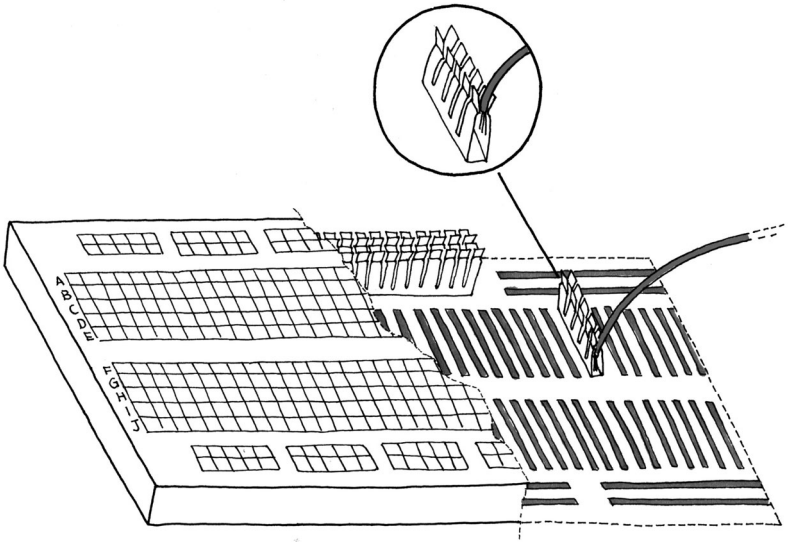


Abbildung A-1:
Die lötfreie Steckplatine

Anhang B/ Das Lesen von Widerständen und Kondensatoren

Um elektronische Bauteile verwenden zu können, müssen Sie in der Lage sein, sie zu identifizieren, was für Neulinge eine große Herausforderung darstellen kann. Die meisten Widerstände, die in Läden erhältlich sind, haben einen zylindrischen Körper, aus dem zwei Anschlussbeinchen herausragen und der ringsherum merkwürdige farbige Markierungen aufweist. Als die ersten kommerziellen Widerstände hergestellt wurden, gab es noch keine Möglichkeit, so kleine Nummern aufzudrucken, dass sie auf den Körper des Widerstands passten. Daher kamen clevere Ingenieure auf die Idee, die Widerstandswerte in Form von farbigen Ringen darzustellen.

Heute müssen Neulinge daher lernen, wie diese Farben zu interpretieren sind. Der Schlüssel hierzu ist recht einfach: Generell gibt es vier Farbringe. Jede Farbe steht für eine Zahl. Einer der Ringe ist üblicherweise goldfarben und repräsentiert den Genauigkeitsgrad dieses Widerstands. Um die Ringe in der richtigen Reihenfolge zu lesen, müssen Sie den Widerstand so halten, dass sich der goldene (oder in einigen Fällen der silberne) Ring rechts befindet. Schauen Sie sich dann die Farben an und ordnen Sie sie den entsprechenden Zahlen zu. Die folgende Tabelle zeigt in übersichtlicher Weise die Farben und den jeweils zugeordneten numerischen Wert.

Farbe	Wert
Schwarz	0
Braun	1
Rot	2
Orange	3
Gelb	4
Grün	5
Blau	6
Violett	7

Farbe	Wert
Grau	8
Weiß	9
Silber	10%
Gold	5%

Eine Markierung mit einem braunen, einem schwarzen, einem orangefarbenen und einem goldenen Ring beispielsweise bedeutet 1 0 3 ±5%. Das ist doch recht einfach, oder? Nicht ganz, denn es gibt noch eine kleine Falle: Der dritte Ring gibt die Anzahl der Nullen im Wert an. Daher handelt es sich bei der Abfolge 1 0 3 um 1 0 gefolgt von 3 Nullen, sodass wir hier im Endeffekt einen Wert von 10.000 Ohm ±5% haben. Elektronik-Geeks verkürzen die Werte, indem sie sie in Kiloohm (für tausend Ohm) und Megaohm (für eine Million Ohm) angeben, so wird dann aus einem 10.000-Ohm-Widerstand in der Kurzform ein 10k-Widerstand und aus 10.000.000 werden 10M. Bitte beachten Sie dies, denn Ingenieure lieben die Optimierung und Sie werden unter Umständen in Schaltskizzen Werte wie 4k7 finden, was nichts anderes als 4,7 Kiloohm oder 4700 Ohm bedeutet.

Kondensatoren sind diesbezüglich ein wenig einfacher: Bei den fassförmigen Kondensatoren (elektrolytische Kondensatoren) sind die Werte normalerweise aufgedruckt. Der Wert eines Kondensators wird in Farad (F) angegeben, doch die meisten Kondensatoren, die Sie finden, weisen einen Wert in Mikrofarad (µF) auf. Wenn also ein Wert von 100 µF aufgedruckt ist, handelt es sich um einen 100-Mikrofarad-Kondensator.

Bei vielen der scheibenförmigen Kondensatoren (Keramikkondensatoren) sind die Einheiten nicht angegeben. Es wird ein numerischer Code angeführt, der aus drei Ziffern besteht und den Wert in Picofarad wiedergibt. 1.000.000 pF sind ein µF. Ähnlich wie beim Widerstand dient die dritte Ziffer dazu, die Anzahl der Nullen anzugeben, die hinter den beiden ersten Ziffern folgen, mit einem Unterschied: Wenn Sie die Ziffern 1-5 lesen, handelt es sich dabei um die Anzahl an Nullen. Die Ziffern 6 und 7 werden nicht verwendet, bei den Zahlen 8 und 9 wird in einer anderen Weise verfahren. Wenn Sie eine 8 lesen, müssen Sie die Zahl, die sich aus den ersten beiden Ziffern ergibt, mit 0,01 multiplizieren, bei einer 9 lautet der Multiplikator 0,1.

Wenn also ein Kondensator mit 104 beschriftet ist, handelt es sich dabei um 100.000 pF oder 0,1 µF. Bei einem Kondensator mit der Beschriftung 229 sind es demnach 2,2 pF.

Anhang C/ Kurzreferenz zu Arduino

An dieser Stelle soll eine kurze Erläuterung der Standardanweisungen erfolgen, die von der Arduino-Sprache unterstützt werden.

Eine detaillierte Referenz steht unter **arduino.cc/en/Reference/Home-Page zur Verfügung.**

STRUKTUR

Ein Arduino-Sketch wird in zwei Teilen ausgeführt:

`void setup()`
Hier wird der Initialisierungscode platziert – die Anweisungen, mit denen das Setup des Boards vor dem Eintritt in die Hauptschleife des Sketch erfolgt.

`void loop()`
Hier ist der Hauptcode des Sketch untergebracht. Die Schleife enthält einen Satz an Anweisungen, die so lange wiederholt werden, bis das Board ausgeschaltet wird.

SONDERZEICHEN

Arduino enthält zahlreiche Sonderzeichern, um Codezeilen, Kommentare und Codeblöcke zu kennzeichnen.

; (Semikolon)

Jede Anweisung (Codezeile) wird mit einem Semikolon beendet. Duch diese Syntax lässt sich der Code frei formatieren. Sie können sogar zwei Anweisungen in derselben Zeile platzieren, solange Sie sie nur durch ein Semikolon voneinander trennen. (Allerdings ist dies der Lesbarkeit des Codes nicht sehr zuträglich.)

Beispiel:
```
delay(100);
```

{} (Geschweifte Klammern)

Diese werden verwendet, um Codeblöcke zu kennzeichnen. Wenn Sie beispielsweise Code für die *loop()*-Funktion schreiben, muss dieser mit der öffnenden geschweiften Klammer eingeleitet und mit der schließenden geschweiften Klammer beendet werden.

Beispiel:

```
void loop() { Serial.println("ciao"); }
```

Kommentare

Diese Textabschnitte werden vom Arduino-Prozessor ignoriert, sind aber sehr hilfreich, um für Sie und andere Nutzer festzuhalten, was der Code tut.

Bei Arduino gibt es zwei Arten von Kommentaren:

```
// einzelne Zeile: dieser Text wird bis zum Ende der Zeile ignoriert
/* mehrere Zeilen:
   Hier findet
   ein ganzes Gedicht Platz
*/
```

KONSTANTEN

Arduino enthält einen Satz vordefinierter Schlüsselwörter mit speziellen Werten.

HIGH und **LOW** werden verwendet, um beispielsweise einen Arduino-Pin ein- oder auszuschalten. Mit **INPUT** und **OUTPUT** wird definiert, ob es sich bei einem Pin um einen Eingangs- oder einen Ausgangspin handeln soll.

Die Werte **true** und **false** zeigen genau das an, was schon der Name vermuten lässt, nämlich ob eine Bedingung oder ein Ausdruck wahr oder falsch ist.

VARIABLEN

Variablen sind benannte Bereiche des Arduino-Speichers, in denen Daten gespeichert werden können, die sich im Sketch verwenden und auch ändern lassen. Wie der Name schon sagt, können Variablen beliebig oft geändert werden.

Weil Arduino ein sehr einfacher Prozessor ist, müssen Sie bei der Variablendeklaration auch den entsprechenden Typ angeben. Es ist also erforderlich, dem Prozessor die Größe des zu speichernden Wertes mitzuteilen.

Hier die verfügbaren *Datatypen*:

boolean

Er kann einen von zwei Werten enthalten: wahr oder falsch.

char

Er enthält ein einzelnes Zeichen, z.B. A. Wie jeder Computer speichert Arduino dieses Zeichen als Zahl, auch wenn Text angezeigt wird. Wenn char-Variablen verwendet werden, um Zahlen zu speichern, können sie Werte von –128 bis 127 enthalten.

Hinweis: Auf Computern stehen zwei größere Zeichensätze zur Verfügung: ASCII und UNICODE. ASCII umfasst 127 Zeichen, die unter anderem dazu verwendet werden, um Text zwischen seriellen Endgeräten und zeitlich verzahnten Computersystemen wie Großrechnern oder Minicomputern zu übertragen. UNICODE umfasst einen viel größeren Satz an Werten und wird von modernen Computerbetriebssystemen verwendet, um Zeichen in einer Vielzahl von Sprachen darzustellen. Dennoch ist auch ASCII nützlich beim Austauch von kurzen Informationen in Sprachen wie Italienisch oder Englisch, bei denen das lateinische Alphabet, arabische Zahlen und häufig verwendete Schreibmaschinenzeichen wie solche zur Interpunktion und dergleichen verwendet werden.

---

byte

Dieser Datentyp enthält eine Zahl zwischen 0 und 255. Wie bei *chars* wird auch bei *bytes* nur ein Byte an Speicher verwendet.

int

Hier werden 2 Byte Speicher genutzt, um eine Zahl zwischen –32.168 und 32.167 darzustellen. Es ist der bei Arduino am häufigsten genutzte Datentyp.

unsigned_int

Wie bei *int* werden auch hier 2 Bytes verwendet, **unsigned** (vorzeichenlos) bedeutet aber, dass keine negativen Zahlen gespeichert werden können, daher reicht der Wertebereich von 0 bis 65.535.

long

Diese Variable ist doppelt so groß wie **int** und enthält Werte von –2.147.483.648 bis 2.147.483.647.

unsigned_long

Dies ist die vorzeichenlose Version von **long** mit einem Wertebereich von 0 bis 4.294.967.295.

float

Hierbei handelt es sich um eine recht große Variable, die Fließkommawerte (ein lustiger Ausdruck, um zu beschreiben, dass Zahlen mit einem Dezimalkomma gespeichert werden können) enthalten kann. Solche Variablen nutzen 4 Byte Ihres wertvollen RAM, und die Funktionen, die mit ihnen arbeiten können, verbrauchen ebenfalls viel Codespeicher. Sie sollten **float**s daher sparsam verwenden.

double

Diese Variable speichert eine Fließkommazahl mit doppelter Genauigkeit, mit einem maximalen Wert von $1.7976931348623157 \times 10^{308}$. Wow, das ist ein wirklich großer Wert!

string

Hierin wird ein Satz von ASCII-Zeichen beherbergt, mit denen Textinformationen gespeichert werden (wenn beispielsweise mittels einer Zeichenfolge eine Nachricht über den seriellen Anschluss gesendet oder auf einem LCD-Display angezeigt werden soll). Zum Speichern wird dabei ein Byte für jedes Zeichen in der Zeichenfolge verwendet, plus ein Nullzeichen, um Arduino mitzuteilen, dass es sich um das Ende des Strings handelt. Folgende Schreibweisen sind gleichbedeutend:

```
char string1 [] = "Arduino"; // 7 chars + 1 Null-char
char string2[8] = "Arduino"; // wie oben
```

array

Hierbei handelt es sich um eine Liste, auf die über einen Index zugegriffen werden kann. Diese Variablen werden verwendet, um Wertetabellen zu erstellen, auf die schnell zugegriffen werden kann. Wenn Sie beispielsweise verschiedene Helligkeitsstufen speichern möchten, die zum Dimmen einer LED verwendet werden sollen, könnten Sie sechs Variablen mit den Namen light01, light02 usw. erzeugen. Besser ist aber die Verwendung einer einzigen Variablen wie der folgenden:

```
int light[6] = {0, 20, 50, 75, 100};
```

Das Wort array wird bei der Variablendeklaration nicht verwendet. Stattdessen kommen die Zeichen [] und {} zum Einsatz.

KONTROLLSTRUKTUREN

Arduino enthält Schlüsselwörter für das Steuern des Logikflusses Ihres Sketch.

if...else

Mit dieser Struktur werden in Ihrem Programm Entscheidungen gefällt. An *if* muss eine Frage anschließen, die als Ausdruck, der in Klammern eingeschlossen ist, angegeben wird. Wenn der Ausdruck wahr ist, wird alles Nachfolgende ausgeführt. Wenn er falsch ist, wird mit dem nächsten Codeblock fortgefahren. Es ist auch möglich, nur *if* ohne Angabe einer Bedingung zu verwenden. *else* clause.

Beispiel:

```
if (val == 1) {
digitalWrite(LED,HIGH);
}
```

for

Bei dieser Struktur wird der Codeblock mit einer angegebenen Häufigkeit wiederholt.

Beispiel:

```
for (int i = 0; i < 10; i++) {
Serial.print("ciao");
}
```

switch case

Die *if*-Anweisung ist mit einer Weggabelung vergleichbar. Die Kontrollstruktur *switch case* hingegen ähnelt eher einem massiven Kreisel. Sie ermöglicht dem Programm, eine Vielzahl von Richtungen einzuschlagen, in Abhängigkeit vom Wert einer Variablen. Diese Kontrollstruktur ist sehr hilfreich, wenn es darum geht, Ihren Code übersichtlich zu halten, da sie lange Listen von *if*-Anweisungen ersetzt.

Beispiel:

```
switch (sensorValue) {
case 23:
```

```
digitalWrite(13,HIGH);
break;
case 46:
digitalWrite(12,HIGH);
break;
default: // wenn nichts zutrifft, wird dies ausgeführt
digitalWrite(12,LOW);
digitalWrite(13,LOW);
}
```

while

Sie ähnelt der *if*-Anweisung. Es wird ein Codeblock ausgeführt, solange eine Bedingung wahr ist.

Beispiel:

```
// LED blinkt solange der Sensor-Wert kleiner 512
sensorValue = analogRead(1);
while (sensorValue < 512) {
digitalWrite(13,HIGH);
delay(100);
digitalWrite(13,HIGH);
delay(100);
sensorValue = analogRead(1);
}
```

do...while

Diese Struktur ähnelt der *while*-Anweisung, mit der Ausnahme, dass der Code ausgeführt wird, bevor die Bedingung ausgewertet wird. Diese Struktur wird verwendet, wenn der enthaltene Code mindestens einmal ausgeführt werden soll, bevor die Bedingung geprüft wird.

Beispiel:

```
do {
digitalWrite(13,HIGH);
delay(100);
digitalWrite(13,HIGH);
delay(100); sensorValue = analogRead(1);
} while (sensorValue < 512);
```

break

Bei dieser Anweisung wird eine Schleife verlassen und mit der Ausführung des Codes fortgefahren, der nach der Schleife folgt. Sie wird auch verwendet, um die verschiedenen Abschnitte einer *switch case*-Anweisung zu separieren.

Beispiel:

```
// LED blinkt solange der Sensor-Wert kleiner 512
do {
 // Schleife wird verlassen, wenn ein Taster gedrückt wurde
(digitalRead(7) == HIGH)
break;
digitalWrite(13,HIGH);
delay(100);
digitalWrite(13,LOW); delay(100);
sensorValue = analogRead(1);
} while (sensorValue < 512);
```

continue

Wenn diese Anweisung innerhalb einer Schleife verwendet wird, veranlasst *continue*, dass der restliche enthaltene Code übersprungen und die Bedingung erneut getestet wird.

Beispiel:

```
for (light = 0; light < 255; light++)
{
// überspringt Intensitäten zwischen 140 und 200
if ((x > 140) && (x < 200))
continue;
analogWrite(PWMpin, light);
delay(10);
}
```

return

Bei dieser Anweisung wird die Ausführung einer Funktion gestoppt und ein entsprechender Wert zurückgegeben. Sie können diese Struktur auch verwenden, um einen Wert, der aus einer Funktion stammt, zurückzuliefern.

Wenn Sie beispielsweise bei einer Funktion mit dem Namen *computeTemperature()* das Ergebnis an den Teil Ihres Codes ausgeben möchten, über den die Funktion aufgerufen wurde, würden Sie etwa Folgendes schreiben:

```
int computeTemperature() {
int temperature = 0;
temperature = (analogRead(0) + 45) / 100;
return temperature;
}
```

ARITHMETIK UND FORMELN

Sie können Arduino für komplexe Berechnungen verwenden, indem Sie eine spezielle Syntax verwenden. Die Zeichen + und – funktionieren so, wie Sie es in der Schule gelernt haben, Multiplikation wird mit einem * und Division mit einem / dargestellt.

Es gibt noch einen zusätzlichen Operator, der als Modulo (%) bezeichnet wird und den Rest aus einer Division von Ganzzahlen zurückliefert. Sie können beliebig viele Klammern nutzen, um Ausdrücke zusammenzufassen und zu schachteln. Im Gegensatz zu dem, was Sie möglicherweise in der Schule gelernt haben, sind eckige und geschweifte Klammern für andere Zwecke reserviert (z.B. für Array-Indizes und Blöcke).

Beispiele:

```
a = 2 + 2;
light = ((12 * sensorValue) - 5 ) / 2;
remainder = 3 % 2;
 // liefert 1 zurück
```

VERGLEICHSOPERATOREN

Zur Angabe von Bedingungen oder Prüfungen für *if-, while-* und *for-*Anweisungen stehen folgende Operatoren zur Verfügung:

==	gleich
!=	ungleich
<	kleiner als
>	größer als
<=	kleiner oder gleich
>=	größer oder gleich

BOOLESCHE OPERATOREN

Dieser Operator wird verwendet, wenn Sie mehrere Bedingungen verknüpfen möchten. Wenn Sie beispielsweise überprüfen möchten, ob der von

einem Sensor zurückgelieferte Wert zwischen 5 und 10 liegt, würden Sie Folgendes schreiben:

```
if ((sensor => 5)  && (sensor  <=10))
```

Es gibt drei Operatoren: „und“, durch **&&** dargestellt, „oder“, repräsentiert durch **||**, und schließlich „nicht“, dargestellt durch **!**.

KOMBINIERTE OPERATOREN

Hierbei handelt es sich um spezielle Operatoren, die verwendet werden, um den Code bei häufig durchgeführten Operationen, wie z.B. das Hochzählen eines Wertes, möglichst kurz zu halten.

Um beispielsweise *value* um 1 zu inkrementieren, würden Sie Folgendes schreiben:

```
value = value +1;
```

Unter Verwendung eines kombinierten Operators wird daraus diese vereinfachte Version:

```
value++;
```

Inkrementieren und Dekrementieren (–– und ++)

Hiermit wird um den Wert 1 hoch- oder heruntergezählt. Seien Sie aber vorsichtig: Wenn Sie *i*++ schreiben, wird *i* um 1 inkrementiert und in Bezug auf das Äquivalent von *i+1* ausgewertet. Bei ++*i* wird in Bezug auf den Wert von *i* ausgewertet und **anschließend** *i* inkrementiert. Dasselbe gilt für ––.

+=, *= und /=

Hierdurch lässt sich die Schreibweise für bestimmte Ausdrücke verkürzen. Die beiden folgenden Ausdrücke sind gleichbedeutend:

```
a = a + 5;
a += 5;
```

INPUT- UND OUTPUT-FUNKTIONEN

Arduino umfasst Funktionen für die Handhabung von Input und Output. Sie haben in diesem Buch bereits einige entsprechende Beispielprogramme gesehen.

pinMode(pin, mode)

Hiermit wird ein digitaler Pin neu definiert, sodass er dann als Eingangs- oder Ausgangspin dient.

Beispiel:

```
pinMode(7,INPUT); // definiert Pin 7 als Input
```

digitalWrite(pin, value)

Hiermit wird ein digitaler Pin ein- oder ausgeschaltet. Pins müssen mittels *pinMode* explizit als Output definiert werden, bevor mit *digitalWrite* irgendein Effekt erzielt werden kann.

Beispiel:

```
digitalWrite(8,HIGH); // schaltet den digitalen Pin 8 ein
```

int digitalRead(pin)

Hiermit wird der Zustand eines Eingangspins ausgelesen. Dabei wird HIGH zurückgeliefert, wenn vom Pin eine Spannung festgestellt wurde, und LOW, wenn keine Spannung anliegt.

Beispiel:

```
val =  digitalRead(7); // liest Pin 7 in val ein
```

int analogRead(pin)

Diese Funktion liest die Spannung an einem analogen Pin aus und liefert einen Wert zwischen 0 und 255 zurück, der eine Spannung zwischen 0 und 5 V repräsentiert.

Beispiel:

```
val =  analogRead(0 ); // liest analogen Input 0 in val ein
```

analogWrite(pin, value)

Hiermit wird die PWM-Frequenz für einen der Pins, die als PWM definiert wurden, geändert. Dabei kann *pin* 11,10, 9,6,5 oder 3 sein. Die Variable *value* kann Werte zwischen 0 und 255 enthalten, die eine Skala von 0 und 5 V für die Spannung am Output-Pin repräsentieren.

Beispiel:

```
analogWrite(9,128); // Dimmt eine LED an Pin 9 auf 50%
```

shiftOut(dataPin, clockPin, bitOrder, value)

Diese Funktion sendet Daten an ein *Schieberegister*, ein logisches Schaltewerk, das verwendet wird, um die Anzahl der digitalen Outputs zu erweitern. Dieses Protokoll nutzt einen Pin für Daten und einen als Taktgeber. Mit *bitOrder* wird die Reihenfolge der Abarbeitung (**least significant bit** oder **most significant bit**) bestimmt und in *value* ist das zu sendende Byte gespeichert.

Beispiel:

```
shiftOut(dataPin, clockPin, LSBFIRST, 255);
```

unsigned long pulseIn(pin, value)

Hiermit wird die von einem der digitalen Pins eingehende Pulsdauer gemessen. Dies ist zum Beispiel dann nützlich, wenn ein Infrarotsensor oder ein Beschleunigungsmesser ausgelesen werden soll, bei dem die Werte in Form von Impulsen bezogen auf die Änderungsdauern ausgegeben werden.

Beispiel:

```
time =  pulsein(7,HIGH); // misst die  Zeitdauer, die der nächste
                         // Impuls HIGH bleibt
```

ZEITFUNKTIONEN

Arduino umfasst Funktionen für das Messen von abgelaufener Zeit und für Pausenzeiten von Sketches.

unsigned long millis()

Diese Funktion gibt die Anzahl an Millisekunden zurück, die seit dem Start des Sketch vergangen sind.

Beispiel:

```
duration = millis()-lastTime; // berechnet die vergangene Zeitdauer seit
                              // "lastTime"
```

delay(ms)

Hiermit wird eine Pause des Programms für die angegebene Zeit in Millisekunden veranlasst.

Beispiel:

```
delay(500 ); // das Programm wird für eine halbe Sekunde gestoppt
```

delayMicroseconds(us)

Das Programm wird veranlasst, für eine gegebene Anzahl an Millisekunden zu pausieren.

Beispiel:

```
delayMicroseconds(1000 ); // wartet eine 1 Millisekunde
```

MATHEMATISCHE FUNKTIONEN

In Arduino sind viele häufig genutzte mathematische und trigonometrische Funktionen enthalten:

min(x, y)

Es wird ein Wert kleiner als *x* und *y* zurückgeliefert.

Beispiel:

```
val =  min(10,20); // val ist nun 10
```

max(x, y)

Es wird ein Wert größer als *x* und *y* ausgegeben.

Beispiel:

```
val =  max(10,20); // val ist nun 20
```

abs(x)

Es wird der absolute Wert von *x* zurückgeliefert, der negative Zahlen in positive umwandelt. Wenn also *x* 5 ist, wird 5 zurückgeliefert, und wenn *x* –5 ist, lautet der Rückgabewert ebenfalls 5.

Beispiel:

```
val = abs(-5); // val ist nun 5
```

constrain(x, a, b)

Gibt den Wert von *x* zurück, der aber auf einen Bereich zwischen a und b beschränkt ist. Wenn *x* kleiner als *a* ist, wird einfach *a* zurückgeliefert, und wenn *x* größer als *b* ist, wird *b* ausgegeben.

Beispiel:

```
val = constrain(analogRead(0), 0, 255); // weist Werte größer als 255
                                        // zurück
```

map(value, fromLow, fromHigh, toLow, toHigh)

Hiermit wird ein Wert aus dem Bereich *fromLow* und *maxLow* dem Bereich *toLow* und *toHigh* zugewiesen. Dies ist sehr nützlich bei der Verarbeitung von Werten, die von analogen Sensoren stammen.

Beispiel:

```
val = map(analogRead(0 ),0,1023,100, 200 ); // weist den Wert von
                                            // analog 0 einem Wert
                                            // zwischen 100 und 200 zu
```

double pow(base, exponent)

Es wird das Ergebnis einer Zahl (*Basis*) im Hinblick auf eine Potenz (*Exponent*) zurückgeliefert.

Beispiel:

```
double x = pow(y, 32); // setzt x auf den um die Potenz 32 erhöhten Wert von y
```

double sqrt(x)

Es wird die Quadratwurzel einer Zahl zurückgeliefert.

Beispiel:

```
double a =  sqrt(1138); // etwa 33.73425674438
```

double sin(rad)

Es wird der Sinus eines Winkels als Bogenmaß zurückgeliefert.

Beispiel:

```
double sine =  sin(2); // etwa 0.90929737091
```

double cos(rad)

Es wird der Kosinus eines Winkels als Bogenmaß zurückgeliefert.

Beispiel:

```
double cosine =  cos(2); // etwa -0.41614685058
```

double tan(rad)

Es wird die Tangente eines Winkels als Bogenmaß zurückgeliefert.

Beispiel:

```
double tangent =  tan(2); // etwa -2.18503975868
```

ZUFALLSZAHLENFUNKTIONEN

Zum Erzeugen von Zufallszahlen können Sie den Pseudozufallszahlen-Generator von Arduino verwenden.

randomSeed(seed)

Hiermit wird der Pseudozufallszahlen-Generator von Arduino zurückgesetzt. Die Verteilung der von *random()* zurückgelieferten Zahlen ist zwar grundsätzlich zufällig, aber die Abfolge ist vorhersehbar. Daher sollten Sie den Generator auf einen Zufallswert zurücksetzen. Wenn ein nicht verbundener analoger Pin vorhanden ist, wird er einige zufällige Geräusche aus der Umgbebung auffangen (Radiowellen, kosmische Strahlung, elektromagnetische Interferenzen von Mobiltelefonen und fluoreszierendem Licht usw.).

Beispiel:

```
randomSeed(analogRead(5 )); // erzeugt Zufallszahlen mithilfe von Geräu-
                            // schen an Pin 5
```

long random(max)
long random(min, max)

Es wird ein ganzzahliger Zufallswert vom Typ *long* zwischen *min* und *max – 1* zurückgeliefert. Wenn kein Minimum angegeben wurde, ist die untere Grenze 0.

Beispiel:

```
long randnum = random(0, 100); // eine Zahl zwischen 0 und 99
long randnum = random(11); // eine Zahl zwischen 0 und 10
```

SERIELLE KOMMUNIKATION

Wie Sie in Kapitel 5 gesehen haben, können Sie über den USB-Port mit anderen Geräten kommunizieren, wobei ein serielles Kommunikationsprotokoll zum Einsatz kommt. Im Folgenden sind die seriellen Funktionen aufgelistet.

Serial.begin(speed)

Mit dieser Funktion wird Arduino darauf vorbereitet, serielle Daten zu versenden und zu empfangen. Normalerweise arbeitet der serielle Monitor der Arduino-IDE mit einer Geschwindigkeit von 9600 Bit pro Sekunde

(bps), es stehen aber auch andere Werte zur Verfügung, üblicherweise aber nicht mehr als 115.200 bps.

Beispiel:

```
Serial.begin(9600);
```

Serial.print(data)
Serial.print(data, encoding)

Diese Funktion schickt Daten an den seriellen Anschluss. Die Zeichencodierung ist dabei optional; wenn keine Angaben getroffen werden, werden die Daten so weit wie möglich als Klartext behandelt.

Beispiele:

```
Serial.print(75); // druckt "75"
Serial.print(75, DEC); // wie oben
Serial.print(75, HEX); // "4B" (75 als Hexadezimalzahl)
Serial.print(75, OCT); // "113" (75 als Oktalzahl)
Serial.print(75, BIN); // "1001011" (75 als Binärzahl)
Serial.print(75, BYTE); // "K" (das Byte
                        // das zufällig 75 im ASCII-Zeichensatz ist)
```

Serial.println(data)
Serial.println(data, encoding)

Diese Funktion arbeitet wie *Serial.print()* mit der Ausnahme, dass ein Wagenrücklauf und ein Zeilenvorschub (\r\n) angefügt wird, als ob nach der Dateneingabe die Return- oder Enter-Taste gedrückt worden wäre.

Beispiele:

```
Serial.println(75);       // druckt "75\r\n"
Serial.println(75, DEC);  // wie oben
Serial.println(75, HEX);  // "4B\r\n"
Serial.println(75, OCT);  // "113\r\n"
Serial.println(75, BIN);  // "1001011\r\n"
Serial.println(75, BYTE); // "K\r\
```

int Serial.available()

Diese Funktion liefert zurück, wie viele Daten am seriellen Anschluss für das Auslesen mittels der *read()*-Funtion bereitstehen. Nachdem mit *read()* alle verfügbaten Daten ausgelesen wurden, liefert *Serial.available()* so lange 0 zurück, bis neue Daten am seriellen Anschluss vorliegen.

Beispiel:

```
int count = Serial.available();
```

int Serial.read()

Es wird ein Byte der eingehenden seriellen Daten abgerufen.

Beispiel:

```
int data = Serial.read();
```

Serial.flush()

Da die Daten über den seriellen Anschluss möglicherweise schneller eintreffen, als dein Programm sie verarbeiten kann, speichert Arduino alle eingehenden Daten in einem Puffer. Wenn der Puffer gelöscht und Platz für neue Daten geschaffen werden soll, wird hierzu die *flush()*-Funktion verwendet.

Beispiel:

```
Serial.flush();
```

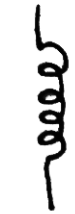
Anhang D/ Das Lesen von Schaltplänen

Bisher haben wir noch keine sehr detaillierten Illustrationen verwendet, um zu beschreiben, wie Ihr Schaltkreis aufgebaut werden muss. Sie können sich aber bestimmt vorstellen, dass es schon recht zeitaufändig ist, zu Dokumentationszwecken für jedes Projekt eine Schaltskizze zu zeichnen.

Ähnliche Probleme werden früher oder später in jeder Disziplin auftauchen. Wenn Sie beispielsweise im Bereich Musik einen schönen Song geschrieben haben, müssen Sie ihn mittels Musiknoten zu Papier bringen.

Da Ingenieure praktisch veranlagte Menschen sind, haben sie einen Weg entwickelt, die Essenz eines Schaltkeises zu erfassen, um sie später zu dokumentieren oder an andere Personen weiterzuleiten.

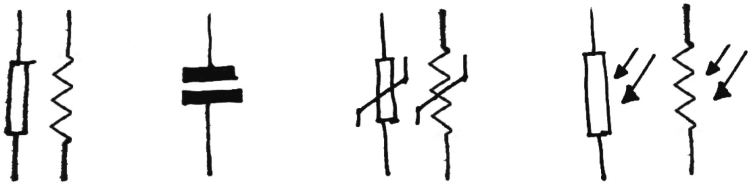
Im Bereich Elektronik ermöglichen *Schaltdiagramme* Schaltungen in einer Weise zu beschreiben, dass sie von den anderen Personen in einer Community verstanden werden. Einzelne Komponenten werden in Form von Symbolen dargestellt, bei denen es sich um eine Art Abstraktion der tatsächlichen Form der Komponente oder ihrer Essenz handelt. Der Kondensator beispielsweise besteht aus zwei Metallplättchen, die durch Luft oder Plastik voneinander separiert werden. Das entsprechende Symbol sieht demnach wie folgt aus:



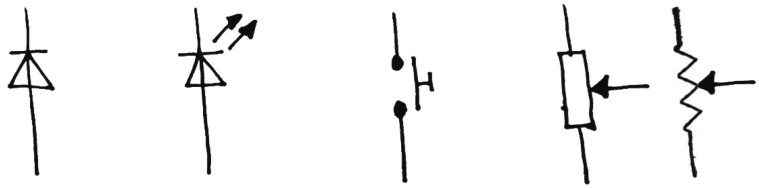
Ein weiteres schönes Beispiel ist der Induktor, der aus einem um einen Zylinder gewickelten Kupferdraht besteht. Daher sieht das entsprechende Symbol folgerichtig wie das auf der linken Seite dargestellte aus.

Die Verbindungen zwischen den Komponenten bestehen üblicherweise aus Drähten oder Leiterbahnen auf der Platine und werden in der Schaltskizze als einfache Linien dargestellt. Wenn zwei Drähte verbunden werden, wird diese Verbindung als großer Punkt an der Kreuzung der beiden Linien dargestellt, wie in der Abbildung links dargestellt.

Dies sind alle Informationen, die Sie für das Verständnis von Basis-Schaltskizzen benötigen. Hier eine Liste mit weitern Symbolen und den entsprechenden Bedeutungen:



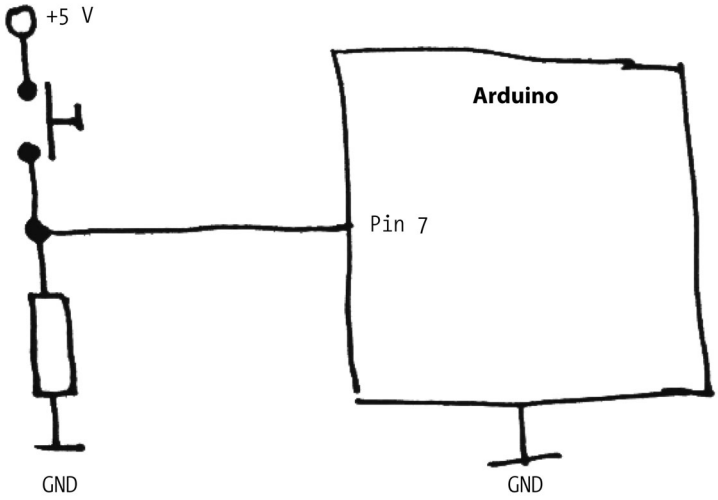
Widerstand Kondensator Thermistor LDR Licht-sensor



Diode LED Drucktaster Potentiometer

Möglicherweise werden Sie Variationen dieser Symbole begegnen (z.B. beide hier aufgeführten Symbole für den Widerstand). Eine umfangreichere Liste von Elektronik-Symbolen finden Sie unter **en.wikipedia.org/wiki/Electronic_symbol**. Konventionell werden Schaltdiagramme von links nach rechts gezeichnet. Beim Zeichnen eines Radios würden Sie demnach mit der Antenne auf der linken Seite beginnen und dann mit dem Weg fortfahren, den das Radiosignal bis zum Lautsprecher (der auf der rechten Seite gezeichnet wird) zurücklegt.

In der folgenden Schaltskizze ist der weiter vorne in diesem Buch erläuterte Drucktaster-Schaltkreis beschrieben:



Index

C

D

E

F

G

H

I

K

L

M

N

O

P

U

V

W

X

Z