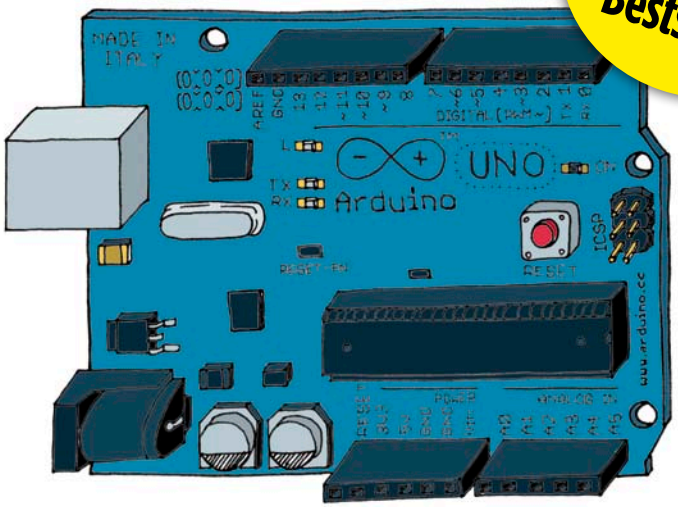
O'REILLY®

3. Auflage

Arduino für Einsteiger

Aktualisierter und erweiterter Bestseller



Die OPEN-SOURCE-PLATTFORM für MAKER

Massimo Banzi, Mitbegründer von Arduino, & Michael Shiloh

Übersetzung von Tanja Feder

3. AUFLAGE

Arduino für Einsteiger

Massimo Banzi &
Michael Shiloh

O'REILLY®
Beijing · Cambridge · Farnham · Köln · Sebastopol · Tokyo

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

Die Originalausgabe erschien im Dezember 2014 unter dem Titel
Getting Started with Arduino, 3rd bei Maker Media, Inc.

Bibliografische Information Der Deutschen Bibliothek

Die Deutsche Bibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über *http://dnb.dnb.de* abrufbar.

Lektorat: Volker Bombien, Köln
Übersetzung: Tanja Feder, Bonn
Korrektorat: Sabine Krämer, Gütersloh
Umschlaggestaltung: Karen Montgomery, Boston & Michael Oreal, Köln
Produktion: Karin Driesen, Köln
Satz: le-tex publishing services GmbH, Leipzig, *www.le-tex.de*
Belichtung, Druck und buchbinderische Verarbeitung:
Druckerei Kösel, Krugzell; *www.koeselbuch.de*

ISBN: 978-3-95875-055-5

Dieses Buch ist auf 100% chlorfrei gebleichtem Papier gedruckt.

Inhaltsverzeichnis

Vorwort

Der dritten Auflage von *Arduino für Einsteiger* wurden zwei neue Kapitel hinzugefügt: Bei Kapitel 8, *Ein System zur automatischen Gartenbewässerung*, auf Seite 93 handelt es sich um ein ehrgeiziges Projekt, in dem ein komplexerer Schaltkreis und ein komplizierteres Programm veranschaulicht werden. In diesem Kapitel werden auch Projektdesign, Testen und Konstruktion erläutert und Schaltpläne benutzt, die in Anhang D beschrieben werden.

Im zweiten neuen Kapitel, Kapitel 6, *Der Arduino Leonardo*, auf Seite 69, wird der Arduino Leonardo vorgestellt. Beim Leonardo handelt es sich um eine andere Art Arduino, weil der USB-Controller in der Software implementiert ist und sich nicht auf einem separaten Chip befindet, wie es vor dem Leonardo der Fall war. Dadurch kann das USB-Verhalten auf dem Board modifiziert werden.

Abgesehen von diesen neuen Kapitel hat es weitere Updates gegeben:

Die dritte Ausgabe ist für die Version 1.0.5 der Arduino-IDE geschrieben. Zwischenzeitlich ist die Arduino-IDE-Version 1.5 erschienen. In Vorwegnahme der Veröffentlichung von Version 1.5 haben wir die Unterschiede zwischen 1.0.5 und 1.5 aufgeführt.

Zahlreiche Vorschläge von Studenten und anderen Lesern wurden implementiert.

Michael

Vorwort zur 2. Auflage

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

Ich folgte unterbewusst meinem Instinkt, Elektronik auf die Weise zu lehren, wie ich es von der Schule her kannte. Später wurde mir klar, 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 einem sehr empirischen Weg erworben: mit sehr wenig Theorie, aber mit viel praktischer Erfahrung.

Ich begann also, darüber nachzudenken, auf welche Weise ich Elektronik wirklich verstanden habe:

- Ich nahm alle elektronischen Geräte auseinander, die ich in die Finger bekommen konnte.
- So lernte ich langsam alle 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 Explosion oder mit einer Rauchwolke.
- Ich baute Bausätze aus Beilagen von Elektrozeitschriften zusammen.
- Ich kombinierte von mir gehackte Geräte mit zweckentfremdeten Bausätze und anderen 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 auseinandergenommen. Je mehr nicht benutzte elektronische Objekte, die ich irgendwo im Haus fand und in ihre Einzelteile zerlegte, desto größer wurde mein Interesse. Schließlich brachten die Leute alle möglichen Geräte zu mir, damit ich sie auseinandernehmen sollte. 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 stapelten sich in unserem Haus große Mengen elektronische 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.

Dieses immer wieder erneute Lesen der Artikel auf der äußerst hilfreichen Grundlage des Wissens, das ich durch das Zerlegen von Schaltungen erworben hatte, erwies sich als langsamer, wirkungsvoller Prozess.

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, so dass eine Verbindung entstand. Oben auf diesen Würfeln war das jeweilige elektronische Symbol angeführt. Ich wusste noch wenig davon, dass dieses Spielzeug auch ein Meilenstein des Designs "Made in Germany" war, denn es war bereits in den 1960ern von Dieter Rams entwickelt worden.

Mit diesem neuen Tool konnte ich schnell Schaltkreise zusammenbauen, ausprobieren und mir dann 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 wiedergibt, bei der man einfach von einer bestimmten Idee ausgeht und bei einem völlig unerwarteten Resultat landet. Schließlich stieß ich auf das Wort Tinkering, das sich mit dem Begriff Tüfteln ins Deutsche übertragen ließe. Ich erkannte, dass 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 (*www.exploratorium.edu/tinkering*) 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.

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

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 ihnen folgte, lernte ich die Grundlagen der Binärmathematik und der Programmierung.

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

Nach der Rückkehr begann ich 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 Code-Zeilen weniger zeitaufwendig ist als das Aufbauen komplexer Schaltungen.

Mehr als 20 Jahre später denke ich, dass es diese Erfahrung ist, die es 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 mir seitdem bewahrt habe.

Massimo

Danksagung von Massimo Banzi

Dieses Buch ist Ombretta gewidmet.

Danksagung von Michael Shiloh

Dieses Buch ist meinem Bruder und meinen Eltern gewidmet.

Zuallererst möchte ich mich bei Massimo dafür bedanken, dass er mich eingeladen hat, an der dritten Auflage dieses Buches mitzuarbeiten und überhaupt in die Welt des Arduino einzutauchen. Es war ein wirkliches Privileg und ein wahres Vergnügen, an diesem Projekt teilzuhaben zu können.

Als Nächstes möchte ich Brian Jepson für seine Beratung, seine Betreuung, seine Ermutigung und seine Unterstützung danken. Bei Frank Teng möchte ich mich dafür bedanken, dass er mich stets in der Spur hielt. Ein weiter Dank gebührt Kim Cofer und Nicole Shelby, die einen fantastischen Job im Lektorat und der Herstellung gemacht haben.

Bei meiner Tochter Yasmine möchte ich mich dafür bedanken, dass sie eine solch hohe Meinung von mir hat, und dafür, dass sie mich unentwegt unterstützt und mich ermutigt hat, meine Interessen konsequent zu verfolgen. Ich danke ihr auch dafür, dass sie mich für irgendwie cool hält, obwohl ich ihr Vater bin. Ohne ihre Unterstützung hätte ich diese Arbeit nie geschafft.

Zu guter Letzt möchte ich mich bei bei meiner Partnerin Judy Aime' Castro für die endlosen Stunden bedanken, die sie damit verbracht hat, mein Gekritzel in schöne Illustrationen umzuwandeln, für die Diskussionen, die wir zu verschiedenen Aspekten dieses Buches geführt haben, und für ihre grenzenlose Geduld mit mir.

In diesem Buch benutzte Konventionen

Die folgenden typografischen Konventionen werden in diesem Buch verwendet:

Kursiv
: Steht für neue Begriffe, URLs, E-Mail-Adressen, Dateinamen und Dateierweiterungen.

`Feste Breite`
: Programmlistings und Programmelemente im Fließtext, zum Beispiel Variablen oder Funktionsnamen, Datenbanken, Datentypen, Umgebungsvariablen, Anweisungen und Schlüsselwörter.

`Feste Breite Fettdruck`
: Befehle oder anderer Text, der genau so vom Anwender eingegeben werden sollte.

`Feste Breite kursiv`
: Text, der vom Anwender durch eigene Werte ersetzt werden sollte.

Dieses Symbol zeigt einen Tipp, eine Empfehlung oder eine allgemeine Anmerkung.

Dieses Symbol zeigt eine Warnung oder Vorsichtsmaßnahme.

1/Einleitung

Arduino ist eine Open-Source-Plattform für *Physical Computing*, mit der sich interaktive Objekte herstellen lassen. Dabei kann es sich um Stand-Alone-Objekte handeln oder um solche, die mit der Software auf deinem Computer interagieren. Arduino ist für Künstler, Designer und andere Leute gedacht, die Physical Computing in ihr Design integrieren möchten, ohne dafür erst Elektrotechnik studieren zu müssen.

Die Arduino-Hardware und -Software ist Open Source. Durch die Open-Source-Philosophie wird eine Community gefördert, die ihr Wissen großzügig teilt. Für Anfänger ist dies eine tolle Sache, denn Hilfe ist meistens online jederzeit verfügbar, und zwar auf vielen verschiedenen Niveaus und in einer verblüffenden Bandbreite an Themen. Projekte werden nicht nur in Form von Bildern vorgestellt, sondern du findest auch Anweisungen, wie du dein eigenes Projekt entwickelst oder das vorgestellte Objekt als Ausgangspunkt für den Einbau in einer abgeleiteten oder ähnlichen Version nutzen kannst.

Die Arduino-Software, bekannt als Integrierte Entwicklungsumgebung (Integrated Development Environment, IDE), ist frei und steht unter *www.arduino.cc* jedem zum Download bereit. Die Arduino-IDE basiert auf der Sprache Processing *http://www.processing.org*, die entwickelt wurde, um Künstlern zu helfen, Computer-Art zu entwickeln, ohne erst zu Software-Ingenieuren werden zu müssen. Die Arduio-IDE kann unter Windows, Macintosh und Linux betrieben werden.

Das Arduino-Board ist preiswert (etwa 40 Euro) und recht tolerant gegenüber typischen Anfängerfehlern. Wenn du etwas tust, wodurch die Hauptkomponente auf dem Arduino Uno beschädigt wird, kostet es weniger als 4 Euro, sie zu ersetzen.

Das Arduino-Projekt wurde in einer Lernumgebung entwickelt und ist ein sehr beliebtes Lehrwerkzeug. Aufgrund derselben Open-Source-Philosophie, auf der die Communitys basieren, in denen großzügig Informationen, Antworten und Projekte geteilt werden, werden auch Lehrmethoden, Lehrinhalte und andere Informationen geteilt. Bei Arduino gibt es eine spezielle Mailing-Liste (*http://bit.ly/1vKh0wb*, um Diskussionen unter Nutzern zu erleichtern, die daran interessiert sind, mithilfe von oder über Arduino zu unterrichten.

Da die Arduino-Hardware und -Software Open Source sind, kannst du das Arduino-Hardware-Design herunterladen und dein eigenes entwickeln oder es als Ausgangspunkt für eigene Projekte nutzen, deren Design auf Arduino

basiert (oder Arduino enthält), oder einfach um zu verstehen, wie Arduino funktioniert. Dasselbe gilt für die Software.

Dieses Buch soll Einsteigern ohne Vorkenntnisse helfen, erste Schritte mit dem Arduino zu wagen.

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 lässt. Einer nannte die einleitenden Kapitel meines ersten Entwurfs sogar "fluff" (Staubfusseln). 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 "Staubfusseln" eintauchen.

Dieses Buch soll kein Lehrbuch für Elektronik und Programmierung sein, du wirst aber beim Lesen etwas über Elektronik und Programmierung lernen.

> Als Arduino allmählich beliebter wurde, konnte ich beobachten, 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 auch euch gewidmet.
>
> Massimo

Arduino basiert auf einer Diplomarbeit von Hernando Barragan, die er über die Wiring-Plattform schrieb, als er unter Casey Reas und mir am Interaction Design Institute (IDII) studierte.

Was ist Interaction Design?

Arduino wurde aus der Idee geboren, Interaction Design, eine Design-Disziplin, bei der das Prototyping im Zentrum der Methodik steht, zu vermitteln. Es gibt viele Definitionen für den Begriff Interaction Design, wir bevorzugen die folgende:

Interaction Design ist das Design einer beliebigen 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.

Früher musste man beim Einsatz von Elektronik ständig Ingenieure hinzuziehen, weil Schaltkreise aus kleinen Bauteilen zusammengesetzt wurden. Dadurch wurden kreative Menschen daran gehindert, mit dem Medium direkt zu experimentieren. Die meistens Tools waren für Ingenieure gedacht und setzten erhebliche Kenntnisse voraus.

In den letzten Jahren wurden Mikrocontroller billiger und einfacher in der Handhabung. Gleichzeitig wurden sie schneller und leistungsfähiger, wodurch die Herstellung besserer (und einfacherer) Tools für die Entwicklung ermöglicht wurde.

Der Fortschritt, den wir mit dem Arduino erlangten, bestand darin, dass diese neuen Tools Neulingen näher gebracht wurden, so dass es ihnen möglich wurde, nach nur einem zwei- oder dreitägigen Workshop bereits beeindruckende Dinge zu bauen. Mit Arduino können sich Designer und Künstler die Grundlagen von Elektronik und 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, Design zu erstellen, anstatt darüber zu sprechen. Sie besteht in einem andauernden Suchen nach schnelleren und leistungsstärkeren 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 liebgewonnen haben – grenzenlos mit einem Medium herumexperimentieren und dabei das Unerwartete entdecken. Bei dieser Suche nach Wegen, bessere Prototypen herzustellen, haben wir auch eine Reihe von Software-Paketen ausgewählt, die einen Prozess der ständigen Veränderung des Software- oder des Hardware-Mediums 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 Einsteiger gehen zunächst mit der Vorstellung an Elektronik heran, dass sie lernen müssen, alles von Grund auf selbst zu bauen. Das ist reine Energieverschwendung: Was du wirklich möchtest, ist die Bestätigung, dass etwas sehr schnell funktioniert, so dass du selbst motiviert bist, den nächsten Schritt zu unternehmen, oder dass du sogar jemand anderen motivierst, entsprechend großzügig in dich zu investieren.

Daher haben wir das *opportunistische Prototyping* entwickelt: Warum sollten wir Zeit und Energie darauf verschwenden, Dinge von Grund auf zu bau-

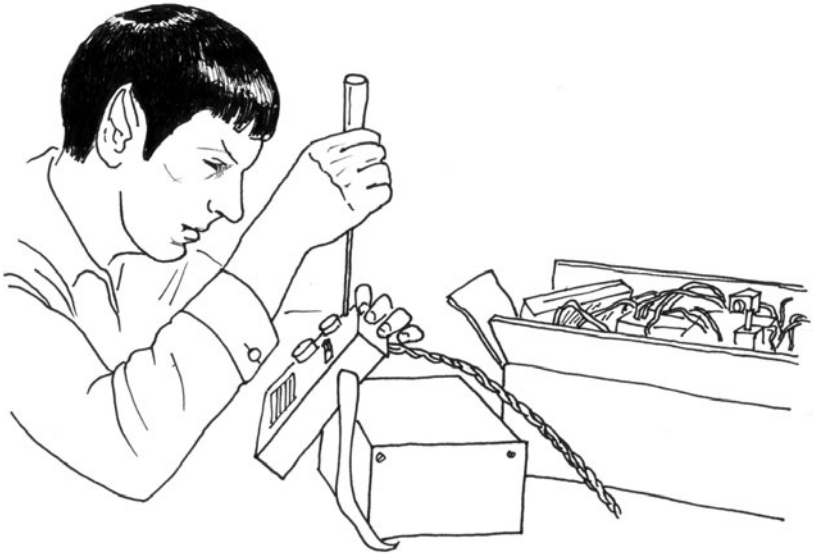
en, ein Prozess, der viel Zeit und tiefgehendes technisches Wissen erfordert, wenn wir fertige Geräte hacken und so die harte Arbeit nutzen können, die von großen Unternehmen und fähigen Ingenieuren bereits getan wurde?

Unser Held ist James Dyson, der 5.127 Prototypen seines Vakuumstaubsaugers baute, bevor er mit dem Resultat zufrieden war (*www.dyson.co.uk/*).

Tüfteln

Wir glauben, dass es essenziell ist, mit Technologie herumzuexperimentieren und verschiedene Möglichkeiten 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.

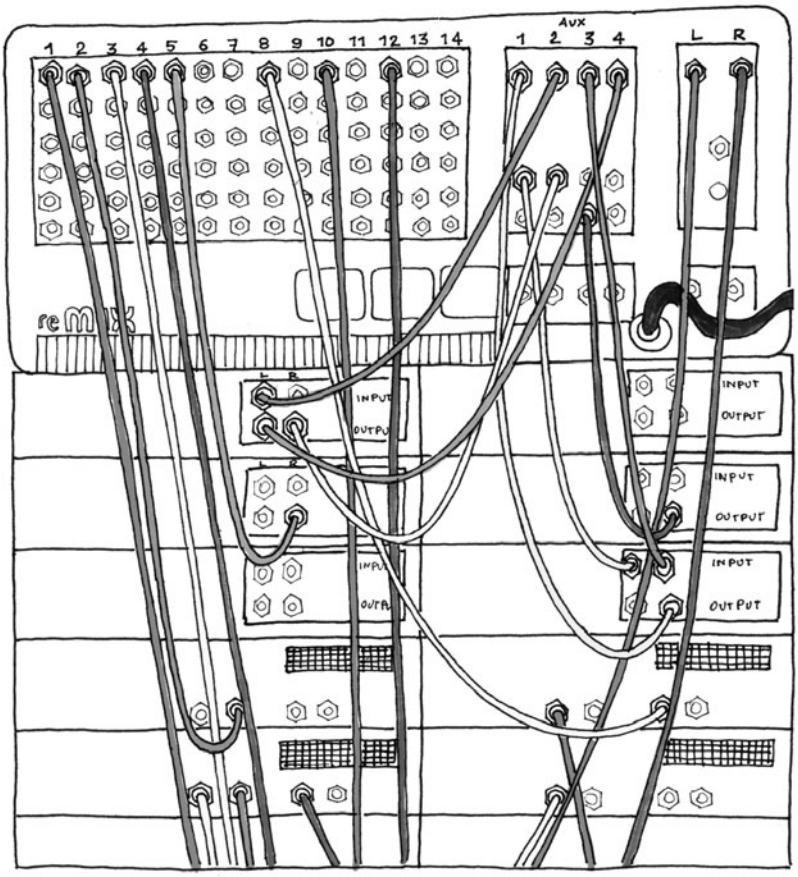


Patching

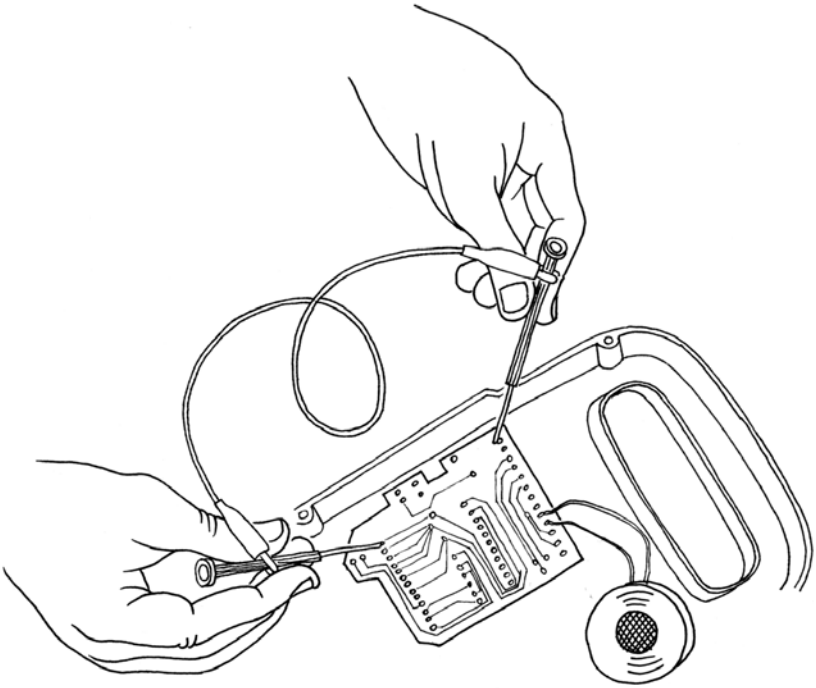
> Ich war immer fasziniert vom Baukastenprinzip und von der Möglichkeit, durch das Verbinden einfacher Geräte komplexe Systeme zu schaffen. Diese Methode wird sehr gut durch Robert Moog und seine analogen Synthesizer repräsentiert. Musiker erzeugten Sounds, indem sie verschiedene Module mit Kabeln zusammensteckten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

Möglichst wenige Unterbrechungen im Prozess zu haben, ist sehr wichtig für die Kreativität – je nahtloser der Prozess ist, desto besser funktioniert das Tüfteln.

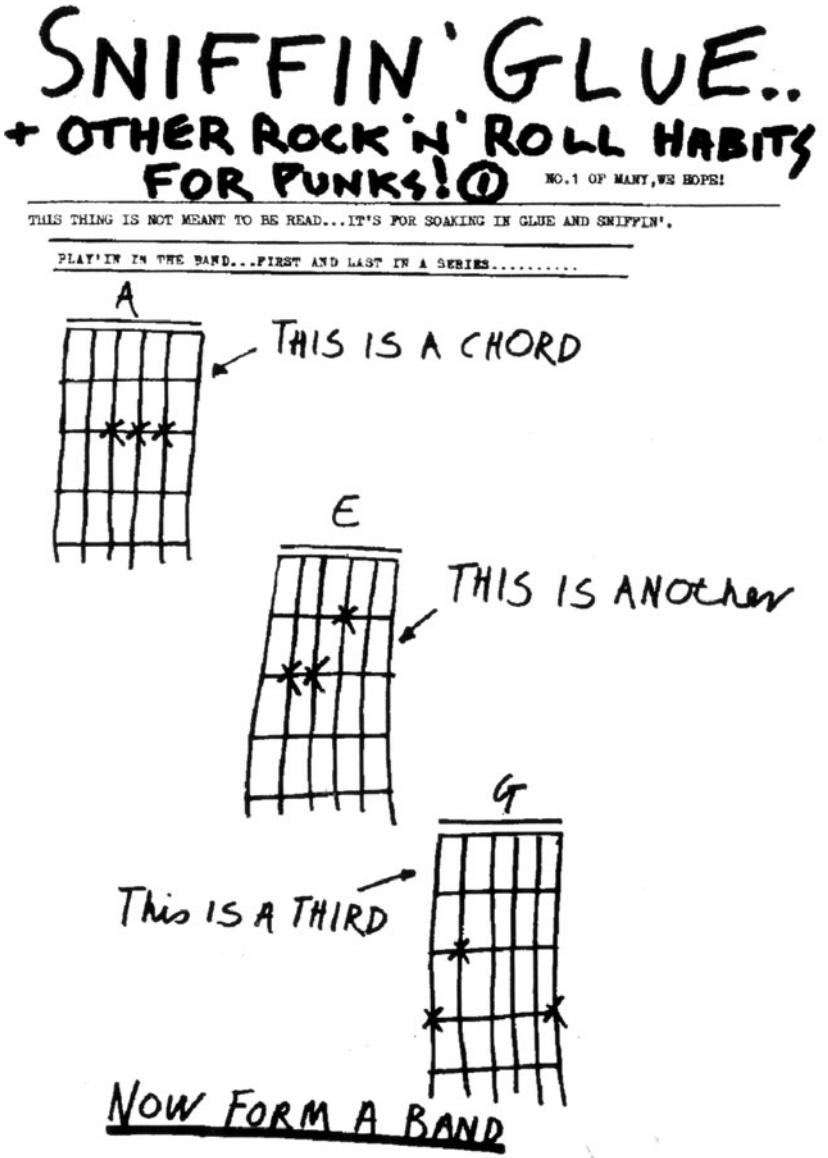
Diese Technik wurde in Form von Entwicklungsumgebungen für das visuelle Programmieren wie Max, Pure Data oder VVVV in die Welt der Software übertragen. Diese Tools lassen sich als Behälter für verschiedene Funktionen visualisieren, die sie bereitstellen. Indem sie diese Behälter miteinander verbinden, stellen die Nutzer dann *Patches* her. 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 du also eher visuell orientiert bist, empfehle ich dir, 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

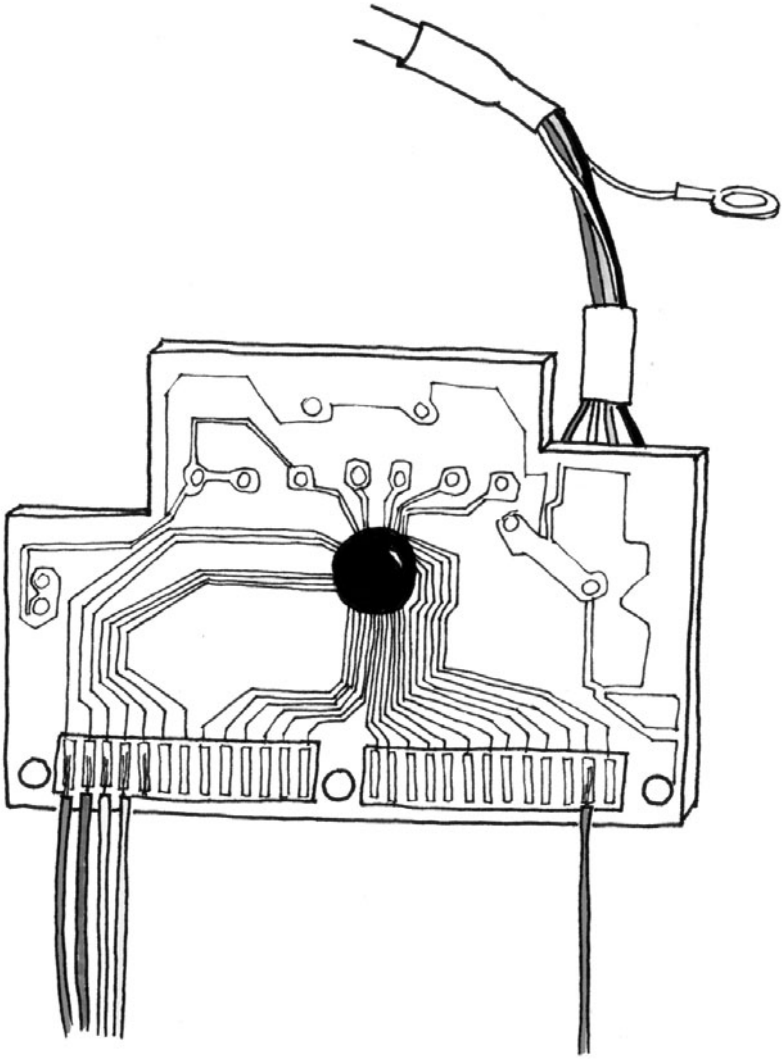


Modifizieren von Schaltkreisen ist eine der interessantesten Formen des Tüftelns. Es handelt sich um das kreative Kurzschließen von akustischen Geräten, die mit einer Niederspannungsbatterie betrieben werden, z. B. Pedale für Gitarreneffekte, Kinderspielzeug und kleine Synthesizer, um neue Musikinstrumente 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 seiner Schreibtischschublade kurzschloss, woraus eine Flut ungewöhnlicher Töne resultierte. Das Modifizieren von Schaltkreisen ist eine hervorragende Möglichkeit, durch das Zweckentfremden oder Modifizieren von Technologie die wildesten Geräte zu erschaffen, ohne dass dazu notwendigerweise ein theoretisches Verständnis ihrer Funktion erforderlich wäre.



Es ist ein wenig wie beim Fanzine *Sniffin Glue*, aus dem hier ein Auszug zu sehen ist: Während der Punk-Ära reichte die Kenntnis von drei Gitarren-Akkorden aus, um eine Band zu gründen. Lasst nicht zu, dass Experten in einem bestimmten Bereich euch vermitteln, dass ihr niemals zu ihnen gehören werdet. Ignoriert sie und überrascht 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: "Ent-

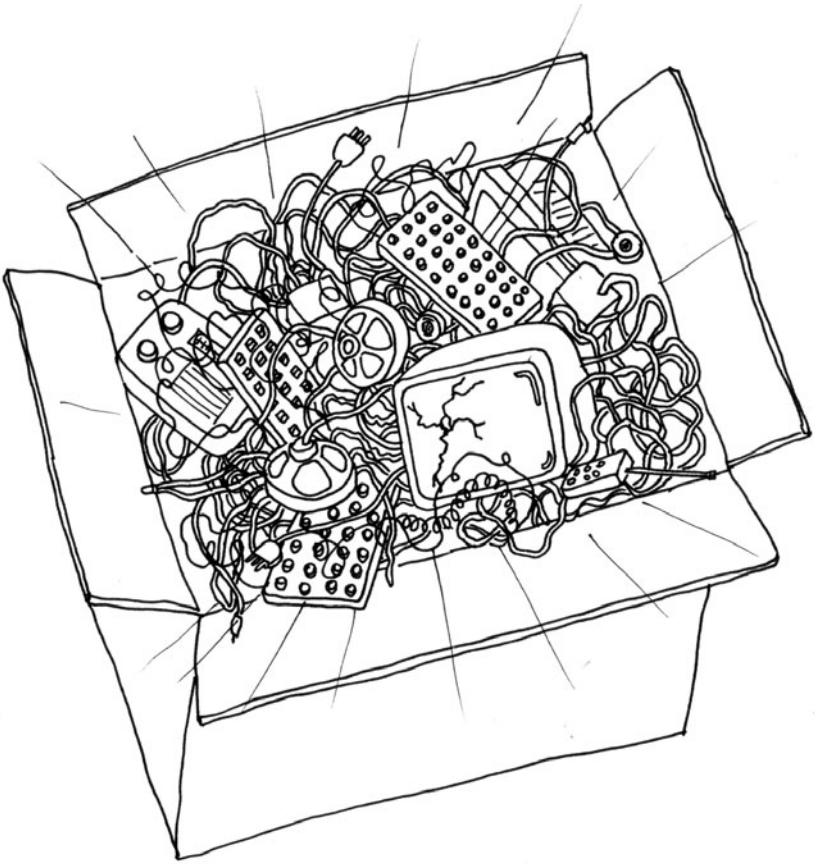
schuldigen Sie die Ausdrucksweise, aber Männerurinale sind intelligenter als Computer. Computer sind von ihrer Umgebung isoliert."[1]

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 Braun mit zwei Gruppen von Kontakten, die zu zwei Plastikschichten führen, die die Verbindungen zwischen den einzelnen Tasten beherbergen. Wenn du den Schaltkreis entfernst und zwei Kontakte mit einem Draht verbindest, erscheint ein ganzer Roman auf dem Computerbildschirm. Wenn du nun einen Bewegungssensor kaufst und mit der Tastatur verbindest, siehst du, dass jedes Mal, wenn jemand vor dem Computer herläuft, eine Taste gedrückt wird. In Verbindung mit der bevorzugten Software wird dein Computer so intelligent wie ein Urinal. Keyboard-Hacking ist eine Schlüsseldisziplin beim Prototyping und beim Physical Computing.

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 1990er entsorgte die Firma alles auf Schrottplätzen in der Gegend, unter anderem 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: Sammle Elektroschrott und schau ihn durch, bevor du etwas von Grund auf neu baust.

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



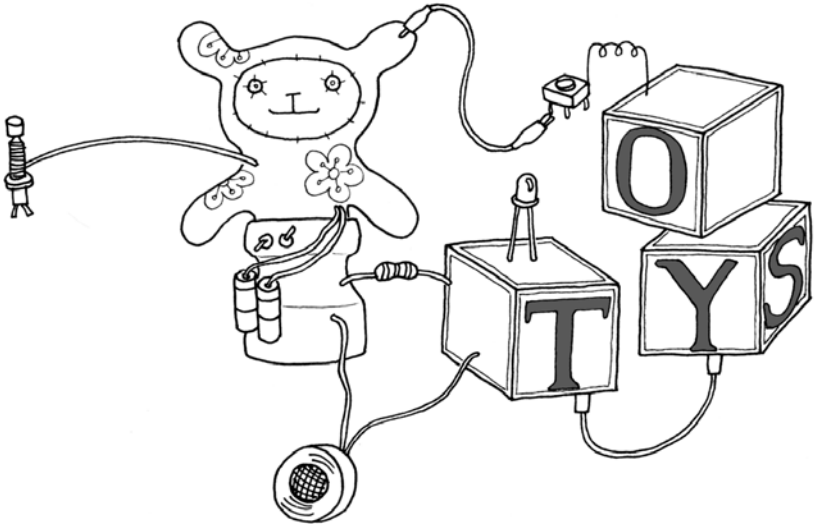
Hacken von Spielzeug

Spielzeug ist eine fantastische Quelle für billige Technologie zum Hacken und Weiterverwenden, wie es auch schon vorher beim Modifizieren von Schaltkreisen erwähnt wurde. Mit der aktuellen Flut Tausender billiger Spielzeuge aus China kannst du schnell kleine Ideen umsetzen, beispielsweise mit wenigen Spielzeugkatzen, die Geräusche von sich geben, und einigen Laserschwertern.

> Ich habe das einige Jahre lang getan, um meinen Studenten zu zeigen, dass Technologie nichts Furchteinflößendes hat und auch nicht schwierig 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 den-

ke, dass besagte Technik in diesem Handbuch perfekt beschrieben wurde, und ich verwende sie schon seit jeher.

Massimo



Kooperation

Die Kooperation von Nutzern ist eines der Schlüsselprinzipien der Arduino-Welt – über das entsprechende Forum unter *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 ein Wiki namens Playground eingerichtet (*www.arduino.cc/playground*), auf dem Benutzer ihre Erkenntnisse dokumentieren können. Es ist wirklich erstaunlich, wie viele Informationen diese Leute im Web bereitstellen, so dass sie sich jeder zunutze machen kann.

Diese Kultur des Teilens und gegenseitigen Helfens ist einer der Aspekte, die mich im Hinblick auf Arduino am meisten mit Stolz erfüllen.

Massimo

3/Die Arduino-Plattform

Arduino besteht aus zwei Hauptteilen: dem Arduino-Board, d.h. der Hardware, mit der du arbeitest, wenn du deine Objekte herstellst, und der Arduino-IDE, also der Software, die du auf deinem Computer ausführst. Mit der IDE kannst du einen Sketch (so nennen wir ein kleines Computerprogramm, das auf dem Arduino lauffähig ist) erstellen, das 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 verschiedener 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. um 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.
>
> Massimo

Schau dir dein Arduino-Board einmal an: Du kannst einen Chip mit 28 Beinchen erkennen. Dieser Chip ist vom Typ ATmega328 und das Herzstück deines Boards.

Es gibt in der Tat eine Vielzahl Arduino-Boards, das gängigste ist bisher das Arduino Uno, das hier beschrieben wird. In Kapitel 6, *Der Arduino Leonardo*, auf Seite 69 wird eines der anderen Arduino-Boards vorgestellt.

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 den Arduino Uno verwenden, der im Hinblick auf seine Handhabung am einfachsten ist und auf dem man am besten lernen kann. Die Anweisungen gelten für alle Arduinos, einschließlich der meisten neueren, aber auch der früheren Versionen. In Abbildung 3-1 ist der Arduino Uno abgebildet.

Wie in Abbildung 3-1 zu erkennen ist, weist der Arduino etliche Anschlussmöglichkeiten und zahlreiche Beschriftungen auf der Ober- und der Unterseite auf. Bei diesen Anschlüssen handelt es sich um Zugänge, mittels derer *Sensoren* und *Aktoren* angefügt werden. (Ein Aktor ist das Gegenteil eines Sensors: Ein Sensor erspürt etwas in der physikalischen Welt und wandelt es in ein Signal um, das vom Computer verstanden wird, wohingegen ein Aktor ein Signal von einem Computer in eine Aktion in der physikalischen Welt umsetzt. Du wirst in diesem Buch noch mehr über Sensoren und Aktoren erfahren.)

Zu Beginn sind all diese Anschlüsse möglicherweise ein wenig verwirrend. Im Folgenden werden die Eingangs- und Ausgangs-Pins erläutert, die du in diesem Buch benutzen wirst. Mach dir keine Gedanken, wenn du nach der Lektüre immer noch verwirrt bist – in diesem Buch werden zahlreiche neue Konzepte vorgestellt, bei denen es möglicherweise eine Weile dauert, bis du mit ihnen vertraut bist. Die betreffenden Erläuterungen werden wir mehrfach und auf verschiedene Weise wiederholen. Sie werden vor allem dann Sinn ergeben, wenn du damit beginnst, eigene Schaltkreise aufzubauen und die Resultate zu beobachten.

14 digitale I/O-Pins (Pins 0 bis 13)
: Hierbei kann es sich um Eingangs- oder Ausgangs-Pins handeln. Eingänge werden genutzt, um Informationen von einem Sensor auszulesen, während Ausgänge der Steuerung von Aktoren dienen. Die Richtung (AN oder AUS) wird im betreffenden Sketch definiert, der in der IDE erstellt wurde. Digitale Eingänge können nur einen von zwei Werten auslesen, und digitale Ausgänge können nur einen von zwei Werten ausgeben (HIGH oder LOW).

6 analoge In-Pins (Pins 0 bis 5)

Die analogen Eingangs-Pins werden benutzt, um Spannungswerte von einem analogen Sensor zu lesen. Im Gegensatz zu digitalen Eingängen, die nur zwischen zwei verschiedenen Levels (HIGH und LOW) unterscheiden können, verfügen analoge Eingänge über die Fähigkeit, 1.024 verschiedene Spannungswerte zu erfassen.

6 analoge Ausgangs-Pins (Pin 3, 5, 6, 9, 10 und 11)

Hierbei handelt es sich eigentlich um sechs der digitalen Pins, die eine dritte Funktion erfüllen können: Sie können einen analogen Ausgang bereitstellen. Wie bei den digitalen I/O-Pins kannst du in deinem Sketch festlegen, was ein Pin tun soll.

Das Board kann mit dem USB-Anschluss deines Computers, meistens sind das USB-Stecker, oder einem AC-Adapter (9 Volt, 2,5-mm-Klinkenstecker, Zentrum positiv) mit Strom versorgt werden. Wann immer eine entsprechende Verbindung zur Steckdose besteht, erfolgt die Stromzufuhr über diesen Weg. Anderenfalls wird für die Stromversorgung der USB-Anschluss benutzt. Beide Wege, die Steckdose und der USB-Anschluss, sind sichere Alternativen, um das Board mit Strom zu versorgen.

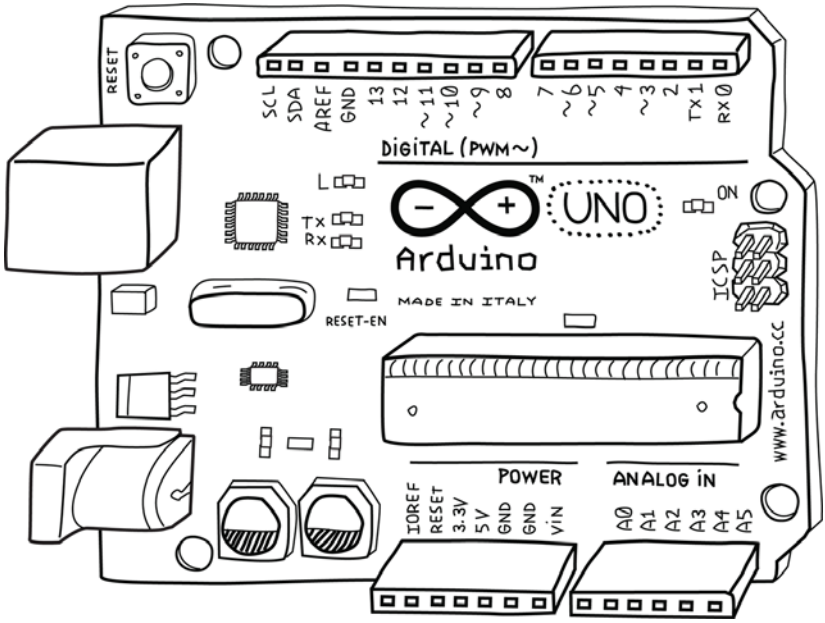


Abbildung 3-1: *Das Arduino-Uno-Board*

Die Integrierte Entwicklungsumgebung (IDE)

Die IDE (Integrated Development Environment = Integrierte Entwicklungsumgebung) ist ein spezielles Programm, das auf deinem Computer ausgeführt wird und das dir ermöglicht, Sketche für dein Arduino-Board in einer einfachen Sprache zu schreiben, die auf der Sprache Processing (*www.processing.org*) basiert. Die Magie offenbart sich, wenn du die entsprechende Schaltfläche drückst, um den Sketch auf das Board zu laden: Der Code, den du geschrieben hast, wird in die Sprache C übersetzt (die im Allgemeinen für einen Anfänger recht schwierig zu verstehen ist) und dann zum *avr-gcc-Kompiler* übertragen, bei dem es sich um einen sehr wichtigen Bestandteil von Open-Source-Software handelt und der die endgültige Übersetzung in die Sprache vornimmt, die vom Mikrocontroller verstanden wird. Dieser letzte Schritt ist sehr wichtig, da genau an dieser Stelle Arduino die Sache erheblich erleichtert, indem der größte Teil der Komplexität, die mit der Programmierung von Mikrocontrollern verbunden ist, ausgeblendet wird.

Der Programmierzyklus von Arduino umfasst folgende Schritte:

1. Schließe dein Board an einen USB-Anschluss deines Computers an.
2. Schreibe einen Sketch, der dem Board Leben einhaucht.
3. Lade den Sketch über die USB-Verbindung auf dein Board und warte einige Sekunden auf den Neustart des Boards.
4. Das Board führt den von dir geschriebenen Sketch aus.

Die Installation von Arduino auf dem Computer

Um das Arduino-Board zu programmieren, musst du zunächst die IDE installieren, indem du die entsprechende Datei von der Arduino-Website *www.arduino.cc/en/Main/Software* herunterlädst. Wähle die für dein Betriebssystem passende Version und folge dann den passenden Instruktionen in den folgenden Abschnitten.

Anweisungen zur Linux-Installation findest du im Abschnitt *Learning Linux* auf der Arduino-Website (*http://playground.arduino.cc/Learnung/Linux*).

Installieren der IDE: Macintosh

Wenn der Download der Datei beendet ist, öffnest du sie mit einem Doppelklick. Es wird ein Disk-Image geöffnet, das die Arduino-Anwendung enthält.

Zieh die Arduino-Anwendung in deinen *Applications*-Ordner.

Konfigurieren der Treiber: Macintosh

Beim Arduino Uno kommt ein Treiber zum Einsatz, der vom Macintosh-Betriebssystem bereitgestellt wird,so dass keine Installation erforderlich ist.

Da nun die IDE installiert ist, verbindest du deinen Arduino Uno mit deinem Macintosh via USB-Kabel.

Die grüne LED mit der Bezeichnung *PWR* auf dem Board sollte sich einschalten und die gelbe LED mit der Bezeichnung *L* sollte blinken.

Möglicherweise wird ein Popup-Fenster angezeigt, das darauf hinweist, dass eine neue Netzwerkschnittstelle entdeckt wurde.

Klicke 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. Schließe dann das Fenster System Preferences.

Nachdem du die Software konfiguriert hast, musst du den korrekten Port für die Kommunikation mit dem Arduino Uno auswählen.

Identifizierung des Ports: Macintosh

Rufe die Arduino IDE auf, entweder über den Ordner *Applications* oder unter Verwendung von Spotlight.

Wähle in der Arduino-IDE im Tools-Menü die Option Serial Port und dann den Port, der mit */dev/cu.usbmodem* oder */dev/tty.usbmodem* beginnt. Beide Ports gehören zu deinem Arduino-Board, es macht keinen Unterschied, welchen du auswählst.

In Abbildung 3-2 findest du eine Liste mit Ports.

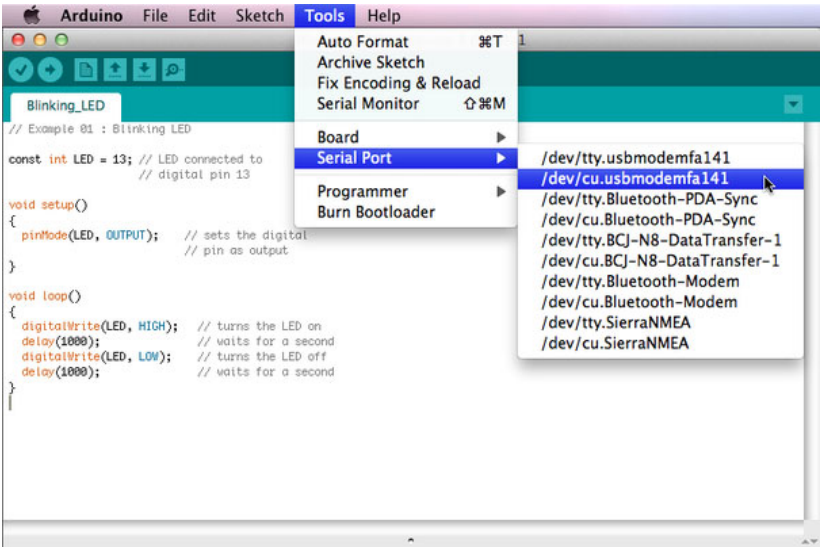


Abbildung 3-2: *Die Liste serieller Ports auf einem Macintosh in der Arduino IDE*

Du bist nun fast fertig. Als Letztes solltest du nur noch überprüfen, ob dein Arduino für den Board-Typ konfiguriert ist, den du nutzt.

Wähle in der Arduino-IDE aus dem Tools-Menü die Option Board und anschließend Arduino Uno. Wenn du ein anderes Board besitzt, musst du den entsprechenden Board-Typ auswählen (der Name des Boards ist neben dem Arduino-Symbol aufgedruckt).

GlüWu! Deine Arduino-Software ist installiert, konfiguriert und einsatzbereit. Du kannst nun mit Kapitel 4, *Die wirklich ersten Schritte mit Arduino*, auf Seite 23 fortfahren.

Falls dir einige dieser Schritte Schwierigkeiten bereiten, schau dir Kapitel 9, *Troubleshooting*, auf Seite 171 an.

Installieren der IDE: Windows

Wenn der Download der Datei beendet ist, öffne den Installer mit einem Doppelklick.

Dir wird nun eine Lizenz angezeigt. Lies die Lizenz, und wenn du einverstanden bist, klick auf die Schaltfläche I Agree.

Du erhältst eine Liste zu installierender Komponenten, bei der standardmäßig alle ausgewählt sind. Belasse es bei dieser Auswahl und klicke auf Next.

Als Nächstes wirst du aufgefordert, einen Installationsordner auszuwählen, wobei der Installer standardmäßig einen entsprechenden Vorschlag macht. Akzeptiere den Vorschlag, wenn kein guter Grund dagegen spricht, und klicke auf Install.

Der Installer wird den Fortschritt des Prozesses anzeigen, während er die Dateien extrahiert und installiert.

Wenn die Dateien installiert sind, wird ein Fenster mit der Frage geöffnet, ob die Treiber installiert werden dürfen. Klicke auf Install.

Wenn der Installer-Prozess abgeschlossen ist, klicke zum Beenden auf Close.

Konfigurieren der Treiber: Windows

Wenn die IDE installiert ist, schließe den Arduino Uno mit einem USB-Kabel an deinen Computer an.

Die grüne LED mit der Bezeichnung *PWR* auf dem Board sollte sich einschalten und die gelbe LED mit der Bezeichnung *L* sollte zu blinken beginnen.

Der Found New Hardware Wizard wird geöffnet, und Windows sollte automatisch die richtigen Treiber finden.

Falls bei diesen Schritten Probleme auftauchen, findest du Hilfestellungen unter Abschnitt *Probleme beim Installieren von Treibern unter Windows*, auf Seite 176 im Kapitel Kapitel 9, *Troubleshooting*, auf Seite 171.

Nachdem du die Software konfiguriert hast, musst du den korrekten Port für die Kommunikation mit dem Arduino Uno auswählen.

Port-Identifikation: Windows

Rufe die Arduino-IDE auf, entweder unter Verwendung eines Desktop-Shortcuts oder über das Startmenü.

Wähle in der Arduino-IDE im Tools-Menü die Option Serial Port. Es werden ein oder mehrere COM-Ports mit unterschiedlichen Nummern angezeigt. Notiere dir, welche Nummern verfügbar sind.

Trenne nun den Arduino von deinem Computer. Wirf erneut einen Blick auf die Liste mit den Ports und schau dir an, welche COM-Ports gelöscht werden. Dies wird wahrscheinlich einen Augenblick dauern, und möglicherwei-

se musst du das Tools-Menü verlassen und erneut öffnen, um die Liste der Ports zu aktualisieren.

Falls die Identifikation des von deinem Arduino Uno benutzten COM-Ports Schwierigkeiten bereitet, findest du Hilfe unter Abschnitt *Identifizierung der Arduino-COM-Ports unter Windows*, auf Seite 177 im Kapitel 9, *Troubleshooting*, auf Seite 171.

Wenn du die COM-Port-Zuweisungen kennst, kannst du diesen Port über Tools→Serial Port in der Arduino-IDE auswählen.

Du bist nun fast fertig. Als Letztes solltest du nur noch überprüfen, ob der Arduino für den Board-Typ konfiguriert ist, den du benutzt.

Wähle in der Arduino-IDE aus dem Tools-Menü die Option Board und anschließend Arduino Uno. Wenn du ein anderes Board besitzt, musst du den entsprechenden Board-Typ auswählen (der Name des Boards ist neben dem Arduino-Symbol aufgedruckt).

Gratulation! Deine Arduino-Software ist installiert, konfiguriert und einsatzbereit. Du kannst nun mit Kapitel 4, *Die wirklich ersten Schritte mit Arduino*, auf Seite 23 fortfahren.

4/Die wirklich ersten Schritte mit Arduino

Als Nächstes wirst du erfahren, wie du ein interaktives Gerät bauen und programmieren kannst.

Der Aufbau eines interaktiven Gerätes

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 in der Software beschrieben ist. Das Gerät ist dann in der Lage, mithilfe von *Aktoren* (elektronische Komponenten, die ein elektrisches Signal in physikalisches Verhalten umwandeln können) mit der Welt zu interagieren.

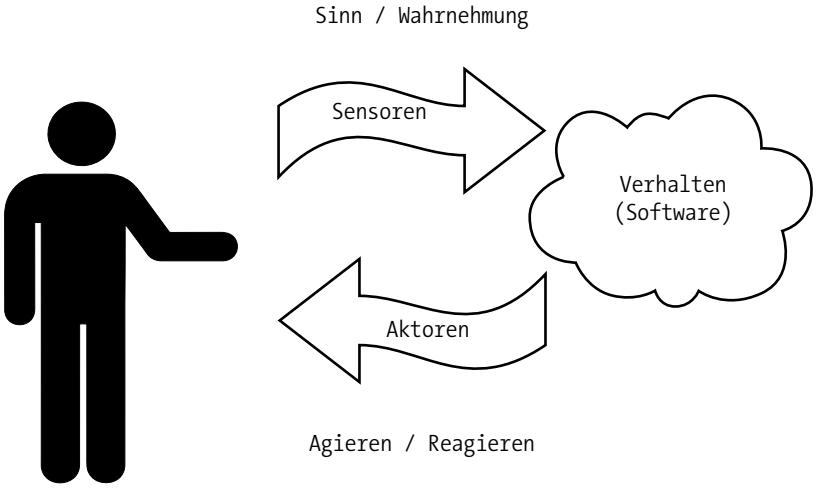


Abbildung 4-1: *Das interaktive Gerät*

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 sie in Elektrizität umgewandelt werden. In unserem Körper wandelt das Auge Licht in Signale um, die über Nerven ans Gehirn weitergeleitet werden. In der Elektronik können wird dazu ein spezielles Bauteil verwenden, nämlich einen *lichtabhängigen Widerstand* oder *LDR*, auch als *Fotowiderstand* bekannt, der die auftreffende Lichtmenge messen und als Signal wiedergeben kann, das der Mikrocontroller versteht.

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 unseren 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 wirst du 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 du ausführen solltest, um zu testen, ob dein 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.

Dein Arduino-Board wird mit einer vorinstallierten LED geliefert, die mit einem *L* auf dem Board gekennzeichnet ist. Diese vorinstallierte LED ist mit dem Pin Nummer 13 verbunden. Merke dir die Nummer, denn wir werden sie später noch brauchen. Du kannst auch deine eigene LED hinzufügen. Schließe sie so an, wie es in Abbildung 4-2 dargestellt ist. Achte darauf, sie in das Verbindungsloch mit der Nummer 13 zu stecken.

Wenn die LED über längere Zeit leuchten soll, solltest du einen Widerstand verwenden, wie in Abschnitt *Steuerung von Licht mittels PWM*, auf Seite 50 beschrieben wird.

K kennzeichnet die Kathode (negativ), bei der es sich um den kürzeren Anschlussdraht handelt, und A kennzeichnet die Anode (positiv), die den längeren Anschlussdraht aufweist.

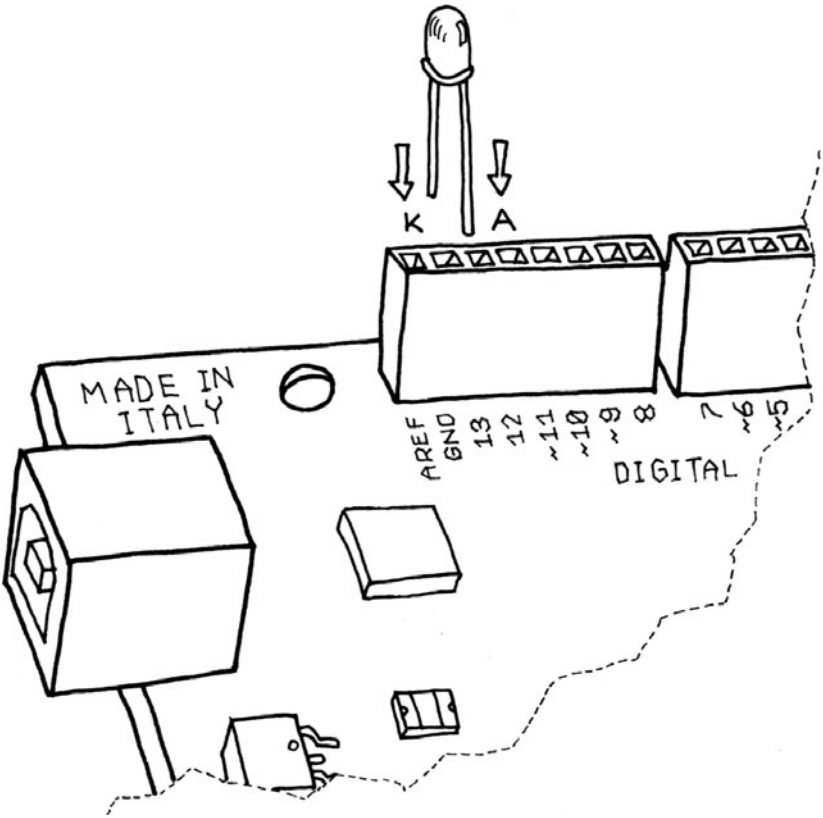


Abbildung 4-2: *Anschließen einer LED an das Arduino-Board*

Wenn die LED angeschlossen ist, muss du dem Arduino mitteilen, was zu tun ist. Dies erfolgt mittels Code, einer Liste von Anweisungen, die du wiederum dem Mikrocontroller übermittelst und mit der du ihn dazu bringst, das zu tun, was du möchtest. (Die Begriffe *Code*, *Programm* und *Sketch* beziehen sich alle auf dieselbe Liste mit Anweisungen).

Öffne auf deinem Computer die Arduino-IDE (beim Mac befindet sie sich im Ordner Applications, unter Windows findest du einen entsprechenden Shortcut entweder auf dem Desktop oder im Startmenü). Starte die IDE mit einem Doppelklick auf das entsprechende Symbol. Wähle File → New aus. Du wirst nun aufgefordert, einen Ordnernamen für den Sketch anzugeben. Hier wird der Sketch dann gespeichert. Nenne ihn *Blinking_LED* und klicke auf OK. Gib dann den folgenden Sketch (Beispiel 4-1) in den Sketch-Editor von Arduino (das Hauptfenster der Arduino-IDE) ein. Du kannst ihn auch über den Link zu den Code-Beispielen auf der Katalogseite zu diesem Buch (*http://bit.ly/start_arduino_3e*) herunterladen.

Eine dritte Möglichkeit besteht darin, den Sketch einfach durch Klicken auf File → Examples → 01.Basics → Blink zu laden; den größten Lerneffekt erzielst du aber, wenn du ihn selbst eingibst.

Er sollte wie in Abbildung 4-3 aussehen.

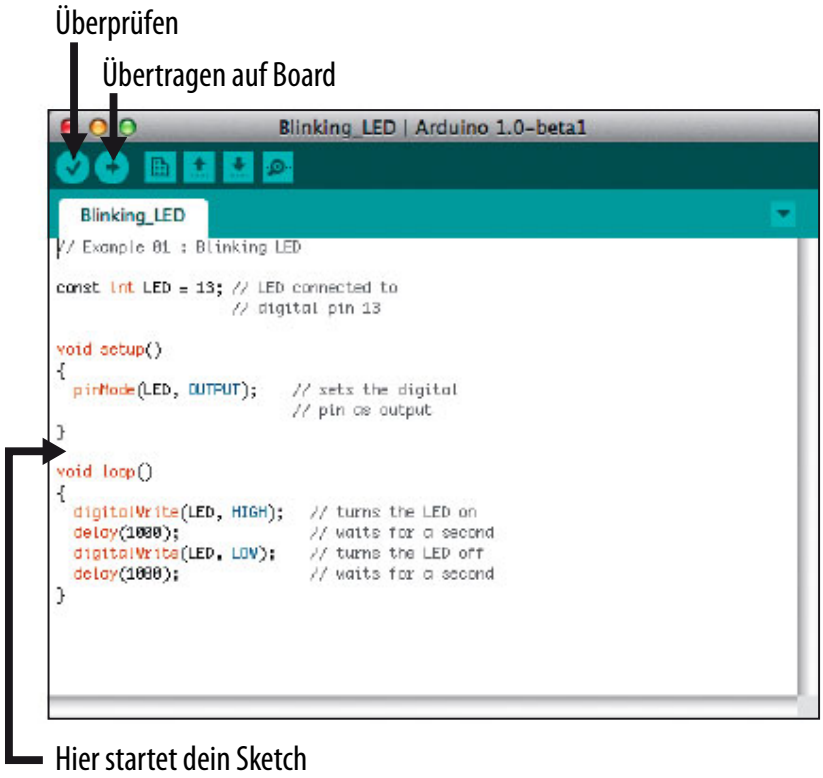


Abbildung 4-3: *Die Arduino-IDE mit dem ersten geladenen Sketch*

Beispiel 4-1: *Eine LED zum Blinken bringen*

```
// 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
}
```

Nun, da sich der Code in deiner IDE befindet, musst du ihn auf Fehler überprüfen. Klicke auf die Schaltfläche Überprüfen (Abbildung 4-3; wenn alles korrekt ist, wird dir die Nachricht *Done compiling* am unteren Rand der Arduino-IDE angezeigt. Diese Nachricht bedeutet, dass die Arduino-IDE deinen Sketch in ein ausführbares Programm übersetzt hat, das auf deinem Board ausgeführt werden kann, ähnlich wie das bei einer .exe-Datei unter Windows oder bei einer .app-Datei unter Mac der Fall ist.

Wenn du eine Fehlermeldung erhältst, hast du sehr wahrscheinlich einen Fehler bei der Eingabe des Codes gemacht. Schau dir jede Zeile genau an und überprüfe jedes einzelne Zeichen, besonders solche wie runde Klammern, eckige Klammern, Semikolons und Kommata. Vergewissere dich, dass du Groß- und Kleinschreibung sorgfältig übernommen hast und dass du den Buchstaben O und die Zahl 0 korrekt verwendet hast.

Wenn der Code fehlerfrei ist, kannst du deinen Sketch auf dein Board laden: Klicke auf die Schaltfläche Übertragen auf Board (siehe Abbildung 4-3). Dadurch wird die IDE angewiesen, den Upload-Prozess zu starten, wodurch das Arduino-Board zurückgesetzt und somit alle laufenden Prozesse beendet werden. Dann wartet das Board auf Instruktionen, die vom USB-Port kommen. Anschließend sendet die Arduino-IDE den Sketch zum Arduino-Board, das ihn wiederum in seinem permanenten Speicher speichert. Sobald die IDE den gesamten Sketch übertragen hat, führt das Arduino-Board den Sketch aus.

Dies geschieht recht schnell. Wenn du die Schaltfläche der Arduino-IDE im Auge behältst, wirst du einige Nachrichten im schwarzen Bereich am unteren Rand des Fenster sehen, und genau über diesem Bereich die Meldungen

Compiling, dann Uploading und schließlich Done uploading, mit denen du darüber informiert wirst, dass der Prozess erfolgreich abgeschlossen wurde.

Es sind zwei LEDs mit den Bezeichnungen RX und TX auf dem Arduino-Board vorhanden. Diese flackern jedes Mal auf, wenn ein Byte vom Board geschickt oder empfangen wird. Während des Upload-Prozesses flackern sie kontinuierlich. Auch dies geschieht recht schnell, und wenn du nicht zur richtigen Zeit auf dein Arduino-Board schaust, wirst du diesen Vorgang verpassen.

Falls du kein Flackern der LEDs erkennen kannst oder anstelle der Nachricht Done Uploading eine Fehlermeldung erhältst, besteht ein Kommunikationsproblem zwischen deinem Computer und dem Arduino. Vergewissere dich, dass du den richtigen seriellen Anschluss (siehe Kapitel 3, *Die Arduino-Plattform*, auf Seite 15) unter Tools→Serial Port ausgewählt hast. Überprüfe außerdem, ob unter Tools→Board das richtige Arduino-Modell ausgewählt wurde.

Wenn weiterhin Probleme bestehen, schau dir das Kapitel Kapitel 9, *Troubleshooting*, auf Seite 171 an.

Sobald der Code auf dein Arduino-Board übertragen wurde, verbleibt er dort, bis du ihn mit einem anderen Sketch überschreibst. Der Code bleibt gespeichert, wenn du beim Board einen Reset durchführst oder es ausschaltest, das ist ähnlich wie bei den Daten auf deiner Computer-Festplatte.

Wenn der Sketch korrekt geladen wurde, wird die LED L für eine Sekunde aufleuchten und dann für eine Sekunde dunkel bleiben. Wenn du eine separate LED installiert hast, wie vorher in Abbildung 4-2 zu sehen ist, wird auch diese LED blinken. Das, was du geschrieben hast, ist ein *Computer-Programm* oder auch *Sketch*, wie ein Arduino-Programm genannt wird. Wie schon erwähnt, handelt es sich beim Arduino um einen kleinen Computer, der sich nach Bedarf programmieren lässt. Um eine Serie von Anweisungen in die Arduino-IDE einzugeben, wird eine Programmiersprache verwendet, die die Anweisungen so umwandelt, dass sie vom Arduino-Board ausgeführt werden können.

Als Nächstes möchten wir ein Verständnis für den Code vermitteln. Zunächst ist zu erwähnen, dass Arduino den Code der Reihe nach von oben nach unten ausführt. Die erste Zeile oben ist also die, die zuerst gelesen wird. Dann wird der Prozess nach unten fortgesetzt. Dies erinnert ein wenig an die Statusanzeige bei einen 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 du dich gerade befindest.

Reich mir den Parmesan

Achte auf die geschweiften Klammern, die dazu dienen, Code-Zeilen zusammenzufassen. Sie sind insbesondere dann sehr nützlich, wenn du eine Gruppe Anweisungen mit einem Namen versehen möchtest. Mit der Aufforderung "Bitte reich' mir den Parmesankäse" beim Abendessen wird beispielsweise eine ganze Reihe von Aktionen angestoßen, die in diesem kleinen Satz zusammengefasst sind. Weil wir Menschen sind, erfassen wir das ganz selbstverständlich, beim Arduino hingegen müssen alle einzelnen kleinen Aktionen ausformuliert werden, weil die Plattform nicht so leistungsfähig wie unser Gehirn ist. Um also mehrere Anweisungen in einer Gruppe zusammenzufassen, platzierst du ein { vor dem Code-Block und ein } hinter dem Code.

Du siehst, dass in unserem Beispiel zwei Blöcke auf diese Weise definiert wurden. Vor jedem dieser Blöcke sind ein paar merkwürdige Wörter angeführt:

```
void setup()
```

Mit dieser Zeile wird dem Code-Block ein Name zugewiesen. Wenn du eine Liste mit Anweisungen schreiben würdest, die Arduino beibringen, dir den Parmesankäse zu reichen, würdest du `void passTheParmesan()` am Anfang des Blocks schreiben – und dieser Block würde zu einer Anweisung, die du von jeder beliebigen Stelle im Arduino-Code aus aufrufen könntest. Solche Blöcke werden als Funktionen bezeichnet. Wenn du also aus diesem Code-Block eine Funktion erstellt hast, kannst du anschließend `passTheParmesan()` an jeder beliebigen Stelle im Sketch schreiben, und der Arduino wird zur Funktion `passTheParmesan()` springen, die betreffenden Anweisungen ausführen, dann zurückspringen und an der Stelle fortfahren, an dem der Code vor den Anweisungen verlassen wurde.

Dies zeigt einen wichtigen Aspekt eines jeden Arduino-Programms. Beim Arduino können nicht mehrere Prozesse gleichzeitig ablaufen, das bedeutet, es wird zu jedem Zeitpunkt immer nur eine Anweisung ausgeführt. Da der Arduino dein Programm Zeile für Zeile ausführt, wird zum gegebenen Zeitpunkt immer nur diese eine Zeile *ausgeführt*. Nach einem Sprung zu einer Funktion wird diese Funktion Zeile für Zeile ausgeführt, bevor dann eine Rückkehr zum Ausgangspunkt erfolgt. Der Arduino kann nicht zwei Sätze an Anweisungen gleichzeitig ausführen.

Arduino ist nichts für Zauderer

Arduino erwartet immer, dass du zwei Funktionen erstellt hast: 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 werden soll. Dies liegt daran, dass der 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 du die Ausführung beenden möchtest, musst du dazu einfach das Board von der Stromversorgung trennen.

Wirkliche Tüftler schreiben Kommentare

Jeglicher Text, der mit `//` beginnt, wird vom Arduino ignoriert. Diese Zeilen sind Kommentare, die du für dich selbst im Programm hinterlässt, um dich daran zu erinnern, was du mit dem Code bezweckt hast, oder die du für andere schreibst, damit sie den Code verstehen.

Es ist absolut üblich (und wir wissen das, weil wir es kontinuierlich tun), einen Code-Abschnitt 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. Du wirst dir den Code anzeigen lassen, und wenn du dann keine entsprechenden Kommentare in deinem ursprünglichen Programm eingefügt hast, wirst du sehr schnell denken: "Oh Mann, was für ein Chaos! Wo fange ich denn da bloß an?" Wenn wir in diesem Buch weiter voranschreiten, wirst du noch einige Tricks kennenlernen, wie du dein Programm lesbarer und einfacher im Hinblick auf die Wartung gestaltest.

Wenn du später deine eigene Sketche schreiben wirst und glaubst, dass sie auch für Tüftler interessant sind, die nicht Deutsch verstehen, dann solltest du auf jeden Fall dich darum bemühen, die Kommentare in englischer Sprache zu verfassen. Du erreichst dann viel mehr Leute mit deinem Sketch.

Der Code – Schritt für Schritt

> Womöglich wird dir 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* – oh, was für ein Albtraum!). Für jede Textzeile gab es hundert Zeilen Kommentar. Wenn du beginnst, eigene Programme zu schreiben, sind solche Erläuterungen wesentlich nützlicher.
>
> Massimo

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

Ein Kommentar ist eine hilfreiche Möglichkeit, kleine Hinweise einzufügen. 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 die Bezeichnung für eine Ganzzahl handelt, die nicht geändert werden kann (man nennt das auch eine Konstante) und für die der Wert 13 festgelegt wurde. Das ist vergleichbar mit einem automatischen Suchen-&-Ersetzen-Vorgang im Code. In unserem Fall wird der Arduino angewiesen, immer wenn das Wort *LED* erscheint, die Zahl 13 zu schreiben.

Der Befehl wird hier verwendet, um festzulegen, dass die vorinstallierte LED, die wir schon erwähnt haben, an den Arduino-Pin 13 angeschlossen ist. Eine übliche Konvention besteht darin, Großbuchstaben für Konstanten zu verwenden.

```
void setup()
```

Mit dieser Zeile wird dem Arduino mitgeteilt, dass es sich beim nächsten Code-Block um eine Funktion namens `setup()` handelt.

```
{
```

Mit der öffnenden geschweiften Klammer wird ein Code-Block eingeleitet.

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

Endlich eine wirklich interessante Anweisung! Mittels `pinMode()` wird dem Arduino mitgeteilt, wie ein bestimmter Pin konfiguriert werden soll. Digitale Pins können entweder als INPUT oder OUTPUT (Eingang oder Ausgang) verwendet werden. Wir müssen dem Arduino aber mitteilen, wie wir einen Pin benutzen möchten.

In unserem Beispiel benötigen wir einen Ausgangs-Pin, um die LED zu steuern.

Bei `pinMode()` handelt es sich um eine Funktion, und die Wörter (oder Zahlen) innerhalb der Klammern werden als Argumente bezeichnet. Argumente sind jegliche Informationen, die eine Funktion benötigt, um ihre Aufgabe zu erfüllen.

Die Funktion `pinMode()` benötigt zwei Argumente. Mit dem ersten Argument wird `pinMode()` mitgeteilt, auf welchen Pin wir uns beziehen, und mit dem zweiten Argument erhält `pinMode()` die Information, ob wir den Pin als Input oder Output nutzen möchten. `INPUT` und `OUTPUT` sind vordefinierte Konstanten in der Arduino-Sprache.

Erinnere dich daran, dass es sich bei dem Wort *LED* um den Namen der Konstanten handelt, für die die Zahl 13 festgelegt wurde, die Nummer des Pins, an den die LED angeschlossen ist. Das erste Argument ist also `LED`, der Name der Konstanten.

Das zweite Argument ist `OUTPUT`, denn wenn ein Arduino mit einem Aktor kommuniziert, werden Informationen *aus*gesendet.

```
}
```

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

```
void loop()
{
```

In `loop()` wird das hauptsächliche Verhalten des interaktiven Bauteils festgelegt. Die Funktion wird immer weiter wiederholt, und zwar solange das Board mit Strom versorgt wird.

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

Wie der Kommentar schon sagt, ist `digitalWrite()` in der Lage, jeden Pin, der als OUTPUT konfiguriert wurde, ein- oder auszuschalten. Genau wie wir es bei der Funktion `pinMode()` gesehen haben, erwartet auch `digitalWrite()` zwei Argumente, und wie bei der Funktion `pinMode()` teilt das erste Argument `digitalWrite()` mit, auf welchen Pin wir uns beziehen, und wie bei der Funktion `pinMode()` benutzen wir den Konstantennamen `LED` und beziehen uns so auf den Pin mit der Nummer 13, an den die vorinstallierte LED angeschlossen ist.

Das zweite Argument unterscheidet sich davon: In diesem Fall teilt das Argument `digitalWrite()` mit, ob der Spannungspegel auf 0 (`LOW`) oder auf 5V (`HIGH`) gesetzt werden soll.

Stell dir vor, dass der Ausgangs-Pin so eine kleine Steckdose ist wie die in den Wänden deiner Wohnung. Europäische Steckdosen liefern 230V, amerikanische 110V und Arduino arbeitet mit gemäßigten 5V. Die Magie offenbart sich hier, wenn die Software die Hardware steuern kann. Wenn du `digitalWrite(LED, HIGH)` schreibst, wird der Ausgangs-Pin auf 5V gesetzt. Schließt du 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 *Aktor*.

Auf dem Arduino bedeutet HIGH, dass 5V am Pin anliegen, wohingegen bei LOW der Pin auf 0V gesetzt ist.

Du fragst dich vielleicht, warum wir HIGH und LOW anstelle von ON und OFF verwenden. Es stimmt, dass HIGH oder LOW normalerweise ON bzw. OFF entsprechen, dies hängt aber davon ab, wie der Pin benutzt wird. Eine LED beispielsweise, die zwischen 5V und einem Pin angeschlossen ist, wird eingeschaltet, wenn dieser Pin LOW ist, und ausgeschaltet, wenn er HIGH ist. In den meisten Fällen kannst du allerdings nur vorspiegeln, dass HIGH gleichbedeutend ist mit ON und LOW mit OFF.

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

Obwohl der Arduino sehr viel langsamer als dein Laptop ist, ist er doch immer noch sehr schnell. Wenn wir die LED schnell an- und sofort wieder ausschalten würden, könnten deine Augen dies nicht sehen. Wir müssen die LED eine Weile eingeschaltet lassen, damit wir es sehen können. Hierzu müssen wir den Arduino anweisen, eine Zeitlang zu warten, bevor mit dem nächsten Schritt fortgefahren wird. Mit `delay()` weist du im Grunde genommen den Prozessor an, zu pausieren und nichts zu tun, und zwar für die Zeitdauer in Millisekunden, die du als Argument übergibst. Eine Millisekunde ist ein Tausendstel einer Sekunde, also sind 1.000 Millisekunden eine Sekunde. In unserem Beispiel wird die LED demnach eine Sekunde lang leuchten.

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

Mit dieser Anweisung wird die LED, die wir vorher eingeschaltet haben, ausgeschaltet.

```
    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. Wenn der Arduino hier angelangt ist, wird wieder beim Beginn von `loop()` gestartet.

Zusammengefasst macht das Programm Folgendes:

- Pin 13 wird als Ausgangs-Pin definiert (nur einmal 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.

Wir hoffen, dass dir dieser Code noch keine allzu großen Kopfschmerzen bereitet hat. Lass dich nicht entmutigen, wenn du nicht alles verstanden hast. Dir sind diese Konzepte ja noch neu. Es wird eine Weile dauern, bis sich ihr

Sinn wirklich erschließt. Du wirst 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, so dass wir unterschiedliche Blinkmuster beobachten können. Insbesondere solltest du darauf achten, was geschieht, wenn die Verzögerungen sehr klein sind und sich in den Ein- und Ausphasen unterscheiden ... du kannst dabei nämlich für einen Moment etwas beobachten, das später in diesem Buch, wenn wir zum Stichwort *Pulsweitenmodulation* in Abschnitt *Steuerung von Licht mittels PWM*, auf Seite 50 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 Glück, 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.
>
> Massimo

In diesem Kapitel sowie in den Kapiteln 5 und 7 werden wir uns damit befassen, wie sich *interaktive Lampen* herstellen lassen. Mithilfe des Arduino, den wir hier verwenden, werden wir die Grundlagen kennenlernen, die erforderlich sind, um interaktive Geräte zu bauen. Denk dabei immer daran, dass der Arduino nicht wirklich versteht und es ihm gleichgültig ist, was du am Ausgangs-Pin anschließt. Der Arduino setzt einen Pin einfach auf HIGH oder LOW, wodurch sich ein Licht, ein elektrischer Motor oder dein Automotor steuern lässt.

Im nächsten Abschnitt werden wir versuchen, die Grundlagen der Elektrizität auf eine Art und Weise zu erläutern, die zwar einen Ingenieur sicherlich langweilen würde, aber dafür auch einen Einsteiger in die Arduino-Programmierung nicht sofort abschreckt.

Was ist Elektrizität?

Wenn du zu Hause schon einmal Klempnerarbeiten durchgeführt hast, wirst du in puncto Elektronik keine Verständnisschwierigkeiten haben. Der beste Weg, zu vermitteln, wie Elektrizität und elektrische Schaltungen funktionieren, ist die "Wasser-Analogie". Nehmen wir ein einfaches Gerät wie den batteriebetriebenen, tragbaren Ventilator, der in Abbildung 4-4 gezeigt wird.

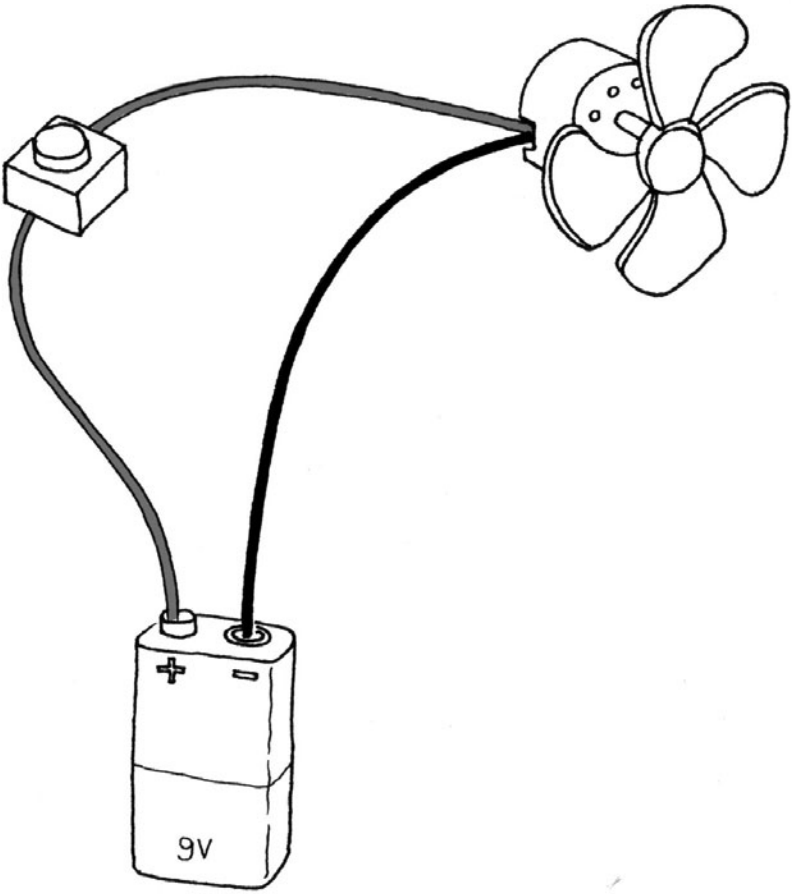


Abbildung 4-4: *Ein portabler Ventilator*

Wenn du den Ventilator auseinanderbaust, wirst du 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 du den Schalter betätigst und den Motor einschaltest, beginnt er, sich zu drehen, und sorgt so für die Luftzirkulation, die dir dann die gewünschte Abkühlung bringt.

Wie funktioniert das? Stell dir einfach vor, die Batterie sei zugleich ein Wasserreservoir und eine Pumpe, der Schalter ein Ventil und der Motor eines von diesen Wasserrädern, die du sicher schon bei Windmühlen gesehen hast. Wenn du das Ventil öffnest, fließt das Wasser von der Pumpe zum Wasserrad 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 entgegensetzt, abhängt).

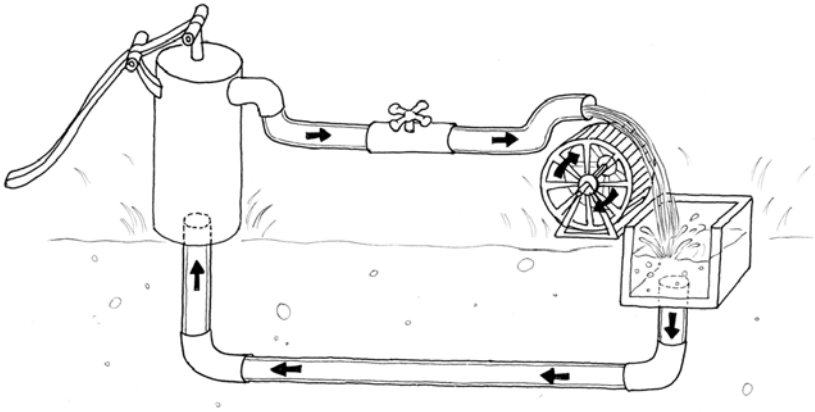


Abbildung 4-5: *Ein Hydrauliksystem*

Du wirst schnell bemerken, dass es 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 Wasserdruck nicht ausreicht. Wenn dieser Punkt erreicht wurde, muss die Pumpleistung erhöht werden. Diese Möglichkeit der Beschleunigung des Wasserrades funktioniert 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 die gesamte zugeführte Energie in Bewegung umgewandelt wird, sondern ein Teil der Energie verlorengeht. Sie zeigt sich dann im Allgemeinen 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 fließen). Elektrizität funktioniert ein wenig wie Wasser. Du hast eine Art Pumpe (jede Art von Energiequelle, z.B. eine Batterie oder eine Steckdose in der Wand), die elektrische Ladungen (die du dir am besten als kleine elektrische "Tropfen" vorstellst) durch Leitungen drückt, die in Form von Drähten realisiert sind. Diese elektrischen Tropfen werden von einigen Geräten verwendet, um Wärme zu produzieren (Großmutters Heizdecke), Licht zu erzeugen (deine Nachttischlampe), Sound herzustellen (deine Stereoanlage), Bewegung anzustoßen (unser Ventilator) und für viele weitere Dinge.

Wenn du auf einer Batterie die Angabe 9V liest, stell dir 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 Beispiels 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 du bestimmt schon erraten hast, 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 du dir wirklich merken musst. 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. Schließe eine 9V-Batterie an einen einfachen Schaltkreis an. Wenn du nun den Strom misst, wirst du feststellen: Je mehr Widerstände du einbaust, desto geringer wird der Strom, der hindurchfließt. Wenn wir noch einmal auf das Beispiel mit dem Wasserdurchfluss in den Leitungen zurückkommen, heißt das Folgendes: Wenn ich hier ein Ventil einbaue (das sich mit einem variablen Widerstand in der Elektrizität vergleichen lässt) und dieses Ventil immer weiter schließe, erhöhe ich den Widerstand in Bezug auf den Wasserdurchfluss, und es fließt immer weniger Wasser durch die Leitungen. Ohm hat dieses Gesetz in folgenden Formeln zusammengefasst:

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

Bei diesem Gesetz ist ein intuitives Verständnis wichtig, daher bevorzuge ich die letzte Version (`I = V/R`), da Strom etwas ist, das als Ergebnis entsteht, wenn du eine bestimmte Spannung (der Druck) zu einem bestimmten Kreislauf (der Widerstand) hinzufügst. Die Spannung existiert, unabhängig davon, ob sie tatsächlich genutzt wird oder nicht, und der Widerstand existiert, egal, ob Elektrizität zum Einsatz kommt oder nicht. Strom entsteht aber erst dann, wenn diese beiden Größen zusammenkommen.

Steuerung einer LED mit einem Drucktaster

Eine LED zum Blinken zu bringen, war recht einfach, aber ich glaube nicht, dass du glücklich wirst, wenn deine Nachttischlampe ständig blinkt, während du versuchst, ein Buch zu lesen. Daher musst du sie irgendwie steuern können. Im vorherigen Beispiel war die LED ein Aktor, der von Arduino gesteuert wurde. Was fehlt, ist ein Sensor.

Für unser Beispiel verwenden wir die einfachste verfügbare Ausführung eines Sensors: einen Schalter in Form eines Drucktasters.

Wenn du einen Drucktaster in seine Einzelteile zerlegen würdest, wäre dir 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 du den Taster aber drückst, stellst du eine Verbindung her.

Alle Schalter bestehen grundlegend aus folgenden Elementen: zwei (oder mehr) Metallteile, die miteinander in Berührung gebracht werden können, damit Elektrizität von einem zum anderen Metallteil fließen kann, oder die getrennt werden können, um den Stromfluss zu unterbinden.

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

Bau' die Schaltung, die in Abbildung 4-6 dargestellt ist. Hierzu benötigst du einige Bauteile (sie werden auch bei anderen Projekten nützlich sein):

- Eine lötfreie Steckplatine
- Einen Satz vorgefertigter Steckbrücken
- Einen 10K-Ohm-Widerstand – 10er-Pack)
- Einen Drucktastschalter, 10er-Pack)

Alternativ zum Kauf vorgefertigter Steckbrücken kannst du auch Massivdraht vom Typ 22 AWG, der auf kleinen Spulen aufgewickelt ist, verwenden und ihn mit Drahtschneider und Abisolierzange abisolieren.

GND auf dem Arduino-Board steht für *Masse* (engl. ground). Der Begriff ist historisch, aber in unserem Fall bezeichnet er einfach den Minuspol beim Strom. Wir tendieren dazu, die Begriffe GND und Masse synonym zu verwenden. Du kannst dir die Masse als die Leitung vorstellen, die sich in der Wasseranalogie in Abbildung 4-5 quasi underground befindet.

Bei den meisten Schaltkreisen wird GND oder Masse sehr häufig genutzt. Daher besitzt dein Arduino-Board drei Pins mit der Bezeichnung GND. Sie sind alle miteinander verbunden, daher macht es keinen Unterschied, welchen du benutzt.

Der Pin mit der Bezeichnung *5V* ist der positive Pol beim Strom und immer 5 Volt höher als die Masse.

In Beispiel 4-2 wird der Code gezeigt, mit dem wir die LED mittels unseres Drucktastschalters steuern werden.

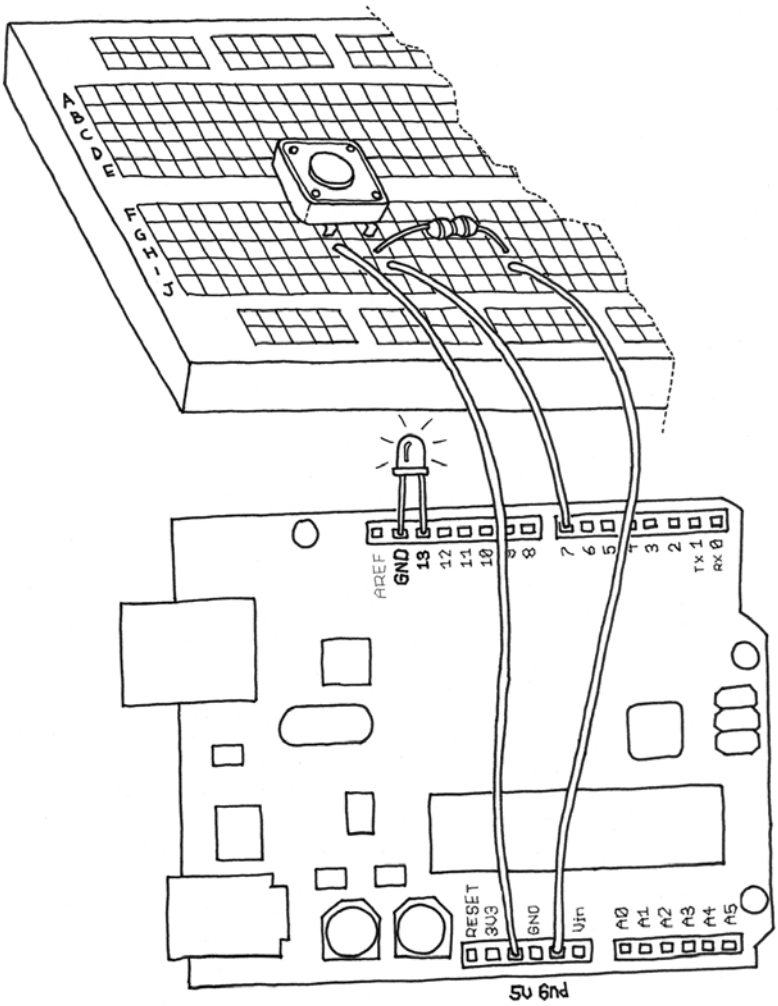


Abbildung 4-6: *Anschließen eines Drucktasters*

Beispiel 4-2: *LED mit Drucktaster anschalten*

```
// 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);
  }
}
```

Wähle im Arduino File→New aus (falls du einen anderen Sketch geöffnet hast, schließe ihn). Wenn du vom Arduino aufgefordert wirst, einen Namen für den neuen Sketch-Ordner anzugeben, gib PushButtonControl ein. Gib dann den Code für Beispiel 4-2 ein oder lade ihn von der Katalogseite zu diesem Buch (*http://bit.ly/start_arduino_3e*) herunter und kopiere ihn in die Arduino-IDE. Ist alles korrekt abgelaufen, wird die LED leuchten, wenn du den Taster drückst.

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 (denk immer daran, dass der 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 Code-Block ausgeführt, anderenfalls der Code-Block nach `else`.

Beachte hier, dass sich das Symbol == von dem Symbol = stark unterscheidet. Das erste Symbol wird verwendet, wenn zwei Dinge miteinander verglichen werden. Es wird dann entsprechend das Ergebnis TRUE oder FALSE zurückgeliefert; mit dem zweiten Symbol wird einer Variablen oder Konstanten ein Wert zugewiesen. Achte 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. Wir wissen, wovon wir sprechen, denn nach jahrelanger Programmiererfahrung unterläuft uns dieser Fehler immer noch.

Es ist wichtig zu wissen, dass der Schalter nicht direkt mit der LED verbunden ist. Dein Arduino-Sketch überprüft den Schalter und fällt dann die

Entscheidung, ob die LED ein- oder ausgeschaltet wird. Die Verbindung zwischen dem Schalter und der LED erfolgt wirklich in deinem Sketch.

Es ist natürlich sehr unpraktisch, mit dem Finger die ganze Zeit den Taster gedrückt halten zu müssen, wenn du Licht benötigst. Auch wenn man bedenkt, wie viel Energie verschwendet wird, wenn du dich von der Lampe fortbewegst und sie nicht nutzt, sie aber eingeschaltet lässt, wollen wir dennoch herausfinden, wie wir bewirken können, dass der Taster im aktivierten Modus fixiert wird.

Ein Schaltkreis – 1.000 Verhaltensweisen

Der große Vorteil von digitaler, programmierbarer Elektronik gegenüber klassischer Elektronik wird hier offensichtlich: Ich werde dir nun zeigen, auf welche Weise viele verschiedene Verhaltensweisen unter Verwendung desselben Schaltkreises wie im vorherigen Abschnitt implementiert werden können, indem einfach 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 "Gedächtnis" implementieren, und zwar in Form eines Software-Mechanismus, 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. Du kannst sie dir als Post-it vorstellen, das du verwendest, um dich an etwas zu erinnern, z.B. eine Telefonnummer: Du schreibst beispielsweise "Luisa 02 555 1212" darauf und klebst ihn an deinen Computerbildschirm oder an den Kühlschrank. In der Arduino-Sprache ist das ähnlich einfach: Du entscheidest einfach, welcher Datentyp gespeichert werden soll (z.B. eine Zahl oder ein Text) und vergibst einen Namen. Du kannst 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, so dass du später im Programm

```
val =112;
```

schreiben kannst.

Hierdurch wird deiner Variablen ein neuer Wert, nämlich 112, zugewiesen.

Hast du bemerkt, dass beim Arduino jede Anweisung mit einem Semikolon endet? Auf diese Weise wird dem Compiler (dem Teil des Arduino, der deinen Sketch in ein vom Mikrocontroller ausführbares Programm umwandelt) angezeigt, dass eine Anweisung beendet ist und eine neue beginnt. Wenn du ein Semikolon an einer Stelle vergisst, an der es erforderlich ist, wird dein Sketch für den Compiler keinen Sinn ergeben.

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

Beispiel 4-3: *LED mit Drucktaster anschalten und angeschaltet lassen*

```
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);
  }
}
```

Teste nun diesen Code. Du wirst sehen, dass er funktioniert ... irgendwie. Das Licht wechselt allerdings so schnell, dass du es gar nicht zuverlässig mit einem Tastendruck ein- oder ausschalten kannst.

Schauen wir uns einmal die interessanten Zeilen des Codes an: `state` ist eine Variable, 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 ihn von 0 in 1 oder umgekehrt. Da `state` immer nur 1 oder 0 sein kann, verwende ich hier einen kleinen Trick. Er beinhaltet einen kleinen mathematischen Ausdruck, der auf der Idee basiert, dass 1-0 = 1 ist, und 1-1 = 0:

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

Die Zeile macht 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 kannst du sehen, dass wir `state` verwenden, um zu ermitteln, ob die LED ein- oder ausgeschaltet sein muss. Wie bereits erwähnt, führt das zu eher ungenauen Ergebnissen.

Dies liegt an der Art und Weise, wie der Taster ausgelesen wird. Der Arduino ist wirklich schnell. Die eigenen internen Anweisungen werden mit einer Rate von 16 Millionen pro Millisekunde ausgeführt – das sind einige Millionen Code-Zeilen pro Sekunde. Während du also mit deinem Finger den Taster drückst, liest der 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 eine falsch laufende Uhr zweimal am Tag die Zeit korrekt wiedergibt, kann auch das Programm gelegentlich ein korrektes Verhalten aufweisen, sehr oft aber wird es falsch sein.

Wie kannst du dieses Problem beheben? Nun, du musst den exakten Zeitpunkt ermitteln, an dem der Taster gedrückt wird – das ist dann der einzige Moment, in dem `state` geändert werden muss. Dazu möchten wir den Wert von `val` speichern, bevor wir einen neuen Wert auslesen. Dadurch wird es

möglich, die aktuelle Position des Tasters mit der vorherigen zu vergleichen und `state` nur dann zu ändern, wenn der Taster von LOW zu HIGH wechselt.

Beispiel 4-4 enthält den entsprechenden Code:

Beispiel 4-4: *Neue und verbesserte Formel für den Tastendruck!*

```
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);
 }
}
```

Probier' das Programm aus, du hast es fast geschafft!

Möglicherweise hast du bemerkt, dass dieser Ansatz nicht ganz perfekt ist, was an einem anderen Problem bei mechanischen Schaltern liegt.

Bei Drucktastern handelt es sich um zwei Metallteilchen, die durch einer Feder auseinandergehalten werden und die miteinander in Berührung kommen, wenn du den Schalter drückst. Das klingt möglicherweise so, als wäre der Schalter vollständig eingeschaltet, wenn du den Taster drückst. Was aber tatsächlich geschieht, ist, dass die Metallteilchen voneinander abprallen wie ein Ball, der auf dem Boden hüpft.

Dieses Abprallen erfolgt zwar nur über eine kurze Distanz und für den Bruchteil einer Sekunde, dennoch verursacht es beim Schalter einen mehrmali-

gen Wechsel zwischen AN und AUS, bis dieses Prellen endet und der Arduino schnell genug ist, dies zu erfassen.

Wenn der Taster prellt, erhält Arduino eine Reihe rasch aufeinanderfolgender Ein- und Aussignale. Es wurden viele Möglichkeiten zum Entprellen entwickelt, aber in diesem einfachen Code-Abschnitt reicht es völlig aus, eine Verzögerung von 10 bis 50 Millisekunden einzubauen. Mit anderen Worten: Du wartest einfach ein wenig, bis das Prellen endet.

Beispiel 4-5 enthält den finalen Code:

Beispiel 4-5: *Eine weitere neue und verbesserte Formel für Tastendrücke – mit einfachem Entprellen!*

```
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);
 }
}
```

Tami (Masaaki) Takamiya hat ein wenig Extra-Code eingefügt, mit dem du möglicherweise ein besseres Ergebnis beim Entprellen erzielst:

```
    if ((val == LOW) && (old_val == HIGH)) {
        delay(10);
    }
```

5/Erweiterter Input und Output

In Kapitel 4, *Die wirklich ersten Schritte mit Arduino*, auf Seite 23 haben wir die grundlegendsten Operationen kennengelernt, die wir mit dem Arduino durchführen können: Steuern des digitalen Outputs und Auslesen des digitalen Inputs. Wenn es sich beim Arduino um eine menschliche Sprache handeln würde, wären dies zwei Buchstaben ihres Alphabets. Angesichts der Tatsache, dass dieses Alphabet aus nur 5 Buchstaben besteht, kannst du schon abschätzen, wie wenig Arbeit uns noch bevorsteht, bevor wir Arduino-Poesie schreiben können.

Der Einsatz anderer Ein/Aus-Sensoren

Nachdem du nun erfahren hast, wie der Drucktasterschalter verwendet wird, solltest du wissen, dass es viele andere ziemlich einfache Sensoren gibt, die nach demselben Prinzip funktionieren:

Kippschalter

Bei dem Drucktaster, den du bisher benutzt hast, handelt es sich um einen Tastertyp, der als *nicht einrastender Schalter* bezeichnet wird, weil er, sobald du ihn loslässt, wieder in die Ausgangsposition zurückspringt. Ein bekanntes Beispiel für einen nicht einrastenden Schalter ist eine Türklingel.

Im Gegensatz dazu bleibt ein *Kippschalter* in der Position, in die du ihn gebracht hast. Ein bekanntes Beispiel für einen Kippschalter ist ein Lichtschalter.

Trotz dieser technisch korrekten Definitionen werden wir in diesem Buch die gebräuchlichen Namen für diese Schalter verwenden: mit Drucktaster meinen wir einen nicht einrastender Schalter, wohingegen ein Schalter einen Kippschalter bezeichnet.

Vielleicht denkst du bei dem Begriff Schalter nicht unbedingt an einen Sensor, aber es handelt sich tatsächlich um einen: Ein Drucktaster (nicht einrastender Schalter) erfasst, dass du ihn drückst, und ein (Kipp-)Schalter erfasst und speichert den letzten Zustand, in den du ihn gebracht hast.

Thermostat

Hierbei handelt es sich um einen Schalter, der seinen Zustand ändert, wenn die Temperatur einen festgelegten Wert erreicht.

Magnetische Schalter (auch als Reed-Schalter bekannt)

Sie verfügen über zwei Kontakte, die in Berührung kommen, wenn sie sich in der Nähe eines Magnets befinden. Diese Schalter kommen oft 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 zu erfassen, wenn eine Personen (oder eine schwergewichtigen Katze) darauf tritt.

Neigungsschalter

Hierbei handelt es sich um einen einfachen, aber cleveren Sensor, der aus zwei (oder mehr) Kontakten und einem kleinen Metallball besteht (oder einem Quecksilbertropfen, aber ich empfehle, solche Schalter nicht zu verwenden). In Abbildung 5-1 ist ein typisches Modell abgebildet.

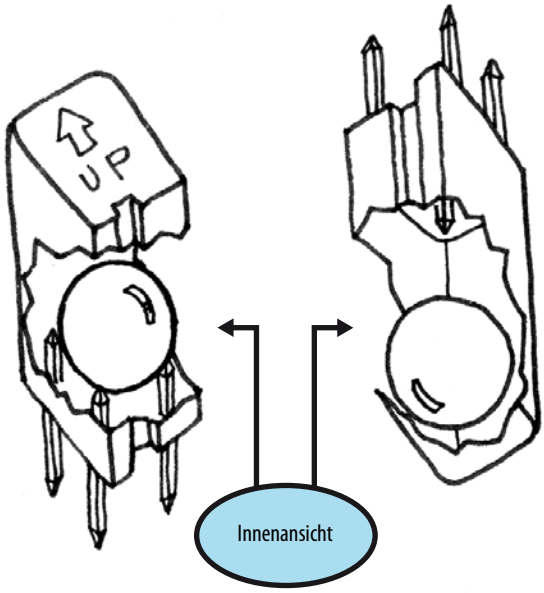


Abbildung 5-1: *Das Innere eines Neigungssensors*

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 du den Sensor kippst, 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.

Ein weiterer nützlicher Sensor, der häufig bei Alarmanlagen eingesetzt wird, ist der passive Infrarot-Sensor oder PIR-Sensor (siehe Abbildung 5-2). Dieses Bauteil ändert seinen Zustand, wenn eine Person sich in der näheren Umgebung bewegt. Diese Sensoren sind oft so konstruiert, dass sie nur Menschen und keine Tiere entdecken, damit der Alarm nicht von Tieren ausgelöst wird.

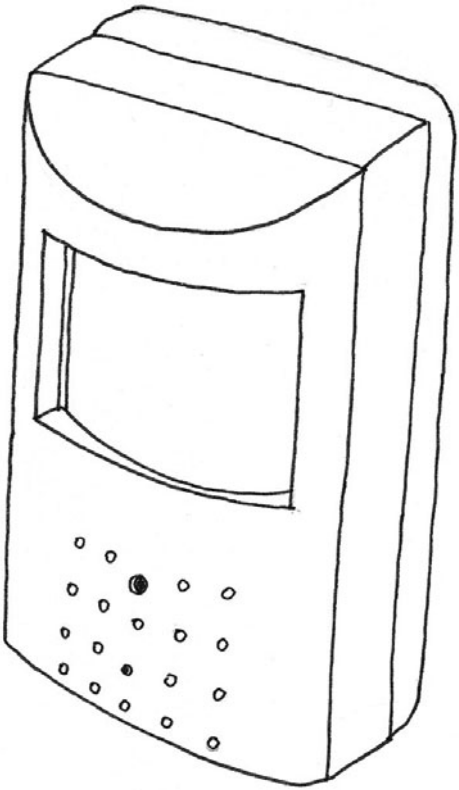


Abbildung 5-2: *Ein typischer PIR-Sensor*

Das Erfassen von Menschen ist tatsächlich ziemlich kompliziert, und der PIR-Sensor ist im Inneren recht komplex. Glücklicherweise befassen wir uns nicht wirklich mit diesem Inneren. Das einzige, was wir wissen müssen, ist, dass mittels eines digitalen Signals erkannt wird, ob eine Person anwesend ist oder nicht. Daher handelt es sich beim PIR-Sensor um einen digitalen Sensor.

Selbstgebaute Schalter (DIY-Schalter)

Du kannst mit einem Metallball, einigen Nägeln und ein wenig Draht, der um die Nägel gewickelt wird, deinen eigenen Neigungsschalter bauen. Wenn der Ball auf eine Seite rollt, und bei zwei der Nägel liegenbleibt, wird er die entsprechenden Drähte verbinden.

Einen nicht einrastender Schalter kannst du mit einer Wäscheklammer herstellen, indem du um jeden Schenkel einen Draht wickelst. Wenn du die Schenkel drückst, ist der Schalter geschlossen.

Alternativ kannst du den Draht auch um die Klemmbacken der Wäscheklammer wickeln und ein Stück Pappe dazwischen klemmen, damit die Drähte nicht miteinander in Berührung kommen. Binde an der Pappe ein Stück Schnur an. Befestige das andere Ende der Schnur an einer Tür. Wenn die Tür geöffnet wird, wird das Stück Pappe herausgezogen, die Drähte berühren sich, und der Schalter wird geschlossen.

Weil es sich bei all diesen Sensoren um digitale Sensoren handelt, kann jeder von ihnen anstelle des Drucktasters, der in Kapitel 4, *Die wirklich ersten Schritte mit Arduino*, auf Seite 23 zum Einsatz kam, verwendet werden, ohne dass irgendwelche Änderungen an deinem Sketch vorgenommen werden müssen.

Wenn du beispielsweise den Schaltkreis und den Sketch aus Abschnitt *Steuerung einer LED mit einem Drucktaster*, auf Seite 38 verwendest, den Drucktaster aber durch einen PIR-Sensor ersetzt, kannst du erreichen, dass deine Lampe auf die Anwesenheit von Personen reagiert. Du kannst auch einen Neigungsschalter verwenden und so eine Lampe konstruieren, die sich ausschaltet, wenn sie auf eine Seite geneigt wird.

Steuerung von Licht mittels PWM

Mit den bisher erworbenen Kenntnissen kannst du eine interaktive Lampe bauen. Bisher ist das Ergebnis allerdings ein wenig langweilig, weil die Lampe nur ein- oder ausgeschaltet ist. Eine schicke Lampe muss aber auch dimmbar sein. Hierzu können wir uns eines Phänomens bedienen, das viele Dinge wie Fernsehen oder Kino erst ermöglicht. Dieses Phänomen wird als *Trägheit des Auges* oder *POV* (Persistance of Vision) bezeichnet.

Nach dem ersten Beispiel in Kapitel 4, *Die wirklich ersten Schritte mit Arduino*, auf Seite 23 wurde schon angedeutet, dass von LEDs abgegebenes Licht gegenüber ihrer normalen Helligkeit gedimmt erscheint, wenn du die Angaben für die Verzögerungen in der Funktion `delay` so änderst, dass du kein Blinken der LED mehr wahrnehmen kannst. Wenn du nun ein wenig herumexperimentierst, wirst du feststellen, dass eine LED heller scheint, wenn die On-Phase länger dauert, und abgedimmt wird, wenn die OFF-Phase länger ist. Diese Technik heißt *Pulsweitenmodulation* (PWM), weil du die Helligkeit der LEDs änderst, indem du die Pulsweite änderst (modulierst). In Abbildung 5-3 ist dargestellt, wie es funktioniert. Die Methode basiert darauf, dass deine Augen Einzelbilder nicht unterscheiden können, wenn sie schnell wechseln.

Diese Technik funktioniert auch bei anderen Bauteilen, nicht nur bei LEDs. Die Schnelligkeit eines Motors lässt sich auf dieselbe Weise ändern. In diesem Fall sind es nicht unsere Augen, die dies ermöglichen, sondern der Motor selbst, weil ein Motor nicht augenblicklich starten oder anhalten kann. Es bedarf einer kleinen Zeitspanne, um zu beschleunigen oder zu drosseln. Wenn wir die Ausgabe (mittels `digitalWrite()`) schneller ändern, als der Motor reagieren kann, ist das Resultat eine mittlere Geschwindigkeit, abhängig davon, wie lang die ON-Phasen und wie lang die OFF-Phasen sind.

Auch wenn dieser Trick sehr nützlich ist, wirst du beim Herumexperimentieren bemerken, dass das Steuern von Helligkeit bei einer LED durch Einfügen von Verzögerungen im Code recht unpraktisch sein kann. Dies gilt besonders dann, wenn du einen Sensor auslesen, Daten an den seriellen Anschluss schicken oder du etwas anderes tun möchtest, dass die Helligkeit der LED ändert, weil für jede Zeile Code, die hinzugefügt wird, Zeit für ihre Ausführung erforderlich wird, wodurch sich die Zeitspanne ändert, in der eine LED ein- oder ausgeschaltet ist.

Glücklicherweise verfügt der Prozessor, der bei Arduino verwendet wird, über eine spezielle Hardware, die recht effizient LEDs blinken lassen kann, während dein Sketch etwas anderes tut. Beim Uno ist diese Hardware über die Pins 3, 5, 6, 9, 10 sowie 11 und beim Leonardo über die Pins 3, 5, 6, 9, 10, 11 sowie 13 implementiert. Diese Hardware wird über die Anweisung `analogWrite()` gesteuert.

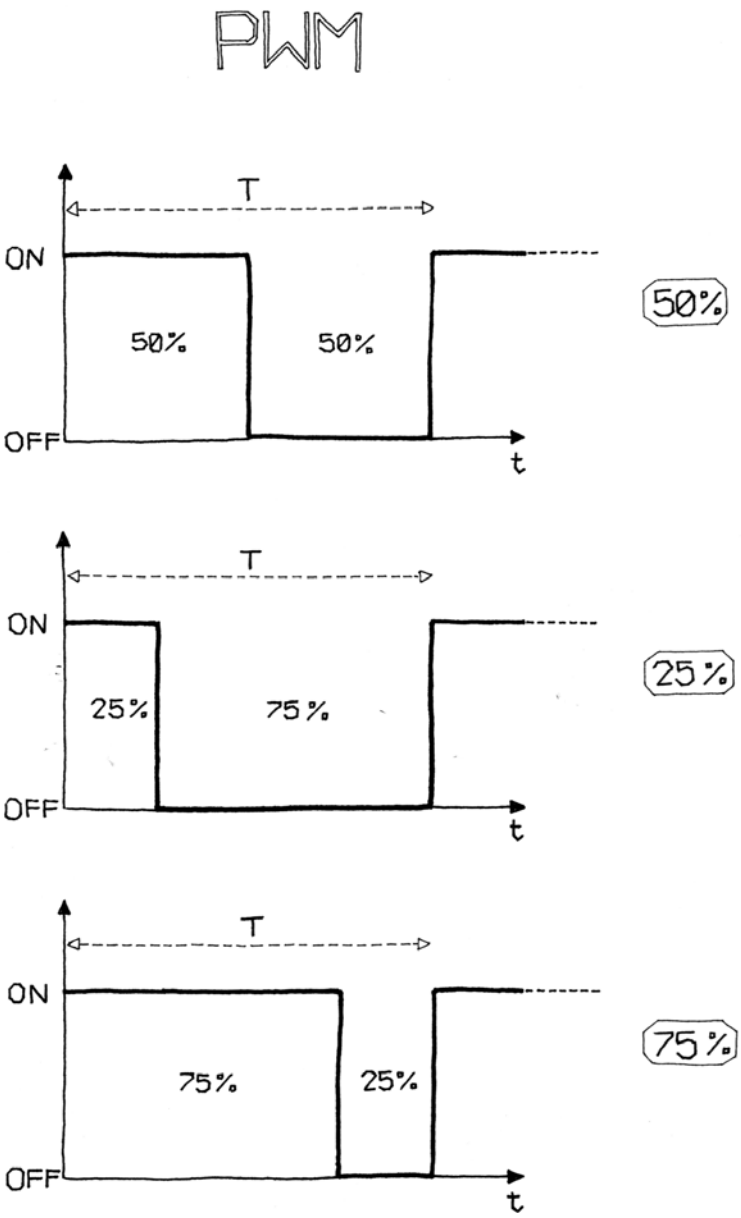


Abbildung 5-3: *PMW in Aktion*

Wenn du zum Beispiel `analogWrite(9,128)` schreibst, wird die Helligkeit der LED, die an Pin 9 angeschlossen ist, auf 50 % gedimmt. Wenn du hingegen `analogWrite(9,255)` schreibst, wird die Helligkeit der LED auf volle Helligkeit gesetzt. Die Anweisung `analogWrite()` kann Zahlen von 0 bis 255 übernehmen, wobei 255 volle Helligkeit bedeutet und bei 0 die LED ausgeschaltet ist.

Eine feine Sache, dass dir mehrere PWM-Pins zur Verfügung stehen. Wenn du z.B. rote, grüne und blaue LEDs kaufst, kannst du ihre Lichter mischen und Licht in beliebiger Farbe erzeugen.

Das wollen wir nun einmal ausprobieren. Bau den Schaltkreis, der in Abbildung 5-4 abgebildet ist. Du benötigst eine LED einer beliebigen Farbe, z.B. in Grün (*http://arduino.cc/GreenLED*) und einige 220-Ohm-Widerstände, die praktisch überall erhältlich sind.

Beachte, dass die LEDs gepolt sind, d. h., es spielt eine Rolle, in welcher Richtung der Strom durch sie hindurchfließt. Bei dem langen Beinchen handelt es sich um die *Anode* oder den positiven Anschluss. Er sollte in unserem Fall nach rechts zeigen, so dass er mit Pin 9 auf dem Arduino verbunden werden kann. Das kurze Beinchen ist die *Kathode*, also der negative Anschluss, und sollte in unserem Fall nach links zeigen, damit sie an den Widerstand angeschlossen werden kann.

Außerdem weisen die meisten Widerstände auf der Kathodenseite eine abgeflachte Ecke auf, wie in der Abbildung zu sehen ist. Das kannst du dir leicht merken, indem du dir vorstellst, dass der flache Teil wie ein Minuszeichen aussieht.

Wie ich bereits in Abschnitt *Eine LED zum Blinken bringen*, auf Seite 24 erwähnt habe, solltest du immer einen Widerstand verwenden, um ein Durchbrennen der LED zu verhindern. Verwende einen 220-Ohm-Widerstand (Rot – Rot – Braun).

Erzeuge dann im Arduino einen neuen Sketch mit dem Code aus Beispiel 5-1. Du kannst den Code auch über den Link zu den Code-Beispielen auf der Katalogseite zu diesem Buch (*http://bit.ly/start_arduino_3e*) herunterladen.

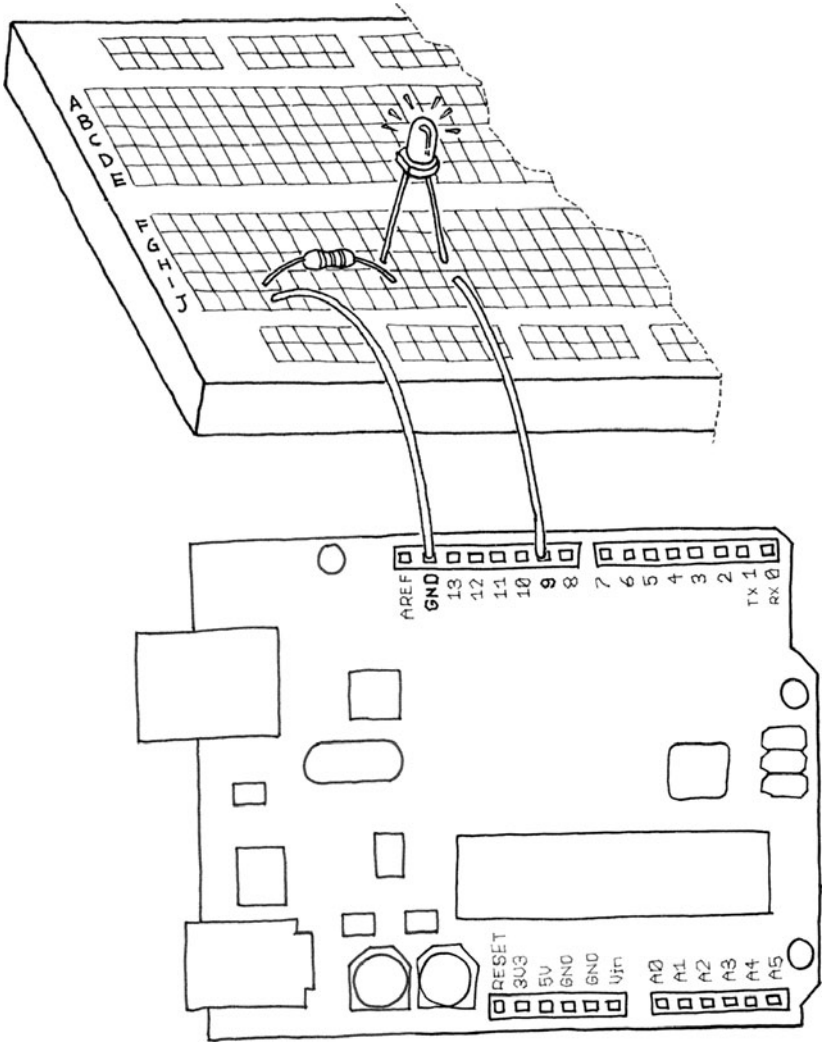


Abbildung 5-4: *LED am PWM-Pin*

Beispiel 5-1: *Ein- und Ausblenden einer LED*

```
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
  }

}
```

Lade den Sketch und die LED wird auf- und dann abgeblendet. Gratuliere! Du hast nun ein nettes Laptop-Feature nachgebildet.

Vielleicht ist es auch ein wenig Verschwendung, den Arduino für so eine simple Angelegenheit zu benutzen, aber du kannst anhand solcher Beispiele sehr viel lernen.

Wie du bereits erfahren hast, wird mittels `analogWrite()` die Helligkeit der LED geändert. Der andere wichtige Teil ist die `for`-Schleife: Sie wiederholt `analogWrite()` und `delay()` immer wieder, wobei jedes Mal ein anderer Wert für die Variable `i` zum Einsatz kommt, wie im Folgenden beschrieben wird.

Bei der ersten `for`-Schleife wird für die Variable `i` der Wert 0 benutzt. Dieser Wert wird dann bis auf 255 gesteigert, wodurch die LED mit voller Helligkeit leuchtet.

Bei der zweiten `for`-Schleife wird für die Variable `i` der Wert 255 benutzt und dann bis auf 0 verringert, wodurch die LED immer weiter gedimmt wird, bis sie schließlich völlig ausgeschaltet ist.

Nach der zweiten `for`-Schleife startet der Arduino unsere `loop()`-Funktion erneut.

Die Funktion `delay()` dient ausschließlich dazu, diesen Prozess ein wenig zu verlangsamen, damit du die Änderungen der Helligkeit auch sehen kannst. Anderenfalls verliefe der Vorgang zu schnell.

Wir wollen nun dieses Wissen einsetzen, um unsere Lampe zu verbessern.

Füge auf der Steckplatine die Schaltung hinzu, die wir verwendet haben, um den Drucktaster auszulesen (weiter vorne in Kapitel 4, *Die wirklich ersten Schritte mit Arduino*, auf Seite 23). Vermeide bitte, in diesem Buch zurückzublättern, weil ich möchte, das du damit beginnst, jeden hier gezeigten Basisschaltkreis als einen der Bausteine zu sehen, aus denen sich immer größere Projekte realisieren lassen. Wenn du doch nachschauen musst, macht das nichts; wichtig ist, dass du dir vorher ein wenig Gedanken darüber gemacht hast, wie das Ganze aussehen könnte.

Um den gewünschten Schaltkreis zu bauen, musst du die Schaltung, die du gerade gebaut hast (die aus Abbildung 5-4), mit der Drucktasterschaltung kombinieren, die in Abbildung 4-6 dargestellt ist. Wenn du möchtest, kannst du die beiden Schaltkreise auf verschiedene Bereiche der Platine bauen; es gibt hier viel Platz. Mehr über lötfreie Steckplatinen erfährst du in Anhang A.

Wenn du noch nicht so weit bist, dies auszuprobieren, ist das nicht weiter schlimm: Verdrahte einfach beide Schaltkreise mit dem Arduino-Board, wie es in den Abbildungen Abbildung 4-6 und Abbildung 5-4 dargestellt ist.

Kommen wir zu unserem Beispiel zurück. Wenn du nur einen Drucktaster hast – wie kannst du dann die Helligkeit der Lampe steuern? An dieser Stelle wirst du nun eine weitere Technik aus dem Interactive Design kennenlernen: das Ermitteln, wie lange ein Taster gedrückt wurde. Hierzu müssen wir ein Upgrade von Beispiel 4-5 aus dem Kapitel 4, *Die wirklich ersten Schritte mit Arduino*, auf Seite 23 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.

Schau dir den Sketch in Beispiel 5-2 an. Die LED wird eingeschaltet, wenn der Taster gedrückt wird, und sie bleibt ausgeschaltet, nachdem der Taster losgelassen wurde. Wenn der Knopf gedrückt bleibt, ändert sich die Helligkeit.

Beispiel 5-2: *Sketch zur Helligkeitsveränderung, wenn du den Knopf drückst*

```
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 500 ms.
       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 du siehst, nimmt unser Interaktionsmodell Formen an. Wenn du den Taster drückst und sofort wieder loslässt, schaltest du die Lampe ein- bzw. aus. Wenn du den Taster gedrückt hältst, ändert sich die Helligkeit. Nimm einfach den Finger vom Taster, wenn die gewünschte Helligkeit erreicht ist.

Wie wir bereits beim Schaltkreis angemerkt haben, solltest du ein wenig Zeit darauf verwenden, das Programm zu verstehen.

Die verwirrendste Zeile ist wahrscheinlich folgende:

```
if (state == 1 && (millis() - startTime) > 500) {
```

Hiermit wird überprüft, ob der Taster länger als 500 ms gedrückt wurde. Zu diesem Zweck wird eine integrierte Funktion mit dem Namen `millis()` verwendet, mit der einfach die Millisekunden gezählt werden, die seit dem Start des Sketches vergangen sind. Indem nachverfolgt wird, wann der Taster gedrückt wurde (in der Variablen `startTime`), können wir die aktuelle Zeit mit der Startzeit vergleichen, um zu sehen, wie viel Zeit vergangen ist.

Natürlich macht das nur Sinn, wenn der Taster aktuell gedrückt ist, weshalb wir am Beginn der Zeile überprüfen, ob `state` auf den Wert 1 gesetzt wurde.

Wie du sehen kannst, handelt es sich bei Schaltern wirklich um recht leistungsfähige Sensoren, auch wenn sie so simpel erscheinen. 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 benötigst du einen Lichtsensor, wie er in Abbildung 5-5 zu sehen ist. Solche Sensoren erhältst du bei jedem Elektronikhändler, bei deutschsprachigen Elektronikgeschäften wird dieser Sensor auch Fotowiderstand genannt .

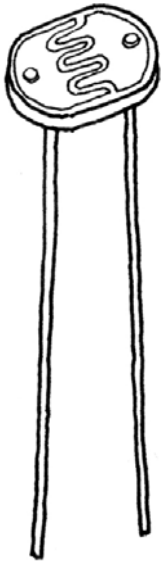


Abbildung 5-5: *Ein Lichtsensor*

Wie der Name schon sagt, handelt es sich bei dem *Lichtsensor* (*LDR, light-dependent resistor*) um einen Widerstand, der vom Licht abhängig ist. Bei Dunkelheit weist der lichtabhängige Widerstand 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 von durch Licht aktivierten Schalter.

Baue den Schaltkreis aus Beispiel 4-2 (siehe auch Abschnitt *Steuerung einer LED mit einem Drucktaster*, auf Seite 38 in Kapitel 4, *Die wirklich ersten Schritte mit Arduino*, auf Seite 23) auf und lade dann den entsprechenden Code aus Beispiel 4-2 auf das Arduino-Board. Drücke den Drucktaster, um sicherzustellen, dass alles funktioniert.

Entferne nun vorsichtig den Drucktaster und stecke den LDR exakt an der Stelle im Schaltkreis ein, an der sich der Drucktaster befand. Die LED sollte nun leuchten. Wenn du den LDR mit der Hand bedeckst, wird die LED ausgeschaltet.

Du hast gerade deine erste wirklich sensorgesteuerte LED konstruiert. Dies ist ein wichtiger Schritt, weil du zum ersten Mal in diesem Buch eine elektronische Komponente verwendest, bei der es sich nicht bloß um ein mechanisches Bauteil handelt: Es ist ein wirklich komplexer Sensor. Natürlich ist dies nur ein kleines Beispiel dafür, wozu ein LDR verwendet werden kann.

Analoger Eingang

Wie du aus dem vorherigen Abschnitt weißt, kann Arduino feststellen, ob eine Spannung an einem seiner Pins anliegt, und das Ergebnis mittels der `digitalRead()`-Funktion zurückmelden. Eine solche 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 oder nicht) und einem analogen Sensor, der uns mitteilen kann, wie viel von etwas vorhanden ist.

Um einen solchen Sensor auszulesen, benötigen wir einen speziellen Arduino-Pin.

Drehe deinen Arduino so, dass die Ansicht der in Abbildung 5-6 entspricht.

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 sie können auch mittels der Funktion `analogRead()`die Größe der Spannung messen. Die Funktion `analogRead()` liefert einen Wert zwischen 0 und 1023 zurück, der eine Spannung zwischen 0 und 5 Volt repräsentiert. Wenn beispielsweise eine Spannung von 2,5 Volt an Pin 0 anliegt, gibt `analogRead(0)` den Wert 512 zurück.

Wenn du nun den Schaltkreis baust, der in Abbildung 5-6 dargestellt ist, dabei einen 10K-Ohm-Widerstand verwendest und den Code ausführst, der in Beispiel 5-3 angeführt ist, wirst du sehen, dass die Board-eigene LED in einer Geschwindigkeit blinkt, die von der auf den Sensor treffenden Lichtmenge abhängig ist.

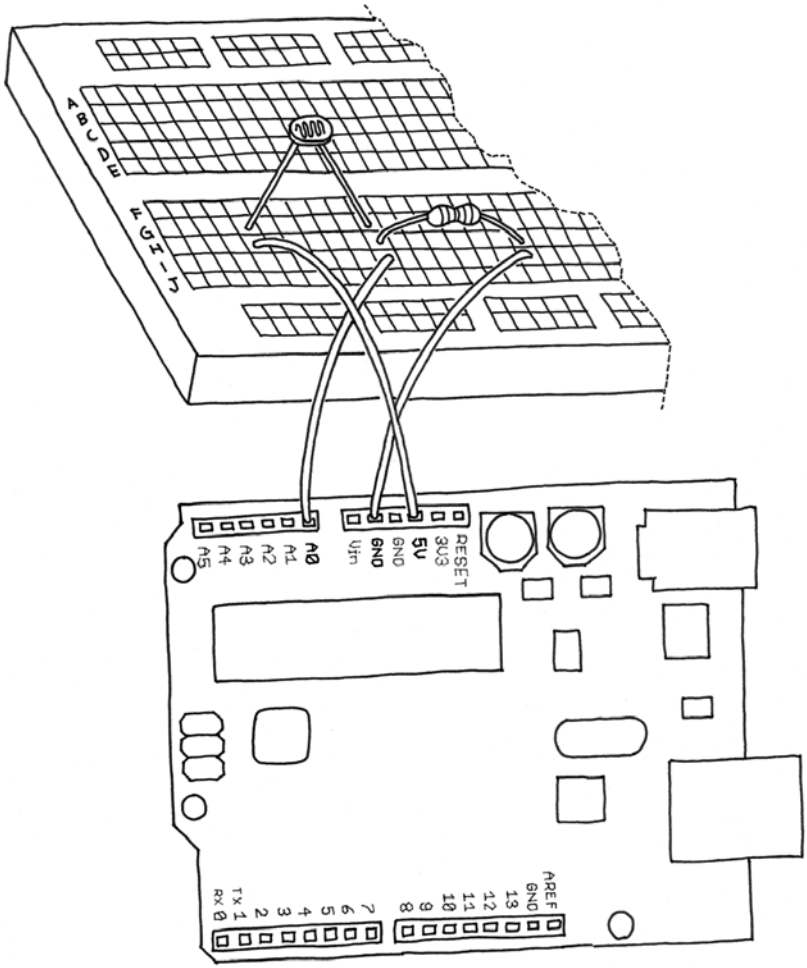


Abbildung 5-6: *Eine analoge Sensor-Schaltung*

Beispiel 5-3: *LED blinkt in Geschwindigkeit abhängig der Lichtmenge*

```
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
}
```

Schließe nun die LED an Pin 9 an, wie vorher bei dem Schaltkreis, der in Abbildung 5-4 abgebildet ist. Weil du bereits einige Komponenten auf deiner Steckplatine angebracht hast, musst du erst eine Stelle finden, an der die LED, die Drähte und der Widerstand sich nicht mit dem LDR-Schaltkreis überschneiden. Möglicherweise musst du einige Dinge verschieben, aber mach dir keine Sorgen. Das ist eine gute Übung, weil es deinem Verständnis von Schaltkreisen und Steckplatinen dient.

Wenn du die LED zu deinem LDR-Schaltkreis hinzugefügt hast, gib den Code aus Beispiel 5-4 ein und lade den Sketch auf deinen Arduino.

Beispiel 5-4: *LED-Helligkeit abhängig vom Wert des analogen Inputs*

```
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
}
```

Wenn das Programm ausgeführt wird, decke den LDR ab, hebe die Abdeckung wieder auf und beobachte, was mit der LED geschieht.

Wie zuvor solltest du versuchen zu verstehen, was geschieht. Das Programm ist wirklich recht einfach, eigentlich sogar sehr viel einfacher als die beiden vorherigen.

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

Der lichtabhängige Widerstand ist ein sehr nützlicher Sensor, aber Arduino kann einen Widerstand nicht direkt lesen. Der Schaltkreis in Abbildung 5-6 wandelt den Widerstand des LDR in eine Voltzahl um, die der Arduino lesen kann.

Dieselbe Schaltung funktioniert bei jedem so genannten resistiven Sensor, und es gibt viele Arten resistiver Sensoren, z.B. Sensoren, die die Kraft messen, die Ausdehnung, die Krümmung oder die Hitze. Du könntest beispielsweise einen *Thermistor* (wärmeabhängigen Widerstand) anstelle des LDR anschließen und eine LED nutzen, die ihre Helligkeit abhängig von der Temperatur ändert.

Wenn du mit einem Thermistor arbeitest, musst du dir darüber bewusst sein, dass es keine direkte Verbindung zwischen dem Wert, den du liest, und der tatsächlich gemessenen Temperatur gibt. Wenn du exakte Daten benötigst, solltest du die Werte vom analogen Ausgangs-Pin lesen und dabei gleichzeitig Messungen mit einem Thermometer vornehmen. Du kannst diese Werte in einer Tabelle einander gegenüberstellen und eine Möglichkeit suchen, die analogen Ergebnisse so abzustimmen, dass sie den Temperaturen in der realen Welt entsprechen. Alternativ kannst du einen digitalen Temperatursensor wie den TMP36 erwenden.

Bisher haben wir eine LED als Bauteil für die Ausgabe genutzt. Es wäre aber schwierig, wenn du die Temperatur anhand der Einschätzung, wie hell eine LED leuchtet, messen wolltest. Wäre es nicht nett, wenn du tatsächlich die Werte erhalten könntest, die der Arduino vom Sensor direkt ausliest? Wir könnten uns die Werte durch eine LED im Morsealphabet blinken lassen, aber beim Arduino gibt es einen sehr viel einfacheren Weg, Informationen an uns Menschen zu senden, indem nämlich das USB-Kabel benutzt wird, das du verwendest, um Sketche auf den Arduino zu laden.

Serielle Kommunikation

Zu Beginn des Buches hast du erfahren, dass der Arduino über einen USB-Anschluss verfügt, über den die IDE Code in den Mikrocontroller lädt. Die gute Nachricht: Nachdem der Sketch geladen ist und ausgeführt wird, kann er dieselbe Verbindung auch benutzen, um Nachrichten an den Computer zu senden oder von ihm zu empfangen. Um Nachrichten von einem Sketch zu senden, verwenden wir *das serielle Objekt*.

Bei einem Objekt handelt es sich um eine Sammlung zusammenhängender Fähigkeiten, die aus Gründen der Bequemlichkeit gebündelt wurden, und das serielle Objekt ermöglicht uns die Kommunikation über die USB-Verbindung . Du kannst dir das serielle Objekt als Mobiltelefon vorstellen, über das du Zugriff auf das Mobilfunknetz hast. Wie ein Mobiltelefon enthält das serielle Objekt sehr viel Komplexität, um die wir uns nicht weiter zu kümmern brauchen. Wir müssen nur wissen, wie das serielle Objekt verwendet wird.

In diesem Beispiel wirst du den letzten Schaltkreis verwenden, den wir mit dem Fotowiderstand gebaut haben, doch anstelle die Helligkeit einer LED zu steuern, sendest du die Werte, die von `analogRead()` gelesen werden, zurück an den Computer. Gib den Code aus Beispiel 5-5 in einen neuen Sketch ein. Du kannst ihn auch über den Link zu den Code-Beispielen auf der englischsprachigen Katalogseite zu diesem Buch (*http://bit.ly/start_arduino_3e*) herunterladen.

Beispiel 5-5: *Senden der vom analogen Eingang 0 gelesenen Daten an den Computer*

```
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 du den Code auf das Arduino-Board übertragen hast, könntest du denken, dass nichts Interessantes geschieht. Tatsächlich arbeitet dein Arduino einwandfrei: Er ist damit beschäftigt, den Lichtsensor auszulesen und die Informationen an deinen Computer zu senden. Das Problem liegt darin, dass auf deinem Computer keinerlei Informationen angezeigt werden, die vom Arduino stammen.

Du benötigst den Serial Monitor, und der ist in die Arduino-IDE integriert.

Die Schaltfläche Serial Monitor befindet sich in der Nähe der oberen rechten Ecke der Arduino-IDE. Sie sieht ein wenig wie ein Vergrößerungsglas aus, als ob du die Kommunikation zwischen dem Arduino und deinem Computer ausspionieren würdest.

Klicke auf die Schaltfläche Serial Monitor, um den Monitor zu öffnen, und du wirst die Zahlen am unteren Rand des Bildschirms vorbeiwandern sehen. Decke den Fotowiderstand ab, damit es dunkler ist, und beobachte, wie die Zahlen sich ändern. Beachte, dass die Zahlen niemals kleiner als Null sind und nie größer als 1023, da dies der Zahlenbereich ist, den `analogRead()` produzieren kann.

Dieser Channel für die Kommunikation zwischen Arduino und deinem Computer eröffnet eine ganze Welt neuer Möglichkeiten. Es gibt viele Programmiersprachen, die es dir ermöglichen, Programme für deinen Computer zu schreiben, die mit dem seriellen Port kommunizieren, und über den Serial Port können diese Programme mit deinem Arduino kommunizieren.

Eine besonders gute Ergänzung zum Arduino ist die Programmiersprache *Processing* (*http://www.processing.org*), weil die Sprachen und IDEs sehr ähnlich sind. Mehr zu Processing erfährst du in Kapitel 7, *Kommunikation mit der Cloud*, auf Seite 81 in Abschnitt *Der Code*, auf Seite 84, und deine Arduino-IDE enthält einige Beispiele, wie z.B. File → Examples → 04.Communication → Dimmer und File → Examples → 04.Communication → Graph. Auch im Internet finden sich viele Beispiele.

Wenn du dich mit der Programmiersprache Processing ausführlich beschäftigen möchtest, dann schau dir mal das deutschsprachige O'Reilly Buch *Processing* von Erik Bartmann (ISBN 978-3-89721-997-7, *http://www.oreilly.de/catalog/processingger/index.html*) an.

Der Umgang mit größeren Lasten

Mit den Pins auf dem Arduino-Boars können nur solche Bauteile mit Strom versorgt werden, die eine geringe Strommenge benötigen, z.B. eine LED. Wenn du versuchst, so etwas wie einen großen Motor oder eine Glühlampe zu betreiben, stellt der Pin möglicherweise seine Arbeit ein und könnte den Mikrocontroller, der das Herzstück deines Arduino bildet, irreparabel beschädigen.

Zur Sicherheit sollte der Strom, der durch den I/O-Pin eines Arduino fließt, auf 20 Milliampere begrenzt werden.

Mach dir deswegen aber keine Sorgen. Es gibt zahlreiche einfache Techniken, mit denen du Bauteile steuern kannst, die erheblich mehr Strom verbrauchen. Dabei gibt es einen Trick, der ähnlich wie ein Hebel mit einem entsprechenden Drehpunkt beim Heben sehr schwerer Lasten funktioniert. Wenn du das Ende eines Stocks unter einen großen Stein schiebst und einen Drehpunkt an der richtigen Stelle setzt, kannst du das lange Ende des Stocks herunterdrücken, und am kurzen Ende unter dem Stein wirkt eine größere Kraft ein. Du drückst zwar mit geringer Kraft, doch durch die Mechanik des Hebels wird eine größere Kraft auf den Stein ausgeübt.

In der Elektronik kann man diesen Effekt mit einem *MOSFET* erzielen. Bei einem MOSFET (Metall-Oxid-Halbleiter-Feldeffekttransistor; englisch: metal-oxide-semiconductor field-effect transistor) handelt es sich um einen elektronischen Schalter, der sich mit einer sehr geringen Strommenge steuern lässt, seinerseits aber eine weitaus größere Menge Strom steuern kann. Ein MOSFET besitzt drei Pins. Du kannst dir einen MOSFET als einen Schalter zwischen zwei Pins vorstellen (dem *Drain*-Pin und dem *Source*-Pin), der wiederum durch einen dritten Pin (dem *Gate*-Pin) gesteuert wird. Das funktioniert ein wenig wie bei einem Lichtschalter, bei dem der Gate-Pin dem Teil entspricht, mit dem das Licht ein- oder ausgeschaltet wird. Ein Lichtschalter ist mechanisch, er wird mit einem Fingers gesteuert. Bei einem MOSFET handelt es sich hingegen um ein elektronisches Bauteil – er wird über einen Pin deines Arduino gesteuert.

MOSFET steht für "Metal Oxide Semiconductor Field Effect Trasistor". Es handelt sich um einen speziellen Transistor-Typ, 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 er

sich sehr einfach als Schnittstelle benutzen lässt. Diese Art Transistor ist ideal, um größere Lasten in hoher Frequenz ein- oder auszuschalten.

In Abbildung 5-7 ist dargestellt, wie du einen MOSFET wie den IRF520 nutzen kannst, um einen kleinen Motor eines Ventilators an- oder auszuschalten. Du siehst außerdem, dass der Motor seine Energiezufuhr vom VIN-Anschluss des Arduino-Boards bezieht, das für eine Spannung zwischen 7 und 12 Volt ausgelegt ist. Dies ist ein weiterer Vorteil des MOSFET: Er ermöglicht das Betreiben von Bauteilen, die eine andere Spannung als die vom Arduino benutzten 5V benötigen.

Bei der schwarzen Komponente mit dem weißen Streifen handelt es sich um eine *Diode*. Im vorliegenden Schaltkreis wird sie verwendet, um den MOSFET zu schützen.

Praktischerweise lässt sich der MOSFET sehr schnell ein- und ausschalten, so dass du immer noch PWM benutzen kannst, um eine Lampe oder einen Motor zu steuern. In Abbildung 5-7 ist der MOSFET an Pin 9 angeschlossen, so dass wir außerdem mit `analogWrite()` die Motorgeschwindigkeit mittels PWM steuern können.

Um den Schaltkreis nachzubauen, benötigst du einen IRF520-MOSFET, den du beispielsweise bei Elektronik Reichelt bestellen kannst und eine 1N4007-Diode (ebenfalls bei Elektronik Reichelt zu beziehen). Wenn der Motor während des Uploads ungewollt anspringt, platziere einen 10K-Ohm-Widerstand zwischen Pin 9 und GND.

In Kapitel 8, *Ein System zur automatischen Gartenbewässerung*, auf Seite 93 wirst du das *Relais* kennenlernen, mit dem eine weitere Möglichkeit zur Verfügung steht, Bauteile zu steuern, die mehr Strom verbrauchen.

Komplexe Sensoren

Wir definieren komplexe Sensoren als Sensoren, die ihre Informationen in einer Form bereitstellen, die nicht allein mit `digitalRead()` oder `analogRead()` gelesen werden kann. Diese Sensoren verfügen üblicherweise über einen eigenen Schaltkreis, möglicherweise mit einem eigenen Mikrocontroller. Einige Beispiele für komplexe Sensoren sind digitale Temperatursensoren, Ultraschall-Ranger, Infrarot-Ranger und Beschleunigungsmesser . Ein Grund für diese Komplexität besteht darin, mehr Informationen oder eine größere Genauigkeit bieten zu können; beispielsweise haben einige Sensoren eindeutige Adressen, so dass du viele Sensoren an dieselben Drähte anschließen und dennoch jeden einzelnen individuell anweisen kannst, seine Temperatur mitzuteilen.

Glücklicherweise bietet der Arduino eine Reihe von Mechanismen, um diese komplexen Sensoren auszulesen, von denen du in diesem Buch einige

kennenlernen wirst. In Kapitel 8, *Ein System zur automatischen Gartenbewässerung*, auf Seite 93 unter Abschnitt *Testen der Echtzeituhr (RTC)*, auf Seite 98 beispielsweise wird beschrieben, wie eine Echtzeituhr ausgelesen wird, und in Abschnitt *Testen des Temperatur- und Feuchtigkeitssensors*, auf Seite 117 wirst du erfahren, wie Informationen von einem Temperatur- und Feuchtigkeitsensor abgerufen werden.

Weitere Beispiele findest du auf unserer Website im Abschnitt Tutorials (*http://bit.ly/11UYi7O*).

In *Making Things Talk – Die Welt hören, sehen, fühlen* (*http://www.oreilly.de/catalog/makingthingstalkger/index.html*) von Tom Igoe, erschienen bei O'Reilly, werden komplexe Sensoren sehr ausführlich beschrieben.

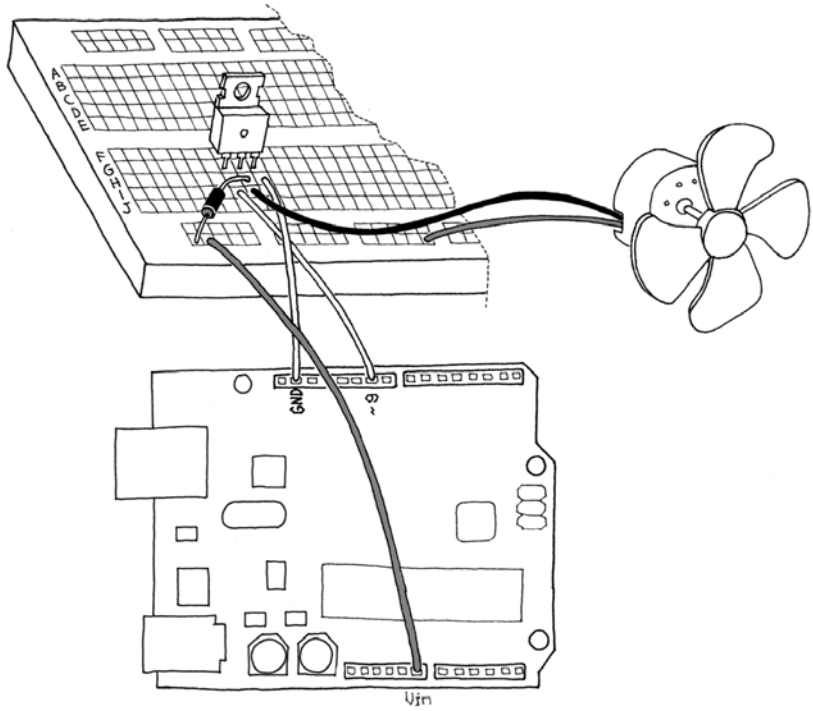


Abbildung 5-7: *Ein Motorschaltkreis für Arduino*

6/Der Arduino Leonardo

Bisher habe ich den Eindruck erweckt, als gäbe es nur einen Typ des Arduino- Boards, den Arduino Uno. Tatsächlich gibt es aber mehrere verschiedene Arduino-Boards, die du dir anschauen kannst, wenn du den Bereich Products auf der Arduino-Website *http://arduino.cc/en/Main/Products* besuchst.

In diesem Kapitel möchte ich auf den Leonardo eingehen, da dieses Board ziemlich einzigartig ist.

Worin unterscheidet sich dieses Arduino-Board von anderen Arduino-Boards?

Wie du möglicherweise noch aus Kapitel 3, *Die Arduino-Plattform*, auf Seite 15 weißt, besteht das Herz des Arduino Uno aus einem Mikrocontroller mit der Bezeichnung ATmega328. Bisher habe ich noch nicht erwähnt, dass es auf dem Uno noch einen zweiten Mikrocontroller gibt, einen *ATmega16U2*, der für die Arbeit mit der USB-Schnittstelle zuständig ist. Vielleicht hast du ihn schon bemerkt, er befindet sich neben dem USB-Anschluss. Ich habe diesen Chip bisher nicht erwähnt, weil er für Anfänger nicht leicht zugänglich ist und seine Aufgabe vor allem darin besteht, dir nicht im Weg zu sein und sich um die USB-Verbindung zu kümmern.

Der Arduino Uno benötigt diesen zweiten Mikrocontroller, weil der ATmega328 nicht mit USB-Verbindungen arbeiten kann, während der ATmega16U2 zu wenige der Features umfasst, durch die Arduino erst nützlich wird.

Glücklicherweise haben einige Mikrocontroller-Designer hart daran gearbeitet, die Arduino-Features des ATmega328 mit den USB-Features des ATmega16U2 zu kombinieren. Das Resultat ist der ATmega32U4.

Auf dem Arduino Leonardo kommt der ATmega32U4 zum Einsatz, damit der Umgang mit USB als auch die typischen Arduino-Aufgaben bewerkstelligt werden können. Das bedeutet, dass der Code, der für die USB-Aufgaben erforderlich ist, auf irgendeine Weise auch Teil deines Sketches ist, was

wiederum bedeutet, dass du das USB-Verhalten über deinen Sketch ändern kannst, indem du spezielle Bibliotheken für Maus und Tastatur benutzt. Diese Bibliotheken erlauben einem Computer, einen mit ihm verbundenen Arduino Leonardo als Maus und/oder Tastatur zu betrachten. Du wirst feststellen, dass du damit einige hübsche Dinge anstellen kannst.

Mittlerweile wird der ATmega32U4 auch auf anderen Arduino-Boards wie z. B. dem Arduino Yun, dem Arduino Micro und dem Arduino Esplora sowie dem Arduino Robot eingesetzt. Die hier beschriebenen Features sind zutreffend für alle Arduinos, die auf dem ATmega32U4 basieren.

Weitere Unterschiede zwischen Arduino Leonardo und Arduino Uno

Bevor ich die netten Dinge beschreibe, die du tun kannst, möchte ich noch einige weitere Unterschiede zwischen dem Arduino Uno und dem Arduino Leonardo erwähnen.

Die wichtigsten Unterschiede, die du kennen solltest, sind folgende:

- In Abschnitt *Steuerung von Licht mittels PWM*, auf Seite 50 hast du erfahren, wie du `analogWrite()` benutzt, um die Helligkeit einer LED zu steuern. Hierfür kann nicht jeder Arduino-Pin verwendet werden. Auf einem Uno funktioniert `analogWrite()` nur mit den Pins 3, 5, 6, 9, 10 und 11, während beim Arduino Leonardo `analogWrite()` auch Pin 13 benutzt werden kann. (Du kannst also die Helligkeit der eingebauten LED steuern, die mit Pin 13 verbunden ist!)
- Wenn du einen Arduino Uno an deinen Computer anschließt, ist die serielle USB-Verbindung eingerichtet und bleibt bestehen, solange der Arduino mit deinem Computer verbunden ist, auch wenn du für deinen Arduino einen Reset durchführst. Beim Arduino Leonardo wird der USB-Port *im Sketch* erzeugt. Wenn also für den Arduino Leonardo ein Reset erfolgt, wird die serielle USB-Verbindung unterbrochen und anschließend neu aufgebaut. Je nachdem, was du tust, hat das möglicherweise Auswirkungen. Unter Windows wird dein Computer einige Male ein Piepgeräusch von sich geben.
- Der Arduino Uno besitzt nur 6 analoge Eingänge, A0–A5, die alle in einer Gruppe zusammengefasst sind. Der Leonardo verfügt über 12 analoge Eingänge, die eine Bezeichnung von A0 bis A11 aufweisen. Die Eingänge A0–A5 befinden sich an derselben Stelle wie beim Uno, während sich die Eingänge A6–A11 auf den digitalen Pins 4, 6, 8, 9, 10 und 12 befinden. Diese zusätzlichen analogen Eingänge sind auf der Rückseite des Leonardo beschriftet.

- Beim Arduino Leonardo wird ein USB-Kabel vom Typ Micro-B verwendet.
- Wenn du einen Leonardo das erste Mal an einen Mac anschließt, wird der Keyboard Setup Assistant gestartet. Hinsichtlich des Leonardo sind keinerlei Konfigurationen erforderlich, schließ' also den entsprechenden Dialog, indem du auf die rote Schaltfläche im oberen linken Bereich des Fensters klickst.

Erscheint es dir ein wenig seltsam, dass einige der digitalen Pins auch als analoge Eingangs-Pins verwendet werden können? Dazu ist zu sagen, dass sich alle analogen Pins als digitale Pins benutzen lassen, auch auf dem Arduino Uno. Du kannst sie über ihren analogen Namenansprechen, z.B. folgendermaßen:

```
pinMode(A4, OUTPUT);
```

oder

```
button = digitalRead(A3);
```

Für einen Mikrocontroller-Pin ist es nicht ungewöhnlich, dass er zu mehr als nur einem Zweck verwendet wird, wobei du den Pin allerdings immer nur für eine dieser Aufgaben gleichzeitig benutzen kannst.

Da du nun die Unterschiede kennst, lass uns den Arduino Leonardo anweisen, sich so zu verhalten, als sei er eine Tastatur.

Vergiss nicht, der Arduino-IDE mitzuteilen, welches Board du benutzt! Wähle aus dem Tools-Menü in der IDE zunächst die Option Board und dann den Leonardo.

Beispiel für eine Tastaturnachricht mit dem Leonardo

Wenn in diesem Beispiel der Button gedrückt wird, wird ein Text-String als Tastatureingabe an den Computer gesendet. Dieser String gibt an, wie häufig der Button gedrückt wurde.

Fertige zunächst den Schaltkreis an. Du benötigst einen einfachen Drucktaster der Art, die in Abbildung 4-6 zu sehen ist, verwende beim Arduino aber Pin 4 anstelle von Pin 7.

Öffne dann einen neuen Sketch in der Arduino-IDE. Du kannst deinen Sketch *KeyboardMessage* nennen. Gib den Code aus Beispiel 6-1 in Arduino ein. Du kannst ihn auch über den Link zu den Code-Beispielen auf der Katalogseite zu diesem Buch (*http://bit.ly/start_arduino_3e*) herunterladen, oder öffne File→Examples→09.USB→Keyboard→KeyboardMessage in der IDE.

Beispiel 6-1: *Der Leonardo gibt vor, eine Tastatur zu sein, und gibt einen Text-String ein, wenn ein Button gedrückt wurde.*

```
/*
 Keyboard Button test

 For the Arduino Leonardo and Micro.

 Sends a text string when a button is pressed.

 The circuit:
 * pushbutton attached from pin 2 to +5V
 * 10-kilohm resistor attached from pin 4 to ground

 created 24 Oct 2011
 modified 27 Mar 2012
 by Tom Igoe

 This example code is in the public domain.

 http://www.arduino.cc/en/Tutorial/KeyboardButton
 */

const int buttonPin = 4;          // input pin for pushbutton
int previousButtonState = HIGH;   // for checking the state of a pushButton
int counter = 0;                  // button push counter

void setup() {
  // make the pushButton pin an input:
  pinMode(buttonPin, INPUT);
  // initialize control over the keyboard:
  Keyboard.begin();
}

void loop() {
  // read the pushbutton:
  int buttonState = digitalRead(buttonPin);
  // if the button state has changed,
  if ((buttonState != previousButtonState)
    // and it's currently pressed:
  && (buttonState == HIGH)) {
    // increment the button counter
    counter++;
    // type out a message
    Keyboard.print("You pressed the button ");
    Keyboard.print(counter);
    Keyboard.println(" times.");
```

```
  }
  // save the current button state for comparison next time:
  previousButtonState = buttonState;
}
```

Lade den Sketch und öffne dann ein neues Dokument in einem beliebigen Texteditor oder Textverarbeitungsprogramm. Wenn du den Button drückst, solltest du ungefähr Folgendes sehen:

```
You pressed the button 1 times.
```

Dies erscheint dir möglicherweise nicht besonders aufregend. Ist es nicht dasselbe, wie einen String mittels `Serial.println()` zum Serial Monitor zu schicken, wie du es in Beispiel 5-5 getan hast? Tatsächlich besteht jedoch ein großer Unterschied.

Wenn du mit `Serial.println()` einen String zum Serial Monitor schickst, sendet Arduino einen speziellen Code für jeden Buchstaben (den sogenannten *ASCII-Code*, siehe Abschnitt *Variablen*, auf Seite 192). Auf Seiten des Computers ist der serielle Monitor dafür zuständig, den ASCII-Code von der seriellen Schnittstelle zu lesen und anschließend deinen Computer anzuweisen, den korrekten Buchstaben anzuzeigen. Dies funktioniert, weil der serielle Monitor weiß, wie mit seriellen Schnittstellen umzugehen ist.

Im Gegensatz dazu dient Beispiel 6-1 dazu, vorzugeben, es handle sich um eine Tastatur, daher wird der Serial Monitor hier nicht benötigt. `Serial.println()` könnte nicht in einen Texteditor oder ein Textverarbeitungsprogramm schreiben, weil diese Programme nicht wissen, wie von einer seriellen Schnittstelle gelesen wird.

Lass es mich anderes ausdrücken, weil diese Unterscheidung so immens wichtig ist:

Die Verwendung von `Serial.println()` zum Senden einer Nachricht vom Arduino an ein Programm auf deinem Computer wird nur bei Programmen funktionieren, die von einer seriellen Schnittstelle lesen, während Beispiel 6-1 mit jedem Programm funktioniert, das Eingaben über eine USB-Tastatur erwartet.

Wie funktioniert das?

Der Teil mit dem Drucktaster weist keine Besonderheiten auf, daher weise ich hier nur auf die neuen Konzepte hin.

In `setup()`wird das diesem Beispiel zugrunde liegende Keyboard-Objekt initialisiert. Wie im seriellen Objekt, das in Abschnitt *Serielle Kommunikation*, auf Seite 64 vorgestellt wurde, wurden im Keyboard-Objekt einige Fähigkeiten gebündelt.

Im `loop()`-Bereich wird darauf gewartet, dass ein Button gedrückt wird. Anschließend wird die Nachricht mittels `Keyboard.print()` und `Key-`

`board.println()` eingegeben, womit vorgegeben wird, dass du die Nachricht über eine USB-Tastatur eingegeben hast.

Das bedeutet, dass der Arduino jede Taste senden kann, also nicht nur Zeichen wie Buchstaben oder Zahlen. Tatsächlich können zusätzlich zu Tastaturtasten auch Mausklicks und Mausbewegungen gesendet werden, was du im nächsten Beispiel sehen wirst.

Wenn du den Befehl `Keyboard.print()` verwendest, wird der Arduino Leonardo die Tastatur deines Computers übernehmen. Wenn die Bibliothek für die Tastatur (oder die Maus) kontinuierlich läuft, wird es schwer sein, deinen Leonardo zu programmieren.

Um sicherzustellen, dass du nicht die Kontrolle über deinen Computer verlierst, während du einen Sketch mit diesen Funktionen ausführst, solltest du immer ein vertrauenswürdiges Kontrollsystem einrichten, bevor du `Keyboard.print()` oder `Mouse.move()` aufrufst. Dieser Sketch umfasst einen Drucktaster, um die Tastatur umzuschalten, so dass sie nur ihren Dienst tut, wenn der Button gedrückt wurde.

Beispiel für eine Steuerung der Maustaste mit dem Leonardo

Wenn in diesem Beispiel ein Button gedrückt wird, wird die Bewegung eines Maus-Cursors an den Computer geschickt, als ob du deine reale Maus bewegt hättest. Vier Buttons werden benutzt, um die Mausbewegungen nach oben, nach unten, nach links sowie nach rechts zu senden, und ein fünfter Button wird zum Senden eines linken Mausklicks verwendet.

Baue zunächst den Schaltkreis. Du benötigst fünf Drucktaster, wie in Abbildung 6-1 gezeigt.

Öffne wie üblich einen neuen Sketch in der Arduino IDE. Nenne deinen Sketch *ButtonMouseControl* und gebe den Code aus Beispiel 6-2 in Arduino ein. Du kannst ihn auch über den Link zu den Codebeispielen auf der Katalogseite zu diesem Buch (*http://bit.ly/start_arduino_3e*) herunterladen, oder du öffnest File→Examples→09.USB→Mouse→ButtonMouseControl in der IDE.

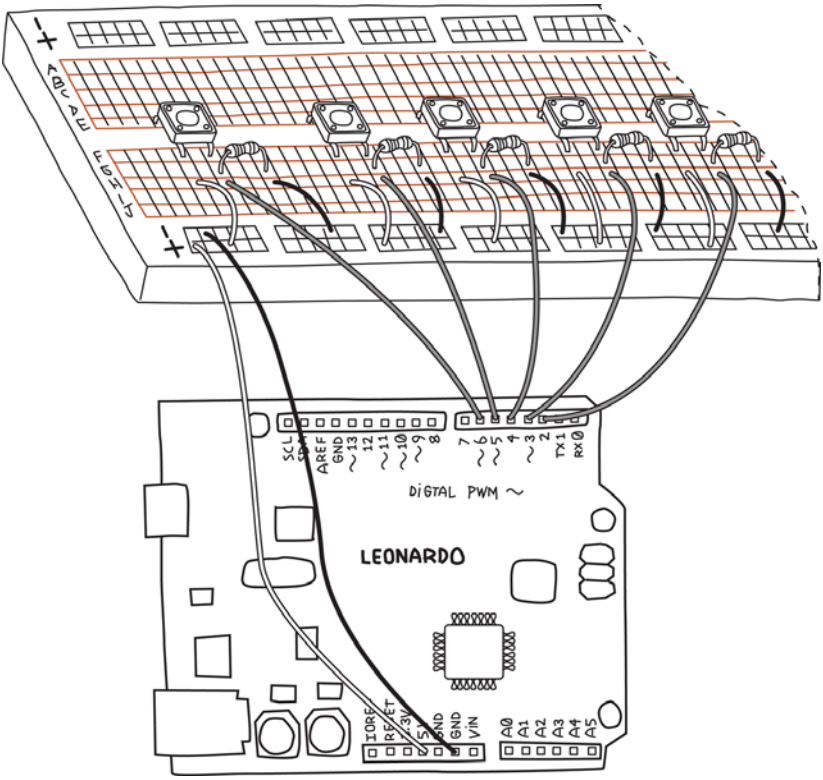


Abbildung 6-1: *Die Schaltung für das Beispiel zur Button-Maussteuerung per USB*

Beispiel 6-2: *Der Leonardo gibt vor, eine Maus zu sein, und sendet Mausbewegungsereignisse oder linke Mausklicks, wenn der entsprechende Button gedrückt wird.*

```
/*
  ButtonMouseControl

 Controls the mouse from five pushbuttons on an Arduino Leonardo or Micro.

 Hardware:
 * 5 pushbuttons attached to D2, D3, D4, D5, D6


 The mouse movement is always relative. This sketch reads
 four pushbuttons, and uses them to set the movement of the mouse.

 WARNING:  When you use the Mouse.move() command, the Arduino takes
```

```
 over your mouse!  Make sure you have control before you use the mouse
 commands.

 created 15 Mar 2012
 modified 27 Mar 2012
 by Tom Igoe

 this code is in the public domain

 */

// set pin numbers for the five buttons:
const int upButton = 2;
const int downButton = 3;
const int leftButton = 4;
const int rightButton = 5;
const int mouseButton = 6;

int range = 5;         // output range of X or Y movement; affects movement speed

int responseDelay = 10; // response delay of the mouse, in ms

void setup() {
  // initialize the buttons' inputs:
  pinMode(upButton, INPUT);
  pinMode(downButton, INPUT);
  pinMode(leftButton, INPUT);
  pinMode(rightButton, INPUT);
  pinMode(mouseButton, INPUT);
  // initialize mouse control:
  Mouse.begin();
}

void loop() {
  // read the buttons:
  int upState = digitalRead(upButton);
  int downState = digitalRead(downButton);
  int rightState = digitalRead(rightButton);
  int leftState = digitalRead(leftButton);
  int clickState = digitalRead(mouseButton);

  // calculate the movement distance based on the button states:
  int  xDistance = (leftState - rightState)*range;
  int  yDistance = (upState - downState)*range;

  // if X or Y is non-zero, move:
  if ((xDistance != 0) || (yDistance != 0)) {
    Mouse.move(xDistance, yDistance, 0);
  }

  // if the mouse button is pressed:
  if (clickState == HIGH) {
    // if the mouse is not pressed, press it:
    if (!Mouse.isPressed(MOUSE_LEFT)) {
      Mouse.press(MOUSE_LEFT);
```

```
    }
  }
  // else the mouse button is not pressed:
  else {
    // if the mouse is pressed, release it:
    if (Mouse.isPressed(MOUSE_LEFT)) {
      Mouse.release(MOUSE_LEFT);
    }
  }

  // a delay so the mouse doesn't move too fast:
  delay(responseDelay);
}
```

Lade den Sketch und drücke dann nacheinander die Bewegungs-Buttons (die mit den Pins 2, 3, 4, oder 5 verbunden sind). Du solltest nun sehen, wie sich der Cursor auf deinem Computer bewegt. Bewege den Cursor über ein partiell verborgenes Fenster und drücke den Button, der dem Klicken mit der linken Maustaste entspricht. Das verborgene Fenster sollte nun im Vordergrund angezeigt werden.

Wie funktioniert das?

Die wichtigen Funktionen hier sind `Mouse.move()`, `Mouse.press()` und `Mouse.release()`.

Abgesehen von der Tatsache, dass fünf Buttons vorhanden sind, ist Beispiel 6-2 auch deshalb komplizierter als Beispiel 6-1 , weil die Anzahl der Mausbewegungen berechnet werden muss. In diesem Sketch wird eine Schrittgröße von 5 verwendet, und diese Zahl wird dann entweder zur Variablen `xDistance` hinzugefügt oder von ihr abgezogen, je nachdem, ob der linke oder rechte Button gedrückt wird, und außerdem zur Variablen `yDistance` hinzugefügt oder von ihr abgezogen, je nachdem, ob der Button für aufwärts oder für abwärts gedrückt wurde. Sobald die Anzahl der Bewegungen berechnet wurde, wird der Cursor mittels `Mouse.move(xDistance, yDistance, 0)` bewegt.

Wenn der fünfte Button – also der, der dem Klicken mit der linken Maustaste entspricht – gedrückt wird, sendet der Leonardo mittels `Mouse.press(MOUSE_LEFT)` eine entsprechende Nachricht hierüber.

Wie du siehst, bedeutet dies, dass du jedes Programm auf deinem Computer mit dem Leonardo steuern kannst. Du kannst deinen eigenen (physikalischen) Shortcut erstellen, mit dem du automatisch eine E-Mail an einen Freund erstellen und senden kannst, z.B. mit folgendem Inhalt: Entschuldige, ich werde mich zum Mittagessen verspäten; wir sehen uns in einer halben Stunde. Du kannst das Cockpit eines Flugzeugs mit verschiedenen Schaltern, Steuerknüppeln und Knöpfen simulieren und für dein bevorzugtes Video-Game benutzen. Du kannst den Leonardo mit einem großen ro-

ten Button versehen, der deinen Computer anweist, die Arduino-IDE für die Neuprogrammierung des Leonardo zu nutzen. Du kannst auch einen Lichtsensor an deinem Leonardo anbringen, damit du von deinem Computer abgemeldet wirst, sobald es dunkel wird, und du gezwungen wirst, nach Hause zu gehen (siehe File→Examples→09.USB→Keyboard→KeyboardLogout.)

Die letzten beiden Beispiele zeigen, dass du diese neuen Möglichkeiten sehr vorsichtig nutzen solltest. Alles, wozu du Maus und Tastatur benutzt, kann nun mittels deines Leonardo erfolgen, einschließlich des Löschens von Dateien, des Rebootens deines Computers und der Änderung von Passwörtern. Du musst wirklich wissen, was du tust, damit du dich nicht auf unkalkulierbare Abenteuer einlässt.

Mehr über das Leonardo-USB-Keyboard und Maus-Bibliotheken sowie entsprechende Beispiele findest du auf der Arduino-Website (*http://arduino.cc/en/Reference/MouseKeyboard*) oder auch im deutschsprachige Buch Die elektronische Welt mit Arduino entdecken (http://www.oreilly.de/catalog/elekarduinobas2ger/) von Erik Bartmann, das im O'Reilly Verlag erschienen ist und auf über 1.000 Seiten über allle Arduino Boards ausführlich informiert.

Weitere Unterschiede beim Leonardo

Es gibt noch weitere Unterschiede, die du wahrscheinlich als Einsteiger nicht entdecken wirst, die ich hier aber trotzdem erwähne, damit du sie kennst. Die meisten Unterschiede betreffen Anschlüsse, die sich auf abweichenden Pins befinden.

So wie dein Computer verschiedene Arten von Anschlüssen besitzen kann (USB, Video, 1394, parallel), ist das auch bei deinem Arduino der Fall. Du hast bereits die serielle Schnittstelle benutzt. Hier nun die anderen Anschlüsse:

- Ein Anschlusstyp ist der sogenannte I2C-Bus, den wir in Kapitel 8, *Ein System zur automatischen Gartenbewässerung*, auf Seite 93 benutzen werden. Sowohl der Uno als auch der Leonardo besitzen einen solchen I2C-Bus, der sich allerdings auf unterschiedlichen Pins befindet. Beim Arduino Uno sitzt der I2C-Bus auf den analogen Pins A4 und A5, beim Leonardo hingegen auf den digitalen Pins D2 und D3.

  Um Verwirrungen zu vermeiden, weisen der Arduino Uno R3, Arduino Leonardo und alle jüngeren Arduinos eine Kopie der I2C-Pins über den AREF-Pins auf. Diese neuen Pins tragen die Bezeichnung SCL und SDA. Sie werden sich immer an derselben Stelle befinden, nämlich wie bereits erwähnt oberhalb der AREF-Pins, unabhängig davon, welcher Typ Mikrocontroller auf dem Board zum Einsatz kommt.

- Alle Arduino-Boards verfügen über eine spezielle Methode, brandneue Mikrocontroller zu programmieren (z.B. für den Fall, dass du den Mikrocontroller ersetzen musst). Zu diesem Zweck wird ein anderer spezieller Anschluss, der sogenannte ICSP-Header, benötigt.

  Wie du vielleicht schon erwartest, befindet er sich beim Uno und beim Leonardo auf jeweils abweichenden Pins. Anders als der I2C-Bus besitzt der ICSP-Header bereits seinen eigenen Anschluss: Es handelt sich dabei um die Gruppe von sechs Pins, die in zwei Reihen mit je drei Pins angeordnet sind.

  Bei einigen älteren Bibliotheken und Shields, die entwickelt wurden, *bevor* es den Leonardo gab, wurde jedoch der Umstand genutzt, dass der ICSP-Header sich auch auf bekannten digitalen Pins befand. Diese Bibliotheken und Shields werden beim Leonardo nicht einwandfrei funktionieren, da der ICSP-Header sich nicht auf diesen Pins befindet. (Bei Shields handelt es sich um Boards, die in die Pins eines Arduino gesteckt werden und zusätzliche Funktionalität bereitstellen.)
- In Abschnitt *Serielle Kommunikation*, auf Seite 64 hast du erfahren, wie die serielle Schnittstelle genutzt wird, um Sensorinformationen vom Arduino an deinen Computer zu senden. Es wird dich nun nicht überraschen, dass die serielle Schnittstelle nicht nur mit dem USB-Anschluss verbunden ist, sondern sich außerdem auf dem Arduino Uno auf den digitalen Pins 0 und 1 befindet.

  Der Uno besitzt nur eine serielle Schnittstelle, der Leonardo zwei, und diejenige, die mit den digitalen Pins 0 und 1 verbunden ist, trägt die Bezeichnung *Serial1*, während sie auf dem Uno *Serial* heißt. Die serielle Schnittstelle, die mit dem USB-Anschluss verbunden ist, wird auf allen Arduino-Boards immer *Serial* genannt.

Es gibt auch ein paar Unterschiede, die sich nicht auf die Anschlüsse beziehen.

- Wenn du einen analogen Eingangs-Pin benutzt und dabei die Bezeichnungen A0–A5 verwendest, werden diese Namen in Zahlen übersetzt; alle analogen und digitalen Funktionen arbeiten dabei auf die gleiche Weise, egal, ob du den Namen oder die Zahlen benutzt. Wenn du auf dem Uno

  ```
  pinMode(A0,OUTPUT);
  ```

  eingibst, ist dies vollkommen identisch mit folgender Funktion:

  ```
  pinMode(14,OUTPUT);
  ```

  Auf dem Arduino Uno sind die Pins, die auf D13 folgen, fortlaufend nummeriert, so dass 14 identisch ist mit A0, 15 mit A1 usw.

  Beim Leonardo sind die Pins auf andere Weise nummeriert, so dass Bibliotheken oder Beispiele, in denen Zahlen anstelle der Namen verwendet werden, nicht einwandfrei funktionieren. Von nun an sollten in Bi-

bliotheken und Beispielen immer die analogen Namen der Pins genutzt werden, selbst wenn sie in digitaler Weise verwendet werden, z.B. so:

```
digitalWrite(A4, HIGH);
```

- Beim Arduino Uno wird *immer* ein Reset durchgeführt, wenn du den Serial Monitor öffnest. In einigen Bibliotheken und Beispielen hat man sich diesen Vorteil zunutze gemacht. Da beim Leonardo aber kein Reset erfolgt, wenn der Serial Monitor geöffnet wird, funktionieren diese Bibliotheken möglicherweise nicht einwandfrei.
- Ein fortgeschrittenes Leistungsmerkmal des Arduino, die sogenannten *External Interrupts*, verhalten sich auf dem Leonardo und dem Uno jeweils abweichend. Der Uno verfügt nur über zwei External Interrupts, der Leonardo besitzt fünf. Außerdem ist anzumerken, dass sich die Interrupts beim Leonardo zwar auf denselben beiden Pins befinden wie beim Uno, sich aber in der Nummerierung unterscheiden.

Mehr über den Arduino Leonardo erfährst du auf der Arduino-Website (*http://arduino.cc/en/Guide/ArduinoLeonardoMicro*).

7/Kommunikation mit der Cloud

In den vorangegangenen Kapiteln hast du die Grundlagen des Arduino und die grundlegenden zur Verfügung stehenden Bausteine kennengelernt. Ich möchte hier die Bestandteile des Arduino-Alphabets auflisten:

Digitaler Output
: Wir verwenden ihn zur Steuerung einer LED, aber mit einem entsprechenden Schaltkreis lassen sich hierüber auch Motoren steuern, Sound 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 damit sogar die Geschwindigkeit eines Motors steuern.

Digitaler Input
: Er ermöglicht uns, den Zustand von Sensoren auszulesen, bei denen es nur ja oder nein bzw. an oder aus gibt, wie Drucktaster oder Neigungsschalter.

Analoger Input
: Wir können Signale von Sensoren auslesen, die mehr Informationen bereitstellen als nur AN oder AUS. Ein Potentiometer kann beispielsweise Auskunft darüber geben, wo das Licht eingeschaltet wurde, und ein Lichtsensor darüber, wie groß die entsprechende Lichtmenge ist.

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

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

An dieser Stelle wird der Möchtegern-Designer in mir geweckt. Wir werden eine Version einer Lampe meines italienischen Lieblingsdesigners Joe Colombo bauen. Das Objekt, das wir bauen werden, ist inspiriert von der Lampe Aton aus dem Jahr 1964.

Massimo

Die Lampe ist, wie du in Abbildung 7-1 sehen kannst, eine einfache Kugel, die auf einem Sockel sitzt; der Sockel hat ein Loch, um zu verhindern, dass die Kugel von deinem Schreibtisch rollt. Durch dieses Design kann die Lampe in verschiedene Richtungen ausgerichtet werden.

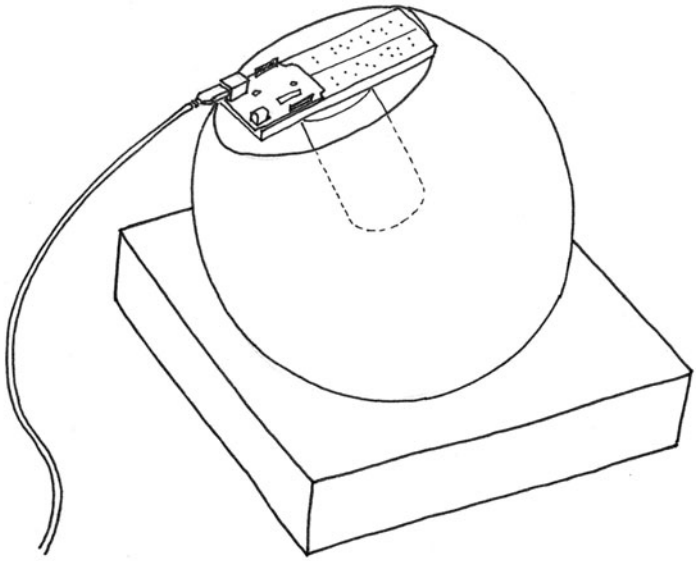


Abbildung 7-1: *Die fertige Lampe*

Hinsichtlich der Funktionalität möchten wir ein Gerät bauen, das mit dem Internet verbunden werden kann, wo es die aktuelle Liste der Artikel im Make-Blog (*http://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 und die Bauteile und Komponenten an, die dazu erforderlich sind. Zuallererst benötigen wir einen Arduino, um uns mit dem Internet zu verbinden. Da der Arduino nur über einen USB-Anschluss 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 dem 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. Andererseits verfügt unser Laptop oder Desktop-Computer über sehr viel mehr RAM und ist für diese Art von Arbeit wesentlich besser geeignet. Daher werden wir mittels Processing ein Proxy-Programm zur Vereinfachung der XML-Datei implementieren, um sie auf dem Computer auszuführen.

Processing

Processing war der Ursprung von Arduino. Wir lieben diese Sprache und benutzen sie, um Einsteigern das Programmieren beizubringen und um schönen Code zu schreiben. Processing und der 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 du findest Tausende von fertigen Programmbeispielen.

Du erhältst Processing unter *https://processing.org/download*.

Das Proxy-Programm erledigt folgende Aufgaben: Es lädt den RSS-Feed unter *http://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 den Arduino. Das Arduino-Board wiederum liefert dem Computer die vom Sensor gemessene Lichtmenge zurück, die dann mittels des Processing-Codes auf dem Computerbildschirm angezeigt wird.

Auf der Hardware-Seite kombinieren wir die Beispiele Drucktaster, Lichtsensor, LED-Steuerung mittels PWM (mal 3!) und serielle Kommunikation. Schau, ob du alle Schaltkreise identifizieren kannst, wenn du sie in Abschnitt *Das Zusammenbauen des Schaltkreises*, auf Seite 90 zusammenbaust. Auf diese Weise werden typischerweise Projekte entwickelt.

Weil es sich beim Arduino um ein einfaches Gerät handelt, müssen wir die Farben auf einfache Weise kodieren. Wir benutzen 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 aus zwei Zeichen bestehenden Gruppen in ein Byte um, das die Helligkeit einer der drei LEDs repräsentiert.

Der Code

Es werden zwei Sketche ausgeführt: der Processing- -Sketch und der Arduino-Sketch. Beispiel 7-1 ist der Code für den Processing-Sketch. Du kannst ihn auch über den Link zu den Code-Beispielen auf der Katalogseite zu diesem Buch (*http://bit.ly/start_arduino_3e*) herunterladen.

Beispiel 7-1: *Die Arduino-Netzwerklampe*

Teile des Codes sind durch einen Blog-Beitrag von Tod E. Kurt (*http://todbot.com*) inspiriert.

```
import processing.serial.*;
import java.net.*;
import java.io.*;
import java.util.*;

String feed = "http://makezine.com/feed/";

int interval = 5 * 60 * 1000;  // retrieve feed every five minutes;
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 = createFont("Helvetica", 24);
  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 = millis();
  fetchData();
}

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

Eine Sache gilt es noch zu tun, bevor der Processing-Sketch einwandfrei läuft: Du musst bestätigen, dass der Sketch den korrekten seriellen Anschluss für die Kommunikation mit dem Arduino verwendet. Dazu musst du 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, lies dir den Kommentar unter der Überschrift IMPORTANT NOTE im Processing-Sketch durch und folge den betreffenden Anweisungen. In diesem Kommentar wirst du gebeten, die erste Zeile nach dem IMPORTANT-NOTE-Kommentar (`//println(Serial.list());`) auszukommentieren, wenn dein Sketch nicht zu laufen scheint.

Wenn du auf einem Mac arbeitest, besteht eine gute Chance, dass sich dein Arduino auf dem letzten seriellen Port in der Liste befindet. Wenn das der Fall ist, kannst du die `0` in `Serial.list()[0]` durch `Serial.list().length -1` ersetzen. Dadurch wird 1 von der Länge der Liste aller seriellen Ports subtrahiert; bei Array-Indizes wird von 0 an gezählt, `length` gibt hingegen Auskunft über die Länge der Liste (es wird von 1 an gezählt), so dass du 1 subtrahieren musst, um den tatsächlichen Index zu erhalten.

Beispiel 7-2 ist der entsprechende Arduino-Sketch. Du kannst ihn auch über den Link zu den Code-Beispielen auf der Katalogseite zu diesem Buch (*http://bit.ly/start_arduino_3e*) herunterladen.

Beispiel 7-2: *Die Arduino-Netzwerklampe (Arduino Sketch)*

```
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 7-2 ist dargestellt, wie der Schaltkreis zusammenzubauen ist. Benutze wie in Abschnitt *Steuerung von Licht mittels PWM*, auf Seite 50 unter Kapitel 5, *Erweiterter Input und Output*, auf Seite 47 einen 220-Ohm-Widerstand (Rot – Rot – Braun) für jede LED, und verwende wie in Abschnitt *Analoger Eingang*, auf Seite 60 einen 10K-Ohm-Widerstand für den Fotowiderstand.

Aus Abschnitt *Steuerung von Licht mittels PWM*, auf Seite 50 weißt du, dass LEDs gepolt sind. In unserem Schaltkreis hier sollte die Anode (langer Pin, positiv) nach rechts und die Kathode (kurzer Pin, negativ) nach links zeigen. In Abbildung 7-2 wird auch die abgeflachte Seite der LED gezeigt, die die Kathode kennzeichnet.

Bau' die Schaltung wie dargestellt nach und verwende dabei eine rote, eine grüne und eine blaue LED. Lade die Sketche in Arduino und Processing und überprüfe dann den Processing-Sketch, indem du ihn ausführst (du musst den Taster drücken, damit die Lampe eingeschaltet wird). Falls Probleme auftreten, schlage im Kapitel Kapitel 9, *Troubleshooting*, auf Seite 171nach.

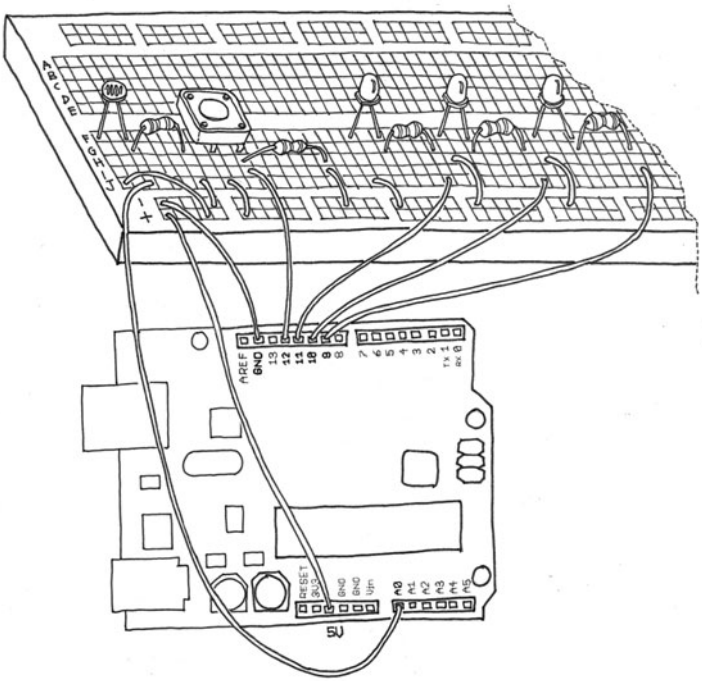


Abbildung 7-2: *Der Schaltkreis für die Netzwerk-Lampe*

Anstelle von drei separaten LEDs kannst du auch eine einzelne RGB-LED verwenden, die über vier Anschlüsse verfügt. Sie lässt sich auf dieselbe Weise anschließen wie die LEDs aus Abbildung 7-2, mit einem Unterschied: Anstelle von drei separaten Verbindungen zum Masse-Pin auf dem Arduino-Board gibt es nur einen einzigen Anschluss zur Masse (der als *gemeinsame Kathode* bezeichnet wird).

In praktisch jedem Elektronfachgeschäft oder bei jedem Elektronikversender werden RGB-LEDs mit vier Anschlüssen recht preisgünstig vertrieben. Anders als bei den separaten einfarbigen LEDs ist es der längste Anschluss der RGB-LED, der zur Masse führt. Die anderen drei werden mit Pin 9, 10 und 11 des Arduino-Boards verbunden (mit jeweils einem 220-Ohm-Widerstand zwischen den Anschlüssen und den Pins, genau wie bei den separaten roten, grünen und blauen LEDs).

Der Arduino-Sketch wurde so geschrieben, dass mit einer RGB-LED mit *gemeinsamer Kathode* (bei der der lange Anschlussdraht mit Masse verbunden wird) arbeitet. Wenn du die falsche Ausgabe erhältst, verwendest du möglicherweise eine RGB-LED mit einer *gemeinsamen Anode*. Wenn dies der Fall ist, musst du die Stellen im Code, an denen die Intensität der LED definiert ist, folgendermaßen ändern (du vertauschst im Grunde genommen die Werte; an Stellen, an denen du 0 verwendet hast, setzt du nun 255 ein):

```
if (state == 1) { // if the lamp is on
    analogWrite(R_LED, 255 - r);  // turn the leds on
    analogWrite(G_LED, 255 - g);  // at the colour
    analogWrite(B_LED, 255 - b);  // sent by the computer
  } else {
    analogWrite(R_LED, 255);  // otherwise turn off
    analogWrite(G_LED, 255);
    analogWrite(B_LED, 255);
  }
```

Nun wollen wir die Konstruktion vervollständigen, indem wir die Steckplatine in einer Glaskugel platzieren. Die preiswerteste Möglichkeit besteht darin, bei IKEA eine Tischlampe vom Typ Fado zu kaufen. Sie kostet zur Zeit etwa 15 €.

So funktioniert das Zusammenbauen

Packe die Lampe aus und entferne das Kabel, das in den Sockel der Lampe führt. Du wirst sie nicht mehr an die Steckdose anschließen.

Du kannst ein Gummiband verwenden, um das Arduino-Board auf einer Steckplatine zu befestigen. Klebe dann die Platine mit Heißkleber auf der

Rückseite der Lampe fest, wie in Abbildung 7-1 gezeigt. Lass ein wenig Platz, damit du die LED einfügen und festkleben kannst.

Löte längere Drähte an die RGB-LED und klebe sie an die Stelle, an der sich normalerweise die Glühbirne befindet. Verbinde die Drähte der LED mit dem Breadboard (an der Stelle, an der sie sich befanden, bevor du sie entfernt hast). Du kannst ein wenig Zeit sparen, indem du dir vergegenwärtigst, dass du nur eine Verbindung zur Masse benötigst, ganz gleich, ob du eine RGB-LED oder drei separate LEDs verwendest.

Suche entweder ein hübsches Holzstück mit einem Loch in der Mitte, das als Sockel für die Kugel dienen kann, oder schneide einfach den oberen Teil des Pappkarton, in dem die Lampe geliefert wurde, so zurecht, dass er eine Höhe von etwa 5cm (oder 2") hat. Schneide dann ein Loch mit einem solchen Durchmesser aus, dass die Lampe gehalten wird. Verstärke den Karton, indem du an den Innenkannten Heißleim aufträgst, der den Sockel stabiler macht.

Platziere die Kugel auf dem Sockel, führe das USB-Kabel an der Oberseite heraus und schließe es an deinen Computer an.

Starte deinen Processing-Code, drücke den Ein/Aus-Schalter und beobachte, wie die Lampe zum Leben erwacht. Lade deine Freunde ein und lass sie das Ergebnis bestaunen.

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

- Baue Neigungssensoren ein, um die Lampe durch Rotation in unterschiedliche Richtung ein- oder auszuschalten.
- Füge einen PIR-Sensor hinzu, um festzustellen, ob jemand in der Nähe ist, und um die Lampe auszuschalten, wenn niemand wahrgenommen wird.
- Erzeuge verschiedene Modi, so dass du die Farbe manuell steuern oder mehrere Farben überblenden kannst.

Denk' dir verschiedene Projekte aus, experimentiere und hab' Spaß!

8/Ein System zur automatischen Gartenbewässerung

In Kapitel 7, *Kommunikation mit der Cloud*, auf Seite 81 hast du dein bisheriges Arduino-Wissen einbringen können. Ein Teil des Vergnügens bestand darin, einige der einfachen Übungen aus den ersten Kapiteln in ein praktisches Projekt zu übertragen. Du hast Kenntnisse über die Programmiersprache Processing erworben, und darüber, wie sie sich zur Herstellung eines Proxy auf deinem Computer für Aufgaben benutzen lässt, die mit deinem Arduino nur sehr schwer oder unmöglich zu bewerkstelligen wären.

In diesem Kapitel wirst du wieder einfache Beispiele mit etlichen neuen Ideen kombinieren und sie in einem praktischen Projekt umsetzen. Dabei wirst du mehr über Elektronik, Kommunikation sowie Programmierung erfahren. Außerdem werden wir uns mit Konstruktionstechniken beschäftigen.

Das Ziel dieses Projekts besteht darin, das Wasser jeden Tag zur richtigen Zeit an- und auszustellen, außer wenn es regnet.

Auch wenn du keinen Garten besitzt, kannst du an diesem Projekt Spaß haben. Falls du nur eine kleine Zimmerpflanze mit Wasser versorgen möchtest, bau das Ganze mit nur einem Ventil. Wenn jeden Tag um fünf Uhr nachmittags ein leckeres Getränk deiner Wahl ausgegeben werden soll, zieh in Erwägung, eine Nahrungsmittelpumpe anstelle eines Wasserventils zu nutzen. Bei Adafruit ist beispielsweise eine Schlauchpumpe mit Silikonschlauch erhältlich (*http://bit.ly/15oTtpH*). Ähnliche Pumpen findest du auch bei deutschen Anbietern, sie kosten um die 20 Euro.

> Als Professor bringe ich vielen Studierenden bei, Dinge zu bauen. Dabei habe ich manchmal den Eindruck, die Studenten denken, dass ich sofort genau wüsste, wie ein Projekt umzusetzen ist. Tatsächlich handelt es sich beim Entwickeln eines Projekts aber um einen höchst iterativen Prozess.
>
> Michael

Um ein Projekt zu erstellen, starte mit einer Idee und entwirf grob kleinere Teillösungen; wenn du weiter fortschreitest, werden manchmal Änderungen an der ursprünglichen Idee erforderlich. Wir müssen oft einen Umweg nehmen, um zu lernen, wie ein neues elektronisches Teil funktioniert, um uns ein Programmierkonzept zu erarbeiten, dem wir vielleicht noch nie begegnet sind, oder um uns noch einmal daran zu erinnern, wie ein bestimmtes Feature des Arduino funktioniert, das wir lange Zeit nicht mehr benutzt haben oder das neu für uns ist. Manchmal müssen wir wieder ein Fachbuch zur Hand nehmen, im Internet recherchieren oder jemanden um Rat fragen. Wir sehen uns viele Beispiele, Tutorials und Projekte an, die Bestandteile von dem enthalten, womit wir uns gerade beschäftigen. Wir entnehmen verschiedenen Quellen Häppchen und Teile und fügen sie zusammen, vielleicht zu Beginn ein wenig grob bei wie Frankenstein, um zu sehen, wie alles im Zusammenspiel funktionieren wird.

Wenn das Projekt vom Konzept zum groben Design fortschreitet, damit Teile der Hardware und Software getestet werden können, wird es weiterhin erforderlich sein, immer wieder Schritte zurückzugehen und Änderungen an Dingen vorzunehmen, die wir vorher gemacht haben, damit alles einwandfrei funktionieren kann. Wir kennen keinen einzigen Ingenieur, der mit einem leeren Blatt Papier beginnt und ein ganzes Projekt von vorne bis hinten durchkonzipiert, das dann auch genau wie geplant funktioniert, ohne dass Schritte zurück gegangen und Änderungen vorgenommen werden müssten.

Dies alles gilt sowohl für Hardware als auch für Software.

Entscheidend ist, dass du auch als Einsteiger *bereits dazu in der Lage* bist, Projekte zu realisieren. Beginne mit dem, was du kennst, und füge langsam Features hinzu, jeweils immer eine neue Idee oder ein neues Teil. Scheu' nicht davor zurück, auch interessante Ideen zu verfolgen, die keinen unmittelbaren Nutzen haben.

> Immer wenn ich von möglicherweise interessanten elektronischen Teilen, Programmierkonzepten oder Tricks höre, probiere ich sie aus, auch wenn ich aktuell keine entsprechende Verwendung dafür habe. Dieses Wissen wird dann zu einem weiteren Werkzeug in meinem Werkzeugkasten. Wenn du feststeckst oder etwas nicht weißt, denke immer daran, dass selbst professionelle Ingenieure ständig dazulernen müssen.
>
> Michael

Dank der weitverzweigten und großzügigen Arduino-Community stehen dir zahlreiche Ressourcen über das Internet zur Verfügung. Wenn du nicht gerade ein Einsiedler auf einem Berggipfel bist, wirst du wahrscheinlich immer eine lokale Arduino-Gruppe, einen entsprechenden Club, Makerspace, Hackerspace oder auch einzelne Personen finden, die dir helfen können. Einige Hinweise, wie du Online-Ressourcen am besten nutzen kannst, findest du in Abschnitt *So findest du Online-Hilfe*, auf Seite 180.

Ich werde dir also nicht nur Näheres zur Elektronik, Programmierung und Konstruktion erläutern, sondern auch ein wenig detaillierter auf den Designprozess eingehen. Du wirst sehen, dass einige der vereinfachten Schaltkreise oder Sketche immer wieder modifiziert werden, bis wir beim fertigem Projekt angelangt sind. Trotzdem habe ich einige der Iterationsschritte übersprungen, da dieses Kapitel sonst ein komplett eigenes Buch geworden wäre. Iterationen brauchen Zeit!

Planung

Beginne wie in Kapitel 7, *Kommunikation mit der Cloud*, auf Seite 81 damit, darüber nachzudenken, was du erreichen möchtest und welche Bausteine und Teile du benötigen wirst.

In diesem Projekt werden wir gängige elektrische Wasserventile für den Garten nutzen, die in Heimwerkerläden erhältlich sind. Wenn du schon einmal dort bist, kannst du auch gleich die benötigte Stromversorgung bzw. einen passenden Transformator kaufen. In Kapitel 5, *Erweiterter Input und Output*, auf Seite 47 hast du erfahren, wie sich ein MOSFET nutzen lässt, um einen Motor zu steuern. Dies kann auch für die Wasserventile funktionieren. Einige dieser Ventile arbeiten aber möglicherweise mit *Wechselstrom* (AC = Alternating Current), MOSFETs können jedoch nur mit *Gleichstrom* (DC = Direct Current) arbeiten. Um Wechselstrom zu kontrollieren, benötigen sie ein *Relais*, mit dem sowohl Wechselstrom als auch Gleichstrom gesteuert werden kann.

In Abschnitt *Der Umgang mit größeren Lasten*, auf Seite 66 unter Kapitel 5, *Erweiterter Input und Output*, auf Seite 47 hast du erfahren, dass ein MOSFET eine Art Transistor ist, bei dem über den *Gatter*-Pin gesteuert werden kann, ob Strom zwischen den *Drain*- und den *Source*-Pins fließt. So betrachtet, handelt es sich bei einem MOSFET um einen Schalter. Dies gilt auch für das Relais, das im Inneren einen kleinen mechanischen Schalter hat, der über ein Elektromagnet gesteuert wird: Indem du den Elektromagnet an- oder ausschaltest, kannst du steuern, ob Strom durch den mechanischen Schalter fließt.

Um zu wissen, wann das Wasser auf- oder abgedreht werden soll, wird irgendeine Art von Uhr benötigt. Du könntest in deinem Programm den beim Arduino integrierten Timer nutzen. Das wäre allerdings recht kompliziert und – noch schlimmer– nicht besonders exakt. Es empfiehlt sich vielmehr die Verwendung eines eigens für diesen Zweck gedachten und preisgünstigen Bauteils, das sich leicht mit dem Arduino einsetzen lässt. Es handelt sich um die RTC (Real Time Clock), die auch auf deinem Computer vorhanden ist; sie liefert Datum und Uhrzeit, auch wenn sie für längere Zeit ausgeschaltet war.

Des Weiteren ist ein Sensor erforderlich, um festzustellen, ob es regnet. Wir werden dazu einen Temperatur- und Feuchtigkeitssensor verwenden, denn solche Sensoren sind preisgünstig und lassen sich einfach nutzen. Du musst zwar nicht die Temperatur kennen, du erhältst dieses Feature aber gratis dazu, und es kann eventuell noch nützlich werden.

Zu guter Letzt benötigen wir noch eine Möglichkeit, die An- und Auszeiten festzulegen, d.h. die *Benutzerschnittstelle*. Um dieses Projekt nicht ausufern zu lassen, werde ich den Serial Monitor für die Benutzerschnittstelle verwenden. Wenn du mit Arduino vertrauter sein wirst, kannst du stattdessen eine LCD-Anzeige und Drucktaster verwenden.

Bevor du mit dem Programmieren beginnst, musst du darüber nachdenken, wie die Hardware angeschlossen wird. Ich verwende gerne ein grobes Blockdiagramm, das mir hilft, alle Teile im Blick zu haben und eine Vorstellung davon zu entwickeln, wie sie verbunden werden sollten. Später wirst du genau wissen müssen, wie die einzelnen Komponenten zu verbinden sind, im Blockdiagramm (Abbildung 8-1) verwenden wir aber nur eine Linie, um irgendeine Art von Verbindung zu kennzeichnen.

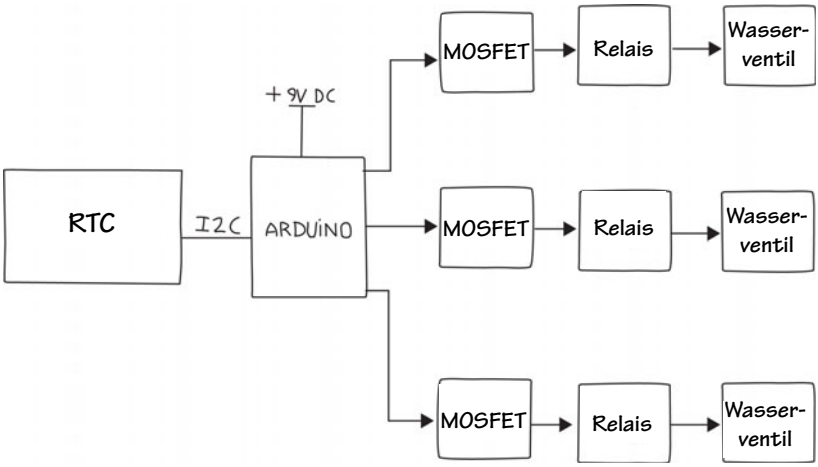


Abbildung 8-1: *Ein Blockdiagramm mit einer RTC, dem Arduino, MOSFETs, Relais, Ventilen und Stromanschlüssen*

Bei diesem Diagramm gehen wir von drei separaten Wasserventilen aus, aber du erkennst schon das Prinzip und kannst es daher an deine jeweiligen Bedürfnisse anpassen.

Da es sich hier um ein Projekt für Fortgeschrittene handelt, werde ich auch Konstruktionstechniken vorstellen. Dieses Projekt soll viele Monate oder vielleicht sogar Jahre zuverlässig arbeiten, du hast also ein anderes Ziel als bei einem Beispiel, das nur demonstrieren soll, wie etwas funktioniert. Die lötfreie Steckplatine, die du bei früheren Projekten genutzt hast, ist bestens fürs Prototyping oder zum Experimentieren geeignet, aber der Zuverlässigkeit halber löten wir bei diesem Projekt die Komponenten auf ein *Proto Shield*. Wir werden außerdem Überlegungen dazu anstellen, wie die Stromversorgung aussehen soll, und alle Verbindungen zu den verschiedenen externen Teilen wie den Wasserventilen realisieren. Wir werden uns sogar darum kümmern, das Projekt durch ein entsprechendes Gehäuse zu schützen.

Bei *Shields* handelt es sich um Boards, die genau in die Pins auf einem Arduino passen und zusätzliche Funktionalität bereitstellen. Das Arduino-Proto-Shield ist ein spezielles Shield und dazu gedacht, dass du deine eigene Schaltung hierauf aufbauen kannst.

Ein weiteres Feature, das nützlich ist, wenn deine Projekte komplexer werden, ist ein Hinweis, was gerade jeweils geschieht. Dies ist hilfreich beim Debuggen und erweist sich besonders dann als nützlich, wenn Teile eines Systems eine gewisse örtliche Entfernung aufweisen, wie beispielsweise die Wasserventile. Wir werden LEDs hinzufügen, um anzuzeigen, dass die Wasserventile aktiviert sind. Denke dabei auch an die Widerstände für die LEDs.

Da wir nun einige zusätzliche Details kennen, fertige ich eine vorläufige Einkaufsliste an. Bei komplexeren Systemen werden erwartungsgemäß Änderungen erforderlich: Wenn ich an einem Sketch arbeite, bemerke ich womöglich, dass ich ein weiteres Teil benötige. (Die vollständige Einkaufsliste mit entsprechenden Links findet sich unter Abschnitt *Die Einkaufsliste für das Bewässerungsprojekt*, auf Seite 169.)

Mach dir keine Sorgen, wenn du nicht bei allen Teilen weißt, was sich dahinter verbirgt. Wir werden sie zu gegebener Zeit vorstellen:

- eine RTC (Real Time Clock = Echtzeituhr)
- einen Temperatur- und Feuchtigkeitssensor
- ein Proto Shield
- drei elektrische Wasserventile
- einen Transformator oder eine Stromversorgung für die Wasserventile
- drei Relais zum Steuern der Wasserventile

- drei Relais-Sockel
- drei LEDs als Aktivierungsanzeige für die Ventile
- drei Widerstände für die LEDs
- eine Stromversorgung für den Arduino (Er soll ja auch dann arbeiten können, wenn kein Computer angeschlossen ist.)

Nun, da du über eine vorläufige Einkaufsliste verfügst, lass uns die einzelnen Elemente anschauen und die Details ausarbeiten. Beginnen wir mit der RTC.

Testen der Echtzeituhr (RTC)

Wenn ich plane, ein mir neues Bauteil einzusetzen, möchte ich sichergehen, dass ich seine Funktionsweise verstanden habe, bevor ich das Design für das ganze System entwerfe. Da die RTC neu für dich ist, lass uns zunächst einmal anschauen, wie sie arbeitet.

Der Hauptbestandteil einer RTC ist der Chip selbst. Der gängigste Chip ist der DS1307, bei dem ein Kristall für eine zuverlässige Zeitkontrolle benutzt wird und eine Batterie, um sie aufrechtzuerhalten, wenn das übrige System abgeschaltet ist. Wir werden die RTC nicht selbst anfertigen, sondern eins der vielen verfügbaren RTC-Module benutzen und auf diese Weise zu sehr geringen Kosten Zeit sparen.

Glücklicherweise funktionieren sie alle ähnlich und stellen auf eine ähnliche Weise Schnittstellen her. Ich habe mich für das TinyRTC-Modul, das du in Deutschland beispielsweise bei Ebay oder bei Amazon für unter 5 Euro bestellen kannst.

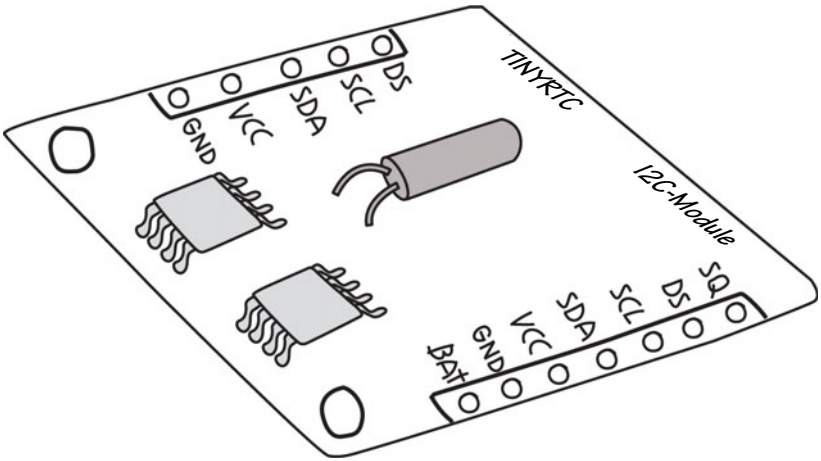


Abbildung 8-2: *Das TinyRTC-Modul*

Beachte, dass die männlichen Stiftleisten nicht immer im Lieferumfang enthalten sind. Du musst sie möglicherweise separat bestellen und einlöten. Männliche Stiftleisten kannst du von verschiedenen Quellen beziehen. Jeder Elektronikversender führt diese Stiftleisten.

Wenn du Einsteiger im Bereich des Lötens bist, findest du einen Link zu einem sehr guten Tutorial dazu in Abschnitt *Dein Projekt auf dem Proto Shield fest verlöten*, auf Seite 150.

Bei diesem Gerät kommt eine Schnittstelle mit der Bezeichnung I2C zum Einsatz, die manchmal auch Two Wire Interface (TWI) oder ganz einfach Wire genannt wird. Arduino stellt hierfür eine integrierte Bibliothek bereit, die ebenfalls den Namen Wire trägt, und bei Adafruit ist eine Bibliothek für den DS1307 erhältlich (*https://github.com/adafruit/RTClib*).

Um diese Bibliothek zu benutzen, klickst du auf die Schaltfläche Download ZIP und entpackst dann diese Datei im Ordner *libraries* in deinem Arduino-Sketch-Ordner.

Den Sketch-Ordner findest du, indem du die Voreinstellungen deines Arduino öffnest (File → Preferences oder Arduino → Preferences auf dem Mac) und dann das Feld mit der Bezeichnung Sketchbook Location suchst.

Wenn du die Bibliothek entpackst, befindet sie sich in einem Ordner namens *RTClib-master*, den du in *RTClib* umbenennen musst.

Wenn die Arduino-IDE geöffnet ist, wirst du sie schließen und erneut öffnen müssen, damit die neue Bibliothek erkannt wird.

Anhand des Beispiels, das mit der Bibliothek geliefert wird, kannst du überprüfen, ob die Bibliothek korrekt installiert wurde. Du musst dafür keinen Schaltkreis aufbauen, technisch gesehen ist es noch nicht einmal erforderlich, dass du deinen Arduino zur Hand hast. Öffne in der Arduino-IDE das File-Menü und wähle dann Examples → RTClib → ds1307, um ein Programmbeispiel zu öffnen. Klicke anschließend nicht auf die Schaltfläche Upload, sondern stattdessen auf die Schaltfläche Verify (siehe Abbildung 4-3). Wenn du die MeldungDone compiling erhältst, hast du die Bibliothek korrekt installiert.

Nach der Installation einer neuen Bibliothek tust du gut daran, zu überprüfen, ob die Bibliothek und auch alle Bibliotheken, auf die sie sich stützt, korrekt installiert sind, bevor du versuchst, dein eigenes Programm zu schreiben.

Die meisten Arduino-Bibliotheken enthalten Beispiele, die vermutlich von denselben Personen stammen, die auch die Bibliothek geschrieben haben und die daher sehr wahrscheinlich korrekt sind.

Das TinyRTC-Modul wird mit zwei Sätzen von Pins ausgeliefert: Einer besteht aus fünf Positionen, der andere aus sieben Positionen. Die meisten Pins liegen in doppelter Ausführung vor, andere Pins bieten zusätzliche Features. Um die RTC zu testen, musst du dich nur um vier Pins kümmern: die zwei Pins, die die I2C-Schnittstelle bilden, Strom, und Masse; die RTC muss über ihre als VCC bezeichneten Pins mit 5V versorgt und ihre Masse (GND) muss mit der Masse des Arduino verbunden werden.

Auf der Hardware-Seite wird I2C von zwei spezifischen Arduino-Pins (SDA und SCL) unterstützt, wie in der Referenz zur Arduino Wire Library (*http://arduino.cc/en/reference/wire*) beschrieben ist.

Für einen Schnelltest kannst du einen gängigen Trick verwenden: Bei Bauteilen wie der RTC, die sehr wenig Strom verbrauchen, kannst du für die Stromversorgung die digitalen Ausgänge verwenden, indem du einen Pin auf HIGH und einen anderen auf LOW setzt, wenn die Pins korrekt angeordnet sind. Ein I/O-Pin, der als digitaler Ausgang auf HIGH gesetzt ist, entspricht im Grunde genommen 5V, und ein I/O-Pin, der als digitaler Ausgang auf LOW gesetzt wurde, der Masse.

Auf einem Uno verhält es sich folgendermaßen: SCL ist A5 und SDA ist A4. Zusätzlich zu diesen beiden Schnittstellen-Pins müssen wir VCC und GND mit anderen I/O-Pins verbinden, um 5V und Masse bereitzustellen. Hierzu können wir das TinyRTC-Modul wie in Abbildung 8-3 dargestellt auf den analogen Eingangs-Pins des Arduino Uno ausrichten. Du kannst auch eine Steckplatine und Schaltdraht verwenden, um die Verbindung herzustellen.

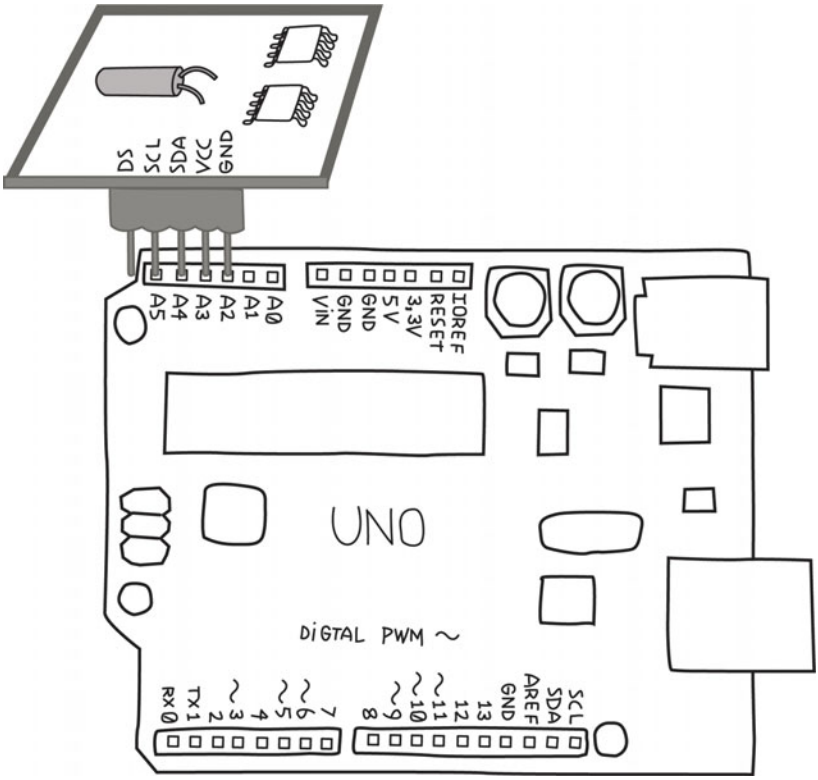


Abbildung 8-3: *Ein TinyRTC wurde direkt in die analogen Eingangs-Pins des Arduino Uno gesteckt. Die Verschiebung ist beabsichtigt – auf diese Weise kann SCL mit A5 und SDA mit A4 verbunden werden.*

Beachte, dass bei neueren Arduinos mit abweichenden Mikrocontrollern, zum Beispiel beim Arduino Leonardo, für die I2C-Signale (SDA und SCL) andere Pins genutzt werden. Aus diesem Grund wird nun bei allen neuen Arduinos ein neues Standard-Pin-Layout befolgt, bei dem die 12C-Pins hinter AREF angefügt werden. Sie stellen eine Ergänzung dar. Andere Pins, die als 12c-Pins fungieren, werden dupliziert.

Die analogen Eingänge A4 und A5 sind für die I2C-Kommunikation zuständig, während A2 und A3 Strom und Masse bereitstellen. A3 muss dem RTC-Pin mit der Bezeichnung VCC 5V liefern, so dass wir ihn auf HIGH setzen, während A2 auf LOW gesetzt wird, um Masse bereitzustellen.

Nun sind wir bereit zum Testen! Öffne in der Arduino-IDE das File-Menü und wähle dann Examples → RTClib → ds1307, um das Programmbeispiel zu öffnen. Denk vor den Kompilieren und Hochladen daran, dass die Pins A2 und A3 so eingerichtet werden müssen, dass sie dem TinyRTC Strom liefern. Füge `setup()` gleich zuoberst die folgenden vier Zeilen hinzu:

```
void setup() {
  pinMode(A3, OUTPUT);
  pinMode(A2, OUTPUT);
  digitalWrite(A3, HIGH);
  digitalWrite(A2, LOW);
```

Falls du ein Board benutzt, das anders verdrahtet ist, benötigst du eine Steckplatine und Schaltdraht, um eine entsprechende Verbindung herzustellen. Daher solltest du diese Extrazeilen an Code nicht hinzufügen. Achte einfach darauf, das Board so zu verdrahten, wie es vom Anbieter vorgegeben ist.

Du kannst das Beispiel mit dieser Änderung auch über den Link zu den Code-Beispielen auf der Katalogseite zu diesem Buch *http://bit.ly/start_arduino_3e* herunterladen.

Da wir schon einmal dabei sind: Beachte, dass in diesem Beispiel der serielle Anschluss bei 57.600 Baud geöffnet wird.

Nun kann der Upload erfolgen. Öffne nach dem Upload den Serial Monitor, den du unter dem Menüeintrag Werkzeuge findest. Aktivere den Button zur Auswahl der Baudrate im sich nun öffnenden Serial Monitor (du findest ihn rechts unten) und wähle dann 57600 baud. Du solltest ungefähr folgende Ausgabe erhalten:

```
2013/10/20 15:6:22
 since midnight 1/1/1970 = 1382281582s = 15998d
 now + 7d + 30s: 2013/10/27 15:6:52

2013/10/20 15:6:25
 since midnight 1/1/1970 = 1382281585s = 15998d
 now + 7d + 30s: 2013/10/27 15:6:55
```

Datum und Uhrzeit sind möglicherweise falsch, aber du solltest sehen, dass die Zahl der Sekunden ansteigt. Wenn du eine Fehlermeldung erhältst, überprüfe nochmals, ob sich die RTC an der richtigen Position befindet, d.h., dass SCL mit A5 verbunden ist und dass die Pins A2 und A3 in `setup()` korrekt eingerichtet sind.

Um in der RTC die korrekte Zeit festzulegen, schau' in die `setup()`-Funktion. Gegen Ende wirst du folgende Zeile finden:

```
rtc.adjust(DateTime(__DATE__, __TIME__));
```

Diese Zeile enthält das Datum und die Uhrzeit, zu dem der Sketch kompiliert wurde, (`__DATE__` bzw. `__TIME__`) und verwendet diese Information für die Einstellung der RTC. Natürlich kann die RTC für ein oder zwei Sekunden lang aus sein, aber das ist für unsere Zwecke exakt genug.

Kopiere die Zeile so, dass sie sich außerhalb der `if()`-Bedingung, d.h. direkt unter `rtc.begin()` befindet:

```
    rtc.begin();
    rtc.adjust(DateTime(__DATE__, __TIME__));
```

Kompiliere den Sketch und lade ihn hoch. Der Serial Monitor sollte nun das korrekte Datum und die korrekte Uhrzeit anzeigen:

```
014/5/28 16:12:35
 since midnight 1/1/1970 = 1401293555s = 16218d
 now + 7d + 30s: 2014/6/4 16:13:5
```

Wenn dein Computer ein falsches Datum oder eine falsche Uhrzeit ausweist, spiegelt sich das natürlich an dieser Stelle wieder.

Nachdem du auf der RTC die Zeit eingestellt hast, solltest du `rtc.adjust` auskommentieren (und den Code hochladen), anderenfalls wird der Reset der Uhrzeit ebenfalls über Jahre beibehalten.

Mehr Information zur Bibliothek und zu den Beispielen findest du unter Arduino Library (*http://bit.ly/1vJTRul*) und unter Understanding the Code (*http://bit.ly/11t22xq*) sowie im Tutorial von Adafruit zum DS1307 Breakout Board Kit (*http://bit.ly/1xSkpbl*).

Dabei ist anzumerken, dass das Adafruit-Board zwar Unterschiede aufweist, der Code jedoch derselbe ist.

Wenn du dich nun vertraut mit der RTC fühlst, können wir zu den Relais übergehen.

Testen der Relais

Welche Art Relais brauchen wir? Das hängt davon ab, wie viel Strom die Wasserventile benötigen. Die meisten Gartenventile scheinen mit 300 Milliampere zu arbeiten. Dies ist eine geringe Strommenge, daher reicht auch ein kleines Relais. Es sind verschiedene Relais erhältlich, die mit unterschiedlichen Voltzahlen betrieben werden können; wir werden eines verwenden, das 5V nutzt, so dass wir keine weitere Stromquelle benötigen. In Abbildung 8-4 ist ein gängiges 5V-Relay abgebildet.

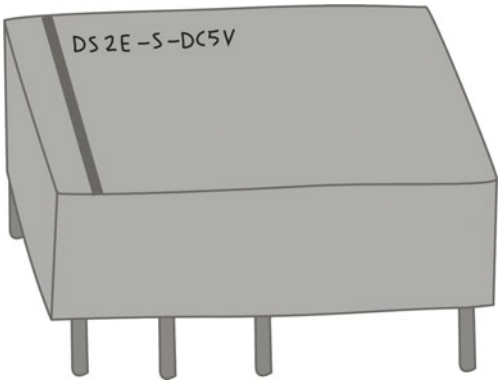


Abbildung 8-4: *Ein 5V-Relais*

Du kannst es bei Digi-Key (*http://www.digikey.de*), aber auch bei vielen anderen Anbietern kaufen.

Zu fast jedem elektronischen Gerät gibt es ein sogenanntes *Datenblatt*, in dem alle technischen Information zu dem Gerät detailliert dokumentiert sind. Für Einsteiger kann das zunächst ein wenig erdrückend wirken, da sehr viele Information enthalten sind, von denen nur ein geringer Teil benötigt wird. Wenn du etwas erfahrener bist, wirst du lernen zu erkennen, was wichtig ist und wie du es schnell findest. Auf dem Datenblatt zu dem von uns ausgewählten Relais (du findest einen Link zum Download des Datenblattes auf der Digi-Key-Webseite) kannst du sehen, dass es mit bis zu 2 Ampere und 30 Volt Gleichstrom (DC) oder 1 Ampere und 125 Volt Wechselstrom (AC) arbeiten kann, was für unsere Zwecke mehr als ausreichend ist. Dieses Relais hat außerdem den Vorteil, dass es mit unserer lötfreien Steckplatine und dem Proto Shield, das du später benutzen wirst, kompatibel ist.

Wann immer du etwas mittels eines Arduino-Ausgangs steuern möchtest, musst du daran denken, dass ein Arduino-Pin nur solche Geräte mit Strom versorgen sollte, die mit bis zu 20 Milliampere betrieben werden (siehe Abschnitt *Der Umgang mit größeren Lasten*, auf Seite 66). Wenn du im Datenblatt nach der Stromstärke suchst, die von diesem Relais benutzt wird, wirst du diese Angabe nicht finden. Allerdings ist der Widerstand angegeben. Nun ist ein wenig Mathematik erforderlich, denn wenn du den Widerstand des Relais (125 Ohm) und die Voltzahl kennst, die Arduino an den I/O-Pins ausgibt (5V), kannst du den Strom mithilfe des Ohm'schen Gesetzes, das du am Ende von Abschnitt *Was ist Elektrizität?*, auf Seite 34 kennengelernt hast, berechnen. Wenn wir die Voltzahl (5) durch den Widerstand (125) teilen, erhalten wir die Stromstärke: 40 Milliampere.

Da dieser Wert über unserem Limit liegt, benötigen wir MOSFETs. Zur Abwechslung benutzen wir einen anderen MOSFET als in Abschnitt *Der Umgang mit größeren Lasten*, auf Seite 66. Wir verwenden den 2N7000. Das

entsprechende Datenblatt steht auf der Website von Fairchild Semiconductor (*http://bit.ly/1vck9Cd*) zur Verfügung.

Wie in Abschnitt *Der Umgang mit größeren Lasten*, auf Seite 66 wird der Gate-Pin durch den I/O-Pin des Arduino gesteuert, und der Drain- und der Source-Pin werden den Schalter bilden, mit dem das Relais reguliert wird. Du musst drei 2N7000-MOSFETs zur Einkaufsliste hinzufügen, einen für jedes Relais.

Ergänze die Liste außerdem um drei Widerstände von je 10KOhm, einen für jedes Relais, um bei den Gate-Pins der MOSFETs ein Floaten zu vermeiden.

Wenn du den Arduino einschaltest oder ein Reset durchführst, werden alle digitalen Pins als Eingänge gestartet, bis dein Programm ausgeführt wird und durch `pinMode()` alle Pins in Ausgänge geändert werden. Das ist wichtig, weil in der kurzen Zeitspanne, bevor die Funktion `pinMode()` deinen Pin in einen Ausgang ändert, der Gate-Pin weder HIGH noch LOW sein wird. Er befindet sich im Zustand *Floating*, was bedeutet, dass der MOSFET leicht eingeschaltet werden könnte, wodurch das Wasser rasch fließen würde. Das würde bei vielen Projekten zwar nicht das Ende der Welt bedeuten, trotzdem ist es eine gute Gepflogenheit, dies zu berücksichtigen.

Verhindern lässt sich das Ganze, wie schon unter Abschnitt *Der Umgang mit größeren Lasten*, auf Seite 66 in Kapitel 5, *Erweiterter Input und Output*, auf Seite 47 angemerkt wurde, mit einem Widerstand von 10kOhm zwischen I/O-Pin und Masse. Der Widerstand von 10kOhm ist niedrig genug, um sicherzustellen, dass kein Floating beim Gate-Pin erfolgt, und groß genug, um uns nicht im Weg zu stehen, wenn wir das Wasser anstellen möchten.

Ein Widerstand, der auf diese Weise benutzt wird, wird als *Pull-down*-Widerstand bezeichnet, weil er den Gate-Pin auf Masse herunterzieht. Manchmal muss ein Anschluss auf 5V gezogen werden. In diesem Fall sprechen wir von einem *Pull-up*-Widerstand.

Wann immer wir einen Motor oder ein Relais steuern, sollten wir eine Diode anbringen, um den MOSFET vor der *Rücklaufspannung* zu schützen, die durch das kollabierende Magnetfeld erzeugt wird, wenn das Relais ausgeschaltet ist. Unser MOSFET besitzt zwar eine eingebaute Diode, die allerdings recht klein ist. Für eine höhere Zuverlässigkeit empfiehlt es sich daher, eine zusätzliche externe Diode anzubringen. Weitere Posten deiner Einkaufsliste sind also drei 1N4148-Dioden (oder gleichwertige Dioden). Da wir schon einmal dabei sind, sollten wir auch die Teilenummer des Relais, und

weil wir den Typ des Relais kennen, auch den passenden Relais-Sockel auf die Liste setzen. Hier nun die Ergänzungen, mit denen wir zur sogenannten Revision 0.1 der Einkaufsliste kommen:

- Füge drei MOSFETs vom Typ 2N7000 zur Steuerung der Relais hinzu.
- Füge drei Widerstände von je 10kOhm hinzu.
- Füge drei Dioden vom Typ 1N4148 oder gleichwertige hinzu.
- Füge drei Relais vom Typ DS2E-S-DC5V hinzu.

Das klingt ganz so, als hätten wir es hier mit einer recht komplexen Schaltung zu tun, oder? Es ist schwierig, sich vorzustellen, wie alle diese Komponenten miteinander verbunden werden sollen.

Glücklicherweise gibt es ein cleveres System, mit der all diese Informationen erfasst werden können. Es nennt sich *Schaltplan*.

Der elektronische Schaltplan

Die meisten elektronischen Schaltkreise werden komplett durch zwei Aspekte definiert: 1) welche Komponenten genutzt werden und 2) wie sie verbunden sind. Wenn nur diese Informationen so klar wie möglich erfasst werden, ist ein elektronischer Schaltplan die übersichtlichste Art und Weise, einen elektronischen Schaltkreis zu visualisieren und zu vermitteln.

Ein elektronischer Schaltplan gibt absichtlich keine Auskunft über Größe, Form oder Farben von Komponenten oder darüber, wie sie physisch nebeneinander platziert werden müssen, weil diese Informationen nicht relevant für die Definition des Schaltkreises sind; es sind vielmehr Konstruktionsdetails für eine bestimmte Implementierung dieses Schaltkreises.

Jede Komponente wird durch ein Schaltkreissymbol dargestellt, mit dem die Komponente eindeutig identifiziert wird, die aber nichts über ihre Größe, Farbe usw. aussagt.

In einigen Fällen sehen die Schaltkreissymbole den Komponenten sehr ähnlich, die sie repräsentieren, in anderen Fällen unterscheiden sie sich deutlich. Insbesondere das Schaltkreissymbol für den Arduino sieht gar nicht wie ein Arduino aus. Für einen elektronischen Schaltplan ist der einzig relevante Aspekt eines Arduino, dass er über diverse Pins verfügt (Strom, Eingänge, Ausgänge usw.). Daher wird er als einfacher Kasten dargestellt, bei dem nur die Pins gekennzeichnet sind.

Elektronische Schaltkreissymbole und -diagramme sind dazu gedacht, die entsprechenden Funktionen schnell und deutlich zu veranschaulichen, wobei sich einige Konventionen durchgesetzt haben. Die wichtigsten sind folgende:

- Die niedrigste Voltzahl wird am unteren Rand des elektronischen Schaltplans und die höchste am oberen Rand angegeben. Normalerweise be-

deutet das, dass sich die GND- Verbindung am unteren Rand befindet und 5V (oder eine höhere Voltzahl, falls sie zum Einsatz kommt) am oberen Rand angeführt ist.

- Die Informationen verlaufen von links nach rechts. Daher werden Sensoren und andere Bauteile für den Eingang links und Ausgänge wie Motoren, LEDs, Relais und Wasserventile rechts dargestellt. Wenn Informationen vom Arduino an einen MOSFET, ein Relais oder ein Wasserventil übertragen werden, werden sie im elektronischen Schaltplan in eben dieser Reihenfolge von links nach rechts angeführt, auch wenn du einen Schaltkreis baust, bei dem der konkrete Aufbau anders ist und sich das Layout entsprechend unterscheidet.

Das Schaltkreissymbol für einen Arduino spiegelt diese Konventionen ebenfalls wider: VIN, 5V und 3,3V sind an der oberen Seite zu finden, GND an der unteren. Die verschiedenen Elemente für die Steuerung (RESET, AREF usw) befinden sich links, weil es sich um Eingänge für den Arduino handelt. Der Arduino besitzt zwar drei GND-Pins, im elektronischen Schaltplan wird jedoch nur einer angezeigt, weil sie elektrisch gesehen identisch sind. Da die digitalen und analogen Pins sowohl Eingänge als auch Ausgänge sein können, ist ihre Platzierung ein wenig willkürlich.

Mehr zu elektronischen Schaltplänen erfährst du in Anhang D.

Nun zurück zu unserem konkreten Projekt: In Abbildung 8-5 ist der elektronische Schaltplan für den entsprechenden Schaltkreis abgebildet. Erinnere dich an unser Ziel: Wir wollen sicherstellen, dass wir ein Relais mithilfe eines MOSFET steuern können (das fertige System umfasst zwar drei Wasserventile, aber um sicherzustellen, dass der Plan funktioniert, müssen wir nur ein Ventil überprüfen.)

Notiere die Pin-Nummern neben dem Schaltsymbol für das Relais. Sie sind sehr wichtig, weil sie angeben, welcher Pin wie verbunden ist. Es ist von grundlegender Bedeutung, dies zu verstehen, um deinen Schaltkreis korrekt zu verdrahten. Anzumerken ist außerdem, dass die Abstände zwischen den Pins nicht immer gleich sind: Die Pins 1 und 4 sind weiter voneinander entfernt als die Pins 4 und 8. Beachte auch den schwarzen Streifen oben zwischen den Pins 8 und 9. Die Pin-Nummern sind von der Unterseite gesehen dargestellt.

Zu Referenzzwecken haben wir mit Abbildung 8-6 eine Darstellung derselben Schaltung auf einer lötfreien Steckplatine, wie wir sie bisher verwendet haben, angeführt. Du kannst dir dies in Form einer bebilderten Schaltskizze vorstellen, die wir dem Schaltplan in Abbildung 8-5 gegenüberstellen.

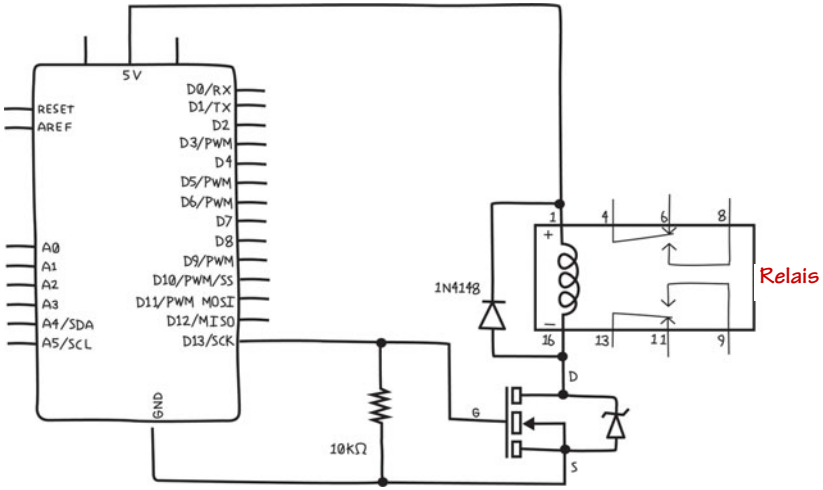


Abbildung 8-5: *Elektronischer Schaltplan zum Testen einer Arduino-Steuerung mittels eines Relais*

Wie beim Relais müssen auch die MOSFET-Pins einwandfrei bestimmt und korrekt verdrahtet werden. Der MOSFET hat eine abgerundete Seite und eine flache Seite, wie in Abbildung 8-6 zu sehen ist. Dies ist notwendig, um die korrekte Anordnung der Pins anzuzeigen. Diese Anordnung ist spezifisch für den 2N7000 und nicht unbedingt universell. Bei anderen MOSFETs unterscheidet sich die Anordnung der Pins möglicherweise. Um die Anordnung der Pins für einen bestimmten MOSFET zu ermitteln, muss immer das Datenblatt herangezogen werden.

Widme den Dioden, dem MOSFET und dem Relais die entsprechende Aufmerksamkeit: Die Dioden sind gepolt, beim MOSFET muss die flache Seite in die richtige Richtung zeigen und der Streifen des Relais muss sich am korrekten Ende befinden, damit alles funktioniert.

Wenn du dir Abbildung 8-5 und Anhang D anschaust, wirst du feststellen, dass einige Komponenten, wie Widerstände, Lichtsensoren und (einige) Kondensatoren Schaltkreissymbole besitzen, die *symmetrisch* sind, bei denen du also die eine Hälfte auf die andere klappen kannst, weil sie exakt gleich aussehen. Andere Komponenten hingegen wie LEDs, Dioden und MOSFETs sind *asymmetrisch*. Widerstände, Lichtsensoren und (einige) Kondensatoren sind nicht *gepolt*, d.h. sie arbeiten unabhängig davon, in welcher Richtung der Strom

durch sie hindurchfließt, auf die gleiche Weise. Andere Komponenten, wie LEDs und Dioden, sind *gepolt*, d.h., sie arbeiten unterschiedlich, abhängig davon, auf welchem Weg der Strom durch sie hindurchfließt. Ganz ähnlich haben MOSFET-Pins sehr spezifische Funktionen und sind darin untereinander nicht austauschbar. Dies gilt zwar nicht einheitlich, aber generell sind Komponenten, die über ein symmetrisches Schaltkreissymbol verfügen, nicht gepolt, und Komponenten mit asymmetrischen Schaltkreissymbolen sind gepolt.

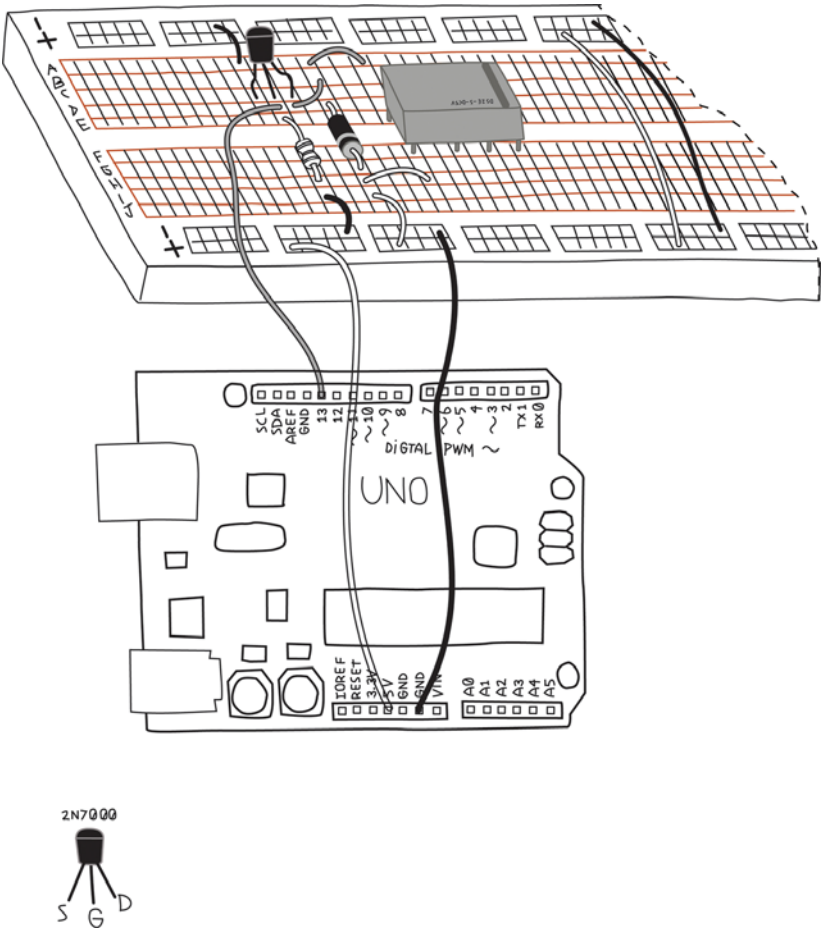


Abbildung 8-6: *Bebilderte Schaltskizze für das Testen der Arduino-Steuerung eines Relais*

Wenn der Schaltkreis aufgebaut ist, besteht der nächste Schritt darin, den Sketch zu schreiben. Für Testzwecke bevorzuge ich, falls möglich, die Nutzung der beim Arduino schon enthaltenen Beispiel-Sketche, da ich mir dann der Korrektheit des Sketches sicher sein kann. Da das Relais ein leichtes Klickgeräusch von sich gibt, wenn es aktiviert ist, solltest du das Relais einmal pro Sekunde klicken hören, wenn du das Blink-Beispiel ausführst. Du musst hier keine einzige Code-Zeile schreiben.

Überprüfe, bevor du den Sketch lädst, dass er den Pin steuert, der mit dem MOSFET verbunden ist. Wenn du dich an den Schaltplan gehalten hast, wirst du mit Pin 13 verbunden werden, bei dem es sich tatsächlich um den Pin handelt, der im Sketch für das Blink-Beispiel gesteuert wird. Die LED wird außerdem zur selben Zeit blinken, zu der das Relais klickt.

Es ist eine gute Angewohnheit, wenn du die Pin-Belegung deines Sketches checkst, bevor du ihn hochlädst. Du kannst einen perfekten Sketch geschrieben haben und du kannst eine perfekte Schaltung aufgebaut haben, aber wenn die Pin-Belegung deines Sketches nicht mit der konkreten Verdrahtung übereinstimmt, dann läuft nichts. Du wirst dann viel Zeit vergeuden, um den Fehler im Nachhinein zu finden.

Wenn du das Relais nicht klicken hörst, lies die Hinweise zur Fehlerbehebung in Kapitel 9, *Troubleshooting*, auf Seite 171. Denk daran, dass das Klicken sehr leise ist; du musst dein Ohr nahe an das Relais halten und die Umgebung muss entsprechend ruhig sein.

Nun können wir das Wasserventil hinzufügen. Das Wasserventil wird mit dem Relais und seiner eigenen Stromversorgung verbunden. Das Wasserventil und die Stromversorgung werden wahrscheinlich mit Litzendraht betrieben, wodurch es fast unmöglich wird, hier eine lötfreie Steckplatine zu benutzen. Daher ist es praktisch, ein kurzes Stück Massivdraht am Litzendraht anzubringen, wenn du mit einer lötfreien Steckplatine wie der in Abbildung 8-7 arbeitest.

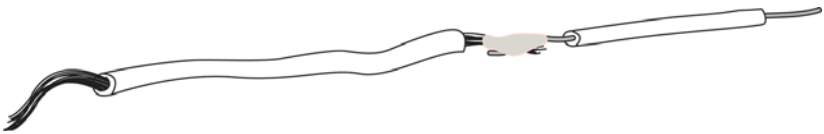


Abbildung 8-7: *Für die Arbeit mit einer lötfreien Steckplatine wurde ein kurzes Stück Massivdraht am Litzendraht angelötet.*

Du musst die Verbindungsstelle mit ein wenig Isolierband oder Schrumpfschlauch isolieren, um unerwünschte Kurzschlüsse zu verhindern.

Wann immer du ein Stück ungeschütztes Metall hast, z.B. die Drähte, die du gerade verbunden hast, oder die langen, blanken Anschlussdrähte eines Fotowiderstandes oder auch Teile, die nicht Bestandteil des Schaltkreises sind, wie eine Schraube, musst du sicherstellen, dass dieses Metall andere Teile des Schaltkreises nicht berühren und damit eine Verbindung herstellen kann, die du nicht möchtest. Dies wird als *Kurzschluss* bezeichnet und kann zu Fehlfunktionen bei deinem Projekt führen. Um einen Kurzschluss zu verhindern, isoliere immer alle ungeschützten Drähte und sichere Dinge immer so, dass sie sich nicht in unerwünschter Weise bewegen und etwas berühren können, das sie nicht berühren sollen.

Du bist wahrscheinlich schon vertraut mit Isolierband, einer gängigen, preiswerten und einfachen Methode. Eine professionellere Technik ist der Einsatz von Schrumpfschlauch. Der Schlauch wird passend abgeschnitten, über die ungeschützten Drähte oder Verbindungen gezogen und dann mit einer Heißluftpistole erhitzt, wodurch der Schlauch schrumpft und sich eng um die Verbindung legt.

An dieser Stelle wirst du vielleicht bemerken, dass du einen Weg finden musst, diese Verbindung herzustellen, wenn du das endgültige System aufbaust. Hier gibt es viele Möglichkeiten, aber Schraubklemmen von guter Qualität sind dabei eine hervorragende Wahl, siehe Abbildung 8-8.

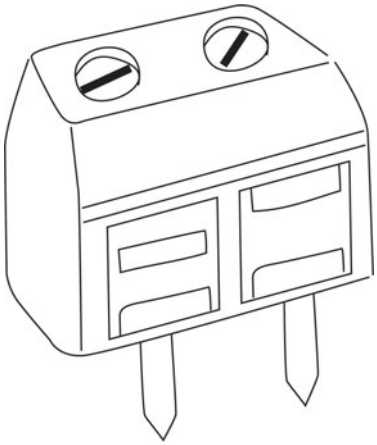


Abbildung 8-8: *Schraubklemme von hoher Qualität mit zwei Positionen*

Du musst also deiner Einkaufsliste ein weiteres Element hinzufügen. Hier die Ergänzungen, die uns zu Revision 0.2 der Einkaufsliste führen:

- vier Schraubklemmen mit zwei Positionen (ein Paar für jedes Wasserventil und ein Paar für die Stromversorgung der Wasserventile), z.B. von Jameco (*http://www.watterott.de*)

In Abbildung 8-9 wird der Schaltplan mit Ergänzung des Wasserventils und der Stromversorgung gezeigt.

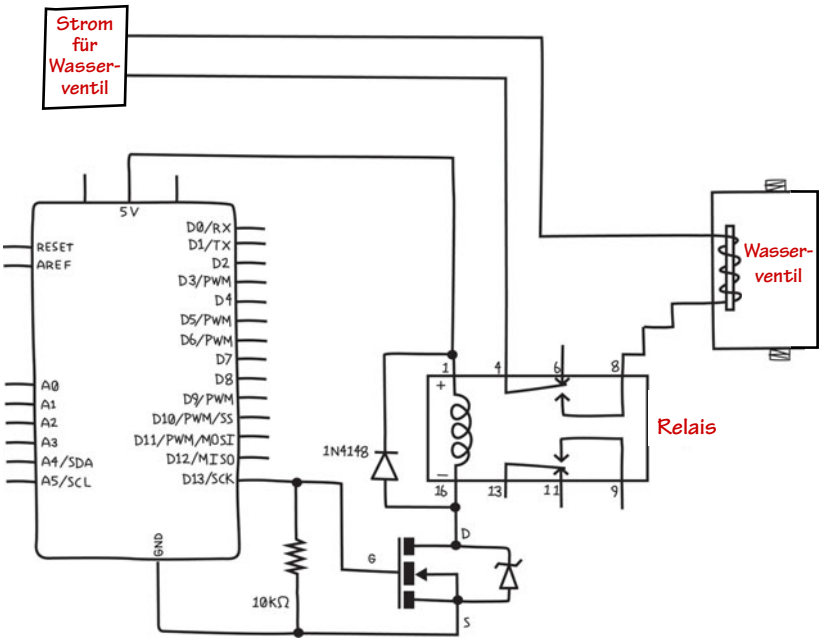


Abbildung 8-9: *Schaltskizze für das Testen der Arduino-Steuerung eines Wasserventils*

In Abbildung 8-10 siehst du die bebilderte Schaltskizze für exakt denselben Schaltkreis.

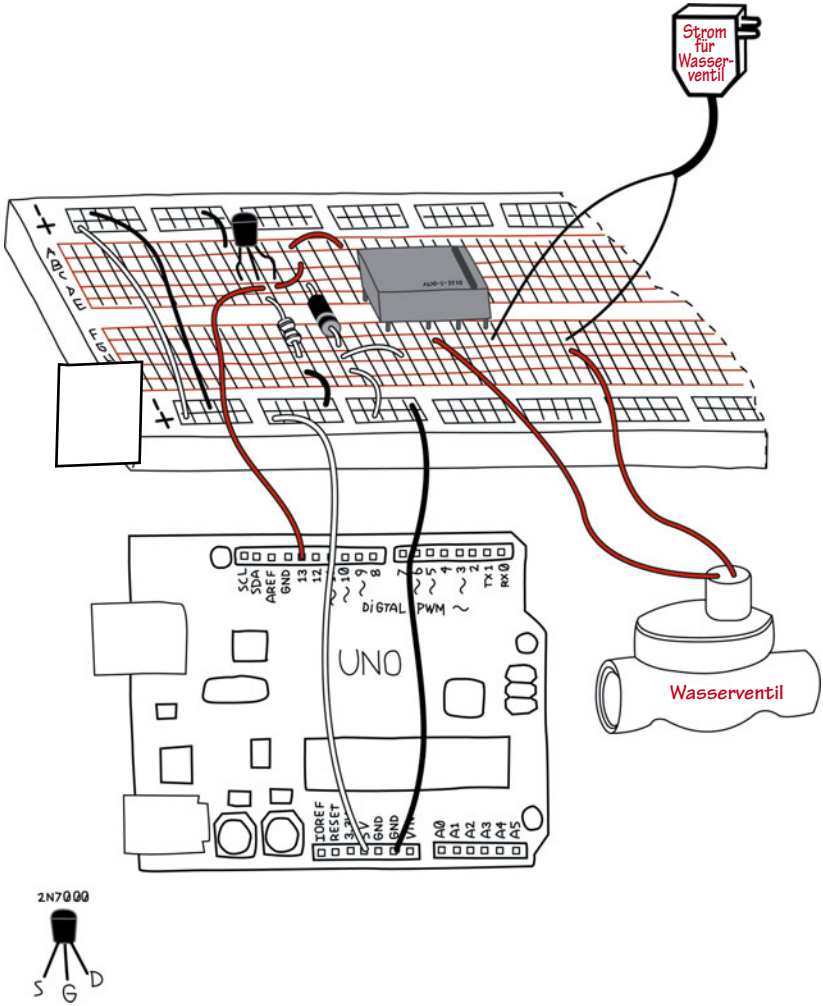


Abbildung 8-10: *Bebilderte Schaltskizze für das Testen der Arduino-Steuerung eines Wasserventils*

Die Anordnung des Wasserventils und der Stromversorgung des Wasserventils wurde in diesen beiden Ansichten geändert. Ich habe das der Übersicht wegen gemacht, so dass nun keine Leitungen quer durch das Diagramm laufen. Wenn eine Komponente über einen Schalter mit einer Stromversorgung verbunden ist, spielt die Anordnung keine Rolle, solange mit

dem Schalter gesteuert wird, ob der Stromkreis geschlossen oder offen ist.

Du kannst denselben Blink-Sketch verwenden; du solltest das Relais klicken hören. Möglicherweise hörst du das Wasserventil nicht klicken, weil Wasserventile nur arbeiten, wenn im Inneren ein entsprechender Wasserdruck vorhanden ist. Ich hatte Glück, meine Wasserventile haben ein lautes Klickgeräusch von sich gegeben, als sie eingeschaltet wurden.

Was ist mit den LEDs? Sie können an zahlreichen verschiedenen Orten installiert werden: an den digitalen Ausgängen, am MOSFET-Ausgang oder am Relais-Ausgang. Ich platziere die LED an der entferntesten Stelle, um möglichst viele Überprüfungen durchzuführen, lass sie uns also am Relais-Ausgang anbringen. (Wenn du willst, kannst du LEDs an allen genannten Stellen anbringen. Dadurch wird es sehr leicht, exakt zu ermitteln, wann das Signal endet.)

Welchen Widerstand sollten wir verwenden? Die LED erhält den Strom von der Stromversorgung für das Wasserventil. Die meisten Wasserventile sind für entweder 12 oder 24 Volt ausgelegt. Aus Sicherheitsgründen ist unser Entwurf für ein 24-Volt–System gedacht. Wenn du über ein 12-Volt-System verfügst, kannst du entweder den Wert des Widerstandes reduzieren oder deine LED ist leicht gedimmt. Sie sollte aber immer noch sichtbar leuchten.

```
LED resistor = (voltage of supply - voltage of LED)/(desired LED current)
```

Wenn du dir nicht sicher bist, welcher Wert dein Widerstand haben sollte, und er bei der maximalen Stromstärke zum Einsatz kommt, ist es immer sicherer, den Widerstand mit dem höheren Wert zu wählen. Das Licht der LED wird bei einer großen Bandbreite von Werten immer noch sichtbar sein, und wenn sie zu sehr gedimmt wird, kannst du den Widerstand immer noch reduzieren.

Die meisten LEDs sind für 2V ausgelegt und ihre Sicherheit ist unter 30 mA gegeben, so dass wir `R = (24-2)V/30mA = 733 Ohm` haben. Du kannst dies ohne Verlust an Sicherheit auf 1kOhm aufrunden; das Resultat wird eine etwas geringere Stromstärke und eine leicht gedimmte LED sein.

Aber warte, in Abschnitt *Planung*, auf Seite 95, habe ich ja angemerkt, dass einige Wasserventile Wechselstrom (AC) nutzen, und später in Abschnitt *Der elektronische Schaltplan*, auf Seite 106 habe ich erläutert, dass LEDs gepolt sind. *Gepolt* bedeutet, dass es für die LEDs von Bedeutung ist, in welche Richtung der Strom fließt, und AC bedeutet, dass der Strom ständig die Richtung wechselt. Werden dadurch die LEDs nicht beschädigt? Wie sich zeigt, können LEDs eine gewisse Spannung in die falsche Richtung aushalten. Wenn allerdings die Spannung zu hoch ist, wird die LED möglicherweise

beschädigt. Glücklicherweise können andere Dioden eine weit höhere Spannung sicher vertragen, daher wollen wir nun weitere drei 1N4148-Dioden nutzen, um die LEDs zu schützen. Mit dieser Ergänzung sind wir bei Revision 0.3 der Einkaufsliste angelangt:

- Ändere die Menge der 1N4148-Dioden oder ähnlicher Dioden auf sechs.
- Setze den Wert des LED-Widerstands auf 1kOhm.

In Abbildung 8-11 wird der Schaltplan mit der Ergänzung von LED und Diode gezeigt. Wir haben die Polung der Stromversorgung des Wasserventils und des Wasserventils selbst gekennzeichnet, die aber nur relevant ist, wenn du ein DC-System hast. Wenn du ein AC-System benutzt (was das häufigere zu sein scheint), ist keine Polung vorhanden.

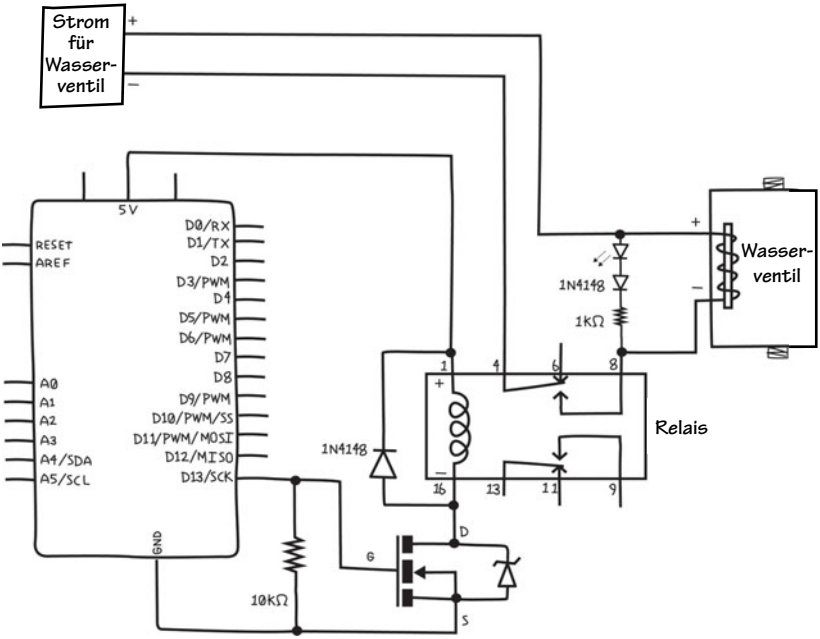


Abbildung 8-11: *Schaltplan für das Testen des Wasserventils mit ergänzter LED*

In Abbildung 8-12 siehst du die bebilderte Schaltskizze. Ich habe die Widerstandswerte ausgelassen, ziehe also den Schaltplan zum Vergleich heran, um den korrekten Widerstand an der richtigen Stelle zu platzieren. Beachte die Polung der LED und der Diode – die Anode der LED ist mit dem Wasserventil verbunden und die Anode der Diode mit der Kathode der LED.

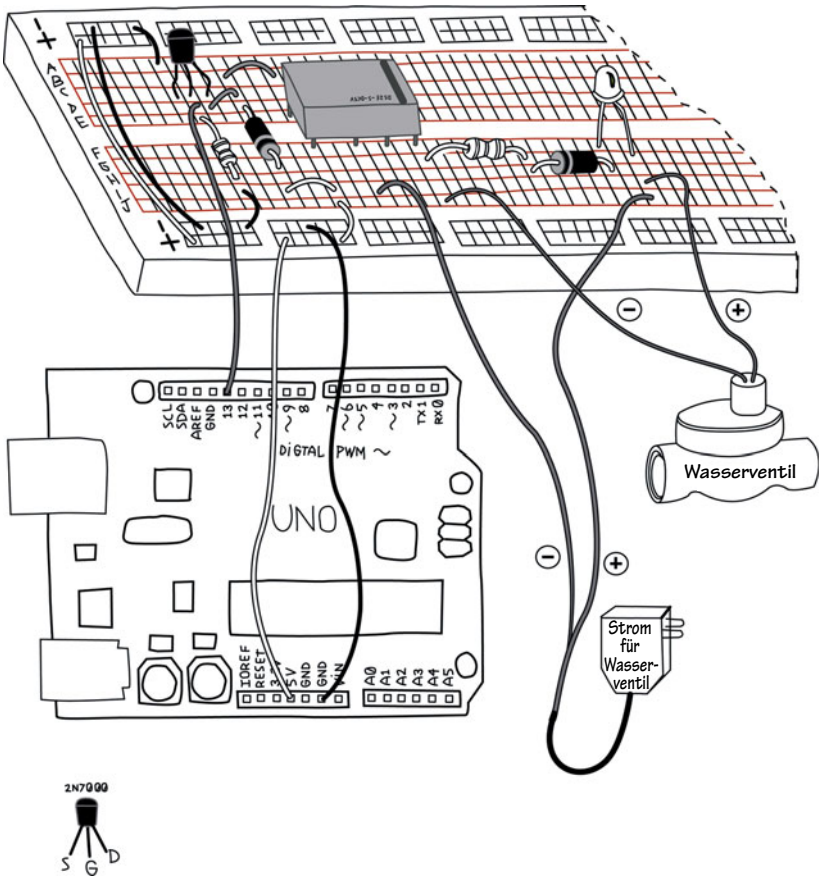


Abbildung 8-12: *Bebilderte Schaltskizze für das Testen des Wasserventils mit ergänzter LED*

Bevor du die Stromversorgung für das Wasserventil anschließt, überprüfe noch einmal die Verdrahtung, insbesondere die Relais-Verbindung. Die Spannung aus der Stromversorgung des Wasserventils sollte nicht in den Arduino gelangen, da sie fast unweigerlich das Board beschädigen würde. Führe einmal mehr den Blink-Sketch aus. Nun wirst du nicht nur das Relais und möglicherweise das Wasserventil klicken hören, sondern auch die LED aufleuchten sehen.

Nachdem wir die Herausforderungen des Relais und des Wasserventils gemeistert haben, wollen wir nun den Temperatur- und Feuchtigkeitssensor testen.

Testen des Temperatur- und Feuchtigkeitssensors

Der DHT11 ist ein gängiger Temperatur- und Feuchtigkeitssensor. Wie die RTC ist er preisgünstig und lässt sich auf einfache Weise mit dem Arduino benutzen. Gemäß dem Datenblatt wird der DHT11 wie in Abbildung 8-13 dargestellt angeschlossen. Beachte den Pull-up-Widerstand am Daten-Pin.

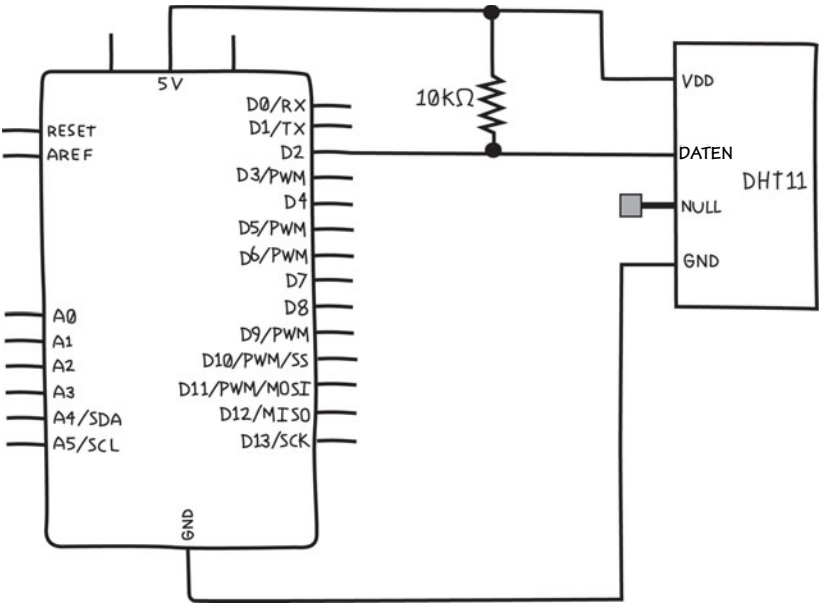


Abbildung 8-13: *Schaltplan für das Testen des Temperatur- und Feuchtigkeitssensors DHT11*

Da wir eine Komponente hinzufügen, für die ein Pull-up-Widerstand erforderlich ist, erweitern wir unsere Einkaufsliste um einen weiteren 10kOhm-Widerstand. Wir sind nun bei Version 0.4 angelangt:

- Füge einen 10kOhm-Widerstand hinzu (für den Temperatur- und Feuchtigkeitssensor)

Wegen des Pull-up-Widerstands können wir nicht den Trick anwenden, der bei der RTC zum Einsatz kam (die wir ja direkt auf dem Arduino untergebracht haben), sondern wir müssen den Sensor auf der Steckplatine anbringen (Abbildung 8-14).

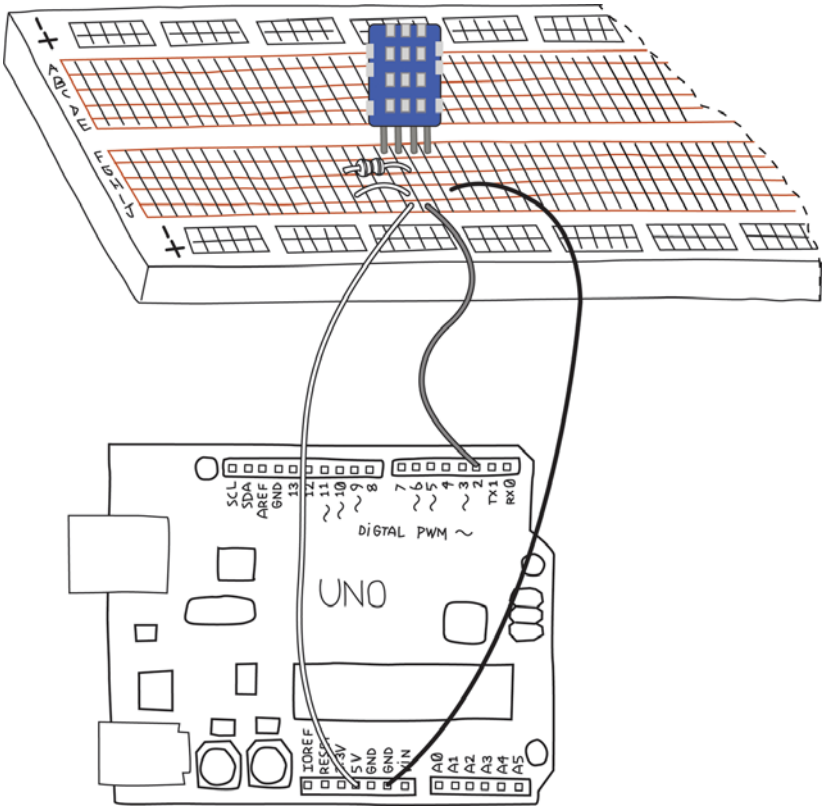


Abbildung 8-14: *Bebilderte Schaltskizze für das Testen des Temperatur- und Feuchtigkeitssensors DHT11*

Die Schaltskizze ist dieselbe, unabhängig davon, ob die Schaltung auf einer Steckplatine aufgebaut wird oder in irgendeiner anderen Weise realisiert wird.

Du kannst die englischsprachige DHT11-Bibliothek von GitHub (*http://bit.ly/1trMqke*) herunterladen.

Klick' wie bei der RTC-Bibliothek auf die Schaltfläche Download ZIP, und entpacke diese Datei im Ordner *libraries* in deinem Arduino-Sketch-Ordner. Die Bibliothek wird in einem Ordner namens *DHT-sensor-library-master* erstellt, den du in *DHT* umbenennen musst. Denk daran, dass ein Neustart der Arduino-IDE erforderlich ist, wenn sie geöffnet ist, damit die neue Bibliothek er-

kannt wird. Überprüfe, ob die Bibliothek korrekt installiert wurde, indem du das Beispiel *DHTtester* der Beispiele aus dem DHT-Verzeichnis öffnest und dann auf die Schaltfläche Verify klickst (siehe Abbildung 4-3). Wenn du die Meldung Done compiling erhältst, wurde die Bibliothek korrekt installiert.

Bevor du den Sketch auf deinen Arduino lädst, beachte, dass von dem Beispiel drei verschiedene Modelle des DHT-Sensors unterstützt werden: DHT11, DHT21 und DHT22. Für die korrekte Auswahl des richtigen Modells wird eine Konstante namens DHTTYPE definiert, die entweder DHT11, DHT21 oder DHT22 ist:

```
// Uncomment whatever type you're using!
//#define DHTTYPE DHT11   // DHT 11
#define DHTTYPE DHT22   // DHT 22  (AM2302)
//#define DHTTYPE DHT21   // DHT 21 (AM2301)
```

Beachte, dass DHT11 und DHT21 ignoriert werden, da es sich bei diesen Zeilen um Kommentare handelt. Programmierer sagen, diese Zeilen werden *auskommentiert*. Da du einen DHT11-Sensor nutzt, musst du die Zeile mit dem DHT22 *auskommentieren* und die Kommentierung der Zeile mit dem DHT11 aufheben:

```
// Uncomment whatever type you're using!
#define DHTTYPE DHT11   // DHT 11
//#define DHTTYPE DHT22   // DHT 22  (AM2302)
//#define DHTTYPE DHT21   // DHT 21 (AM2301)
```

Die Zeilen mit dem DHT22 und dem DHT21 erfüllen keine praktische Funktion, aber sie erinnern uns daran, dass die Bibliothek mit diesen drei Sensoren arbeitet und dass du an dieser Stelle festlegst, welchen Sensor du verwendest.

Möglicherweise hast du noch einen weiteren Konstantentyp entdeckt: die *konstante Variable*. Abgesehen von ihrem ungelenken Namen ist auch sie sehr wichtig und nützlich.

Die Unterschiede zwischen einer *Variablen* (wie einem Integer), einer *konstanten Variablen* und einem *konstanten Wert* (*named constant value*) sind sehr subtil und ein wenig kompliziert. In groben Zügen lässt sich sagen, dass eine *konstante Variable* einen winzig kleinen Teil deines Arduino-Speichers benutzt und *Regeln des Geltungsbereiches* befolgt. Im Gegensatz dazu benutzt ein *konstanter Wert* keinerlei Speicher und besitzt immer einen globalen Wertebereich.

Als allgemeine Regel solltest du den Einsatz von *konstanten Werten* (*named constant values*) vermeiden, du solltest sie nur benutzen, wenn eine Bibliothek sie erfordert.

Auf der Arduino-Website erfährst du mehr über *named constant values* (konstanter Wert) (*http://bit.ly/1uUiDpC*), das Schlüsselwort `const` *http://bit.ly/1vjSHDK* und *Wertebereiche von Variablen* (*http://bit.ly/1zqEKmM*).

Wenn du den korrekten Typ des DHT-Sensors festgelegt hast, überprüfe anschließend, ob im Sketch auch derselbe Pin genutzt wird, der mit dem Sensor verbunden ist. Lade das Beispiel *DHTtester* auf deinen Arduino und öffne den Serial Monitor. Du solltest in etwa Folgendes sehen:

```
DHTxx test!
Humidity: 47.00 % Temperature: 24.00 *C 75.20 *F Heat index: 77.70 *F
Humidity: 48.00 % Temperature: 24.00 *C 75.20 *F Heat index: 77.71 *F
```

Du kannst den Feuchtigkeitssensor überprüfen, indem du ihn leicht anhauchst. Durch die Feuchtigkeit in deiner Atemluft sollte die wahrgenommene Feuchtigkeit beim Sensor ansteigen. Du kannst versuchen, die Temperatur ansteigen zu lassen, indem du deine Finger um den Sensor legst. Da du aber das Plastikgehäuse und nicht den Sensor selbst berührst, wird die Temperatur möglicherweise nicht deutlich ansteigen.

Wenn du nun Vertrauen in die Funktionstüchtigkeit deiner Komponenten hast, kannst du mit dem Software-Design beginnen.

Programmierung

Du wirst es nicht glauben, aber das Schreiben von Code (*Programmierung*) erfordert ebenfalls Planung. Du musst ein wenig darüber nachdenken, was du tun möchtest, bevor du den Code herunterschreibst. Auf ähnliche Weise, wie du die neuen Elektronikbausteine getestet hast, bevor du das komplette Design aufgebaut hast, wirst du auch jeden Code-Abschnitt testen, bevor du fortfährst. Je weniger Code vorhanden ist, umso leichter lassen sich Probleme finden.

Die An- und Aus-Zeiten bestimmen

Wir möchten die Wasserventile zu verschiedenen Tageszeiten ein- und ausschalten. Wir brauchen eine Möglichkeit, die entsprechenden Werte aufzuzeichnen. Da wir drei Ventile haben, könnten wir ein Array mit einem Eintrag für jedes Ventil benutzen. Dadurch wird es auch leichter, später weitere Ventile hinzuzufügen, falls gewünscht. Vielleicht erinnerst du dich daran, dass wir ein Array mit der Bezeichnung `buffer` (siehe Beispiel 7-2 in Kapitel 7, *Kommunikation mit der Cloud*, auf Seite 81) verwendet haben, um die Zeichen zu speichern, die vom Processing-Sketch gesendet wurden. Arrays werden kurz unter Abschnitt *Variablen*, auf Seite 192 beschrieben.

Du kannst z.B. Folgendes ausprobieren:

```
const int NUMBEROFVALVES = 3;
const int NUMBEROFTIMES = 2;

int onOffTimes [NUMBEROFVALVES][NUMBEROFTIMES];
```

Der Einfachheit halber gehen wir hier davon aus, dass das Wasser an jedem Wochentag zur gleichen Zeit auf- oder abgedreht werden soll. Wenn deine Programmierfähigkeiten entsprechend wachsen, kannst du den Sketch an verschiedene Terminpläne für unterschiedliche Wochentagen und sogar für mehrere Male an einem Tag anpassen. Wenn du mit einem Projekt beginnst, tust du gut daran, mit dem einfachsten denkbaren System zu starten und Features hinzuzufügen, wenn du sichergestellt hast, dass alles andere einwandfrei funktioniert.

Beachte, dass wir keine festen Zahlen für die Größe des Arrays verwenden, sondern erst zwei konstante Variablen erstellt haben. Dies erinnert uns daran, was diese Zahlen bedeuten. Außerdem können wir sie auf diese Weise später leichter ändern. Namen von konstanten Variablen werden in Großbuchstaben geschrieben, um uns daran zu erinnern, dass es sich um Konstanten handelt.

Falls du noch niemals ein zweidimensionales Array gesehen hat, ist das kein Grund zur Panik. Denk einfach an eine Tabelle. Die Zahl in der ersten [ ] ist die Anzahl der Zeilen, bei der Zahl in der zweiten [ ] handelt es sich um Anzahl der Spalten. Eine Spalte repräsentiert ein Ventil. Wir verwenden die erste Spalte, um die Zeit zu speichern, zu der das Ventil eingeschaltet werden soll, und die zweite Spalte, um die Zeit zu speichern, zu der das Ventil ausgeschaltet werden soll.

Wir wollen auch konstante Variablen für die Zeilennummern erstellen. Denk daran, dass der Index für Elemente innerhalb eines Arrays bei 0 beginnt:

```
const int ONTIME = 0;
const int OFFTIME = 1;
```

Als Nächstes benötigen wir eine Möglichkeit, diese Informationen als Eingang zu nutzen, d. h. eine Art *Benutzerschnittstelle*. Normalerweise handelt es sich bei einer Benutzerschnittstelle um ein Menü, wir machen aber etwas extrem Einfaches und benutzen den Serial Monitor.

Erinnerst du dich, dass wir dem Arduino in Kapitel 7, *Kommunikation mit der Cloud*, auf Seite 81 mitteilen mussten, welche Farbe das Licht haben soll? Weil der Arduino ein einfaches Gerät ist, haben wir einen simplen Weg gewählt, um die Farbe zu programmieren.

Wir werden hier ähnlich vorgehen – nämlich die Zeiten auf eine möglichst simple Weise programmieren.

Wir müssen in der Lage sein, die EIN-Zeit und die AUS-Zeit für jedes Ventil festzulegen. Wir können eine Zahl verwenden, die das gewünschte Ventil angibt, gefolgt von dem Buchstaben *N* für on und *F* für off, auf den wiederum die Zeit folgt. Wir geben die Zeit in der 24-Stunden-Schreibweise ein, d.h. 01345 für 13.45 Uhr. Wir würden also

2N1345 2F1415

schreiben, um das Ventil 2 um 13.45 Uhr einzuschalten und um 14.15 Uhr auszuschalten.

Um uns das Leben zu erleichtern, verwenden wir konsequent immer Großbuchstaben für *N* und *F*.

In unserem Code würden wir den String, den wir in die korrekten Gruppen schreiben, *parsen* oder separieren.

Eine Gruppe aufeinanderfolgender Zeichen wird als *String* bezeichnet.

Wenn du dir den Arduino-Sketch anschaust (Beispiel 7-2), siehst du, dass wir `Serial.available()` und `Serial.read()` benutzen. Hierbei handelt es sich um Funktionen des seriellen Objektes. Wie sich zeigt, hat das serielle Objekt mehr Funktionen, als bei Arduino (*http://bit.ly/1uCgj1v*) beschrieben sind.

Wir werden die Funktion `Serial.parseInt()` verwenden, die automatisch jede Eingabe liest und in eine Zahl umwandelt. Sie stoppt, wenn ein Zeichen auftaucht, bei dem es sich nicht um eine Ziffer handelt. Die Buchstaben (*N* oder *F*) werden wir direkt mit der Funktion `Serial.read()` lesen.

Zu Testzwecken drucken wir einfach nachdem jede Zeile gelesen wurde, das gesamte Array aus, wie in Beispiel 8-1 dargestellt.

Beispiel 8-1: *Parsen der an das Bewässerungssystem gesendeten Befehle*

```
const int NUMBEROFVALVES = 3;
const int NUMBEROFTIMES = 2;

int onOffTimes [NUMBEROFVALVES][NUMBEROFTIMES];

const int ONTIME = 0;
const int OFFTIME = 1;

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

```
void loop() {
  // Read a string of the form "2N1345" and separate it
  // into the first digit, the letter, and the second number

  // read only if there is something to read
  while (Serial.available() > 0) {

    // The first integer should be the valve number
    int valveNumber = Serial.parseInt();

    // the next character should be either N or F
    // do it again:
    char onOff = Serial.read();

    // next should come the time
    int desiredTime = Serial.parseInt();
      //Serial.print("time = ");
      //Serial.println(desiredTime);

    // finally expect a newline which is the end of
    // the sentence:
    if (Serial.read() == '\n') {
      if ( onOff == 'N') { // it's an ON time
        onOffTimes[valveNumber][ONTIME] = desiredTime;
      }
      else if ( onOff == 'F') { // it's an OFF time
        onOffTimes[valveNumber][OFFTIME] = desiredTime;
      }
      else { // something's wrong
        Serial.println ("You must use upper case N or F only");
      }
    } // end of sentence
    else {
      Serial.println("no Newline character found"); // Sanity check
    }

    // now print the entire array so we can see if it works
    for (int valve = 0; valve < NUMBEROFVALVES; valve++) {
      Serial.print("valve # ");
      Serial.print(valve);
      Serial.print(" will turn ON at ");
      Serial.print(onOffTimes[valve][ONTIME]);
      Serial.print(" and will turn OFF at ");
      Serial.print(onOffTimes[valve][OFFTIME]);
      Serial.println();
    }
  } // end of Serial.available()
}
```

Du kannst diesen Sketch von der Katalogseite zu diesem Buch (*http://bit.ly/start_arduino_3e*) herunterladen.

Wenn du den Sketch in den Arduino geladen hast, öffne den Serial Monitor und markiere die Kontrollkästchen für die Baudrate und das Zeilenende

in der unteren rechten Ecke des Serial Monitor. Wähle Newline und 9600 Baud. Durch den Zeilenendetyp wird sichergestellt, dass jedes Mal, wenn du eine Zeile durch Drücken der Eingabetaste auf deinem Computer beendest, dein Computer ein Zeilenvorschubzeichen an den Arduino sendet.

Wenn beispielsweise Ventil Nummer 1 (#1) um 13.30 Uhr eingeschaltet werden soll, gib **`1N1330`** ein und drücke dann die Eingabetaste. Du solltest Folgendes sehen:

```
valve # 0 will turn ON at 0 and will turn OFF at 0
valve # 1 will turn ON at 1330 and will turn OFF at 0
valve # 2 will turn ON at 0 and will turn OFF at 0
```

Den Sketch habe ich dahingehend überprüft, dass es sich bei dem Zeichen zwischen den Zahlen entweder um *N* oder *F* handelt und dass nach der zweiten Zahl ein Zeilenvorschubzeichen folgt. Dieser Gesundheitscheck ist praktisch, um Fehler abzufangen, die möglicherweise bei der Eingabe entstehen und dein Programm durcheinanderbringen. Diese Art Check ist außerdem nützlich, um Fehler in deinem Programm zu entdecken. Es sind auch andere Gesundheitschecks denkbar; vielleicht möchtest du überprüfen, dass die Zeit gültig ist, d.h, dass sie unter 2359 liegt, und dass die Nummer des Ventils kleiner als `NUMBEROFVALVES` ist.

Ein Programm, das so entworfen wurde, dass es nur mit korrekten Daten arbeiten kann, ist sehr empfindlich, weil es sehr anfällig für Fehler ist, seien dies nun Fehler bei der Benutzereingabe oder an anderen Stellen. Durch die Überprüfung der Daten (bevor mit ihnen gearbeitet wird) wird dein Programm Fehler erkennen können, anstatt zu versuchen, die fehlerhaften Daten zu verwenden, was zu unerwartetem und fehlerhaftem Verhalten führen kann. Dadurch wird dein Programm robust, was eine höchst wünschenswerte Eigenschaft ist, besonders weil Menschen nicht immer das tun, was man erwarten könnte.

Bevor wir fortfahren, möchte ich dir einen Trick zeigen. Der Code, den wir bisher entwickelt haben, ist sehr lang, und wir müssen immer noch viel hinzufügen. Es wird langsam schwierig, das Programm zu lesen und zu verwalten.

Glücklicherweise können wir von einer cleveren und gängigen Programmiertechnik Gebrauch machen. In Abschnitt *Reich mir den Parmesan*, auf Seite 29 haben wir erläutert, was eine Funktion ist und dass es sich bei `setup()` und `loop()` um Funktionen handelt, von denen der Arduino erwartet, dass sie vorhanden sind. Bisher hast du dein ganzes Programm auf diesen beiden Funktionen aufgebaut.

Was ich bisher noch nicht betont habe, ist die Tatsache, dass du auf dieselbe Weise wie `setup()` und `loop()` auch andere Funktionen erstellen kannst.

Warum ist das so wichtig? Weil es ein sehr komfortabler Weg ist, ein langes, kompliziertes Programm in kleine Funktionen aufzuspalten, von denen jede eine bestimmte Aufgabe erfüllt. Außerdem kannst du diese Funktionen beliebig benennen. Wenn du ihnen also Namen gibst, die beschreiben, was sie tun, wird das Lesen des Programms erheblich einfacher.

Immer wenn Code, der eine bestimmte Aufgabe erfüllt, recht umfangreich wird, ist er ein geeigneter Kandidat, um aus ihm eine Funktion zu machen. Was aber ist groß genug? Das ist völlig dir überlassen. Meine Daumenregel: Sobald Code mehr als zwei Bildschirmseiten umfasst, ist er ein Kandidat, den ich in eine Funktion umwandle. Ich kann zwei Bildschirmseiten Code im Kopf behalten, mehr nicht.

Eine wichtige Überlegung ist die, ob der Code leicht extrahiert werden kann. Ist er von vielen Variablen abhängig, die nur innerhalb anderer Funktionen sichtbar sind? Wenn du mehr über den *Wertebereich von Variablen* erfährst, wirst du sehen, dass auch dies ein wichtiger Aspekt ist.

Der Code beispielsweise, den wir gerade entwickelt haben, liest Befehle vom Serial Monitor, parst sie, und speichert dann die Zeiten im Array. Wenn wir daraus eine Funktion mit dem Namen `expectValveSetting()` machen, ist unsere Schleife recht einfach:

```
void loop() {

    expectValveSettings();

}
```

Dies ist viel leichter zu lesen und – am wichtigsten – leichter verständlich, wenn wir den Rest des Programms entwickeln.

Natürlich müssen wir die entsprechende Funktion erstellen, die dann so aussieht:

```
void expectValveSettings() {
  // Read a string of the form "2N1345" and separate it
  // into the first digit, the letter, and the second number

  // read only if there is something to read
  while (Serial.available() > 0) {
        // ... rest of the code not repeated here
         }
```

Der übrige Code ist derselbe wie in Beispiel 8-1. Ich habe den Rest der Funktion weggelassen, weil ich nicht zwei zusätzliche Seiten verschwenden möchte.

Nun können wir uns den anderen Dingen zuwenden, die noch zu erledigen sind, und dafür sorgen, dass auch sie funktionieren.

Überprüfen, ob es an der Zeit ist, ein Ventil an- oder auszuschalten

Als Nächstes werfen wir einen Blick auf die Daten von der RTC und überlegen, wie wir sie für die Entscheidung, etwas ein- oder auszuschalten, verwenden werden. Wenn du zurück zum RTC-Beispiel *ds1307* blätterst, wirst du sehen, wie die Zeit ausgegeben wird:

```
Serial.print(now.hour(), DEC);
```

Komfortablerweise handelt es sich dabei bereits um eine Zahl, so dass der Vergleich mit den von uns gespeicherten Stunden und Minuten recht einfach sein wird.

Um auf die RTC zugreifen zu können, musst du dein Programm mit Teilen aus dem *ds1307*-Beispiel ergänzen. Füge am Anfang vor `setup()` Folgendes hinzu:

```
#include <Wire.h>
#include "RTClib.h"

RTC_DS1307 rtc;
```

Dieses Mal verwenden wir nicht die analogen Eingangs-Pins, um 5V und Masse bereitzustellen. Warum? Weil analoge Eingangs-Pins knapp sind; es sind nur sechs vorhanden, und wir haben bereits zwei für die 12C-Schnittstelle mit der RTC verloren. Aktuell sind für unser Projekt keine analogen Eingangs-Pins erforderlich, wir müssen aber daran denken, dass sich dies später ändern kann.

In `setup()` ist folgender Code erforderlich:

```
#ifdef AVR
        Wire.begin();
    #else
        Wire1.begin(); // Shield I2C pins connect to alt I2C bus
                       // on Arduino Due
    #endif
        rtc.begin();
```

Überlege nun, was zu tun ist: Solange die aktuelle Zeit größer ist als die Zeit, zu der das Ventil eingeschaltet werden soll, und kleiner als die Zeit, zu der das Ventil ausgeschaltet werden soll, soll das Ventil eingeschaltet sein. Zu allen anderen Zeiten soll es ausgeschaltet sein.

Von der RTC-Bibliothek kannst du die aktuelle Zeit wie folgt erhalten:

```
dateTimeNow = rtc.now();
```

Dann kannst du folgendermaßen auf Bestandteile der Zeit zugreifen:

```
dateTimeNow.hour()
dateTimeNow.minute()
```

Kannst du das Problem erkennen? Wir haben die Zeit in einem Format mit vier Ziffern gespeichert, wobei die ersten zwei Zahlen die Stunde und die letzten zwei Ziffern die Minuten wiedergeben. Wir können keinen mathematischen Vergleich vornehmen, ohne diese Zahl in Stunden und Minuten aufzuteilen, und dann wird der Vergleich kompliziert.

Es wäre doch ganz nett, wenn wir nur eine Zahl hätten. Dies können wir erreichen, indem wir die Stunden einfach in Minuten konvertieren und die Minuten seit Mitternacht speichern, statt die Zeit in Stunden und Minuten zu speichern. Auf diesem Weg müssen wir mit nur einer Zahl arbeiten, und der mathematische Vergleich ist einfach. (Wir müssen daran denken, das Wasser nie vor Mitternacht aufzudrehen und nach Mitternacht abzudrehen. Dies wäre eine weitere Gelegenheit für einen Gesundheitscheck, um unser Programm robuster zu gestalten.)

Der folgende Code veranschaulicht dies:

```
int nowMinutesSinceMidnight = (dateTimeNow.hour() * 60) +
    dateTimeNow.minute();
```

Der Vergleich sieht dann so aus:

```
if ( ( nowMinutesSinceMidnight >= onOffTimes[valve][ONTIME]) &
        ( nowMinutesSinceMidnight < onOffTimes[valve][OFFTIME]) ) {
            digitalWrite(??, HIGH);
    }
    else {
            digitalWrite(??, LOW);
    }
```

Warte eine Minute, was ist mit dem Fragezeichen? Wir müssen die Pin-Nummer jedes Ventils kennen. Unsere `for()`-Schleife zählt nur die Ventile ab: 0, 1 und 2. Wir benötigen einen Weg, um anzuzeigen, welche Pin-Nummer mit welchem Ventil verbunden ist. Wir können hierfür ein Array verwenden:

```
int valvePinNumbers[NUMBEROFVALVES];
```

Wenn wir dieselbe konstante Variable benutzen, die wir vorher erstellt haben, wird dieses Array exakt dieselbe Anzahl Zeilen aufweisen wie das andere Array, auch wenn du später die Anzahl der Ventile änderst.

In `setup()` fügen wir die korrekten Pin-Nummern in das Array ein:

```
valvePinNumbers[0] = 6; // valve 0 is on pin 6
valvePinNumbers[1] = 8; // valve 1 is on pin 8
valvePinNumbers[2] = 3; // valve 2 is on pin 3
```

Immer wenn Informationen gesucht werden sollen, die auf einem Index basieren, bietet ein Array hierfür eine gute Möglichkeit. Du kannst es dir als *Nachschlagetabelle* vorstellen.

Nun können wir unsere Fragezeichen auflösen:

```
if ( ( now.hour() > onOffTimes[valve][onTime]) &
     ( now.hour() < onOffTimes[valve][offTime]) ) {

    Serial.println("Turning valve ON");
    digitalWrite(valvePinNumbers[valve], HIGH);
}
else {
    Serial.println("Turning valve OFF");
    digitalWrite([valve], LOW);
}
```

Eine letzte Sache: Wir werden das Zahlenformat mit den vier Ziffern im Array in Stunden und Minuten auflösen müssen. Möglicherweise ist es einfacher, dies dann zu tun, wenn der Nutzer die Informationen eingibt. Wir fordern den Benutzer auf, einen : zwischen den Stunden und Minuten einzufügen, wir lesen sie als separate Zahlen, nehmen die Konvertierung in Minuten seit Mitternacht direkt hier vor und speichern die Minuten seit Mitternacht im Array, wie in Beispiel 8-2 gezeigt wurde.

Du hast wahrscheinlich schon über zwei oder drei verschiedene Wege nachgedacht, dies zu realisieren. Die meisten Probleme beim Programmieren, eigentlich die meisten technischen Probleme, können auf ganz unterschiedliche Weise gelöst werden. Ein professioneller Programmierer denkt möglicherweise über Aspekte wie Effizienz, Geschwindigkeit, Speicher, Verwendung und vielleicht über die Kosten nach, aber als Einsteiger solltest du immer den Weg wählen, der für dich am leichtesten verständlich ist.

Beispiel 8-2: *Die expectValveSetting()-Funktion*

```
/*
 * Read a string of the form "2N13:45" and separate it
 * into the valve number, the letter indicating ON or OFF,
 * and the time
 */

void expectValveSetting() {

  // The first integer should be the valve number
  int valveNumber = Serial.parseInt();
```

```
  // the next character should be either N or F
  char onOff = Serial.read();

  // next should come the hour
  int desiredHour = Serial.parseInt();

  // the next character should be ':'
  if (Serial.read() != ':') {
    Serial.println("no : found"); // Sanity check
    Serial.flush();
    return;
  }

  // next should come the minutes
  int desiredMinutes = Serial.parseInt();

  // finally expect a newline which is the end of
  // the sentence:
  if (Serial.read() != '\n') { // Sanity check
    Serial.println(
      "Make sure to end your request with a Newline");
    Serial.flush();
    return;
  }

  // Convert the desired hour and minute time
  // to the number of minutes since midnight
  int desiredMinutesSinceMidnight
    = (desiredHour*60 + desiredMinutes);

  // Now that we have all the information set it into the array
  // in the correct row and column

  if ( onOff == 'N') { // it's an ON time
    onOffTimes[valveNumber][ONTIME]
    = desiredMinutesSinceMidnight;
  }
  else if ( onOff == 'F') { // it's an OFF time
    onOffTimes[valveNumber][OFFTIME]
    = desiredMinutesSinceMidnight;
  }
  else { // user didn't use N or F
    Serial.print("You must use upper case N or F ");
    Serial.println("to indicate ON time or OFF time");
    Serial.flush();
    return;
  }

  // now print the entire array so user can see what they set
  printSettings();
} // end of expectValveSetting()


void printSettings(){
```

```
  // Print the current on and off settings, converting the
  // number of minutes since midnight back to the time
  // in hours and minutes

  Serial.println();

  for (int valve = 0; valve < NUMBEROFVALVES; valve++) {
    Serial.print("Valve ");
    Serial.print(valve);
    Serial.print(" will turn ON at ");

    // integer division by 60 gives the hours
    // since integer division drops any remainder
    Serial.print((onOffTimes[valve][ONTIME])/60);

    Serial.print(":");

    // the minutes are the remainder after dividing by 60.
    // get the remainder with the modulo (%) operator
    Serial.print((onOffTimes[valve][ONTIME])%(60));

    Serial.print(" and will turn OFF at ");
    Serial.print((onOffTimes[valve][OFFTIME])/60); // hours
    Serial.print(":");
    Serial.print((onOffTimes[valve][OFFTIME])%(60)); // minutes
    Serial.println();
  }
}
```

Regen feststellen

Wie können wir mit dem Feuchtigkeitssensor Regen erfassen? Dies kann gleichzeitig mit dem Ermitteln der Zeit erfolgen, aber daraus wird sich eine sehr lange Zeile ergeben. Es ist in Ordnung, eine weitere `if()`-Anweisung zu benutzen; erfahrene Programmierer werden dir vielleicht sagen, dass dies weniger effizient ist, aber deinem Garten ist es egal, ob das Wasser den Bruchteil einer Sekunde später kommt. Viel wichtiger ist, dass du das Programm lesen kannst und auch verstehst.

Möglicherweise gibt es erfahrene Programmierer, die dir in bester Absicht clevere Tricks zeigen möchten, mit denen du die Anzahl der Zeilen deines Programms reduzieren oder seine Effizienz steigern kannst. Wenn du mehr Erfahrung gesammelt hast, ist es gut, zu verstehen, warum diese Tricks funktionieren, als Einsteiger solltest du aber immer den Weg wählen, der für dich der verständlichste ist.

In Beispiel 8-3 wird eine Möglichkeit gezeigt, das Wasser nur dann anzustellen, wenn es nicht regnet.

Beispiel 8-3: *Das Wasser nur dann anstellen, wenn es nicht regnet*

```
if ( ( nowMinutesSinceMidnight >= onOffTimes[valve][ONTIME]) &
      ( nowMinutesSinceMidnight < onOffTimes[valve][OFFTIME]) ) {
      // Before we turn a valve on make sure it's not raining
      if ( humidityNow > 50 ) { // Arbitrary; adjust as necessary
        // It's raining; turn the valve OFF
        Serial.print(" OFF ");
        digitalWrite(valvePinNumbers[valve], LOW);
      }
      else {
        // No rain and it's time to turn the valve ON
        Serial.print(" ON ");
        digitalWrite(valvePinNumbers[valve], HIGH);
      } // end of checking for rain
 } // end of checking for time to turn valve ON
    else {
      Serial.print(" OFF ");
      digitalWrite(valvePinNumbers[valve], LOW);
    }
```

Natürlich werden wir für diesen Zweck eine weitere Funktion schreiben. Wir wollen sie `checkTimeControlValves()` nennen.

Außerdem schreiben wir eine separate Funktion für das Auslesen des Feuchtigkeitssensors und der RTC. Ihr wollen wir den Namen `getTimeTempHumidity()` geben.

Unsere Schleife sieht nun ungefähr so aus:

```
    void loop() {

      // Remind user briefly of possible commands
      Serial.println("Type 'P' to print settings or 'S2N13:45'");
      Serial.println(" to set valve 2 ON time to 13:34");

      // Get (and print) the current date, time, temperature, and humidity
      getTimeTempHumidity();

      // Check for new time settings:
      expectValveSettings();

      // Check to see whether it's time to turn any valve ON or OFF
      checkTimeControlValves();
      // No need to do this too frequently
      delay(5000);
    }
```

Zusammenfügen aller Teile

Der Sketch ist fast fertig. Wir müssen uns nur noch mit ein paar geringfügigeren Details befassen, dann können wir alles zusammenfügen.

Zunächst einmal wäre da die Frage, wie wir vorgehen, wenn der Nutzer den aktuellen Zeitplan für das Ventil sehen möchte. Das ist recht einfach, aber wie kann der Nutzer uns dies mitteilen? Der Nutzer könnte den Buchstaben `P` für Printeingeben, dann müsste der Sketch aber hier entweder für den Buchstaben `P` oder eine Zahl ausgelegt sein. Das ist knifflig. Es wäre einfacher, wenn wir als erstes Zeichen immer einen Buchstaben erwarten und dann entscheiden würden, ob wir eine Zahl wünschen. Gehen wir einmal von Folgendem aus: Wenn es sich bei dem ersten Buchstaben um ein `P` handelt, werden die aktuellen Einstellungen ausgedruckt, und wenn der erste Buchstabe ein `S` ist, sollen neue Einstellungen befolgt werden. Wenn der Nutzer etwas anderes als `P` oder `S` eintippt, wollen wir ihn daran erinnern, was er korrekterweise eingeben kann:

```
/*
 * Check for user interaction, which will be in the form of something typed
 * on the serial monitor
 * If there is anything, make sure it's proper, and perform the requested
 * action
 */
void checkUserInteraction() {
  // Check for user interaction
  while (Serial.available() > 0) {

    // The first character tells us what to expect for the rest of the line
    char temp = Serial.read();

    // If the first character is 'P' then print the current settings
    // and break out of the while() loop
    if ( temp == 'P') {
      printSettings();
      Serial.flush();
      break;
    } // end of printing current settings

    // If first character is 'S' then the rest will be a setting
    else if ( temp == 'S') {
      expectValveSetting();
    }

    // Otherwise, it's an error. Remind the user what the choices are
    // and break out of the while() loop
    else
    {
      printMenu();
      Serial.flush();
      break;
    }
  } // end of processing user interaction
}
```

Bei dem folgenden Code handelt es sich um die Funktion `printMenu()`. Er ist kurz, aber vielleicht möchten wir ihn anderenorts einsetzen. Außerdem wächst meiner Erfahrung nach das Menü mit der zunehmenden Komplexität des Projekts, so dass diese Funktion wirklich eine nette Möglichkeit ist, das Menü im Sketch zu dokumentieren. Vielleicht möchtest du beispielsweise später eine Menü-Option hinzufügen, um die RTC-Zeit festzulegen:

```
void printMenu() {
  Serial.println("Please enter P to print the current settings");
  Serial.println("Please enter S2N13:45 to set valve 2 ON time to 13:34");
}
```

Wann immer ein Code-Block mehr als einmal verwendet wird, ist er ein geeigneter Kandidat, um in eine Funktion umgewandelt zu werden, wie kurz er auch ist.

Endlich – in Beispiel 8-4 wird der gesamte Sketch gezeigt.

Beispiel 8-4: *Der Sketch für das Bewässerungssystem*

```
/*
 Sketch to accompany the automatic garden irrigation system
 */

#include <Wire.h>   // Wire library, used by RTC library
#include "RTClib.h" // RTC library
#include "DHT.h"    // DHT temperature/humidity sensor library

// Analog pin usage
const int RTC_5V_PIN = A3;
const int RTC_GND_PIN = A2;

// Digital pin usage
const int DHT_PIN  = 2;     // temperature/humidity sensor
const int WATER_VALVE_0_PIN = 8;
const int WATER_VALVE_1_PIN = 7;
const int WATER_VALVE_2_PIN = 4;

const int NUMBEROFVALVES = 3; // How many valves we have
const int NUMBEROFTIMES = 2;  // How many times we have

// Array to store ON and OFF times for each valve
// Store this time as the number of minutes since midnight
// to make calculations easier
int onOffTimes [NUMBEROFVALVES][NUMBEROFTIMES];
int valvePinNumbers[NUMBEROFVALVES];

// Which column is ON time and which is OFF time
const int ONTIME = 0;
const int OFFTIME = 1;

#define DHTTYPE DHT11
```

```
DHT dht(DHT_PIN, DHTTYPE); // Create a DHT object

RTC_DS1307 rtc;    // Create an RTC object

// Global variables set and used in different functions
DateTime dateTimeNow; // to store results from the RTC

// to store humidity result from the DHT11 sensor
float humidityNow;

void setup(){

  // Power and ground to RTC
  pinMode(RTC_5V_PIN, OUTPUT);
  pinMode(RTC_GND_PIN, OUTPUT);
  digitalWrite(RTC_5V_PIN, HIGH);
  digitalWrite(RTC_GND_PIN, LOW);

  // Initialize the wire library
  #ifdef AVR
    Wire.begin();
  #else
    // Shield I2C pins connect to alt I2C bus on Arduino Due
    Wire1.begin();
  #endif

  rtc.begin();        // Initialize the RTC object
  dht.begin();        // Initialize the DHT object
  Serial.begin(9600); // Initialize the Serial object

  // Set the water valve pin numbers into the array
  valvePinNumbers[0] = WATER_VALVE_0_PIN;
  valvePinNumbers[1] = WATER_VALVE_1_PIN;
  valvePinNumbers[2] = WATER_VALVE_2_PIN;

  // and set those pins all to outputs
  for (int valve = 0; valve < NUMBEROFVALVES; valve++) {
    pinMode(valvePinNumbers[valve], OUTPUT);
  }

};

void loop() {

  // Remind user briefly of possible commands
  Serial.print("Type 'P' to print settings or ");
  Serial.println("'S2N13:45' to set valve 2 ON time to 13:34");


  // Get (and print) the current date, time,
  // temperature, and humidity
  getTimeTempHumidity();

  // Check for request from the user
  checkUserInteraction();
```

```
  // Check to see whether it's time to turn any valve ON or OFF
  checkTimeControlValves();

  // No need to do this too frequently
  delay(5000);
}


/*
 * Get, and print, the current date, time,
 * humidity, and temperature
 */
void getTimeTempHumidity() {
  // Get and print the current time
  dateTimeNow = rtc.now();

  if (! rtc.isrunning()) {
    Serial.println("RTC is NOT running!");
    // use this to set the RTC to the date and time
    // this sketch was compiled
    // use this ONCE and then comment it out
    // rtc.adjust(DateTime(__DATE__, __TIME__));
    return; // if the RTC is not running don't continue
  }

  Serial.print(dateTimeNow.hour(), DEC);
  Serial.print(':');
  Serial.print(dateTimeNow.minute(), DEC);
  Serial.print(':');
  Serial.print(dateTimeNow.second(), DEC);

  // Get and print the current temperature and humidity
  humidityNow = dht.readHumidity();
  // Read temperature as Celsius
  float t = dht.readTemperature();
  // Read temperature as Fahrenheit
  float f = dht.readTemperature(true);

  // Check if any reads failed and exit early (to try again).
  if (isnan(humidityNow) || isnan(t) || isnan(f)) {
    Serial.println("Failed to read from DHT sensor!");
    return; // if the DHT is not running don't continue;
  }

  Serial.print(" Humidity ");
  Serial.print(humidityNow);
  Serial.print("% ");
  Serial.print("Temp ");
  Serial.print(t);
  Serial.print("C ");
  Serial.print(f);
  Serial.print("F");
  Serial.println();
} // end of getTimeTempHumidity()
```

```
/*
 * Check for user interaction, which will be in the form of
 * something typed on the serial monitor
 * If there is anything, make sure it's proper, and perform the
 * requested action
 */
void checkUserInteraction() {
  // Check for user interaction
  while (Serial.available() > 0) {

    // The first character tells us what to expect
    // for the rest of the line
    char temp = Serial.read();

    // If the first character is 'P' then
    // print the current settings
    // and break out of the while() loop
    if ( temp == 'P') {
      printSettings();
      Serial.flush();
      break;
    } // end of printing current settings

    // If first character is 'S' then the rest will be a setting
    else if ( temp == 'S') {
      expectValveSetting();
    }

    // Otherwise, it's an error.
    // Remind the user what the choices are
    // and break out of the while() loop
    else
    {
      printMenu();
      Serial.flush();
      break;
    }
  } // end of processing user interaction
}


/*
 * Read a string of the form "2N13:45" and separate it
 * into the valve number, the letter indicating ON or OFF,
 * and the time
 */
void expectValveSetting() {

  // The first integer should be the valve number
  int valveNumber = Serial.parseInt();

  // the next character should be either N or F
  char onOff = Serial.read();
```

```
  // next should come the hour
  int desiredHour = Serial.parseInt();

  // the next character should be ':'
  if (Serial.read() != ':') {
    Serial.println("no : found"); // Sanity check
    Serial.flush();
    return;
  }

  // next should come the minutes
  int desiredMinutes = Serial.parseInt();

  // finally expect a newline which is the end of
  // the sentence:
  if (Serial.read() != '\n') { // Sanity check
    Serial.println(
      "Make sure to end your request with a Newline");
    Serial.flush();
    return;
  }

  // Convert the desired hour and minute time
  // to the number of minutes since midnight
  int desiredMinutesSinceMidnight
    = (desiredHour*60 + desiredMinutes);

  // Now that we have all the information set it into the array
  // in the correct row and column

  if ( onOff == 'N') { // it's an ON time
    onOffTimes[valveNumber][ONTIME]
    = desiredMinutesSinceMidnight;
  }
  else if ( onOff == 'F') { // it's an OFF time
    onOffTimes[valveNumber][OFFTIME]
    = desiredMinutesSinceMidnight;
  }
  else { // user didn't use N or F
    Serial.print("You must use upper case N or F ");
    Serial.println("to indicate ON time or OFF time");
    Serial.flush();
    return;
  }

  // now print the entire array so user can see what they set
  printSettings();
} // end of expectValveSetting()


void checkTimeControlValves() {

  // First figure out how many minutes have passed
  // since midnight, since we store ON and OFF time
```

```
  // as the number of minutes since midnight.  The
  // biggest number will be at 2359 which is
  // 23 * 60 + 59 = 1159 which is less than the
  // maximum that can be stored in an integer so an
  // int is big enough
  int nowMinutesSinceMidnight =
    (dateTimeNow.hour() * 60) + dateTimeNow.minute();

  // Now check the array for each valve
  for (int valve = 0; valve < NUMBEROFVALVES; valve++) {
  Serial.print("Valve ");
    Serial.print(valve);

    Serial.print(" is now ");
    if ( ( nowMinutesSinceMidnight >=
           onOffTimes[valve][ONTIME]) &&
         ( nowMinutesSinceMidnight <
           onOffTimes[valve][OFFTIME]) ) {

      // Before we turn a valve on make sure it's not raining
      if ( humidityNow > 70 ) {
        // It's raining; turn the valve OFF
        Serial.print(" OFF ");
        digitalWrite(valvePinNumbers[valve], LOW);
      }
      else {
        // No rain and it's time to turn the valve ON
        Serial.print(" ON ");
        digitalWrite(valvePinNumbers[valve], HIGH);
      } // end of checking for rain
    } // end of checking for time to turn valve ON
    else {
      Serial.print(" OFF ");
      digitalWrite(valvePinNumbers[valve], LOW);
    }
    Serial.println();
  } // end of looping over each valve
  Serial.println();
}


void printMenu() {
  Serial.println(
    "Please enter P to print the current settings");
  Serial.println(
    "Please enter S2N13:45 to set valve 2 ON time to 13:34");
}


void printSettings(){

  // Print the current on and off settings, converting the
  // number of minutes since midnight back to the time
  // in hours and minutes
```

```
  Serial.println();

  for (int valve = 0; valve < NUMBEROFVALVES; valve++) {
    Serial.print("Valve ");
    Serial.print(valve);
    Serial.print(" will turn ON at ");

    // integer division by 60 gives the hours
    // since integer division drops any remainder
    Serial.print((onOffTimes[valve][ONTIME])/60);

    Serial.print(":");

    // the minutes are the remainder after dividing by 60.
    // get the remainder with the modulo (%) operator
    Serial.print((onOffTimes[valve][ONTIME])%(60));

    Serial.print(" and will turn OFF at ");
    Serial.print((onOffTimes[valve][OFFTIME])/60); // hours
    Serial.print(":");
    Serial.print((onOffTimes[valve][OFFTIME])%(60)); // minutes
    Serial.println();
  }
}
```

Du kannst diesen Sketch auch über den Link zu den Code-Beispielen auf der Katalogseite zu diesem Buch (*http://bit.ly/start_arduino_3e*) herunterladen.

Zusammenbau des Schaltkreises

Endlich! Der Sketch ist fertig und alle Komponenten wurden getestet! Bist du bereit, mit dem Löten zu beginnen? Noch nicht ganz: Wir haben die verschiedenen Komponenten separat getestet, aber noch nicht zusammen. Möglicherweise denkst du, dass dieser Schritt unnötig ist, aber Integrationstests sind von grundlegender Bedeutung. Bei diesem Schritt werden unerwartete Interaktionen zwischen Komponenten, sei es bei der Hardware oder Software, entdeckt. Es könnte beispielsweise für zwei Komponenten dasselbe Feature auf dem Arduino erforderlich sein, das aber nur auf einem Pin verfügbar ist, oder es könnte ein Konflikt zwischen zwei Bibliotheken bestehen oder die Sketch-Logik wird möglicherweise nicht erkannt und muss umgestaltet werden. Es ist daher besser, hier mit einer lötfreien Steckplatine zu arbeiten, für den Fall, dass Änderungen an der Verdrahtung vorgenommen werden müssen.

Um unser vollautomatisches Gartenbewässerungssystem zu erstellen, müssen wir drei Schaltpläne miteinander verbinden: Abbildung 8-11, Abbildung 8-13 und den Schaltplan für die RTC. Unser endgültiges System wird zwar über drei Ventile verfügen, dies bedeutet aber eine Menge Arbeit für wenig Informationsgewinn (und auf der lötfreien Steckplatine kann es zudem ganz schön eng werden). Wenn es bei einem Wasserventil funktioniert, sollte es auch bei dreien funktionieren, daher wollen wir an dieser Stelle nur ein Wasserventil einsetzen.

Sei immer vorsichtig mit Annahmen – sie könnten falsch sein und dir später böse Überraschungen bereiten. Setze niemals voraus, dass einzelne Elemente auch gut zusammenarbeiten, wenn sie einzeln für sich genommen funktionieren. Jeder Ingenieur wird dir sagen, dass Integrationstest von großer Bedeutung sind und hier sehr oft Probleme ans Licht kommen, die vorher nicht erkannt wurden.

Du hast sicher bemerkt, dass ich genau solch eine Vermutung angestellt habe, nämlich dass das Testen mit nur einem Ventil ausreichend ist. Dies ist genau die Art von Annahmen, vor der ich warne. So verbrauchen beispielsweise mehr Ventile und Relais mehr Strom. Können die digitalen Ausgänge des Arduino alle drei Relais mit Strom versorgen, wenn alle zur gleichen Zeit eingeschaltet sind? Kann die Stromversorgung für die Wasserventile alle drei Wasserventile mit Strom versorgen, wenn alle gleichzeitig eingeschaltet sind?

Ich habe mir erlaubt, diese Annahme zu treffen, weil ich eine grobe Berechnung in meinem Kopf angestellt habe und meine jahrelange Erfahrung mir sagt, dass es hier nur ein geringes Risiko gibt. Dennoch solltest du als Einsteiger keine solchen Vermutungen anstellen, sondern alles testen, bevor du mit dem Löten beginnst oder ein Projekt in ein Gehäuse einbaust.

Ich habe zu viele Studenten gesehen, die wunderschön konstruierte Projekte wieder auseinandernehmen mussten, weil etwas nicht erwartungsgemäß funktionierte. (Dabei handelt es sich dann, wie es scheint, ausgerechnet immer um die Komponente, an die man am schlechtesten herankommt.)

Michael

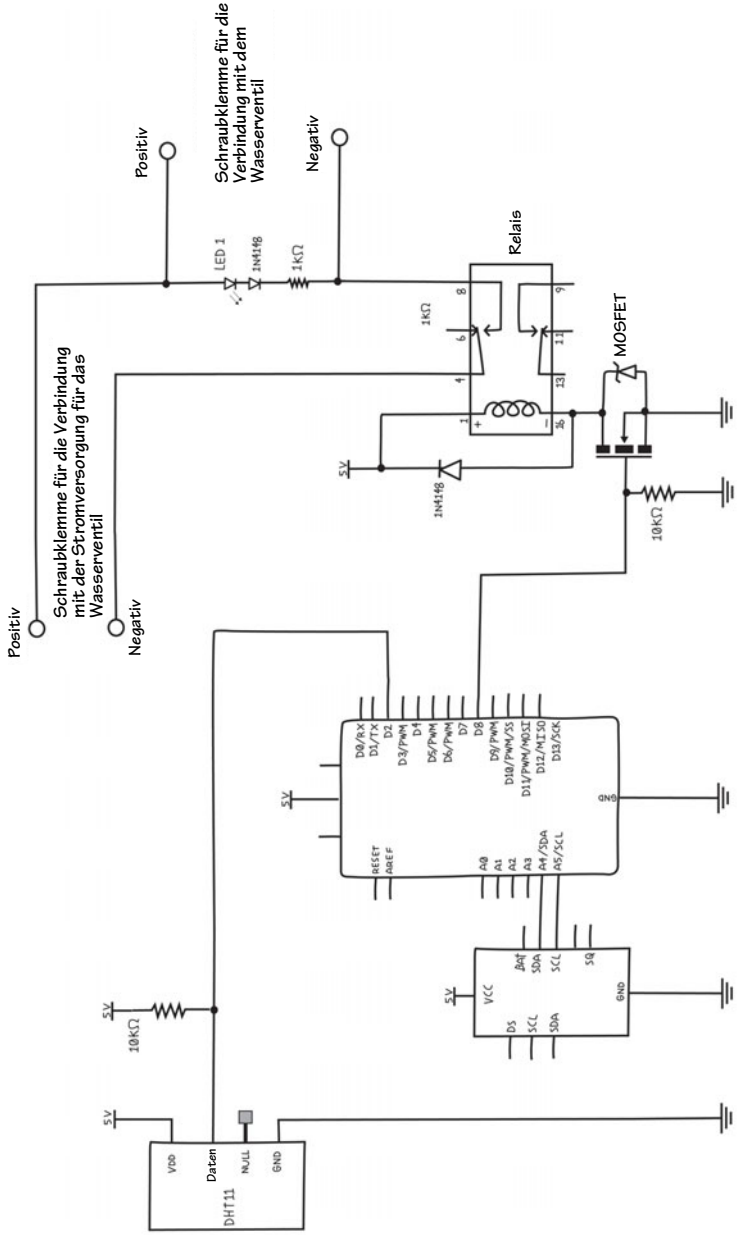


Abbildung 8-15: *Schaltplan für das gesamte automatische Gartenbewässerungssystem mit einem Ventil*

Abbildung 8-15 zeigt die Schaltskizze und Abbildung 8-16 zeigt die bebilderte Schaltskizze für das gesamte System mit einem Ventil. Ich möchte noch einmal in Erinnerung rufen, dass ich die Polung der Stromversorgung für das Wasserventil und die Polung des Wasserventils selbst gekennzeichnet habe, dies ist aber nur relevant, wenn du mit einem DC-System arbeitest. Bei einem Wechselstrom/AC-System (was das gebräuchlichere zu sein scheint) gibt es keine Polung.

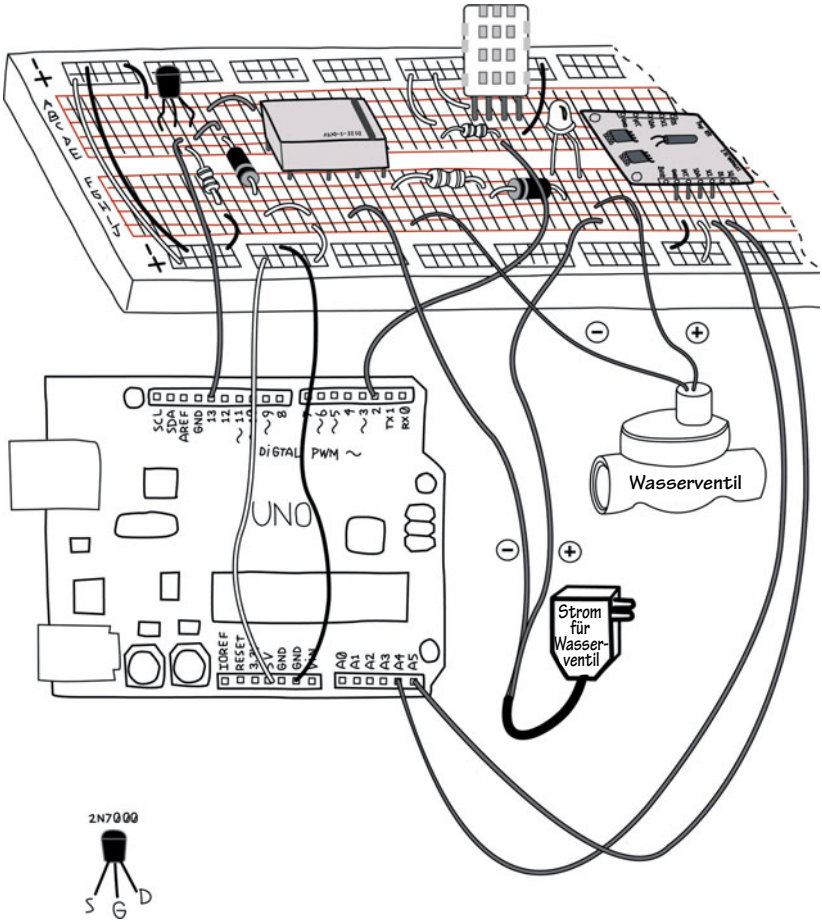


Abbildung 8-16: *Bebilderte Schaltskizze für das gesamte automatische Gartenbewässerungssystem mit einem Ventil*

Drucke den entsprechenden Schaltplan aus, bevor du einen komplexen Schaltkreis aufbaust. Verwende beim Aufbau einen farbigen Stift oder Textmarker, um die Verbindungen zu kennzeichnen, die du schon hergestellt hast. Dadurch kannst du leichter erkennen, was du schon erledigt hast und was nicht.

Das Arbeiten mit einem Stift oder Textmarker ist auch beim Inspizieren des Schaltkreises nützlich, indem du jede Verbindung markierst, die du überprüfst.

Baue alles auf deine Steckplatine und lade den Sketch für das komplette System aus Beispiel 8-4. Es ist in Ordnung, wenn der Sketch für drei Ventile geschrieben ist, in unserem Schaltkreis aber nur eines zum Einsatz kommt: Du kannst entsprechende Zeiten festlegen und die beiden anderen Ventile aktivieren, es wird aber nichts geschehen.

Führe nun den Test durch: Drücke `P`, um die aktuellen Einstellungen anzuzeigen, und vergewissere dich, dass sie alle auf 0 gesetzt sind. Notiere die aktuelle Zeit. Drücke `S` und gib für das Einschalten einen Zeitpunkt an, der eine Minute später liegt, und dann für das Ausschalten einen Zeitpunkt, der eine weitere Minute später folgt. Dein Relais sollte klicken und die LED zu leuchten beginnen. Dein Wasserventil tut nun möglicherweise etwas oder auch nicht, abhängig davon, ob es auch ohne Wasserdruck funktioniert (meins gibt auch ohne Wasser ein sehr beruhigendes Klickgeräusch von sich).

Treten Probleme auf? Überprüfe noch einmal deine Verdrahtung. Achte dabei besonders auf die MOSFETs und das Relais. Denke daran, dass jeder Pin des MOSFETs eine spezielle Funktion erfüllt und du den korrekten Pin verwenden musst. Rufe dir ins Gedächtnis, dass die Diode gepolt ist. Der schwarze Streifen kennzeichnet die Kathode. Erinnere dich auch daran, dass das Relais einen schwarzen Streifen hat, um die Pins 8 und 9 zu kennzeichnen. Wenn deine Wasserventile und die entsprechende Stromversorgung mit Gleichstrom (DC) arbeiten, musst du sicherstellen, dass die positiven und negativen Pole an den richtigen Stellen angeschlossen sind. Weitere Informationen findest du in Kapitel 9, *Troubleshooting*, auf Seite 171.

Diese Schritte sind auch hilfreich, um dich daran zu erinnern, wie wichtig es ist, die MOSFETs, Dioden und Relais korrekt zu verdrahten. Sobald diese Komponenten erst einmal angelötet sind, lassen sie sich nicht mehr so einfach ändern. Wenn also alles funktioniert, vergewissere sich, dass du auch verstanden hast, warum das so ist. Notiere alle Fehler, die du gemacht hast, und wie du sie behoben hast. Vielleicht möchtest du zu Referenzzwecken auch ein Foto deiner Steckplatine machen. Es ist immer gut und sinnvoll, deine Arbeit zu dokumentieren.

Wenn du mit dem Ergebnis zufrieden bist, kannst du mit dem Proto Shield fortfahren.

Das Proto Shield

Wie ich schon zu einem früheren Zeitpunkt erwähnt habe, verwenden wir das Proto Shield, weil es eine sichere und einfache Methode bietet, ein Projekt mit dem Arduino zu verbinden. Du kannst es im Arduino Store (*http://arduino.cc/protoShield*) kaufen. Es sind aber auch zahlreiche andere Proto Shields erhältlich. Du kannst jedes Modell verwenden, musst aber möglicherweise Änderungen am Layout vornehmen, damit es für dein spezielles Shield passt. Einige Shields werden mit männlichen Stiftleisten geliefert, bei anderen musst du sie separat kaufen.

Wie du siehst, hat das Shield männliche Stiftleisten auf der Unterseite, die in deinen Arduino gesteckt werden, so dass alle Arduino-Pins auf dem Shield vorhanden sind. Das Shield weist Löcher neben jedem Arduino-Pin auf, damit Drähte festgelötet werden können, über die eine Verbindung vom Shield zum Arduino hergestellt wird. Um eine Verbindung zu einem Arduino-Pin herzustellen, löte einfach einen Draht in das entsprechende Loch. Dadurch wird eine sicherere Verbindung erzielt, als wenn du die Drähte in die Stiftleisten steckst, wie wir es in der Vergangenheit getan haben.

Die meisten Shields bestehen aus einem Gitter mit kleinen Löchern. Sie erinnern ein wenig an die Löcher in einer lötfreien Steckplatine, in die du Komponenten und Drähte an (fast) jede gewünschte Stelle stecken kannst. Anders als bei einer lötfreien Steckplatine stehen aber nur sehr wenige Leiterbahnen zur Verfügung. Du wirst die meisten Verbindungen herstellen, indem du Drähte an die Komponenten auf dem Board lötest. Du kannst die Anzahl der Verbindungen reduzieren, bei denen du die Verdrahtung selbst vornehmen musst, indem du cleveren Gebrauch von Bussen oder auf andere Weise verbundenen Löchern machst, die auf dem Shield zur Verfügung stehen.

Wenn du ein Proto Shield oder eigentlich eine perforierte Steckplatine jeglicher Art benutzt, werden die Komponenten üblicherweise auf der Oberseite angebracht und auf der Unterseite gelötet. Dies ist besonders bei einem Proto Shield wichtig, weil sich die Oberseite des Shields sehr nah am Arduino befinden wird und dazwischen nicht sehr viel Platz vorhanden ist. Erinnere dich daran, dass keine der Verbindungen auf der Oberseite des Shields mit Metall auf der Oberseite des Arduinos in Verbindung kommen darf, z.B. mit Komponenten, Leiterbahnen oder dem USB-Anschluss.

Wenn du also Komponenten oder Drähte auf der Oberseite platzieren musst, sollten sie möglichst flach anliegen.

Das Layout deines Projekts auf dem Proto Shield

Der erste Schritt, über den du nachdenken musst, ist der, welche Anforderungen erfüllt werden müssen und wo. Wir müssen Platz für die MOSFETs, Relais, LEDs und Schraubklemmen schaffen. Die Schraubklemmen sollten an einer Seite angebracht werden, die zugänglich ist, und es wäre nett, wenn die LEDs nahe an den entsprechenden Schraubklemmen platziert würden. Die MOSFETs sind recht klein und können überall untergebracht werden, aber es wäre schön, wenn sich jeder neben dem Relais befände, das er steuert.

Bei den Relais handelt es sich um die größten Objekte. Daher müssen wir ihnen Platz einräumen, bevor wir das Shield weiter bestücken.

Es ist besteht leicht die Gefahr, die Ober- und die Unterseite des Shields zu verwechseln. Vergewissere dich also, dass du die Komponenten auf der richtigen Seite platziert hast. In den folgenden Abbildungen habe ich die Kennzeichnungen TOP und BOTTOM verwendet, um dich entsprechend zu erinnern. Ich würde dir vorschlagen, TOP und BOTTOM mit Permanentmarker auf dein Shield zu scheiben.

Verwende keine Löcher, die schon eine Funktion haben wie die ICSP-Schnittstelle. In den Abbildungen habe ich diese Löcher mit einem schwarz gefüllten Kreis markiert.

Vermeide den Bereich oberhalb des USB-Anschlusses. Wenn du das Arduino Proto Shield nutzt, weist dieser Bereich absichtlich keinerlei Löcher auf.

Wann immer du einen Schaltkreis lötest, solltest du immer gut überlegen, wo du die einzelnen Elemente platzieren möchtest, bevor du irgendetwas lötest. Beginne mit den Anschlüssen sowie den großen Bauteilen und platziere dann die kleineren Komponenten nahe an der Stelle, an der sie angeschlossen werden müssen. Du kannst die Anschlussdrähte der Komponenten verwenden, indem du sie auf der Unterseite des Shields biegst und sie direkt an die korrekten Pins anlötest.

Beginne erst mit dem Löten, wenn du mit der Platzierung zufrieden bist. Dokumentiere die Bestückung, entweder mit einer entsprechenden Zeichnung oder Fotos, bevor du mit dem

Löten beginnst, für den Fall, dass etwas herunterfällt, bevor es an der gewünschten Stelle festgelötet wurde.

Ich habe ja versprochen, dass ich noch erläutern werde, welchen Zweck die Sockel erfüllen. Wie du siehst, müssten Relais in das Proto Shield gelötet werden. Was geschieht aber, wenn eines der Relais nicht mehr funktioniert? Glücklicherweise passen die Relais auf einen Sockel. Der Sockel wird auf das Proto Shield gelötet und das Relais auf den Sockel gesteckt.

Wenn die Relais Sockel erhalten, warum erhält dann nicht jede Komponente einen Sockel? Hierfür gibt es mehrere Gründe: Widerstände lassen sich mit entsprechenden Entlöttechniken leicht entfernen. Im schlimmsten Fall können sie herausgeschnitten werden. Dasselbe gilt für die MOSFETs. Relais ließen sich nur sehr schwer mit Entlöttechniken entfernen, da sie über acht Pins verfügen. Wenn wir den zweiten Pin erhitzen würden, wäre der erste bereits wieder abgekühlt. Außerdem kann das Relais, sobald es einmal an eine Stelle gelötet wurde, nicht mehr entfernt werden. Und schließlich handelt es sich bei einem Relais um ein mechanisches Bauteil mit beweglichen Teilen, und bei beweglichen Teilen ist die Fehlergefahr größer, als bei rein elektronischen Teilen. (Dennoch sollte das Relais viele Jahre seinen Dienst verrichten.)

Beachte, dass die Sockel eine bestimmte Ausrichtung haben: Ein kleiner Halbkreis im Plastik zeigt die Oberseite oder den Anschluss für Pin 1 an. Dem Sockel ist das egal, er würde in jeder Richtung funktionieren. Die Kennzeichnung ist dazu gedacht, dir zu helfen, die Komponenten korrekt rund um den Sockel zu platzieren, um sicherzustellen, dass du diesen so positioniert hast, dass die gewünschte Verdrahtung erfolgen kann. Es soll an dieser Stelle noch einmal erwähnt werden, dass Zeichnungen und Notizen, die du für dich selbst anfertigst, dir später sehr helfen werden. Beachte, dass, wenn du das Shield umdrehst, sich die Ausrichtung des Sockels umkehrt. Ich zeichne gerne einen Kreis um Pin 1 jeden Sockels auf der Unterseite des Boards, um sicherzustellen, dass meine Ausrichtung stimmt.

Wenn du das Shield umdrehst, fallen die Sockel heraus. Biege also die Anschlussdrähte um, damit sie an ihrem Platz gehalten werden. Die Anschlussdrähte können beliebig gebogen werden, solange sie keine anderen Löcher berühren.

Wann immer du einen Schaltkreis lötest, solltest du Sockel für Relais und Chips verwenden.

In Abbildung 8-17 wird ein mögliches Layout gezeigt.

Beachte, dass das Bild ein wenig verzerrt ist, um bestimmte Abschnitte zu vergrößern. Wir werden an dieser Stelle später sehr viel Arbeit haben, und

ich möchte daher, dass die Details leichter zu erkennen sind. Die Anzahl der Löcher und die Ausrichtung der Reihen sind akkurat dargestellt.

Wenn wir die kleineren Komponenten hinzufügen, werde ich dir einen Trick zeigen. Wir werden deren Anschlussdrähte nutzen, um einige unserer Verbindungen herzustellen.

Schau dir die Schaltskizze an. Du siehst, dass die drei Dioden nahe am Relais sich von Pin 1 bis zu Pin 16 erstrecken. Wenn wir die Dioden an diesem Ende platzieren, können wir die Anschlussdrähte auf der Unterseite des Shields einfach biegen und sie direkt an die richtigen Pins anlöten. Beachte dabei die Polung der Dioden oder du wirst später fluchen. (Ich kann ein Lied davon singen, weil ich das auch viele Male getan habe.) Die Kathode ist durch einen Ring nahe an einem Ende der Diode gekennzeichnet und wird mit Pin 1 des Sockels verbunden.

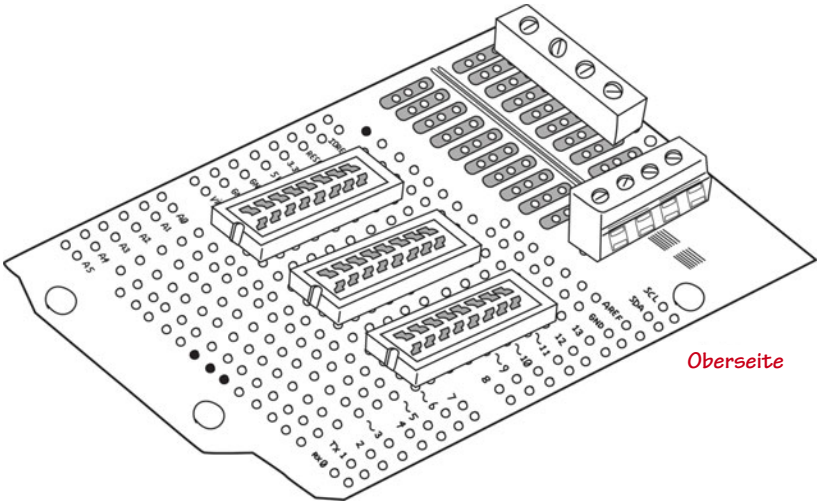


Abbildung 8-17: *Eine mögliche Platzierung der großen Komponenten auf dem Proto Shield (Beachte die Ausrichtung des Relaissockels!)*

Das Umbiegen der Anschlussdrähte der Diode hilft auch dabei, sie an Ort und Stelle zu halten, wenn das Board umgedreht wird, um Lötarbeiten an der Unterseite durchzuführen.

Der MOSFET verfügt über einen Pin (den Drain-Pin), der mit Pin 16 des Relais verbunden wird. Wir können ihn direkt neben der Diode platzieren, dann den Anschlussdraht des MOSFETs umbiegen und diesen anschließend an der Diode festlöten. Der 10K-Ohm-Widerstand, der den Gate-Pin jeden MOSFETs mit Masse verbindet, befindet sich zwichen dem Gate-Pin und dem Source-Pin des MOSFETs, da der Source-Pin auch mit Masse verbunden ist. Dazu stellen wir den Widerstand senkrecht auf ein Ende

und verwenden die Anschlussdrähte, um die erforderlichen Verbindungen herzustellen, ohne dass zusätzlicher Draht zum Einsatz kommt.

Alle Komponenten sollten möglichst nah am Shield anliegen. Auch die Dioden sollten möglichst wenig Abstand zum Shield aufweisen. Du kannst die Anschlussdrähte der MOSFETs ein wenig biegen, damit sie stärker anliegen, biege sie aber nicht zu sehr, sonst brechen sie. Der Widerstand steht senkrecht, aber eines seiner Enden sollte sich auf dem Shield befinden.

Ich werde gleich beschreiben, wie all diese Verbindungen hergestellt werden.

In Abbildung 8-18 wird die Oberseite mit den angefügten Relaissockeln, MOSFETs, Dioden und Widerständen gezeigt.

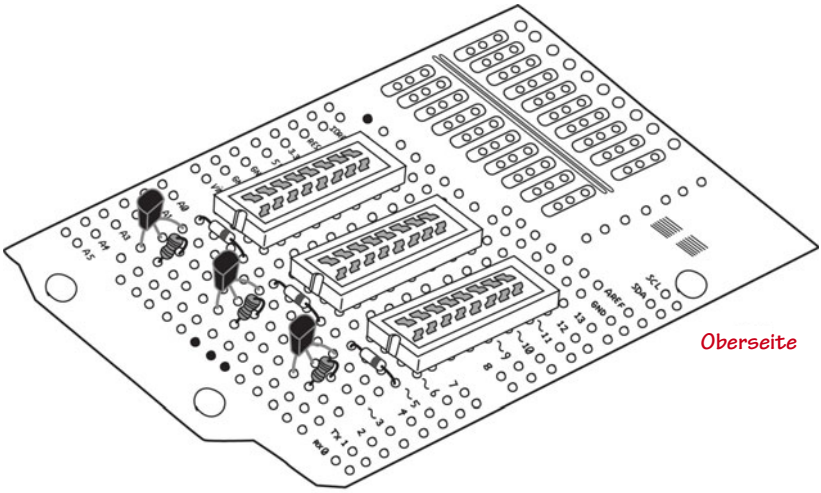


Abbildung 8-18: *Die Oberseite mit den angefügten Relaissockeln, MOSFETs, Dioden und Widerständen*

Was ist mit der RTC und dem DHT11? Der DHT11 muss mittels vier langer Drähte in den Garten ausgelagert werden. Wir löten diese Drähte nicht direkt

am Proto Shield an, sondern eine männliche Stiftleiste an den Draht und montieren eine weibliche Stiftleiste auf dem Shield, so dass wir sie nötigenfalls entfernen können. Ich werde später die entsprechenden Arbeitsschritte beschreiben. Der 10K-Ohm Widerstand (auf dem Datenpin des DHT11) kann so angepasst werden, dass er fast überall verwendbar ist, wir können uns also später mit ihm beschäftigen.

Die RTC verfügt bereits über männliche Stiftleisten, eine weitere weibliche Stiftleiste ist für diese Komponente daher perfekt. Bedenke, dass die RTC schon ein wenig Platz erfordert und wähle eine entsprechende Stelle. Die obere Kante des Boards nach den MOSFETs und der entsprechenden Verschaltung könnte hier geeignet sein. Ich habe ihn in der allerletzten Reihe platziert. Dadurch bleibt mir immer noch eine leere Reihe zwischen den MOSFETs und den weiblichen Stiftleisten für eine entsprechende Verdrahtung, wie in Abbildung 8-19 gezeigt.

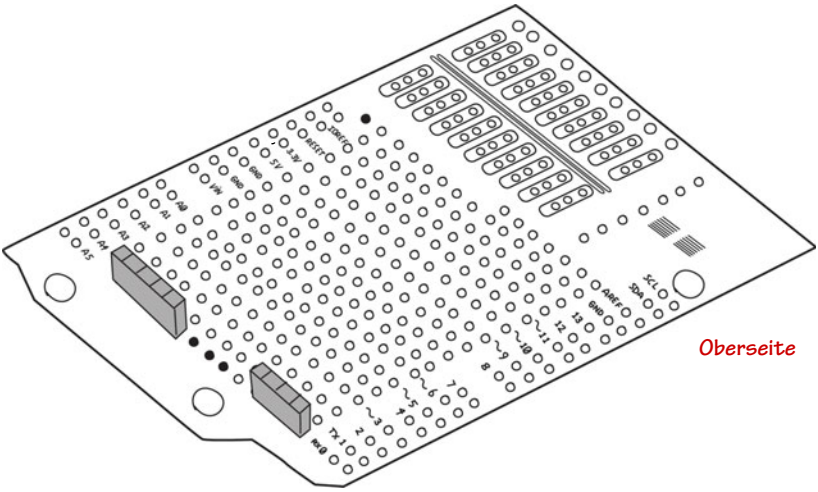


Abbildung 8-19: *Eine weibliche Stiftleiste mit vier Positionen für den DHT11-Sensor und eine weibliche Stiftleiste mit fünf Positionen für die RTC*

Wann immer du Drähte von einem weiter entfernten Objekt mit dem Board verbinden musst, solltest du die Drähte nicht direkt an das Board anlöten, sondern stattdessen eine Art Verbindung herstellen, bei der sich die Drähte auch wieder auf einfache Weise entfernen lassen. Ein Paar aus männlicher und weiblicher Stiftleiste mit der richtigen Anzahl an Positionen ist eine gute, preisgünstige Wahl bei dünnen Drähten. Schraubklemmen sind für dickere Drähte geeignet.

Wenn du ein Modul mit Stiftleisten auf dem Bord montierst, solltest du die Stiftleisten des Moduls niemals direkt an deinem Board festlöten. Montiere stattdessen eine entsprechende Stiftleiste des anderen Geschlechts auf deinem Board. Dadurch hast du die Möglichkeit, das Modul nötigenfalls zu entfernen.

Füge besser all diese Stiftleisten deiner Einkaufsliste hinzu! Die Stiftleisten sind üblicherweise in langen Streifen und mehreren Stücken erhältlich. Sie sind so gestaltet, dass sie sich auf die erforderliche Länge zuschneiden lassen. Wenn du die männlichen Stiftleisten kürzen möchtest, kannst du den Streifen normalerweise direkt an der Stelle durchbrechen, an der es erforderlich ist. Wenn die weiblichen Stiftleisten in der Länge anpasst werden sollen, musst du eine Position opfern. Hier nun die Ergänzungen, mit denen wir zu Version 0.5 der Einkaufsliste gelangen:

- Füge einen Satz weiblicher Stiftleisten, .1"-Raster, z.B. von Adafruit, Produkt-ID 598, hinzu
- Füge einen Satz männlicher Stiftleisten, .1"-Raster, z.B. von Adafruit, Produkt-ID 392, hinzu.

Dein Projekt auf dem Proto Shield fest verlöten

Eine gute Anleitung für das Löten ist der englischsprachige Adafruit Guide to Excellent Soldering (*http://bit.ly/solder-guide*). Ein deutschsprachiges Youtube-Video zum Löten findest du hier: *https://www.youtube.com/watch?v=T5v3illPk8I*.

Nun bist du endlich soweit und kannst mit dem Löten beginnen!

Hetze dich nicht. Gehe vorsichtig vor. Atme durch und entspanne dich. Überprüfe noch einmal jede Verbindung, bevor du sie lötest. Inspiziere deine Arbeit hinsichtlich schlechter Lötverbindungen und sonstiger Fehler.

Versuche nicht, alle Verbindungen auf einmal herzustellen. Nimm dir kleine Gruppen vor und leg zwischen den Gruppen jeweils eine kleine Pause ein. Überprüfe nach der Pause noch einmal, was du gerade getan hat.

Folge nicht blind meinen Anweisungen. Versuche den Arbeitsschritt zu verstehen, der gerade durchgeführt wird, und vergewissere dich, dass du mit ihm einverstanden bist.

Du kannst die Schraubklemmen und weiblichen Stiftleisten nun erst einmal entfernen, so dass du das Shield flach auf die Arbeitsfläche legen kannst, damit die Bauteile an ihrem Platz bleiben.

Löte zunächst die Sockel an ihrem Platz fest, damit sie nicht herausfallen. Dadurch erhältst du ein wenig Lötpraxis.

Als Nächstes folgen Dioden, MOSFETs und Widerstände. Erinnere dich, dass wir ihre Verbindungsdrähte (auf der Unterseite des Boards) verwenden möchten, um die Verbindungen herzustellen. Biege sie sehr nah am Board um, so dass die Komponenten eng an das Board herangezogen werden, und dann in die Richtung, an der die Verbindung erfolgen soll. Du musst den Anschlussdraht nicht um den Pin wickeln; es reicht aus, wenn er die das Shield rund um den Pin berührt. Das Lötzinn muss auf alle Anschlussdrähte fließen, die verbunden werden sollen.

Wenn du eine Verbindung gelötet hast, schneide den überschüssigen Anschlussdraht so nah wie möglich an der Lötstelle ab, denn du möchtest ja nicht, dass ein überschüssiges Stück herausragt und später möglicherweise etwas anderes berührt. Dies wird ausführlich im Tutorial Making a good solder joint (*http://bit.ly/1tgWVHY*) von Adafruit beschrieben.

Wenn du nach dem Löten einen Anschluss oder Draht kürzt, musst du das abgeschnittene Teil festhalten und dann entfernen, damit es nicht auf deine Arbeit zurückfällt, wo es versehentlich etwas berühren und dadurch einen Kurzschluss verursachen kann.

Sämtliche Source-Pins des MOSFETs werden mit Masse verbunden, so dass eine Verbindung untereinander hergestellt werden kann. Du kannst die langen Anschlussdrähte der Widerstände verwenden, um alle drei miteinander zu verbinden, wodurch ein nützlicher GND-Strang für alle anderen Komponenten entsteht, die mit Masse verbunden werden müssen. Diese häufig benutzten Stränge werden oft als *Busse* bezeichnet. Beachte, dass zu diesem Zeitpunkt unser GND-Bus nicht mit dem GND-Pin des Arduino verbunden ist. Ich hebe mir dies normalerweise für später auf, um nicht ein Loch zu belegen, das wir später noch benötigen; wir müssen aber daran denken, zum Schluss diese Verbindung herzustellen.

In Abbildung 8-20 wird die Unterseite mit den umgebogenen und festgelöteten Anschlussdrähten gezeigt. Mit den schattierten Bereichen sind die Relais-Sockel auf der Oberseite gekennzeichnet, die konischen Symbole stehen für das Lötzinn.

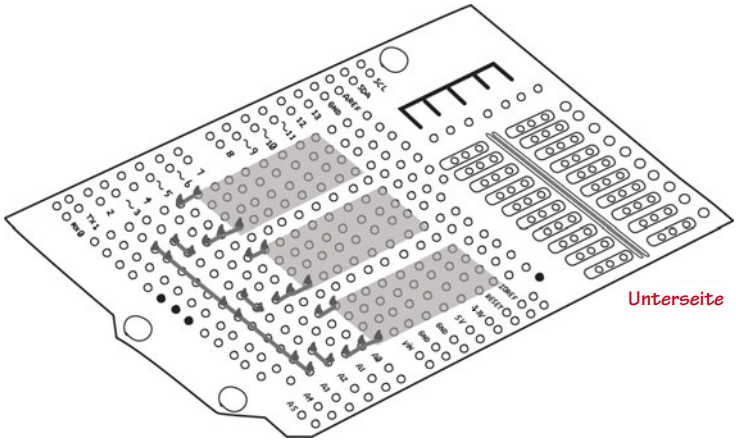


Abbildung 8-20: *Ansicht der Unterseite des Relais-Sockels, des MOSFETs und der Diode*

Nun kannst du die Schraubklemmen wieder einsetzen. Bevor du sie festlötest, musst du dich vergewissern, dass die Öffnung für die Drähte in die richtige Richtung zeigen, nämlich vom Board nach außen! (Dies ist ein weiterer häufiger Fehler, wie ich auf die harte Tour lernen musste.) Wie immer ist auch hier darauf zu achten, dass sie möglichst bündig am Shield anliegen. Heb dir die weibliche Stiftleiste für später auf.

An dieser Stelle ist es sinnvoll, zu entscheiden, welche Schraubklemme welchem Zweck dient. Dokumentiere dies wie in Abbildung 8-21 gezeigt, damit du es nicht vergisst und später keine falsche Verbindung herstellst.

Nun kannst du die LEDs für die Anzeige sowie die entsprechenden Dioden und Widerstände anbringen. Auch hier kannst du wieder durch cleveres Platzieren dieser Komponenten ihre Anschlussdrähte für die Herstellung der Verbindungen benutzen. Beachte, dass dieses spezielle Proto Shield über einige Reihen mit drei verbundenen Pins verfügt. Ich habe sie als Hilfsmittel für die Verbindungen benutzt. Denk wieder daran, dass LEDs und Dioden gepolt sind: Die Anode (mit dem längeren Anschlussdraht) jeder LED wird mit der Schraubklemme und die Anode jeder Diode mit der Kathode der entsprechenden LED verbunden.

Es ist wichtig, zu wissen, dass es zwei Stränge oder Busse gibt, nämlich 5V und Masse, die wir nicht verwenden, so dass du dich sorgfältig vergewissern musst, dass sich die Anschlussdrähte der LED über ihnen befinden und sie nicht berühren. Wenn die LED versehentlich mit den 5V oder Masse in Berührung kommt, können dadurch leicht die 24V der Stromversorgung des Wasserventils in den Arduino geraten, was ihn wahrscheinlich beschädigen würde. Um sicherzugehen, kannst du ein entsprechendes Stück Isolierband zurechtschneiden und über den 5V-Bus und den GND-Bus kleben. Diese

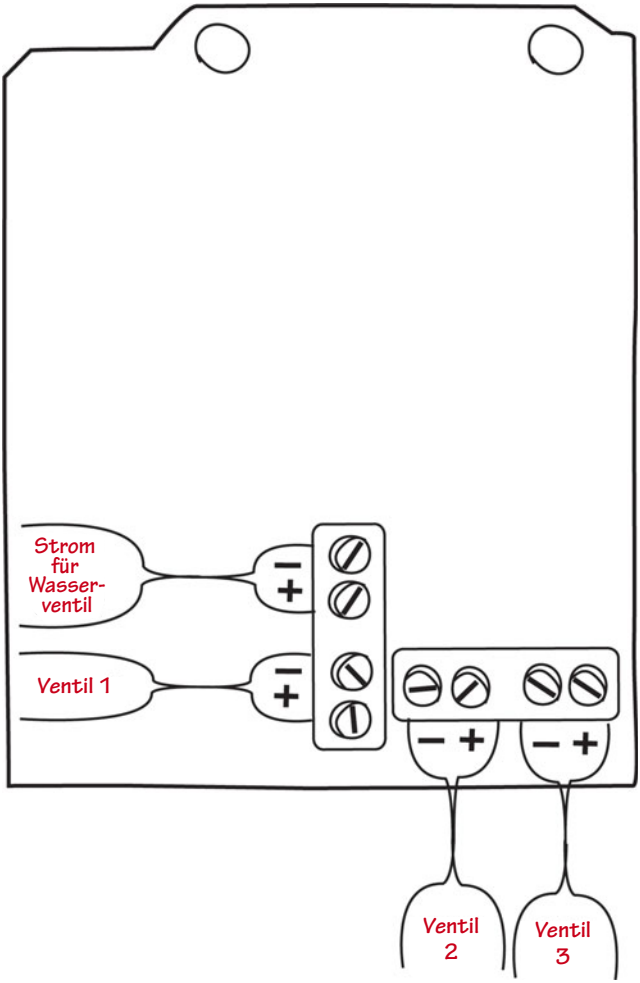


Abbildung 8-21: *Dokumentation der Aufgaben der einzelnen Schraubklemmen*

Busse befinden sich auch auf der Unterseite, und auch hier müssen entsprechende Berührungen vermieden werden.

In Abbildung 8-22 wurden die Komponenten in einem größeren Abstand zum Shield dargestellt, damit du sehen kannst, wo sie alle verbunden sind. Wenn du den Schaltkreis aufbaust, sollten die Komponenten aber, wie schon erwähnt, so dicht wie möglich am Shield anliegen. Denk daran, dass diese Abbildung verzerrt ist, um bestimmte Details zu vergrößern. Die Benutzung der Löcher ist aber akkurat dargestellt.

In diesem und vielen anderen nachfolgenden Abbildungen habe ich die meisten Komponenten aus den früheren Schritten ausgelassen, damit du die Komponenten und ihre Positionen bei jedem Schritt besser erkennen kannst.

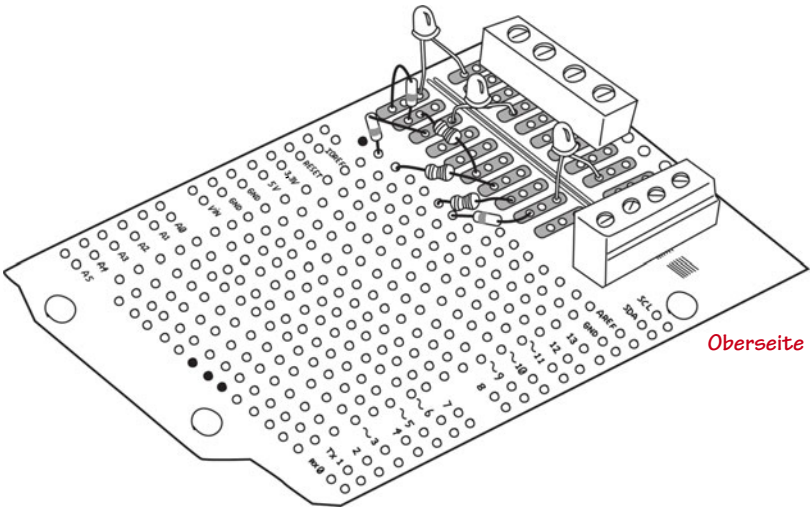


Abbildung 8-22: *Die Oberseite mit der Platzierung von LED, Widerstand und Diode*

Abbildung 8-23 zeigt die Unterseite. Wie zuvor haben wir die Anschlussdrähte der Komponenten benutzt, um sie miteinander und mit den Schraubklemmen zu verbinden. Hier erweist es sich als kluger Schachzug, dass wir die Schraubklemmen dokumentiert haben, da wir sie und die entsprechenden Funktionen sonst durcheinanderbringen würden.

Beachte den schwarzen Kreis beim IOREF. Es handelt sich hier um eine Verbindung zur Masse, die verwendet werden muss, bis du eine eigene Masseverbindung herstellst.

Dies ist ein komplizierter Abschnitt. Lies ihn sorgfältig und beginne erst mit dem Löten, wenn du wirklich verstanden hast, welchen Zweck ein Bauteil erfüllt, und du sicher bist, dass es sich am richtigen Platz befindet.

Wenn sich nun alle Komponenten auf dem Shield befinden, musst du die restlichen Verbindungen herstellen; verwende zu diesem Zweck einen 22 AWG-Massivdraht. Eine dünnere Version würde zwar auch ausreichen, aber mit ihr lässt es sich schwerer arbeiten. Du kannst einfach den Massivdraht verwenden, mit dem du am besten zurechtkommst.

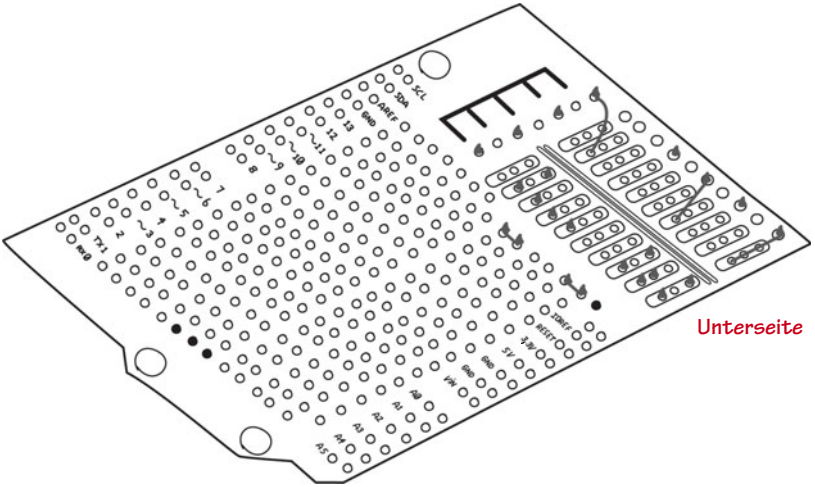


Abbildung 8-23: *Die Unterseite mit den an den Pins der Schraubklemmen angelöteten Anschlussdrähte der LED*

Verwende ein konsistentes Farbschema: Benutze roten Draht für alle Verbindungen mit 5V und schwarzen Draht für alle Verbindungen mit Masse. Für andere Zwecke kannst du dein eigenes Farbensystem verwenden, setze darin aber nicht mehr die Farben Rot oder Schwarz ein. Du könntest Orange für die positiven Anschlüsse der Stromversorgung der Wasserventile und Grün für ihre negativen Anschlüsse verwenden. Alle Drähte, die miteinander verbunden werden, sollten dieselbe Farbe haben, und alle nicht verbundenen Drähte sollten sich davon farblich unterscheiden.

Das grundlegende Prinzip ist folgendes: Die Verdrahtung erfolgt auf der Oberseite über Löcher in der Nähe des Pins, mit dem eine Verbindung hergestellt werden soll. Auf der Unterseite biegst du den Draht um und lötest ihn am entsprechenden Pin an, wie du es vorher auch bei den Anschlussdrähten der Komponenten getan hast.

Alle Arduino-Pins haben ihre eigenen Löcher, so dass du die Drähte nicht umbiegen musst. Löte einfach den Draht in das entsprechende Loch.

Manchmal ist es nicht möglich, von der Oberseite her nah genug an den Pin herankommen. In diesem Fall ist es in Ordnung, die Verdrahtung auf der Unterseite vorzunehmen, sorge aber dafür, dass sie so flach wie möglich anliegen.

Wir wollen nun mit den MOSFETs und der entsprechenden Verschaltung beginnen. Wir haben, soweit es möglich war, alle Verbindungen mit den Anschlussdrähten der Komponenten realisiert. Wir müssen Pin 1 aller Relais mit 5V verbinden. Ich habe roten Draht verwendet, um die Verbindungen zu den 5V herzustellen, wie in Abbildung 8-24 gezeigt (denk daran, dass wir uns um die GND-Verbindung erst zum Schluss kümmern werden).

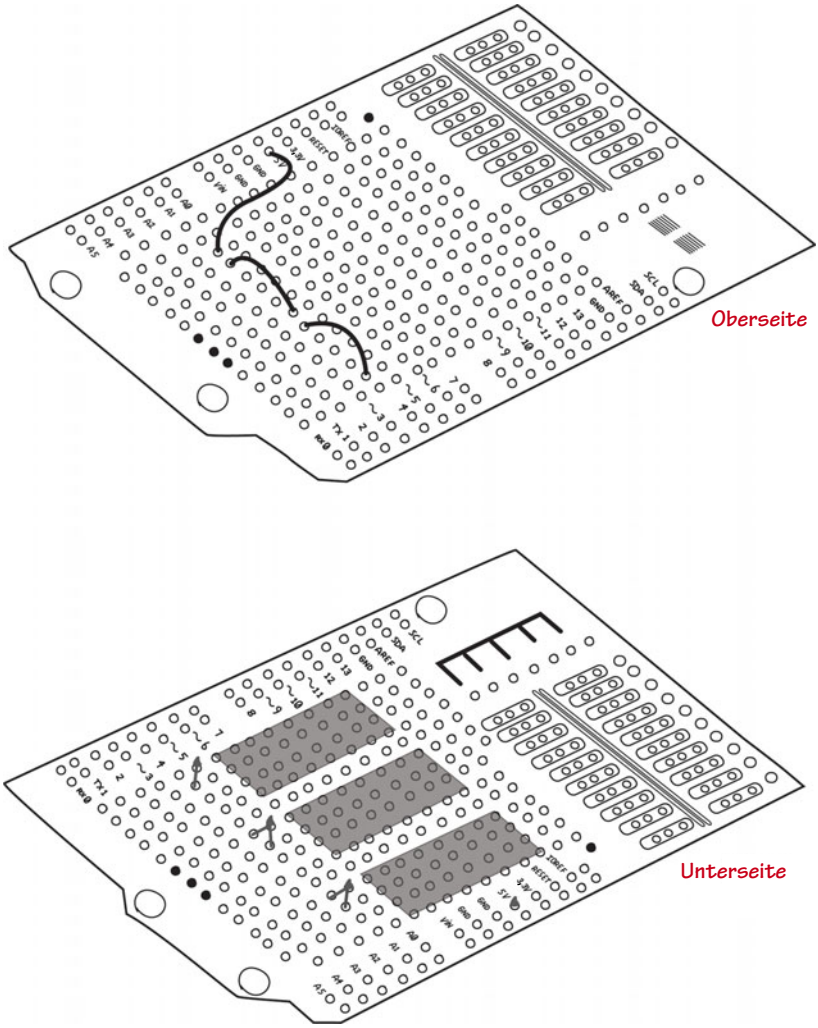


Abbildung 8-24: *Die 5V-Verbindungen der Relais-Schaltung*

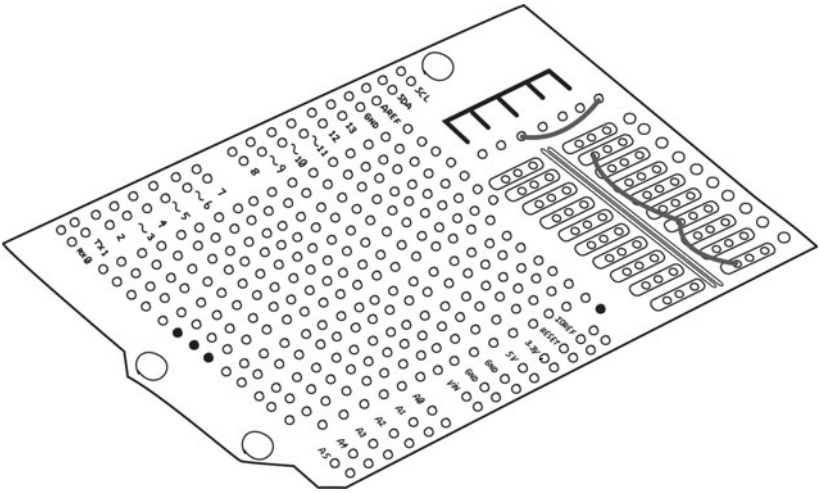


Abbildung 8-25: *Anfügen aller positiven Schraubklemmen*

Füge als Nächstes alle positive Schraubklemmen an. Dies alles erfolgt auf der Unterseite des Boards, wie in Abbildung 8-25 gezeigt. Stelle sicher, dass weder irgendein Anschlussdraht noch Lötzinn diese 5V- und GND-Busse berühren kann!

Verbinde die Kette aus LED, Widerstand und Diode mit der negativen Schraubklemme. Lass das mittlere Loch frei, da wir es im nächsten Schritt benötigen werden, um eine Verbindung zu den Relais herzustellen. In diesem Fall habe ich zwei Drähte auf der Oberseite und einen auf der Unterseite verwendet, wie in Abbildung 8-26 dargestellt.

Verbinde nun Pin 8 jedes Relais mit der entsprechenden negativen Schraubklemme, wie in Abbildung 8-27 gezeigt.

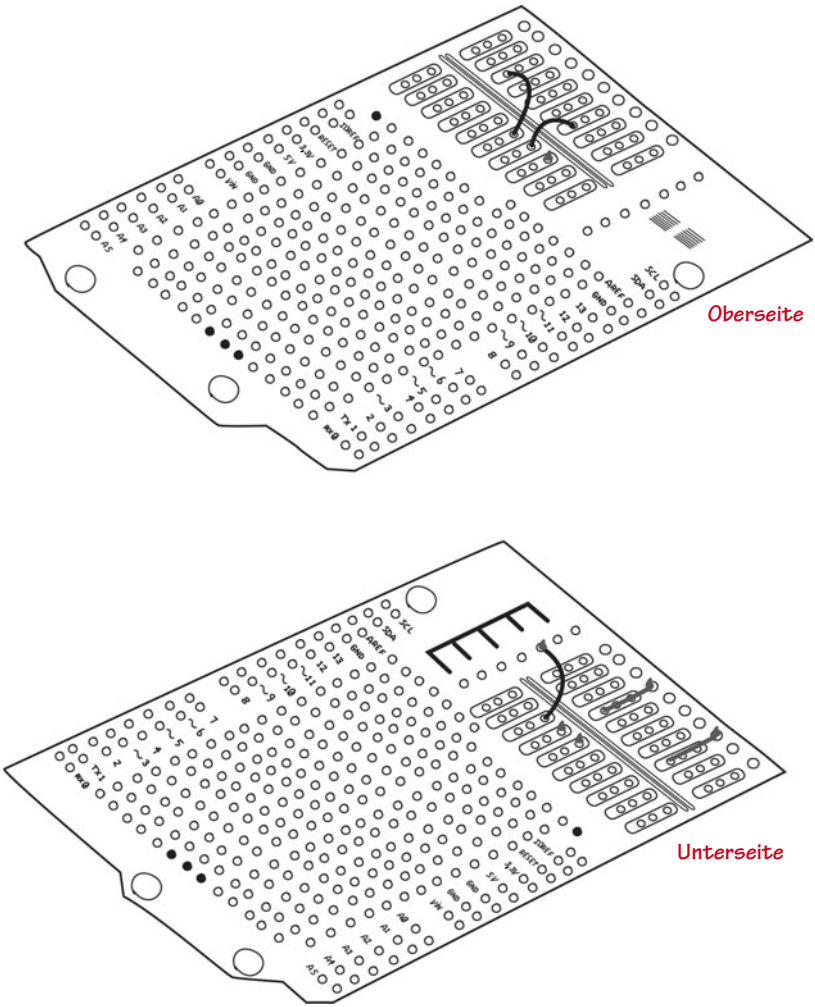


Abbildung 8-26: *Anfügen der negativen Schraubklemmen*

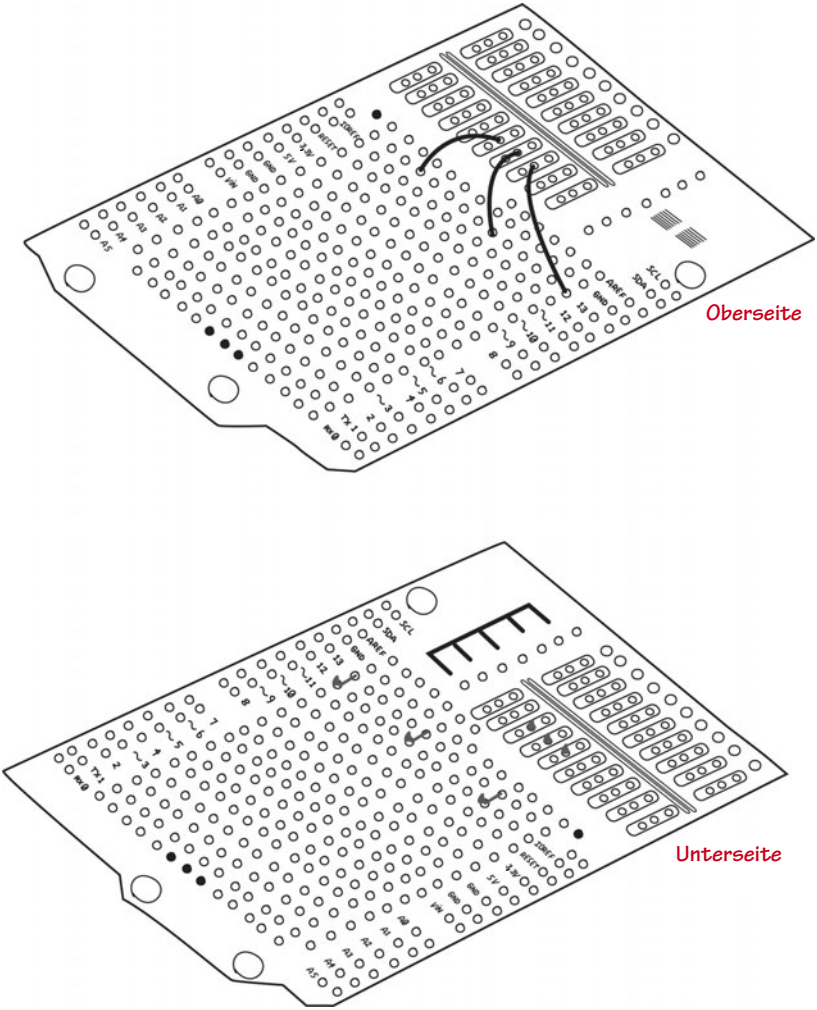


Abbildung 8-27: *Verbinden von Pin 8 jedes Relais mit der entsprechenden negativen Schraubklemme*

Pin 4 aller Relais wird mit der negativen Schraubklemme der Stromversorgung der Wasserventile verbunden. Dazu werden zwei Drähte auf der Oberseite und ein Draht auf der Unterseite verwendet, wie in Abbildung 8-28 gezeigt.

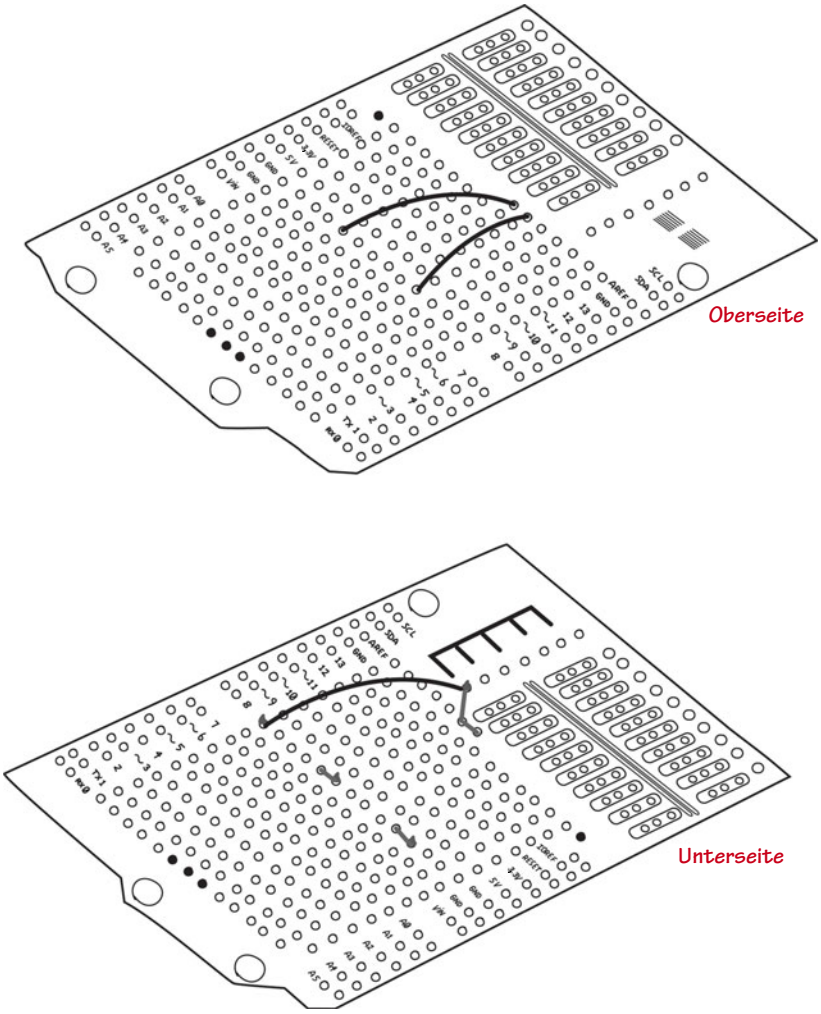


Abbildung 8-28: *Verbinden von Pin 4 jedes Relais mit der negative Schraubklemme für die Stromversorgung der Wasserventile*

Verbinde als Nächstes die digitalen Arduino-Pins mit den Gate-Pins der MOSFETs, wie in Abbildung 8-29 gezeigt. Denke daran, dass die Löcher neben dem Arduino-Pin bereits elektrisch mit den Arduino-Pins verbunden sind, so dass du den Draht nicht biegen und direkt an den Pin anlöten musst.

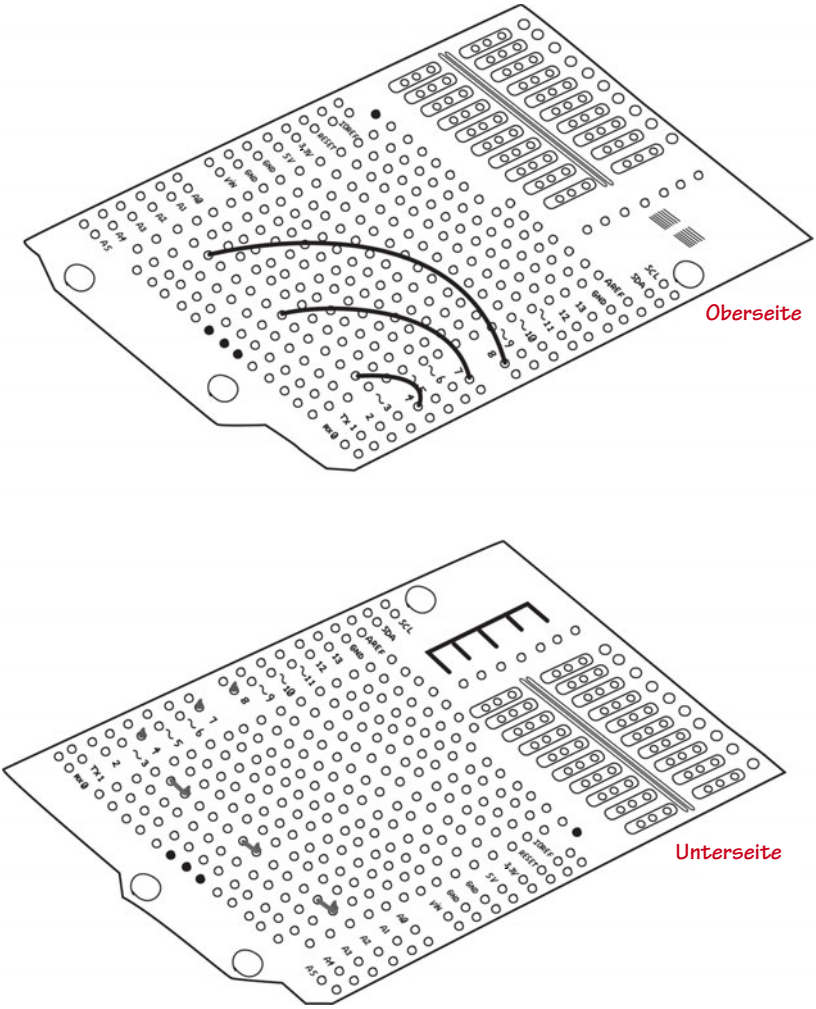


Abbildung 8-29: *Verbinden der Arduino-Pins mit den Gate-Pins der MOSFETs*

Füge schließlich noch die weiblichen Stiftleisten hinzu – eine für die RTC und eine für den DHT11 – und verbinde sie mit den entsprechenden Arduino-Pins. Vergiss nicht den 10kOhm-Widerstand, der für den DHT11 erforderlich ist, wie in Abbildung 8-30 dargestellt. Ich habe außerdem die Gelegenheit genutzt und habe den Masse-Bus, den wir vorher hergestellt haben, mit dem GND-Pin des Arduino verbunden.

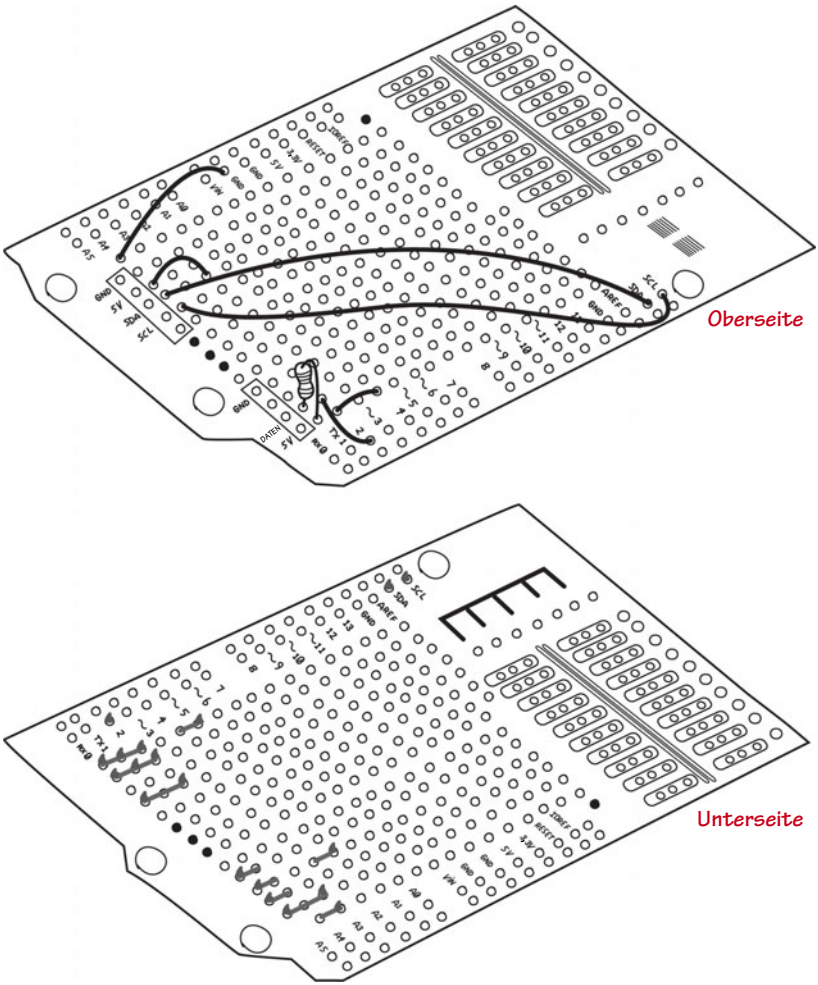


Abbildung 8-30: *Verbinden der Stiftleisten für die RTC und den DHT11 mit den Arduino-Pins*

Dokumentiere, welcher Pin auf der weiblichen Stiftleiste welchen Zweck erfüllt, so dass du die RTC und den DHT11 korrekt anschließen kannst (ein ultrafeiner Sharpie ist für diesen Zweck bestens geeignet).

Der letzte Schritt besteht darin, die männlichen Stiftleisten, die in die Arduino-Pins passen, festzulöten. Dieser Schritt sollte als letzter in Angriff genommen werden, weil die Stiftleisten im Weg wären, wenn du Arbeiten auf der Unterseite des Shields verrichtest. Auch wenn du nicht alle Pins benötigst, ist es klug, sie für die mechanische Stabilität und für mögliche zukünftige Erweiterungen anzubringen.

Vergiss nicht, dass die männlichen Pins nach unten zeigen müssen, d.h. in Richtung Arduino, wie in Abbildung 8-34 gezeigt. Achte darauf, dass die Pins gerade sind, damit sie in die weiblichen Stiftleisten auf deinem Arduino passen.

Wenn du damit fertig bist, besteht der nächste Schritt im Testen.

Das Testen deines bestückten Proto Shields

Teste dein Shield zunächst ohne Ventile und die entsprechende Stromversorgung. Stecke das Shield in deinen Arduino. Vergewissere dich, dass die männlichen Stiftleisten auf dem Shield mit den korrekten Arduino-Pins verbunden werden. Schau dir auch den Zwischenraum an, um sicherzustellen, dass keinerlei Berührung zwischen einer Verbindung auf der Unterseite des Shields und irgendetwas auf dem Arduino stattfindet. Wenn solche Kontakte vorhanden sind, musst du Isolierband verwenden, um sie zu unterbinden.

Verbinde deinen Arduino mit einem USB-Port deines Computers und beobachte die ON-Board-LED auf dem Arduino. Wenn sie aus ist, bedeutet das, dass ein Kurzschluss vorliegt und dein Computer sich geschützt hat, indem er den USB-Anschluss ausgeschaltet hat. Entferne das USB-Kabel und suche das Problem, bevor du fortfährst.

Als Nächstes kannst du die Relais anbringen und testen. Denke daran, dass sie gepolt sind und dass der Streifen die Pins 8 und 9 kennzeichnet. Lade das Blink-Beispiel und teste alle Relais separat. Teste immer nur ein Relais nach dem anderen, um zu überprüfen, ob jedes einzelne funktioniert.

Als Nächstes testen wir die Stromversorgung für die Wasserventile und die LEDs für die Anzeige. Um die Wasserventile kümmern wir uns als Letztes.

Verbinde die Stromversorgung für die Wasserventile mit der korrekten Schraubklemme. Wenn es sich beim Wasserventilsystem um ein Gleichstrom/ DC-System handelt, musst du sorgfältig auf die Polung achten.

Lade das Blink-Beispiel erneut und teste abwechselnd jedes Relais. Dieses Mal sollte die entsprechende LED zu leuchten beginnen.

Füge nun die Wasserventile hinzu und überprüfe sie, ebenfalls unter Verwendung des Blink-Beispiels, für jedes Ventil.

Anschließend möchten wir die RTC und den DHT11 überprüfen. Stecke die RTC in die Stiftleiste, wobei darauf zu achten ist, dass sich die RTC-Pins an der richtigen Stelle befinden. Verwende für den Test das RTC-Beispiel.

Bevor du den DHT11-Sensor überprüfst, füge die langen Drähte an, die nach draußen führen. Verwende aus Konsistenzgründen beim Draht dieselbe Farbe wie beim Shield. Benutze an dieser Stelle Litze, weil sie flexibler sind (siehe Abbildung 8-13). Schiebe für ein professionelleres Erscheinungsbild sechs kurze Stücke Schrumpfschlauch über die Drähte (zwei bei jedem Draht), bevor du lötest. Schiebe, nachdem du die Drähte an den Stiftleisten und am Sensor festgelötet hast, den Schrumpfschlauch über die Lötstellen und schrumpfe ihn, damit er an Ort und Stelle gehalten wird. Ich verwende gerne durchsichtigen Schrumpfschlauch, damit ich sehen kann, wenn eine Lötstelle bricht, und weil mit durchsichtigem Schrumpfschlauch eher das Prinzip von Offenheit transportiert wird.

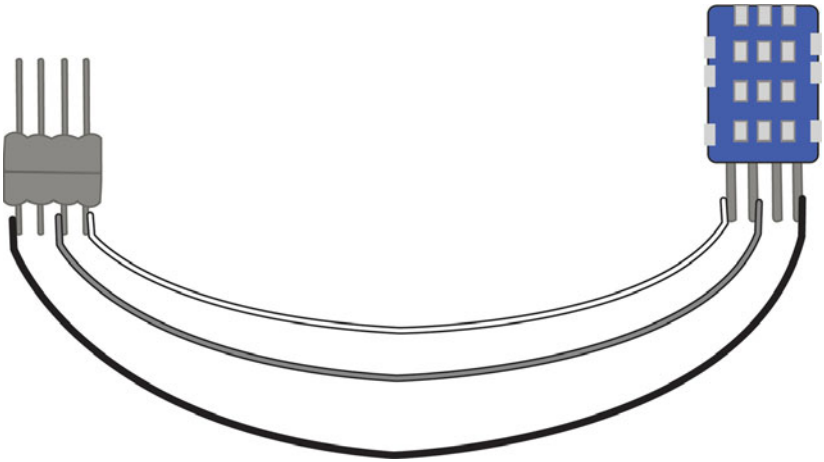


Abbildung 8-31: *Das Anbringen langer Litzen und männlicher Stiftleisten am DHT11-Sensor*

Massivdraht sollte nur an solchen Stellen verwendet werden, an denen er nicht bewegt wird, z.B. wenn beide Enden an dasselbe Board gelötet werden.

Litze sollte immer dann verwendet werden, wenn Komponenten verbunden werden sollen, die zueinander bewegt werden, z.B. zwei Boards oder ein Board mit einem Anschluss.

Ich habe viele Fehlfunktionen bei Projekten gesehen, die deshalb auftraten, weil der Draht durch zu häufiges Bewegen gebrochen war.

Stecke die männliche Stiftleiste des DHT11-Sensors in die entsprechende weibliche Stiftleiste und achte wieder sorgfältig darauf, dass sich der richtige Pin an der richtigen Stelle befindet. Benutze zum Testen das DHT-Testbeispiel.

Einbau deines Projekts in ein Gehäuse

Nun müssen wir das Projekt in ein Gehäuse montieren. Die beste Wahl ist ein Gehäuse, das nicht zu tief ist, damit ein leichter Zugang möglich ist, und bei dem beim Layout noch genügend Platz um die einzelnen Komponenten herum bleibt. Denke daran, dass auf der Oberseite des Arduino ein Shield sitzt und die RTC möglicherweise vertikal angebracht ist, wodurch sich eine größere Höhe ergibt. Der Arduino sollte auf einem kleinen Fuß, der als *Abstandsbolzen* bezeichnet wird, montiert werden. Mit einer Schraube wird der Arduino am Abstandsbolzen befestigt und mit einer weiteren Schraube an der Rückwand des Gehäuses der Abstandsbolzen am Gehäuse, wie in Abbildung 8-32 dargestellt.

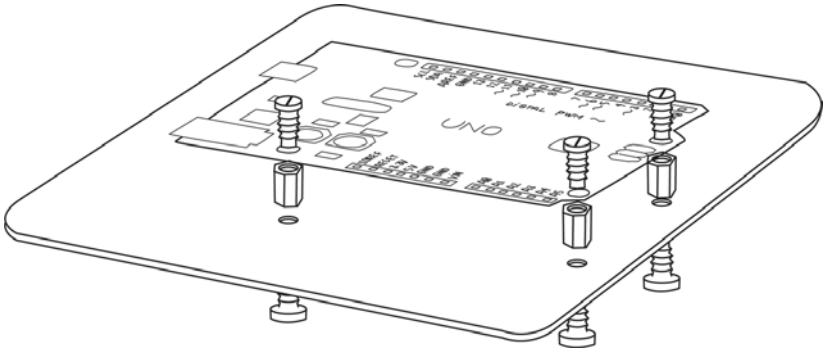


Abbildung 8-32: *Der auf einem Abstandsbolzen in einem Gehäuse montierte Arduino*

Plane immer für ein größeres Gehäuse als du denkst, dass du es benötigen wirst. Vergiss nicht die Stromversorgungen und Anschlüsse, und denke daran, dass auch für die Drähte Platz erforderlich ist. Du solltest die Drähte zwischen Board und Shield und nicht über die Boards führen, damit sie dich nicht stören, wenn du an Komponenten arbeiten oder sie entfernen musst. Für eine schnelle, schmutzige Verkabelung benutze ich gerne solche selbsthaftenden Kabelbinder, wie sie in Abbildung Abbildung 8-33 zu sehen sind.

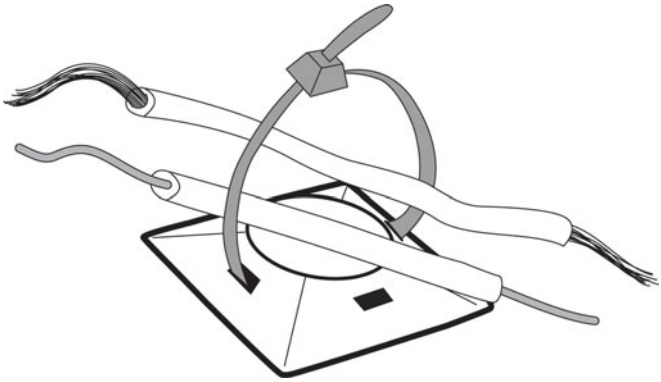


Abbildung 8-33: *Selbsthaftender Kabelbinder mit Drähten*

Bei Projekten mit verschiedenen Stromversorgungen, wie es hier der Fall ist, solltest du dir überlegen, eine Steckerleiste im Gehäuse zu montieren. Verwende ein starkes doppelseitiges Klebeband, um die Steckerleiste zu befestigen. Wenn deine Stromversorgung nur zwei Zinken besitzt, kannst du eine entsprechende Verlängerung mit zwei Zinken und mindestens zwei Steckdosen anstelle der größeren Steckerleiste mit drei Zinken verwenden.

Dies bedeutet eine weitere Revision unserer Einkaufsliste (wir sind nun bei Revision 0.6 angelangt):

- Gehäuse
- Abstandsbolzen
- Befestigungsschrauben und/oder Muttern
- Kabelbinder
- selbsthaftende Kabelbinder, erhältlich beispielsweise bei Jameco (*http://bit.ly/15p5rj5*)
- starkes doppelseitiges Klebeband
- Steckerleiste

Ich halte die Stromversorgungen gerne ein wenig vom Arduino entfernt. Du solltest den Arduino nahe an die Unterseite montieren, so dass die Kabel der Wasserventile und des DHT11-Sensors von der Unterseite aus eingeführt werden können – aber auch nicht zu nah. Du wirst glücklich über genügend Platz für deine Hände und den Schraubenzieher sein, wenn du die Drähte mit den Schraubklemmen verbindest oder wenn du am Arduino arbeiten musst.

Das USB-Kabel kann auf jeder Seite herausgeführt werden.

Die Drähte sollten immer in geraden Linien verlaufen und sauber gebündelt sein. Dadurch wird die spätere Arbeit am Projekt sehr viel leichter.

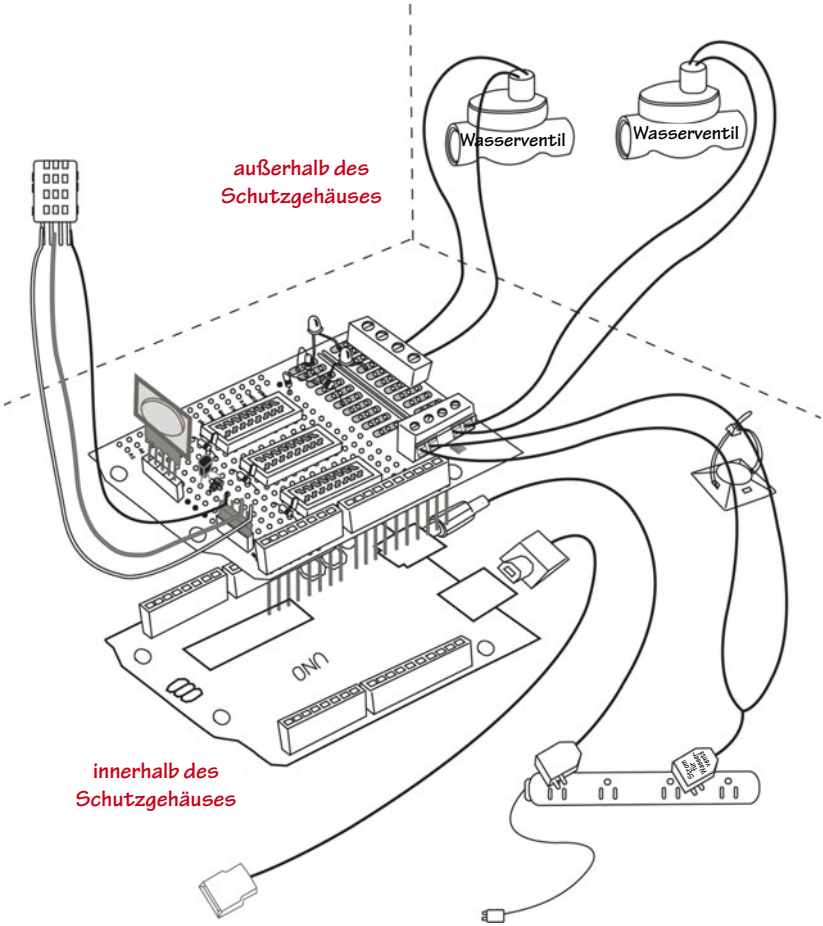


Abbildung 8-34: *Das komplette Gartenbewässerungssystem*

In dieser Abbildung habe ich viele Komponenten beim Proto Shield ausgelassen, damit leichter zu erkennen ist, wie die Bauteile verbunden sind. Ich zeige hier einen Kabelbinder, du solltest aber so viele verwenden, wie erforderlich sind, um die Verkabelung in der gewünschten Ordnung und sauber zu halten. Füge immer einen Kabelbinder an, bevor ein Kabel aus dem Gehäuse herausgeführt wird; dies dient als Zugentlastung, falls am Kabel gezogen wird: In diesem Fall wird nämlich der Kabelbinder belastet und nicht dein empfindlicher Schaltkreis.

Achte wie gehabt auf die Polung der Stromversorgung für die Wasserventile, falls es sich um ein Gleichstromsystem handelt.

Testen des fertigen Systems für die automatische Gartenbewässerung

Teste deine Projekte zunächst immer individuell in Modulen, und zwar auf jede Weise, die das Projekt zulässt.

Beginne mit dem Testen des Arduino und des Proto Shields ohne angeschlossene Stromversorgung. Dies bedeutet, dass dein Computer den Arduino mit Strom versorgt. Benutze wie zuvor das Blink-Beispiel, um den digitalen Ausgang zu testen. Die LEDs werden ohne die Stromversorgung für die Wasserventile nicht leuchten, du solltest aber das Relais klicken hören. Benutze DHTtester, um den DHT11-Sensor zu testen, und das *ds1307*-Beispiel, um die RTC zu überprüfen.

An dieser Stelle entsteht möglicherweise der Eindruck, dass du die Tests, die nach der Bestückung des Proto Shields erfolgten, doppelt durchführst. Hierfür gibt es einen guten Grund: Bevor du fortfährst, solltest du sicherstellen, dass die eben durchgeführten Arbeitsschritte keine Auswirkungen auf funktionierende Teile des Systems haben.

Schließ nun die Stromversorgung des Arduino an und trenne ihn von deinem Laptop, um sicherzustellen, dass dein Arduino ohne dein Laptop mit Strom versorgt wird. Lass eines der Relais klicken (indem du das Blink-Beispiel benutzt), um zu hören, ob dein Arduino immer noch den Sketch ausführt.

Schließ dann die Stromversorgung für das Wasserventil an, lade den realen Sketch und teste die Ventile wie zuvor, indem du drei verschiedene Zeiten in der nahen Zukunft angibst. Überprüfe, ob jede LED zu der entsprechenden Zeit leuchtet, ob sich die Wasserventile öffnen und Wasser fließt.

Nun kannst du dich in deinem Garten entspannen und die wohlverdiente Pause genießen. Du hast eine Menge geschafft!

Gratuliere! Das war ein kompliziertes Projekt! Mach ein paar Fotos hiervon, gib ein wenig damit auf Facebook an und poste es im Arduino-Blog (*http://bit.ly/1rcG2Tz*).

Ideen zum Ausprobieren

Es handelt sich hier um ein sehr komplexes Projekt mit vielen verschiedenen Komponenten. Es gibt zahlreiche mögliche Variationen. Hier einige Vorschläge:

- Modifiziere das Programm dahingehend, dass das Wasser zu verschiedenen Zeiten eines Tages ein- und ausgeschaltet werden kann.
- Füge den Wochentag hinzu und lasse unterschiedliche Zeitpläne an den verschiedenen Wochentagen zu.

- Füge eine LED hinzu, die anzeigt, dass keine ON- und OFF-Zeiten festgelegt wurden. Diese LED soll bei einem Reset angeschaltet werden und zum ersten festgelegten Zeitpunkt erlöschen. Dies ist nützlich, wenn die Stromversorgung des Arduino unterbrochen wird und ein Reset erfolgt, bei dem alle deine ON- und OFF-Einstellungen verloren gehen.
- Füge eine kleine LCD-Anzeige hinzu, um die aktuelle Zeit und die aktuell gültigen Einstellungen anzuzeigen.
- Eine Übung für Fortgeschrittene: Das RTC-Modul, das wir verwendet haben, verfügt ebenfalls über einen kleinen Speicherchip, der nichts vergisst, wenn die Stromzufuhr unterbrochen wird. Du kannst untersuchen, wie sich diese Tatsache nutzen lässt, und die Ventileinstellungen für ON und OFF in diesem Speicher sichern, damit sie im Falle eines Arduino-Resets erhalten bleiben.

Die Einkaufsliste für das Bewässerungsprojekt

Um es dir möglichst einfach zu machen, hier die finale Einkaufsliste. Alle Bauteile sind im deutschsprachigen Versandhandel erhältlich:

- eine Echtzeituhr (RTC)
- ein Temperatur- und Feuchtigkeitssensor vom Typ DHT11
- ein Arduino-Proto-Shield
- drei elektrische Wasserventile)
- ein Transformator oder eine Stromversorgung für die Wasserventile
- drei Relais zur Steuerung der Wasserventile
- drei Relais-Sockel
- drei LEDs als Aktivierungsanzeige für die Ventile
- drei 1K-Widerstände für die LEDs
- eine Stromversorgung für den Arduino
- drei MOSFETs zur Steuerung der Relais, 2N7000, 10er-Packung
- vier Widerstände, 10kOhm, 10er-Packung
- sechs Dioden, 1N4148 oder gleichwertig, 25er-Packung
- drei Relais, z.B. ein DS2E-S-DC5V
- vier Schraubklemmen mit zwei Positionen
- weibliche Stiftleisten, .1"-Raster, 5er-Packung mit 20 Pins pro Streifen
- männliche Stiftleisten, .1"-Raster, 10er-Packung mit 36 Pins pro Streifen
- eine Plastikbox, z.B. von Sterilite (*http://bit.ly/1vK02Pl*), oder eine hübsche Metallbox, erhältlich bei Automation 4 Less (*http://bit.ly/1HElg38*)

- Abstandsbolzen
- Schraube zur Befestigung des Abstandsbolzens am Gehäuse
- Schraube zur Befestigung des Arduino am Abstandsbolzen
- Kabelbinder
- selbsthaftender Kabelbinder
- starkes doppelseitiges Klebeband
- Steckerleiste oder Verlängerungskabel mit mindestens zwei Steckdosen

9/Troubleshooting

Es wird bei deinen Experimenten immer ein Moment kommen, in dem nichts funktioniert und du herausfinden musst, wie sich der Fehler beheben lässt. Troubleshooting und Debugging sind historische Disziplinen, bei denen ein paar einfache Regeln gelten, die meisten Resultate werden aber dadurch erzielt, dass die Arbeit mit großer Sorgfalt ausgeführt und auf die Details geachtet wird.

Das Wichtigste hierbei ist, immer daran zu denken, dass du nicht gescheitert bist! Die meisten Maker, Amateure und Profis verwenden die meiste Zeit darauf, Fehler zu beheben, die sie selbst gemacht haben. (Es stimmt natürlich, dass wir beim Suchen und Beheben von Problemen immer besser werden, wir erzeugen allerdings dabei auch kompliziertere Probleme.)

Wenn du mehr mit Elektronik und Arduino arbeitest, wirst du auch mehr lernen und dein Erfahrungsschatz wird größer, wodurch der Prozess der Fehlerbehebung immer weniger zermürbend wird. Lass dich durch die auftretenden Probleme nicht entmutigen. Alles ist einfacher, als es am Anfang aussieht. Je mehr Fehler du machst und korrigierst, desto besser wirst du darin, sie zu suchen und zu beheben.

Da sich jedes Arduino-basierte Projekt aus Hardware und Software zusammensetzt, gibt es immer mehrere Stellen, an denen du nachschauen musst, ob etwas nicht richtig funktioniert. Wenn du einen Fehler suchst, solltest du dich an drei Prinzipien orientieren: Verständnis, Vereinfachung und Segmentierung sowie Ausschluss und Vergewisserung.

Verständnis

Versuch' so weit wie möglich zu verstehen, wie die Bauteile funktionieren, die du verwendest, und welchen Beitrag sie für das fertige Projekt leisten. Dieser Ansatz ermöglicht es dir, Techniken für das separate Testen jeder einzelnen Komponente zu entwickeln. Wenn du das noch nicht getan hast, zeichne einen Schaltplan deines Projekts. Er erleichtert das Verständnis und ist außerdem hilfreich, wenn du jemanden um Hilfe bitten musst. Schaltpläne werden in Anhang D erläutert.

Vereinfachung und Segmentierung

Von den alten Römern stammt der Ausspruch *divide et impera*: Teile und herrsche! Zerlege dein Projekt (mental) in seine Komponenten, indem du dein Wissen anwendest, und finde heraus, wo die Zuständigkeit jeder Komponente beginnt und wo sie endet.

Ausschließen und Vergewissern

Teste bei der Fehlersuche jede Komponente einzeln, so dass du absolut sicher sein kannst, dass jede für sich genommen einwandfrei funktioniert. Dadurch wirst du 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 real existierende 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. Du musst den unsichtbaren Bug so austricksen, dass er sich wirklich zeigt.

Debugging ist ein wenig wie Detektivarbeit. Es tritt eine Situation ein, die erklärt werden muss. Hierzu führst du einige Experimente durch und erhältst entsprechende Resultate. Von diesen Resultaten versuchst du dann abzuleiten, was die Situation verursacht hat. Dies ist von grundlegender Bedeutung, wirklich!

Testen des Board

Bevor du sehr komplizierte Experimente durchführst, tust du gut daran, erst einmal die einfachen Dinge zu überprüfen, besonders wenn dies nicht viel Zeit erfordert. Als Erstes solltest du überprüfen, ob dein Arduino-Board funktioniert. Unser allererstes Beispiel, Blink, ist immer ein guter Ausgangspunkt, weil du hiermit wahrscheinlich am besten vertraut bist und wegen der LED, die sich bereits auf deinem Arduino befindet und den Vorteil hat, dass du nicht von externen Komponenten abhängig bist.

Folge diesen Schritten, bevor du dein Projekt mit deinem Arduino verbindest. Wenn du dein Projekt bereits mittels Drahtbrücken mit deinem Projekt verbunden hast, entferne sie vorerst, behalte aber sorgfältig im Auge, wohin welcher Draht führen soll.

Öffne das Blink-Basisbeispiel in der Arduino-IDE und lade es auf dein Board. Die eingebaute LED sollte in einem regelmäßigen Muster blinken.

Das Blink-Beispiel funktioniert nicht?

Bevor du mit dem Arduino-Projekt haderst, solltest du dich wie ein Pilot, der vor dem Start eine Checkliste durchgeht, um die Flugtüchtigkeit des Flugzeugs sicherzustellen, zunächst vergewissern, dass einige Dinge in Ordnung sind:

- Wird dein Arduino mit Strom versorgt, sei es über einen Computer mittels eines USB-Kabels oder über eine externe Stromversorgung? Wenn das grüne Lämpchen mit der Bezeichnung PWR leuchtet, bedeutet dies, dass dein Arduino Strom erhält. Wenn die LED nur sehr schwach leuchtet, stimmt etwas mit der Stromversorgung nicht.
- Wenn du einen Computer benutzt, stelle noch einmal sicher, dass er eingeschaltet ist (ich weiß, das klingt dumm, aber hier liegt tatsächlich des öfteren die Ursache eines vermeintlichen Fehlers). Versuche es dann mit einem anderen USB-Kabel und inspiziere den USB-Anschluss des Computers und die USB-Buchse des Arduino-Boards, um sicherzustellen, dass sie nicht beschädigt sind. Verwende erforderlichenfalls einen anderen USB-Anschluss deines Rechners oder gleich einen anderen Computer. Falls eine Menge USB-Kabel auf deiner Arbeitsplatte liegen, vergewissere dich, dass das Kabel, das in deinen Arduino eingesteckt ist, auch wirklich das Kabel ist, das mit deinem Computer verbunden ist (ja, auch solche Verwechslungen kennen wir aus persönlicher Erfahrung).
- Wenn du eine externe Stromversorgung benutzt, vergewissere dich, dass die externe Stromversorgung eingesteckt ist. Stelle sicher, dass die Steckerleiste oder das Verlängerungskabel eingesteckt ist. Falls du eine Steckerleiste mit Schalter benutzt, überprüfe, ob er eingeschaltet ist.

  (Falls du einen sehr alten Arduino benutzt, vergewissere dich, dass der Power Selection Jumper sich in der korrekten Position befindet. Bei modernen Arduinos geschieht dies automatisch, es ist kein solcher Jumper vorhanden.)
- Bei einem brandneuen Computer wird möglicherweise die gelbe LED mit der Bezeichnung *L* bereits blinken, bevor du das Blink-Beispiel geladen hast. Dies ist wahrscheinlich das von Haus aus enthaltene Testprogramm, mit dem das Board getestet wird, und das Blinken ist in Ordnung. Es ist auch völlig in Ordnung, wenn kein Blinken zu beobachten ist – dies bedeutet einfach, dass ein anderes Testprogramm auf dem Arduino ausgeführt wird.
- Vergewissere dich, dass der Sketch erfolgreich geladen wurde.

  Wenn das Laden fehlgeschlagen ist, überprüfe zunächst, ob dein Programm fehlerfrei ist, indem du auf Verify klickst.

  Stelle sicher, dass du das korrekte Board im Tools-Menü ausgewählt hast. Wenn du damit beginnst, mit unterschiedlichen Arduino-Boards zu arbeiten, ist es eine gute Angewohnheit, sich immer noch einmal zu

vergewissern, dass das ausgewählte Board auch tatsächlich das ist, das du angeschlossen hast.

Überprüfe, ob der Port im Tools-Menü korrekt ausgewählt wurde. Wenn du die Verbindung mit deinem Arduino zu irgendeinem Zeitpunkt trennst, wird er möglicherweise auf einem anderen Port angezeigt.

Manchmal ist es erforderlich, die Verbindung mit dem Arduino zu trennen und dann wiederherzustellen. Wenn das Menü für die Auswahl des seriellen Ports geöffnet ist, musst du es schließen (öffne einfach eine andere Registerkarte) und dann wieder über Tools→Serial Port den korrekten Port auswählen.

Starte den Upload erneut. In seltenen Fällen schlägt der Ladevorgang ohne erkennbaren Grund fehl und gelingt beim nächsten Versuch ohne irgendwelche Änderungen.

USB-Kabel von schlechter Qualität sind manchmal der Grund dafür, dass der Treiber den Arduino Uno nicht finden kann. Wenn dein Arduino-Port in der Port-Liste nicht angezeigt wird, verwende ein USB-Kabel von bekanntermaßen guter Qualität.

Sobald du das Blink-Basisbeispiel geladen hast und die LED blinkt, kannst du darauf vertrauen, dass dein Arduino über die grundlegende Funktionalität verfügt, und mit dem nächsten Schritt fortfahren.

Testen des Schaltkreises auf der Steckplatine

Der nächste Schritt besteht darin, dein Projekt auf Kurzschlüsse zwischen 5V-Anschluss und GND-Anschluss zu überprüfen. Verbinde deinen Arduino mit der Steckplatine, indem du mit einer Steckbrücke eine Verbindung vom 5V-Anschluss und vom GND-Anschluss zu der positiven und der negativen Leiterbahn auf der Steckplatine herstellst. (Beachte, dass wir das Teile und herrsche-Prinzip befolgen, indem wir nur diese beiden Drahtbrücken und nicht alle Drahtbrücken dieses Projekts verbinden.) Wenn die grüne PWR-LED nicht mehr leuchtet, entferne sofort die Verdrahtung. In diesem Fall gibt es einen größeren Fehler im Schaltkreis und du hast irgendwo einen Kurzschluss verursacht. Dein Board bezieht dann zu viel Strom, und die Energieversorgung wird abgeschnitten, um das Board zu schützen.

Wenn du nun Angst hast, deinen Computer zu beschädigen, solltest du bedenken, dass bei vielen Computern der Überspannungsschutz sehr gut ist und auch rasch reagiert. Außerdem ist das Arduino-Board mit einer selbstrückstellenden Sicherung ausgestattet, einem speziellen Bauteil für den Überspannungsschutz, das sich selbst zurücksetzt, wenn der Fehler behoben ist.

Wenn du schon Paranoia hast, kannst du das Arduino-Board auch immer über einen Self-Powerd USB-Hub anschließen. In diesem Fall könnte schlimmstenfalls der USB-Hub zerstört werden. Der Computer bliebe unversehrt.

Wenn du einen Kurzschluss verursachst, musst du mit dem Prozess Vereinfachung und Segmentierung fortfahren. Überprüfe jeden Sensor im Projekt und schließe dabei immer nur einen an – bis du das Bauteil oder die Verbindung gefunden hast, durch das bzw. die der Kurzschluss verursacht wurde.

Beginne immer mit der Stromversorgung (die Verbindungen vom 5V-Pin und vom GND-Pin). Schau dir alles an und vergewissere dich, dass jedes Bauteil im Schaltkreis korrekt mit Strom versorgt wird. Der wahrscheinlichste Grund ist eine Drahtbrücke, die sich an der falschen Stelle befindet. Andere Gründe können in falschen Komponenten wie einem Widerstand mit einem zu geringen Wert oder einem Schalter oder Transistor, der 5V mit Masse verbindet, bestehen. Ein unwahrscheinlicherer, aber dennoch möglicher Grund könnte darin bestehen, dass ein Drahtstück oder eine Schraube irgendwo sowohl 5V als auch Masse berührt.

> Das Arbeiten in Einzelschritten mit niemals mehr als einer Änderung zur gleichen Zeit ist die golden Regel bei der Behebung von Fehlern. Diese Regel wurde mir von meinem Professor 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 gleichen Zeit ... immer nur eine Änderung zur gleichen 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 gleichen Zeit durchzuführen.)
>
> Massimo

Mit jeder Debugging-Erfahrung entsteht in deinem Kopf Schritt für Schritt eine Wissensbasis im Hinblick auf Fehler und mögliche Lösungen, und bevor du es weißt, bist du schon ein Experte. Das verleiht dir eine gewisse Souveränität, denn wenn sich dann ein Neuling darüber beklagt, dass etwas nicht funktioniert, schaust du dir die Sache kurz an und hast im Bruchteil einer Sekunde eine Lösung parat.

Das Isolieren von Problemen

Eine weitere wichtige Regel ist das Finden eines zuverlässigen Weges zur Problemreduzierung. Wenn dein Schaltkreis sich an scheinbar zufälligen Zeitpunkten merkwürdig verhält, versuche wirklich mit aller Hartnäckigkeit herauszufinden, worin der Grund hierfür bestehen könnte. Tritt das Problem nur auf, wenn du einen Schalter betätigst? Oder nur, wenn eine LED zu leuchten beginnt? Immer wenn du eine Drahtbrücke bewegst? (Viele Probleme werden durch lose Drähte verursacht, indem sie Komponenten nicht verbinden, die sie verbinden sollten, oder aber solche Teile verbinden, die nicht verbunden werden sollten.) Versuche, die Schritte zu wiederholen, die das Problem verursacht haben. Achte dabei auf die kleinen Details und nimm immer nur eine Änderung gleichzeitig vor: Tritt das Problem jedes Mal auf, wenn die LED zu leuchten beginnt, oder nur dann, wenn du den Schalter betätigst, während die LED eingeschaltet ist? Dieser Prozess ermöglicht dir, über eine mögliche Ursache nachzudenken. Außerdem ist er hilfreich, wenn du jemand anderem erklären möchtest, was eigentlich geschieht.

Eine möglichst präzise Beschreibung des Problems ist auch eine gute Methode, eine Lösung zu finden. Suche jemanden, dem du das Problem erläutern kannst. Oft wird dir eine Lösung einfallen, sobald du das Problem artikulierst. 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 bereitstand. Studenten mit rätselhaften Bugs mussten zuerst dem Stofftier das Problem erläutern, bevor sie sich an einen menschlichen Berater wenden durften.

Probleme beim Installieren von Treibern unter Windows

Manchmal schafft es der Windows Hardware-Installationsassistent nicht, die korrekten Treiber finden. In diesem Fall musst du dem Programm möglicherweise manuell mitteilen, wo sich der Treiber befindet.

Der Hardware-Installationsassistent wird dich zunächst fragen, ob nach Windows Update gesucht werden soll; wähle die Option Nicht jetzt und klicke dann auf Weiter.

Wähle im nächsten Bildschirm Aus einer Liste von Geträtetreibern auf dem Computer auswählen und klicke wieder auf Weiter.

Navigiere zur Treiberdatei für den Uno, die den Namen *ArduinoUNO.inf* trägt und sich im Ordner *Drivers* des Downloads der Arduino-Software befindet (nicht im Unterverzeichnis *FTDI USB Drivers*), und wähle sie aus. Von hier an wird Windows die Treiberinstallation fertigstellen.

Probleme mit der IDE unter Windows

Wenn dir nach einem Doppelklick auf das Arduino-Symbol eine Fehlermeldung angezeigt wird oder einfach nichts geschieht, versuche als alternative Methode, Arduino mit einem Doppelklick auf die Datei `Arduino.exe` zu starten.

Windows-Nutzer werden außerdem mit einem Problem konfrontiert, wenn das Betriebssystem dem Arduino eine COM-Anschlussnummer COM10 oder größer zuweist. In diesem Fall kannst du normalerweise Windows dazu veranlassen, dem Arduino eine kleinere Anschlussnummer zuzuweisen, indem (temporär) ein COM-Port mit einer niedrigeren Nummer freigegeben wird. Öffne dazu zunächst den Geräte-Manager, indem du auf Start und dann mit der rechten Maustaste auf Computer (Vista) oder My Computer (XP) klickst und dann Eigenschaften auswählst. Klicke unter Windows XP dann auf Hardware und wähle Geräte-Manager aus. Unter Vista musst du auf Device Manager klicken (er wird in der Task-Liste in der linken Fensterhälfte aufgelistet).

Suche in der Liste unter Anschlüsse (Com & LPT) die seriellen Geräte. Wähle ein serielles Gerät aus, das du nicht benutzt und dem COM9 oder niedriger zugewiesen wurde. Gute Kandidaten sind ein Modem oder eine serielle Schnittstelle. Wähle nach einem rechten Mausklick auf dieses Gerät Eigenschaften aus dem entsprechenden Menü aus. Klicke auf die Registerkarte Anschlusseinstellungen und dann auf Erweitert. Lege für die COM-Anschlussnummer COM10 oder höher fest, klicke auf OK und dann nochmal auf OK, um den Dialog Einstellungen wieder zu schließen.

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

Identifizierung der Arduino-COM-Ports unter Windows

Verbinde deinen Arduino Uno unter Verwendung eines USB-Kabels mit deinem Computer.

Öffne den Geräte-Manager, indem du auf das Startmenü und dann mit der rechten Maustaste auf Computer (Vista) oder My Computer (XP) klickst und dann Einstellungen auswählst. Klicke unter Windows XP auf Hardware und wähle dann Geräte-Manager. Klicke unter Vista auf Device Manager (das Programm wird in der Task-Liste auf der linken Seite des Fensters angezeigt).

Suche den Arduino in der Liste unter Ports (COM & LPT). Der Arduino wird als Arduino UNO angezeigt und einen Namen wie COM7 haben, wie in Abbildung 9-1 dargestellt.

Auf einigen Windows-Geräten besitzt der COM-Port eine Nummer, die größer als 9 ist; diese Nummerierung erzeugt einige Probleme, wenn der Arduino versucht, mit ihm zu kommunizieren.

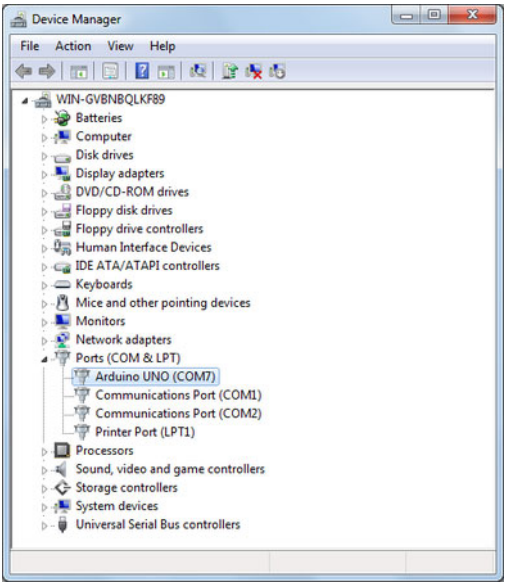


Abbildung 9-1: *Der Windows Geräte-Manager zeigt alle verfügbaren seriellen Schnittstellen an.*

Andere Debugging-Techniken

- Bitte jemand anderen, über dein Projekt zu schauen. Wir sind unseren eigenen Fehlern gegenüber manchmal blind. Sag der anderen Person nicht, welche Verbindungen du beabsichtigt hattest. Lass sie überprüfen, ob du die Schaltskizze, mit der du arbeitest, korrekt umgesetzt hast. Auf diese Weise verhinderst du eine gewisse Voreingenommenheit, d. h. dass die Person genau das sieht, was du beabsichtigt hattest – und prompt die Fehler übersieht. (Wenn du keine Schaltskizze hast, solltest du eine anfertigen.)

- Teile und herrsche funktioniert auch bei Sketchen. Speichere eine Kopie deines Sketches, beginne dann damit, die Code-Bereiche zu entfernen, die nichts mit dem Teil zu tun haben, der dir Probleme bereitet. Möglicherweise findest du eine unerwartete Interaktion zwischen etwas, was einwandfrei zu funktionieren scheint, und dem Bereich, der das Problem verursacht. Wenn das Problem dadurch nicht gelöst wird, steht dir ein minimales Testprogramm zur Verfügung, das das Problem zeigt und eine große Hilfe ist, wenn du Rat suchst.
- Wenn dein Projekt irgendwelche Sensoren (einschließlich Schalter) umfasst, teste jeden dieser Sensoren einzeln mit den einfachsten entsprechenden Beispielen: AnalogReadSerial und DigitalReadSerial, die du unter *File→Examples→01.Basics→AnalogReadSerial/DigitalReadSerial* findest.
- Wenn ein Fehler bei einem Sensor auftritt, stelle sicher, dass der Arduino-Eingang korrekt arbeitet. Nimm den Sensor aus dem Schaltkreis und füge eine Drahtbrücke ein, mit der du die verdächtige Stelle direkt mit 5V und Masse verbindest (nacheinander selbstverständlich), während du mittels AnalogReadSerial oder DigitalReadSerial entsprechende Beobachtungen anstellst. Du solltest 0 sehen, wenn der Eingang mit Masse verbunden ist, und 1 oder 1023, wenn eine Verbindung mit 5V besteht.

  Wenn du mehrere Sensoren einsetzt und bei einem ein Fehler auftritt, solltest du Teile der Schaltung zwischen den funktionierenden Sensoren und dem, bei dem der Fehler aufgetreten ist, austauschen (immer nur ein Teil gleichzeitig). Beobachte die entsprechenden Auswirkungen auf das Problem.
- Wenn dein Projekt irgendwelche Aktoren umfasst, teste jeden einzeln mit den einfachsten entsprechenden Beispielen: Blink oder Fade. Wenn ein Fehler bei einem Aktor auftritt, ersetze ihn durch eine LED, um sicherzustellen, dass der Arduino-Ausgang korrekt arbeitet.
- Wenn dein Sketch Entscheidungstechniken umfasst, z.B. `if`-Anweisungen, verwende die Funktion `Serial.println()`, um anzuzeigen, was jeweils geschieht. Dies ist auch bei Schleifen hilfreich, um sicherzustellen, dass sie auch dann beendet werden, wenn sie beendet werden sollten.
- Wenn du irgendwelche Bibliotheken verwendest, überprüfe, ob sie auch korrekt funktionieren, indem du die jeweiligen mitgelieferten Beispiele benutzt. Wenn Probleme bei einer Bibliothek auftreten, die nicht von Arduino stammen, suche ein entsprechendes Forum oder eine andere Online-Community für diese Bibliothek und nimm Verbindung auf.

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

So findest du Online-Hilfe

Wenn du bei einem Problem festhängst, solltest du nicht tagelang alleine herumbasteln – bitte einfach um Hilfe. Einer der größten Vorteile des Arduino ist seine Community. Du kannst jederzeit Hilfe finden, wenn du das aufgetretene Problem gut beschreibst.

Du solltest dir angewöhnen, per Cut&Paste Stichworte in eine Suchmaschine einzugeben, damit du herausfindest, ob jemand einen entsprechenden Beitrag veröffentlicht hat. Wenn beispielsweise die Arduino-IDE eine merkwürdige Fehlermeldung anzeigt, kannst du sie einfach in die Google-Suchmaschine kopieren und dir die Suchergebnisse anschauen. Möglicherweise muss du die Nachricht in Anführungszeichen setzen, damit die einzelnen Wörter nicht in einer zufälligen Reihenfolge gesucht werden. Das funktioniert auch bei Ausschnitten aus Code, an dem du gerade arbeitest, und bei speziellen Funktionsnamen. Wenn du zu viele unpassenden Treffer erhältst, füge das Wort *Arduino* zu deiner Suche hinzu.

Schau dich einfach um: Alles wurde bereits erfunden und ist auf irgendeiner Webseite gespeichert. Ich bin immer wieder überrascht, wie oft Dinge, von denen ich glaube, dass sie nur mir passieren, im Web bereits gut dokumentiert sind – zusammen mit den entsprechenden Lösungen.

Für weitere Nachforschungen kannst du auch die Hauptseite von Arduino aufrufen (*http://www.arduino.cc*), dir die FAQs anschauen (*http://www.arduino.cc/en/Main/FAQ*) und dann zum Playground wechseln (*www.arduino.cc*). 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 du mit einem Projekt beginnst, durchsuche zunächst den Playground: Du wirst sicherlich Code-Abschnitte und Schaltdiagramme finden, auf denen du aufbauen kannst.

(Erwäge auch, der Open-Source-Community etwas zurückzugeben, indem du ein Projekt veröffentlichst, mit dem du dich beschäftigt hast, oder eine Lösung beiträgst, von der du weißt, dass sie bis dato noch nicht dokumentiert wurde.)

Wenn du über all diese Wege immer noch keine Antwort erhalten hast, besuche das Arduino-Forum (*http://www.arduino.cc/*). Wenn du alles andere versucht hast, kannst du hier eine entsprechende Frage posten. Wähle das richtige Board für deine Frage aus: Es gibt verschiedene Bereiche für Software- und Hardware-Probleme und außerdem Foren in fünf verschiedenen Sprachen. Wenn du dir unsicher bist, welches Board das geeignete ist, poste die betreffende Frage im Project Guidance-Board.

Formuliere deinen Beitrag sorgfältig. Poste möglichst viele Informationen und achte dabei auf Klarheit und Sorgfalt. Nimm dir die Zeit, dein Problem deutlich und korrekt zu beschreiben, denn diese Zeit ist gut investiert. Au-

ßerdem zeigt sich hier, dass du bereits so viel wie möglich selbst getan hast und nicht darauf vertraust, dass das Forum deine Arbeit für dich erledigt. Hier einige Richtlinien:

- Lies den Post mit dem Titel „How to use this forum – please read“ (*http://bit.ly/1y9jh2u*).
- Welches Arduino-Board verwendest du?
- Unter welchem Betriebssystem führst du die Arduino-IDE aus?
- Welche Version der Arduino-IDE verwendest du?
- Füge eine allgemeine Beschreibung dessen an, was du zu tun versuchst. Wenn du ungewöhnliche Bauteile verwendest, poste Links zu den entsprechenden Datenblättern. Stopfe deinen Post nicht mit unnützen Informationen wie dem Projektkonzept oder einer Abbildung des Gehäuses voll, wenn sie nicht in einem Zusammenhang mit dem Problem stehen.
- Poste den Minimal-Sketch und/oder den Schaltkreis (Schaltskizzen sind hier hervorragend geeignet), anhand dessen das Problem deutlich wird. (Du hast es beim Debuggen gefunden, richtig?) Die Posts in How to use this forum zeigen, wie Code zu formatieren ist und wie Anhänge einzufügen sind.
- Wenn du im Forum nach Hilfe suchst, beachte die dort vorherrschende Kultur, besonders im Hinblick darauf, mit welcher Art Fragen gute Hilfestellungen erzielt werden und mit welchen Fragen nicht. Es empfiehlt sich, den Stil der Fragen, die funktionieren, zu kopieren.
- Beschreibe exakt, was geschehen sollte und was stattdessen geschieht. Schreibe nicht einfach Es funktioniert nicht. Falls du eine Fehlermeldung erhältst, poste sie. Wenn bei deinem Programm eine Ausgabe erfolgt, füge diese Ausgabe an.
- Wenn du dein Problem sorgfältig beschrieben hast, schau noch einmal deinen Betreff an und überarbeite ihn gegebenenfalls. Im Betreff sollte das technische Problem zusammengefasst und nicht das Ziel deines Projekts formuliert werden (z.B. Das Betätigen mehrerer Schalter verursacht einen Kurzschluss statt Hilfe bei einem Bedienungspult für eine Rakete).

Denke daran, dass Anzahl und Geschwindigkeit der Antworten davon abhängen, wie gut du deine Frage formulierst.

Deine Chancen steigen, wenn du folgende Dinge unter allen Umständen vermeidest (diese Richtlinien gelten für alle Foren, nicht nur für das Arduino-Forum):

- Die gesamte Nachricht in GROßBUCHSTABEN schreiben. Das nervt die Leute gewaltig und du outest dich sofort als Neuling (Online-Communitys verstehen das Schreiben in Großbuchstaben als Schreien).

- Ein- und denselben Beitrag in verschiedenen Bereichen des Forums posten.
- Ständig nachfassende Kommentare wie "Hey, warum antwortet niemand?" posten oder – noch schlimmer – dieselbe Frage noch einmal senden. Wenn du keine Antwort erhältst, schau dir dein Posting noch einmal an. Wurde das Thema klar formuliert? War die Beschreibung deines Problems verständlich? Warst du freundlich? Sei immer nett.
- Nachrichten schreiben wie "Ich möchte mit dem Arduino eine Raumfähre bauen. Wie mache ich das?" Du signalisierst damit, dass andere die Arbeit für dich erledigen sollen. Diesen Ansatz finden echte Tüftler einfach nicht lustig. Du stellst besser eine spezifische Frage zu einem Teil des Projekts und packst die dazugehörigen Informationen dazu. Zusätzlich zu hilfreichen Antworten erhältst du dann möglicherweise auch nützliche Vorschläge für dein größeres Projekt.
- Eine Variation des vorherigen Punktes ist es, wenn durch die Frage offensichtlich wird, dass die postende Person dafür bezahlt wird. Wenn du eine spezifische Frage stellst, helfen dir die Leute gerne. Wenn sie aber deine Arbeit erledigen sollen (und du den betreffenden Lohn nicht teilst), wird die Antwort wahrscheinlich nicht sehr nett ausfallen.
- Nachrichten posten, die verdächtig nach Hausaufgaben aussehen und mit denen des Forum gebeten wird, deine Arbeit für dich zu erledigen. Professoren wie ich durchkämmen das Netz und strafen solche Studenten gnadenlos ab. Von Stammgästen des Forums werden solche Posts ebenfalls recht zuverlässig enttarnt.

A/Die Steckplatine

Wenn du einen Schaltkreis ans Laufen bringen möchtest, ist das möglicherweise mit vielen Änderungen verbunden, bis alles wirklich funktioniert. Wenn du deinen Schaltkreis immer wieder überarbeitest, kommen dir möglicherweise Ideen, die dir helfen, dein Design zu verfeinern, oder vielleicht auch, das Verhalten dahingehend zu verbessern, dass es zuverlässiger wird und weniger Teile erforderlich sind. Das Design entwickelt sich in deinen Händen, während du verschiedene Kombinationen ausprobierst. Bei diesem Prozess handelt es sich um eine Art elektronisches Pendant zum Schreiben von Sketchen. Idealerweise entwickelst du dabei ein System, bei dem sich Verbindungen zwischen Komponenten auf die schnellstmögliche, praktischste und am wenigsten destruktive Weise ändern lassen. Löten ist zwar bestens geeignet, wenn zuverlässige, permanente Schaltkreise hergestellt werden sollen, du möchtest aber vielleicht eine schnellere Lösung.

Die Antwort auf dieses Problem ist ein sehr praktisches Bauteil, das lötfreie Steckplatine genannt wird. Wie du in Abbildung A-1 siehst, handelt es sich dabei um ein kleines Plastikbrett, das komplett gelocht ist, wobei jedes Loch über einen federgelagerten Anschluss verfügt. Wenn du einen entsprechenden Draht oder den Anschluss-Pin einer Komponente in eines der Löcher steckst, wird durch die Feder die Komponente oder der Draht an Ort und Stelle gehalten. Noch wichtiger ist die Tatsache, dass die Feder mit benachbarten Löchern verbunden ist und dadurch eine elektrische Verbindung mit verschiedenen anderen Löchern hergestellt werden kann.

Im zentralen Bereich (die mit A – J gekennzeichneten Reihen) verlaufen die Federn vertikal, so dass jede Komponente, die in diesen Löchern platziert wurde, sofort mit allen anderen Komponenten verbunden ist, die mittels der Löcher derselben vertikalen Spalte angeschlossen wurden.

Einige lötfreie Steckplatinen verfügen über zusätzliche Reihen: zwei oben und zwei unten, die oft durch rote und blaue Streifen markiert und manchmal mit + und – gekennzeichnet sind. Diese Reihen sind horizontal verbunden und sind für elektrische Signale gedacht, die oft verwendet werden. Diese Reihen sind perfekt für 5V oder Masse (GND) geeignet, die die häufigsten Verbindungen in diesem Buch und bei fast allen elektronischen Projekten ausmachen. Sie werden oft als *Leiterbahnen* oder *Busse* bezeichnet.

Wenn du die rote (oder die mit + gekennzeichnete) Reihe mit 5V und die blaue (oder mit – gekennzeichnete) Reihe mit Masse/GND jeweils auf deinem Arduino verbindest, wirst du immer 5V und Masse nahe an jedem Punkt auf der Steckplatine haben.

Ein gutes Beispiel für diese Leiterbahnen wird in Kapitel 7, *Kommunikation mit der Cloud*, auf Seite 81 gezeigt.

Auf einigen Steckplatinen sind die Leiterbahnen nicht komplett durchgezogen, sondern in der Mitte unterbrochen. Dies wird manchmal durch eine Unterbrechung im roten oder blauen Streifen und manchmal durch einen Abstand zwischen den Pins, der ein wenig größer als üblich ist, angezeigt. Da dieser Umstand oft vergessen wird, überbrücken viele Nutzer diese Unterbrechung in jeder Reihe permanent mit einer Drahtbrücke.

Einige Komponenten, wie Widerstände, Kondensatoren und LEDs verfügen über lange flexible Anschlussdrähte, die sich so biegen lassen, dass alle Löcher an unterschiedlichen Stellen erreicht werden können.

Andere Komponenten, z.B. Chips, haben Anschlüsse, die sich nicht bewegen lassen und bei Techies als *Pins* bekannt sind. Zwischen diesen Pins ist fast immer ein Abstand von 2,54 mm vorhanden, so dass er auch für die Löcher auf der lötfreien Steckplatine gilt.

Die meisten Chips verfügen über zwei Reihen Pins, und wenn die Spalten auf der Platine auf ganzer Länge miteinander verbunden würden, würden (durch die Platine) die Pins auf einer Seite des Chips mit den Pins auf der anderen Seite verbunden werden. Das ist der Grund für die Unterbrechung in der Mitte, durch die alle vertikalen Lochreihen unterbrochen werden. Wenn du einen Chip so platzierst, dass er die Lücke überbrückt, werden die Pins auf der einen Seite nicht mit denen auf der anderen Seite verbunden. Clever, oder?

Einige Platinen weisen Buchstaben auf, die die Reihen kennzeichnen, und Nummern, die die Spalten kennzeichnen. Wir werden sie aber nicht nutzen, weil diesbezüglich nicht alle Platinen gleich sind. Wann immer wir den Begriff Pin-Nummer verwenden, meinen wir den Arduino-Pin und verweisen nicht auf Teile der Platine.

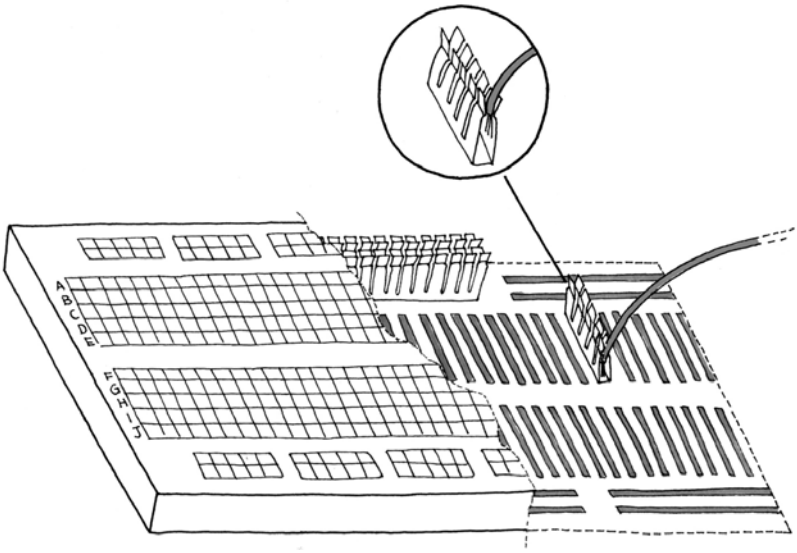


Abbildung A-1: *Die lötfreie Steckplatine*

B/Das Lesen von Widerständen und Kondensatoren

Um elektronische Bauteile verwenden zu können, musst du 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 Anschlussdrähte herausragen und der ringsherum merkwürdige farbige Markierungen aufweist. Als die ersten kommerziellen Widerstände hergestellt wurden, gab es noch keine Möglichkeit, Nummern aufzudrucken, die so klein gewesen wären, dass sie auf den Körper des Widerstands gepasst hätten. 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 ist recht einfach: Generell gibt es 4 Farbringe. Jede Farbe steht für eine Zahl. Einer der Ringe ist üblicherweise goldfarben, er repräsentiert den Toleranzbereich dieses Widerstands. Um die Ringe in der richtigen Reihenfolge zu lesen, musst du den Widerstand so halten, dass sich der goldene (oder in einigen Fällen der silberne) Ring rechts befindet. Schau dir dann die Farben an und ordne 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

Farbe	**Wert**
Blau	6
Violett	7
Grau	8
Weiß	9
Silber	10%
Gold	5%

Eine Markierung mit einem braunen, einem schwarzen, einem orangenfarbenen 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, so dass wir hier im Endeffekt einen Wert von 10.000 Ohm ±5% haben.

Elektronik-Geeks neigen dazu, diese Werte zu verkürzen, indem sie in Kiloohm (kOhm - für tausend Ohm) und Megaohm (mOhm - 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. Da Ingenieure die Optimierung lieben, wirst du unter Umständen in Schaltskizzen Werte wie 4k7 finden, was nichts anderes als 4,7 Kiloohm oder 4.700 Ohm bedeutet.

Manchmal werden dir Widerstände mit einer höheren Präzision von 1 oder 2 % begegnen. Diese Widerstände verfügen über einen fünften Ring, so dass der Wert präziser angegeben werden kann. Es handelt sich um denselben Code, nur dass die ersten drei Ringe den Wert repräsentieren und der vierte Ring die Anzahl an Nullen nach dem Wert angibt. Der fünfte Ring zeigt die Toleranz an: Rot steht für 2% und Braun für 1%. Unser 10K-Ohm-Widerstand (Braun, Schwarz, Orange und Golden) hätte als Widerstand mit einer Toleranz von 1% die Farbcodierung Braun, Schwarz, Schwarz, Rot und Braun.

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 du findest, 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 (pF) 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 du die Ziffern 1–5 liest, handelt es sich dabei um die Anzahl der Nullen. Die Ziffern 6 und 7 werden nicht verwendet, bei den Zahlen 8 und 9 wird in einer andere Weise verfahren. Wenn du eine 8 liest, musst du 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.

Als Erinnerung hier die häufig benutzten Multiplikatoren in der Elektronik.

Multiplikator	**Wert**	**Beispiel**
M (Mega)	10^6 = 1.000.000	1.200,000 Ohm = 1,2 mOhm
K (Kilo)	10^3 = 1.000	470.000 Ohm = 470 kOhm
m (Milli)	10^{-3} = 0,001	0,01 A = 10 mA
µ (Mikro)	10^{-6} = 0,000001	4.700 µ amps = 4,7 mA
n (Nano)	10^{-9}	10.000 n Farad = 10 µF
p (Pico)	10^{-12}	1.000.000 p f = 1 µF

C/Arduino-Kurzreferenz

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

Detaillierte Informationen findest du auf der Seite Language Reference bei Arduino (*http://bit.ly/1ycDNQA*).

Struktur

Ein Arduino-Sketch wird in zwei Teilen ausgeführt:

```
void setup()
```

Hier erfolgen alle Einrichtungen, die einmal erfolgen müssen, bevor die Schleife ausgeführt wird, und dann nicht mehr erforderlich sind.

```
void loop()
```

Hier ist der Hauptcode des Sketches untergebracht. Die Schleife enthält einen Satz Anweisungen, die so lange wiederholt werden, bis das Board ausgeschaltet wird.

Sonderzeichen

Arduino enthält zahlreiche Sonderzeichen, um Code-Zeilen, Kommentare und Code-Blöcke zu kennzeichnen.

; Semikolon

Jede Anweisung (Code-Zeile) wird mit einem Semikolon beendet. Durch diese Syntax lässt sich der Code frei formatieren. Du kannst sogar zwei Anweisungen in derselben Zeile platzieren, solange du sie durch ein Semikolon voneinander trennst (allerdings ist dies der Lesbarkeit des Code nicht sehr zuträglich).
Beispiel:

```
delay(100);
```

{} Geschweifte Klammern

Geschweifte Klammern werden verwendet, um Code-Blöcke zu kennzeichnen. Wenn du beispielsweise Code für die `loop()`-Funktion

schreibst, muss er 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-Mikrocontroller ignoriert, sind aber sehr hilfreich, um für dich und andere Nutzer festzuhalten, was der Code tut.
Beim 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 Ausgangs-Pin handeln soll.

Die Werte true und false werden verwendet, um zu testen, ob eine Bedingung oder ein Ausdruck wahr oder falsch ist. Sie werden hauptsächlich zusammen mit *Vergleichsoperatoren* verwendet.

Variablen

Variablen sind benannte Bereiche des Arduino-Speichers, in denen Daten gespeichert werden können. Dein Sketch kann diese Daten verwenden und ändern, indem er über den entsprechenden Variablen-Namen darauf zugreift. Wie der Name *Variable* schon sagt, können Variablen beliebig oft geändert werden.

Weil Arduino ein sehr einfacher Mikrocontroller ist, musst du bei der Variablen-Deklaration auch den entsprechenden Typ angeben. Es ist also erforderlich, dem Mikrocontroller die Größe des zu speichernden Wertes mitzuteilen.

Im Folgenden sind die verfügbaren *Datatypen* aufgelistet:

boolean

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

char

Er enthält ein einzelnes Zeichen, z.B. den Buchstaben 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. Ein Char belegt 1 Byte an Speicher.

Auf Computern stehen zwei größere Zeichensätze zur Verfügung: ASCII und UNICODE. ASCII umfasst 127 Zeichen, die unter anderem dazu verwendet werden, Text zwischen seriellen Endgeräten und Computer-Systemen wie Großrechnern oder Minicomputern zu übertragen. UNICODE umfasst einen viel größeren Satz an Zeichen und wird von modernen Computer-Betriebssystemen verwendet, um Zeichen in einer Vielzahl von Sprachen darzustellen. Dennoch ist auch ASCII nützlich beim Austausch 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. Anders als char kann byte nur positive Zahlen speichern.

int

Hier werden 2 Bytes an Speicher genutzt, um eine Zahl zwischen –32,768 und 32,767 darzustellen. Es ist der bei Arduino am häufigsten benutzte Datentyp. Wenn du dir unsicher bist, welchen Datentyp du benutzen sollst, versuche es mit int .

unsigned int

Wie bei int werden auch hier 2 Bytes an Speicher 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 netter Ausdruck, um zu beschreiben, dass in float Zahlen mit einem Dezimalkomma gespeichert werden können) enthalten kann. Solche Variablen nutzen 4 Bytes deines wertvollen RAM, und die Funktionen, die mit ihnen arbeiten können, verbrauchen ebenfalls viel Code-Speicher. Du solltest floats daher nur verwenden, wenn es erforderlich ist.

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 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 1 Byte für jedes Zeichen in der Zeichenfolge verwendet plus ein Nullzeichen (1 Byte) am Ende, 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 du beispielsweise verschiedene Helligkeitsstufen speichern möchtest, die zum Dimmen einer LED verwendet werden sollen, könntest du sechs Variablen mit den Namen `light01`, `light02` usw. erzeugen. Besser ist aber die Verwendung eines einfachen Arrays wie das folgende:

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

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

Arrays sind ideal, wenn du eine bestimmte Aktion für eine Menge an Daten in ihrer Gesamtheit durchführen möchtest, weil du das, was du tun möchtest, nur einmal schreiben musst, und du die gewünschte Aktion dann für jede Variable im Array durchführen kannst, beispielsweise sie in einer `for` Schleife verwenden, indem du z.B. einfach den Index änderst.

Der Gültigkeitsbereich von Variablen

Variablen in Arduino besitzen eine Eigenschaft, die *Scope* genannt wird. Variablen können lokal oder global sein, abhängig davon, wo sie deklariert wurden.

Eine globale Variable ist für jede Funktion im Programm sichtbar und kann von ihr verwendet werden. Lokale Variablen sind nur für die Funktion sichtbar, in der sie deklariert wurden.

Wenn Programme länger und komplexer werden, bieten lokale Variablen eine hilfreiche Möglichkeit, sicherzustellen, dass jede Funktion Zugriff auf ihre eigenen Variablen hat. Dadurch lassen sich Programmierfehler vermeiden, die entstehen, wenn eine Funktion unbeabsichtigt die Variablen ändert, die von einer anderen Funktion genutzt werden. Variablen, die von mehreren Funktionen genutzt werden müssen, können als globale Variablen deklariert werden.

In der Arduino-Umgebung handelt es sich bei jeder Variablen, die außerhalb einer Funktion (z.B. `setup()`, `loop()` oder deiner eigenen Funktionen) deklariert wurde, um eine globale Variable. Jede Variable, die innerhalb einer Funktion deklariert wurde, ist eine lokale Variable, auf die nur innerhalb dieser Funktion zugegriffen werden kann.

Manchmal ist es außerdem ganz praktisch, eine Variable innerhalb einer `for`-Schleife zu deklarieren und zu initialisieren. Dadurch entsteht eine Variable, auf die nur innerhalb der Klammern der `for`-Schleife zugegriffen werden kann. Tatsächlich handelt es sich bei einer Variablen immer dann, wenn sie in *geschweiften Klammern* deklariert wurde, um eine lokale Variable, die nur für diesen Code-Block sichtbar ist.

Kontrollstrukturen

Arduino enthält Schlüsselwörter für das Steuern des Logikflusses deines Sketches.

if ... else

Mit dieser Struktur werden in deinem 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 Code-Block fortgefahren. Die `else`-Anweisung ist optional. Beispiel:

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

for

Bei dieser Struktur wird der Code-Block 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, deinen Code übersichtlich zu halten, da sie lange Listen von `if`-Anweisungen ersetzt. Es ist wichtig, an die `break`-Anweisung am Ende jeder `Case`-Anweisung zu denken, weil anderenfalls Arduino die Anweisungen der nachfolgenden `case`-Strukturen ausführen wird, bis ein `break` oder das Ende von `switch case` erreicht wird.
Beispiel:

```
switch (sensorValue) {
    case 23:
      digitalWrite(13,HIGH);
      break;
    case 46:
      digitalWrite(12,HIGH);
      break;
    default: // if nothing matches this is executed
      digitalWrite(12,LOW);
      digitalWrite(13,LOW);
}
```

while

Ähnlich wie bei `if` wird ein Code-Block ausgeführt, wenn eine bestimmte Bedingung wahr (`true`) ist. Bei `if` wird jedoch der Block nur einmal ausgeführt, während bei `while` die Ausführung des Blocks so lange erfolgt, bis die 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

Hiermit kannst du aus einer `while`- oder `for`-Schleife ausbrechen, auch wenn die `loop`-Bedingung besagt, dass die Schleife weiter ausgeführt wird. Sie wird außerdem 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
   if (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`, dassder 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);
}
```

Bei `continue` sind Ähnlichkeiten mit `break` vorhanden. Während bei `break` allerdings die Schleife verlassen wird, wird bei `continue` mit dem nächsten Durchlauf der Schleife fortgefahren.

return

Bei dieser Anweisung wird die Ausführung einer Funktion gestoppt und zu dem Code-Abschnitt zurückgekehrt, von dem die Funktion aufgerufen wurde. Du kannst diese Struktur auch verwenden, um einen Wert, der aus einer Funktion stammt, zurückzuliefern.
Wenn du beispielsweise bei einer Funktion mit dem Namen `computeTemperature()` das Ergebnis an den Teil deines Codes ausgeben möchtest, über den die Funktion aufgerufen wurde, würdest du etwa Folgendes schreiben:

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

Arithmetik und Formeln

Du kannst Arduino für komplexe Berechnungen verwenden, indem du eine spezielle Syntax verwendest. Die Zeichen + und - funktionieren genauso, wie du es in der Schule gelernt hast: Multiplikation wird mit einem * und Division mit einem / dargestellt.

Es gibt noch einen zusätzlichen Operator, der als Modulo (%) bezeichnet wird. Er liefert den Rest aus einer Division von Ganzzahlen zurück. Wie du es in Algebra gelernt hat, kannst du beliebig viele Klammern benutzen, um Ausdrücke korrekt zu schachteln. Im Gegensatz zu dem, was du möglicherweise in der Schule gelernt hast, werden eckige und geschweifte Klammern nicht für arithmetische Formeln verwendet, weil sie für andere Zwecke reserviert sind (z.B. für Array-Indizes und Blöcke).

Beispiel:

```
a =  2 + 2;
light = ((12 * sensorValue) - 5 ) / 2;
remainder = 7 % 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/gleich
>=	größer/gleich

Wenn du auf Gleichheit testest, musst du sorgfältig darauf achten, dass du den Vergleichsoperator == und nicht den Zuweisungsoperator = verwendest, weil sich dein Programm anderenfalls nicht erwartungsgemäß verhält.

Boolesche Operatoren

Dieser Operator wird verwendet, wenn du mehrere Bedingungen verknüpfen möchtest. Wenn du beispielsweise überprüfen möchtest, ob der von einem Sensor zurückgelieferte Wert zwischen 5 und 10 liegt, würdest du Folgendes schreiben:

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

Es gibt drei Boolesche Operatoren: und – durch && dargestellt; oder – durch || repräsentiert; und schließlich nicht – durch ! dargestellt.

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, würdest du Folgendes schreiben:

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

Unter Verwendung eines kombinierten Operators wird daraus diese vereinfachte Version:

```
value++;
```

Es ist völlig in Ordnung, diese kombinierten Operatoren nicht zu verwenden, aber sie sind so häufig, dass es für dich als Einsteiger sehr schwierig werden würde, etwas anhand von Beispielen zu lernen, wenn du diese Operatoren nicht verstehst.

Inkrementieren und Dekrementieren (-- und ++)

Mit diesen Operatoren wird jeweils um den Wert 1 erhöht oder reduziert. Sei aber vorsichtig – sie können beide vor oder hinter einer Variablen eingesetzt werden, aber es besteht ein sehr feiner Unterschied: Wenn du `i++` schreibst, wird zunächst `i` um 1 inkrementiert und dann in Bezug auf das Äquivalent von `i+1` ausgewertet. Bei `++i` wird zunächst in Bezug auf den Wert von `i` ausgewertet und anschließend `i` inkrementiert. Dasselbe gilt für --.

+= , –=, *=, und /=

Diese Operatoren funktionieren ähnlich wie ++ und --, aber sie ermöglichen dir, Werte um mehr als 1 zu erhöhen oder zu reduzieren, sowie Multiplikation und Division. Die beiden folgenden Ausdrücke sind gleichbedeutend:

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

Input- und Output-Funktionen

Eine der Hauptaufgaben von Arduino besteht darin, Informationen von Sensoren als Eingang zu verarbeiten und Werte an Aktoren auszugeben. Du hast in diesem Buch bereits einige entsprechende Beispiel-Programme gesehen.

pinmode(pin, mode)

Hiermit wird ein digitaler Pin (neu) definiert, so dass er dann als Eingangs- oder Ausgangs-Pin dient.
Beispiel:

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

Das Vergessen der Festlegung der Pins als Ausgang mit `pinMode()` ist einer der häufigsten Gründe für eine fehlerhafte oder nicht funktionierende Ausgabe.
Auch wenn sie normalerweise in `setup()` verwendet wird, lässt sich die Funktion `pinMode()` auch in einer Schleife benutzen, wenn das Verhalten der Pins geändert werden muss.
(Wenn ein Funktionsname im Fließtext verwendet wird, wird er oft mit einer leeren Klammer am Ende gekennzeichnet, um anzuzeigen, dass an der betreffenden Stelle von einer Funktion die Rede ist.)

digitalWrite(pin, value)

Hiermit wird ein digitaler Pin auf HIGH oder LOW gesetzt. Pins müssen mittels `pinMode()` explizit als Output definiert werden, bevor mit `digitalWrite()` der erwartete Effekt erzielt werden kann.
Beispiel:

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

Beachte, dass HIGH oder LOW zwar normalerweise AN bzw. AUS entsprechen, dies aber davon abhängt, wie die Pins genutzt werden. Wenn

beispielsweise eine LED zwischen 5V und einem Pin montiert ist, wird sie eingeschaltet, wenn beim Pin der Spannungspegel LOW ist, und ausgeschaltet, wenn der Spannungspegel HIGH ist.

int digitalRead(pin)

Hiermit wird der Zustand eines Eingangs-Pins ausgelesen und 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 1023 zurück, der eine Spannung zwischen 0 und 5V 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 es sich bei `pin` nur um einen Pin handeln, der PWM unterstützt, das sind Pin 3, 5, 6, 9, 10 oder 11 auf dem Uno und Pin 3, 5, 6, 9, 10, 11 oder 13 auf dem Leonardo. Bei `value` muss es sich um eine Zahl zwischen 0 und 255 handeln. Du kannst diesbezüglich an einen Wert denken, der die durchschnittliche Menge an Strom repräsentiert, die der Arduino bereitstellen wird, wobei ein Wert von Null vollständig ausgeschaltet und ein Wert von 255 vollständig eingeschaltet entspricht. Ist `value` 0, wird der Ausgang vollständig auf LOW gesetzt, bei einem Wert von 255 hingegen vollständig auf HIGH.
Beispiel:

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

shiftOut(dataPin, clockPin, bitOrder, value)

Diese Funktion sendet Daten an ein *Schieberegister*, d.h. an ein logisches Schaltwerk, das verwendet wird, um die Anzahl der digitalen Outputs zu erweitern. Dieses Protokoll benutzt einen Pin für Daten und und einen als Taktgeber. Mit `bitOrder` wird die Reihenfolge der Abarbeitung (last significant bit oder most significant bit) bestimmt, und in `value` sind die zu sendenden Daten 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 Infrarot-Sensor oder ein Beschleunigungsmesser ausgelesen werden soll, bei dem die Werte in Form von Impulsen bezogen auf die Änderungsdauer ausgegeben werden.

Beispiel:

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

Zeitfunktionen

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

unsigned long millis()

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

```
duration = millis()-lastTime;
        // berechnet die vergangene Zeit 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(µs)

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 benutzte 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 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
```

Zufallszahlfunktionen

Zum Erzeugen von Zufallszahlen kannst du den Pseudo-Zufallszahlen-Generator von Arduino verwenden. Zufallszahlen sind nützlich, wenn dein Projekt sich jedes Mal, wenn du es nutzt, anders verhalten soll.

randomSeed(seed)

Hiermit wird der Pseudo-Zufallszahlen-Generator von Arduino zurückgesetzt. Die Verteilung der von `random()` zurückgelieferten Zahlen ist zwar grundsätzlich zufällig, aber die Abfolge ist vorhersehbar. Daher solltest du den Generator auf einen Zufallswert zurücksetzen. Eine gute Basis ist dabei ein Wert, der von einem nicht verbundenen analogen Eingang gelesen wird, weil ein nicht verbundener Pin zufällige Geräusche aus der Umgebung (Radiowellen, kosmische Strahlung, elektromagnetische Interferenzen von Mobiltelefonen und fluoreszierendem Licht usw.) auffangen kann, und das Ergebnis daher unvorhersehbar sein wird.

Beispiel:

```
randomSeed(analogRead(5));
        // erzeugt Zufallszahlen mithilfe von Geräuschen 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 du in Kapitel 5, *Erweiterter Input und Output*, auf Seite 47 gesehen hast, kannst du 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 9.600 Baud (Bits pro Sekunde), es stehen aber auch andere Werte zur Verfügung, üblicherweise aber nicht mehr als 115.200 bps. Die Baudrate spielt dabei keine Rolle, solange beide Seiten übereinstimmen und hier dieselbe Rate verwenden.

Beispiel:

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

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

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

Beispiele (Beachte, dass im finalen Beispiel `Serial.write` verwendet wird):

```
Serial.print(75);       // Prints "75"
Serial.print(75, DEC);  // The same as above.
Serial.print(75, HEX);  // "4B" (75 in hexadecimal)
Serial.print(75, OCT);  // "113" (75 in octal)
Serial.print(75, BIN);  // "1001011" (75 in binary)
Serial.write(75);       // "K" (the letter K happens
                        // to be 75 in the ASCII set)
```

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 werden, als wenn nach der Dateneingabe die Return- oder Enter-Taste gedrückt worden wäre.

Beispiele:

```
Serial.println(75);       // Prints "75\r\n"
Serial.println(75, DEC);  // The same as above.
Serial.println(75, HEX);  // "4B\r\n"
Serial.println(75, OCT);  // "113\r\n"
Serial.println(75, BIN);  // "1001011\r\n"
```

int Serial.available()

Diese Funktion liefert zurück, wie viele Daten am seriellen Anschluss für das Auslesen mittels der `read()`-Funktion bereitstehen. Nachdem mit `read()` alle verfügbaren 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 1 Byte der eingehenden seriellen Daten abgerufen.

Beispiel:

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

Serial.flush()

Da die Daten am 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();
```

D/Das Lesen von Schaltskizzen

Im überwiegenden Teil dieses Buches haben wir sehr detaillierte Illustrationen angeführt, um zu beschreiben, wie dein Schaltkreis aufgebaut werden muss. Du kannst dir aber bestimmt vorstellen, dass es schon recht zeitaufwendig ist, für jedes Projekt, das du möglicherweise dokumentieren möchtest, eine Schaltskizze zu zeichnen.

Ähnliche Probleme werden früher oder später in jeder Disziplin auftauchen. Wenn du beispielsweise einen schönen Song geschrieben hast, musst du ihn mittels Musiknoten zu Papier bringen.

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

Im Bereich Elektronik ermöglichen *Schaltpläne* (oder *schematische Darstellungen*), Schaltkreise in einer Weise zu beschreiben, die von den anderen Personen einer Community verstanden wird. Einzelne Komponenten werden in Form von Symbolen dargestellt, bei denen es sich um eine Art Abstraktion der tatsächlichen Form der Komponenten 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 in Abbildung D-1 aus.

Abbildung D-1: *Schaltkreissymbol für einen Kondensator*

Ein weiteres schönes Beispiel ist der Induktor, der aus einem um einen Zylinder gewickelten Kupferdraht besteht. Das entsprechende Symbol sieht folgerichtig wie in Abbildung D-2 aus.

Abbildung D-2: *Schaltkreissymbole für einen Induktor*

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 du in Abbildung D-3 sehen kannst.

Abbildung D-3: *Das Schaltkreissymbol für verbundene Drähte*

Dies sind alle Informationen, die du für das Verständnis von Basis-Schaltskizzen benötigst. In Abbildung D-4 findest du Schaltkreissymbole für Komponenten, die häufig bei Arduino-Schaltkreisen verwendet werden.

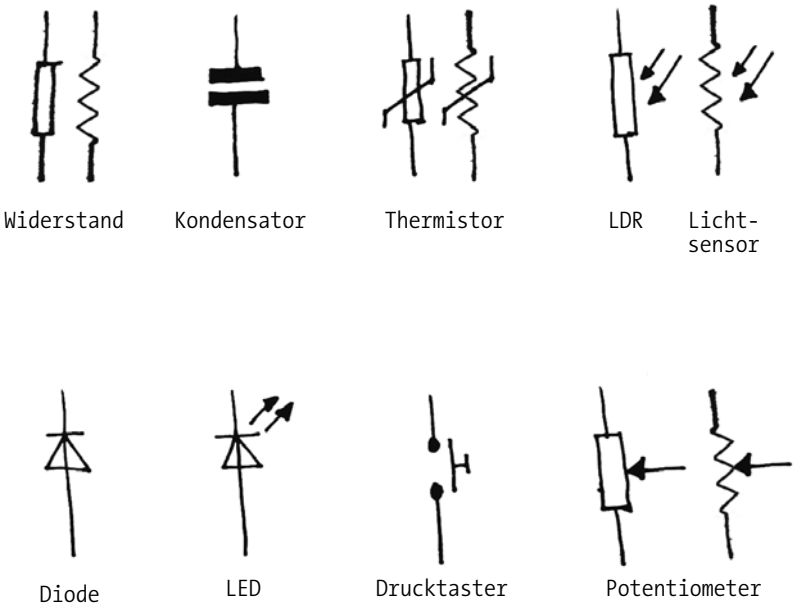


Abbildung D-4: *Häufige Schaltkreissymbole bei Arduino-Schaltkreisen*

Möglicherweise werden dir Variationen dieser Symbole begegnen (z.B. die beiden hier aufgeführten Symbole für den Widerstand). Eine umfangreichere Liste von Elektronik-Symbolen findest du in Wikipedia (*http://bit.ly/1zVpAa3*).

Zusätzlich gibt es auch (eine Art) Standardsatz von Symbolen, bei denen es sich um Konventionen handelt, wie Schaltpläne gestaltet werden sollten. Schaltpläne werden so gezeichnet, dass die Informationen von links nach rechts verlaufen. Beim Zeichnen eines Radios würdest du demnach mit der Antenne auf der linken Seite beginnen und dann mit dem Weg fortfahren, den das Radiosignal bis zum Lautsprecher, bei dem es sich um das letzte Bauteil auf der rechten Seite handelt, zurücklegt.

In Abbildung D-5 ist der Schaltkreis für den Drucktaster dargestellt, der weiter vorne in diesem Buch beschrieben wurde.

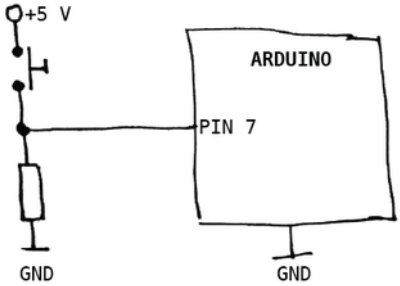


Abbildung D-5: *Ein am digitalen Eingang des Arduino angeschlossener Drucktaster*

Du siehst, dass der Arduino auf einen Kasten mit Pin und GND reduziert wurde, weil dies die einzig wichtigen Teile des Arduino sind, die du bei diesem speziellen Schaltkreis kennen musst. Du siehst außerdem zwei Kabel, die am Label GND angesteckt dargestellt sind. Das bedeutet, dass die Kabel miteinander verbunden sind. Diese Darstellung mithilfe von Labels ist sehr hilfreich bei Verbindungen, die häufig benutzt werden (z.B. Masse) oder die von einer Seite des Schaltplans zur entgegengesetzten Seite verlaufen und dabei viele andere Kabel und Komponenten kreuzen.

In Kapitel 8, *Ein System zur automatischen Gartenbewässerung*, auf Seite 93 werden viele praktische Beispiele für Schaltpläne gezeigt und in Abschnitt *Der elektronische Schaltplan*, auf Seite 106 werden Schaltpläne ein wenig detaillierter erläutert.

Index

D

E

F

G

H

I

K

L

M

N

O

P

W

X

Z

Über die Autoren

Massimo Banzi ist Mitbegründer des Arduino-Projekts und hat bereits für Kunden wie Prada, Artemide, Persol, Whirlpool, V&A Museum und Adidas gearbeitet. Er war vier Jahre beim Interaction Design Institute Ivrea als Associate Professor tätig. Massimo Banzi hat zahlreiche Workshops durchgeführt und wurde schon von vielen Bildungsinstitutionen wie Architectural Association, London; Hochschule für Gestaltung und Kunst, Basel; Hochschule für Gestaltung, Schwäbisch Gmünd; FH Potsdam; Domus Academy; Medialab, Madrid; Escola Superior de Disseny, Barcelona; ARS Electronica, Linz; Mediamatic, Amsterdam, und Doors of Perception, Amsterdam, als Redner eingeladen. Bevor er zum Interaction Design Institute Ivrea kam, war er CTO bei Seat Ventures Incubator. Er arbeitete viele Jahre als Software-Architekt sowohl in Mailand als auch in London für Auftraggeber wie Italia Online, Sapient, Labour Party, BT, MCI WorldCom, SmithKlineBeecham, Storagetek, BSkyB und boo.com.

Michael Shiloh ist Associate Professor am California College of the Arts, wo er Elektronik, Programmierung, Robotik und Elektromechanik lehrt. Ursprünglich als Elektronik-Ingenieur ausgebildet, arbeitete Michael Shiloh für verschiedene Gerätehersteller, bevor er seine Leidenschaft für die Lehre entdeckte. Er wird weltweit zu Konferenzen eingeladen und lehrt an zahlreichen Universitäten. Seit 2013 arbeitet Michael Shiloh für Arduino, er bringt die Open-Source-Plattform neuen Zielgruppen näher und leitet Arduino-Workshops.

Die Schrift auf dem Cover und im Text ist Benton Sans, die Überschriftenschrift ist Serifa und die Code-Schrift ist TheSans Mono.

Arduino Starterset

Die Bauteile zum Buch

Hier gibt es die Bauteile für die Projekte aus diesem Buch. Sorgfältig und praxisgerecht zusammengestellt und übersichtlich in einer Sortimentsbox verpackt.

Optional erhältlich sind: Tiny RTC-Modul, DHT11-Modul, 12V-Magnetventile und alle anderen Arduino-Produkte.

Arduino Starterset V3 € 69,00

142 Teile incl. Arduino Uno

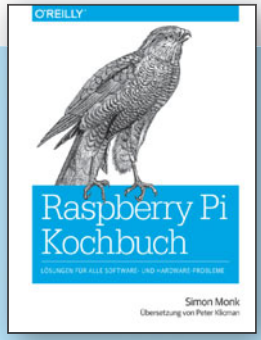
Inhalt:

1	Arduino Uno Rev.3
1	Steckbrett mit 830 Kontakten
1	USB-Kabel 1m A-B
1	Drahtbrücken-Satz (140tlg.)
1	Neigungsschalter
1	Piezoscheibe PLS 3112
20	Widerstände 220 Ω
10	Widerstände 1k Ω
20	Widerstände 10k Ω
1	10k Ω NTC-Widerstand
2	10k Ω Trimmer liegend
2	Steckachsen für die Trimmer
3	Kurzhubtaster 6x6 mm
2	Kurzhubtaster 12x12 mm
1	Krokoklemmen-Set (10tlg. farbig)
1	Relais mit 5V Spulenspannung
1	Motor 1-9V; 2mm Achse; 8000rpm
5	100nF Keramikkondensator
5	10nF Keramikkondensator
5	100uF/35V Elektrolytkondensator
1	Batteriehalter für 6 Mignon-Batterien
6	Mignon-Batterien
1	Anschluss-Clip 9V
2	Optokoppler PC817
10	Universaldioden (1N4007)
3	MOS-FET Transistoren 2N7000
2	MOS-FET Transistoren IRL540N
10	NPN-Transistoren BC337-40
5	LEDs 5mm rot
5	LEDs 5mm gelb
5	LEDs 5mm grün
5	LEDs 5mm blau
1	RGB-LED 4pin (gem.Kathode)
1	Fotowiderstand
3	m Klingeldraht 2x0,6mm ws/rt
1	Sortimentsbox mit var.Einteilung

ERST CODEN, DANN LÖTEN

MAKE & DIY

Elektronikbücher von O'Reilly



O'REILLY®

www.oreilly.de